Institut für Statistik

Ludwig-Maximilians-Universität München

Entwicklung einer Energiemonitoring-Software für Big Data in der universitären Betriebstechnik mittels des relationalen Datenbankmanagementsystems MySQL und der Statistiksoftware R

Masterarbeit

Thomas Rottner

21. Oktober 2015

In Zusammenarbeit mit Paul Eichel von der Betriebstechnik Stammgelände der Ludwig-Maximilians-Universität München

Betreuung Prof. Dr. Christian Heumann

Zusammenfassung

Im Laufe der Gebäudelebensdauer von 60 Jahren betragen die Betriebskosten eines Verwaltungsgebäudes etwa das Zweifache der Baukosten. Somit dient eine laufende Überwachung der Verbräuche neben dem Klimaschutz auch dem Einsparen finanzieller Mittel.

Das Ziel der vorliegenden Arbeit war es, eine Energiemonitoring-Software zu entwickeln, die Berichte zum Energieverbrauch der Gebäude der Ludwig-Maximilians-Universität erzeugt. Dabei wurde insbesondere darauf geachtet, dass die Software möglichst leicht erweiterbar ist und die gemessenen Daten in einer Weise gespeichert und weiterverarbeitet werden, die sich für Messungen im Minutentakt und Sammlung der Daten über mindestens ein Jahrzehnt eignen. Des Weiteren sollte sie Daten aus verschiedenen Quellen in Verbindung bringen können, da die Daten der LMU aufgrund der Größe der Universität dezentral und über verschiedene Systeme erfasst und gespeichert werden. Um die leichte Erweiterbarkeit zu ermöglichen, wurde die Statistiksoftware R verwendet, da sie als Interpretersprache mit tausenden Zusatzpaketen den Programmieraufwand minimiert. Um die anfallenden Datenmengen zu bewältigen, wurde das relationale Datenbankmanagementsystem MySQL verwendet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Im Laufe der Gebäudelebensdauer von 60 Jahren betragen die Betriebskosten eines Verwaltungsgebäudes bei einer angenommenen jährlichen Preissteigerungsrate von 2% etwa das Zweifache der Baukosten. Deshalb müssen die Verbräuche während der Betriebsphase laufend überwacht werden, um ggf. Verbrauchsminderungsmaßnahmen einleiten zu können. Dies trägt sowohl der Knappheit der Haushaltsmittel, als auch dem Klimaschutz Rechnung [1].

In der vorliegenden Arbeit wurde die Energiemonitoring-Software EnMoLMU entwickelt, um eine solche Verbrauchsüberwachung für die Gebäude der Ludwig-Maximilians-Universität durchführen zu können. Dies erfolgt über regelmäßig automatisch generierte Berichte.

Die LMU verfügt über 123 Gebäude (Postadressen) mit 274 Bauteilen. Um hier Fehlbetriebe effizient beheben zu können, muss ein guter Überblick bestehen. Darüber hinaus werden die relevanten Daten aufgrund der Größe und Ausdehnung der Universität dezentral und mithilfe unterschiedlicher Systeme erfasst und gespeichert. Diese konnten bisher nur getrennt voneinander betrachtet werden.

Besondere Bedeutung bei der Umsetzung der Software hatten somit der Umgang mit den verhältnismäßig großen Datenmengen, welche langfristig eine geschätzte Größe im mindestens dreistelligen Gigabyte-Bereich annehmen dürften, die Möglichkeit die Daten aus unterschiedlichen Quellen in Verbindung zu bringen und eine möglichst leichte Erweiterbarkeit mit möglichst kurzer Programmierzeit.

Um die anfallenden Datenmengen zu bewältigen, wurde das relationale Datenbankmanagementsystem MySQL verwendet, das zudem als "populärste Open-Source-Datenbank der Welt"[3] von vielen Leuten beherrscht wird. Um die leichte Erweiterbarkeit zu ermöglichen, wurde die Statistiksoftware R[27] verwendet, da sie als Interpretersprache mit

tausenden Zusatzpaketen (im Januar 2015 über 6000 alleine auf CRAN[2]) den Programmieraufwand minimiert, indem sie für die meisten Aufgaben bereits fertige Funktionen bietet, viel Information zu Ihrer Verwendung im Internet verfügbar ist und viele Komplikationen anderer Sprachen (Zeiger, Zwangs-Objektorientierung usw.) vermeidet und der Code leicht Zeile für Zeile in der Console ausprobiert und somit entwickelt werden kann.

Zuerst wird im nächsten Kapitel **Aufbau der Software** (2) kurz der Aufbau der Software, also die einzelnen R-Pakete und der grobe Ablauf bei der Anwendung skizziert. In Kapitel **Daten**3 werden die hauptsächlich verwendeten Daten und deren Quellen beschrieben. Das nächste Kapitel (**Anwendung der Software**4) dient als Anleitung zur Verwendung der vollständig eingerichteten Software und zur Vorbereitung und Einrichtung von EnMoLMU, zeigt einen Beispielbericht und die gegenwärtig verfügbaren Grafikfunktionen. Eine kurze Abhandlung der verwendeten statistischen Methoden befindet sich im Kapitel **Verwendete statistische Methoden**5. Eine Einführung und Anleitung für besonders relevante Aspekte der zur Entwicklung der Software verwendeten Hilfsmittel, wie R und MySQL, wird in Kapitel **Verwendete Werkzeuge** 6 gegeben. Kapitel **Ausblick**7 schließt den Hauptteil der vorliegenden Arbeit mit einem Ausblick zur Weiterentwicklung der Software ab.

Im **Anhang** (A) befinden sich u.a. die Dokumentationen zu den zu EnMoLMU gehörigen R-Paketen und R- und Batch-Skripte sowie eine Berichtsvorlage.

Der **elektronische Anhang** beinhaltet einen Beispielbericht als Vorlage und als fertig erstellte PDF-Datei, weitere Diagramme zu Abläufen, gegenseitigen Funktionsaufrufen, Skripten und Funktionen, die R-Pakete von EnMoLMU mit ihrer Dokumentation und Batch- und R-Skripte. Auch die R-Codes für die im Ausblick vorgestellten interaktiven Grafiken, die zur Erklärung der statistischen Methoden erstellten Grafiken und die Tests der Methoden zur Schätzung der Famos-Zähler sind vorhanden. Weiter ist die vorliegende Arbeit als PDF-Datei enthalten.

2 Aufbau der Software

Die im Zuge dieser Masterarbeit entwickelte Software **EnMoLMU** (für **En**ergie-**Mo**nitoring **L**udwig-**M**aximilians-**U**niversität München) ist in erster Linie in der statistischen Programmiersprache R implementiert und kann in die Bereiche Daten aufbereiten und speichern, Grafiken zur Auswertung erzeugen und Berichte erstellen gegliedert werden. Daraus ergibt sich auch die Aufteilung in die R-Pakete *enmoDaten*, *enmoGrafiken*, *enmoFamos*, *enmoBerichte*, *enmoSonstiges* und schließlich *EnMoLMU*, das nur die anderen Pakete lädt (siehe A.1).

In *enmoDaten* sind Funktionen implementiert, um über ein R-Skript die aus der Gebäudeleittechniksoftware ProGrafNT (GLT) exportierten Daten und die Wetterdaten des Meteorologischen Instituts München aufzubereiten und in eine MySQL-Datenbank abzuspeichern. Das Paket *enmoGrafiken* enthält Funktionen, die diese Daten abfragen, um damit Grafiken zur Darstellung des Energieverbrauchs erstellen zu können. *enmoFamos* enthält eine weitere Grafik, die sowohl Daten aus der GLT- als auch der Gebäudemanagement-Software Famos verwenden kann. Diese Funktionen können in die Berichtsvorlagen im R Markdown-Format eingebunden werden, so dass mit den Funktionen aus *enmoBerichte* über R- und Batch-Skripte regelmäßig automatisiert Berichte in unterschiedlichen Formaten erstellt werden können. Die einzelnen Berichte können eine beliebige Kombination von Auswertungen enthalten und stellen eine Zusammenstellung einer Reihe von Auswertungen für eine bestimmte Person oder einen bestimmten Bereich dar. Das letzte R-Paket aus EnMoLMU, *enmoSonstiges*, beinhaltet schließlich eine Reihe von Hilfsfunktionen zum Erstellen von Paketdokumentationen, Einrichten von EnMoLMU, etc. Die Diagramme in Abb. 2.1 und Abb. 2.2 skizzieren den Aufbau der Software.

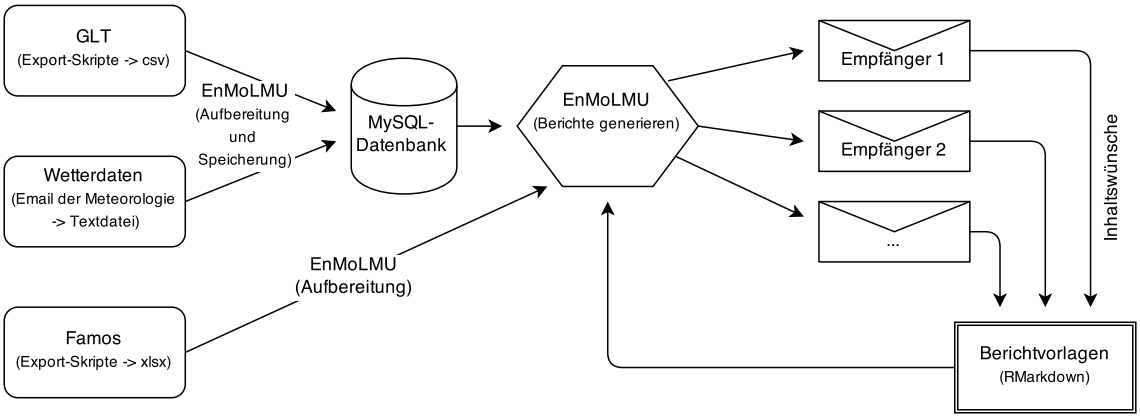


Abbildung 2.1: Die Daten werden von den beiden Programmen GLT und FAMOS gespeichert, von wo sie über Skripte im CSV-Format exportiert werden. Die Wetterdaten des Meteorologisches Institut München werden per E-Mail im Textformat verschickt. Während die Wetterdaten und die automatisch abgelesenen Zähler- und Messdaten der GLT von EnMoLMU erst aufbereitet und in einer MySQL-Datenbank abgelegt werden, aus der sie zur Berichterstellung ausgelesen werden, werden die manuell abgelesenen Zähler etc. aus FAMOS direkt aus den CSV-Dateien weiterverarbeitet. Die Empfänger der Berichte können die gewünschten Auswertungen in die R Markdown-Vorlagen eintragen lassen und erhalten dann die von der EnMoLMU generierten Berichte.

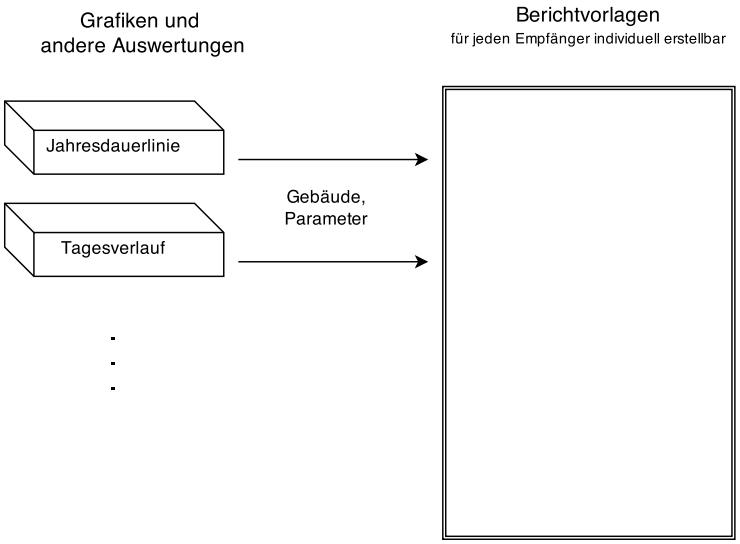


Abbildung 2.2: Die Berichterstellung erfolgt mithilfe von Vorlagen im R Markdown-Format, in welche die R-Grafikfunktionsaufrufe mit den gewünschten Argumenten (Gebäude, Zähler, etc.) als Bausteine eingefügt werden können.

3 Daten

Die Daten, welche für die Auswertungen in den zu generierenden Berichten verwendet werden, werden im Falle der automatisch abzulesenden Zähler von der Gebäudeleittechnik-Software ProGrafNT der Firma Neuberger (im Folgenden als GLT bezeichnet) erhoben und gespeichert, und im Falle der manuell abzulesenden Zähler mithilfe der Gebäudemanagement-Software Famos gespeichert. In den GLT-Daten sind bereits Außentemperaturmessungen enthalten. Weitere Wetterdaten werden vom Meteorologischen Institut der LMU per E-Mail bezogen.

3.1 ProGrafNT-Daten

Die von ProGrafNT erhobenen und gespeicherten Daten werden über ein für die EnMoLMU-Software von der Firma Neuberger geschriebenes Skript am Ende jedes Tages gesammelt in eine CSV-Datei je Subserver geschrieben. Die Daten werden dezentral über einzelne Subserver, die jeweils eine Reihe von Gebäuden zusammenfassen, erfasst und gespeichert. Für EnMoLMU werden aus den Subservern die relevanten Messdaten ausgewählt. Für diese Arbeit wurden beispielhaft die Subserver Leo03, The46 und Oet67 verwendet. Die Messungen werden im Minutentakt erhoben und in die CSV-Dateien ausgegeben. Diese Dateien enthalten eine Spalte mit dem Datum und der Uhrzeit und dem Spaltennamen *Datum* im Format `"Tag.Monat.Jahr Stunde:Minute:Sekunde"`, z. B. `"21.09.2014 00:00:00"`, und je eine weitere Spalte für alle ausgewählten Messdaten.

Die historischen Messwerte für die ausgewählten Messdaten können für die letzten zwei Jahre mithilfe eines weiteren Skriptes, das ebenfalls von der Firma Neuberger für EnMoLMU geschrieben wurde, exportiert werden. Diese werden allerdings in einem anderen Format als CSV-Dateien gespeichert. Hier beinhaltet die Datumsspalte nur das Datum ohne die Uhrzeit (z. B. "20.02.2013") und es gibt jeweils eine weitere Spalte für jede Minute

des Tages (z. B. `"00:00"`), also insgesamt 1.440 Spalten mit den gemessenen Werten. Die einzige zusätzlich vorhandene Spalte ist der Namen des Messwertes. Somit beinhalten diese Dateien eine Zeile für jede Messwert-Datum-Kombination.

Diese Daten werden von EnMoLMU in einer MySQL-Datenbank zusammengeführt, um das Speichern großer Datenmengen (z. B. 10 Jahre im Minutentakt, was 5.256.000 Zeilen entspricht) zu ermöglichen. Des weiteren sollten möglichst wenige Daten zeitgleich im RAM sein müssen, was bei den ursprünglichen Plänen, die Software auf einem 32-Bit Windows 7 Rechner auszuführen, besonders hohe Priorität hatte, aber auch bei dem schließlich verwendeten 64-Bit "Microsoft Windows Server 2012 R2"-Betriebssystems auf einem virtuellen Server scheint es nicht sinnvoll die im CSV-Format auf einen mindestens dreistelligen GB-Bereich geschätzten Datenmengen vollständig in den Hauptspeicher zu laden, insbesondere, da R Hauptspeicher nicht gerade sparsam verwendet.

Die geschätzte langfristig zu erwartende Größe der Daten ist eine lineare Hochrechnung einer CSV-Datei für einen einzelnen Subserver und den Messwerten für einen einzelnen Tag. Diese hat bei etwa 150 Zählpunkten eine Größe von ca. 1 MB. Rechnet man dies auf die erwarteten ungefähr 7.500 Zählpunkte für alle Subserver und die 10 Jahre, die die Daten aufbewahrt und verwendet werden sollen, hoch, so ergibt sich eine Gesamtgröße von 183 GB ($\frac{1}{150} \cdot 7.500 \cdot 10 \cdot 365 = 182.500$). Als weitere Daten im Minutentakt kommen noch die Wetterdaten des Meteorologischen Instituts hinzu. Somit ist es auf jeden Fall notwendig, dass sich nur die gerade benötigten Daten im Hauptspeicher befinden. Da zudem die Möglichkeit eines größeren Wachstums der LMU gegeben ist, in Zukunft weitere Zähler auf die Gebäudeleittechnik umgestellt werden und somit Daten im Minutentakt produzieren werden und Verwendungsmöglichkeiten für Daten, die älter als 10 Jahre denkbar sind, sollte die Software auf jeden Fall ein Vielfaches der genannten Datenmenge bewältigen können. Deshalb ist EnMoLMU für Datenmengen im Terabytebereich ausgelegt.

Von EnMoLMU wird eine Tabelle je Subserver angelegt, um die maximale Spaltenanzahl einer MySQL-Tabelle bzw. des verwendeten Speichersubsystems (Storage Engine) InnoDB nicht zu überschreiten, aber trotzdem eine überschaubare Anzahl von Tabellen zu haben. MySQL kann nur auf äußerst umständliche Weise dazu verwendet werden, Daten im Format der historischen Daten mit einer Zeile pro Tag und einer Spalte pro Minute in eine durchgängige Zeitreihe umzuformen. Somit wurde, um die für die Auswertungen relevanten Abfragen möglichst einfach zu halten und aufgrund der bereits anhand der täglichen

Aktualisierungen gemachten Vorarbeit, das Format der Aktualisierungen beibehalten. Die Tabellen haben also eine Spalte mit Datum und Uhrzeit (*"datum"*) und eine weitere Spalte je Messwert umd damit eine Zeile pro Minute. Ergänzt werden sie um die Spalten *jahr, monat, tag, stunde, minute, sekunde, wochenende, semesterferien, cut15, cut15min* und *"diff_"*-Spalten, um Abfragen und dabei insbesondere Aggregationen und Verknüpfungen mit anderen Tabellen noch weiter zu vereinfachen. Mit *"diff_"*-Spalten sind Spalten gemeint, die zu Zählerspalten, welche die Zählerstände beinhalten, die Differenzen zur jeweils vorigen Minute zeigen. Sie haben dabei den Namen der Zählerspalte mit vorangehendem *"diff_"*. Die MySQL-Syntax, um aus den Zählern deren Differenzen abzufragen ist auch relativ umständlich und, um die Differenzen in R zu berechnen, würde immer eine zusätzliche Zeile benötigt werden. Somit führen diese Spalten zu einer deutlichen Vereinfachung der Abfragen. Die Spalten, welche Zählerdaten beinhalten werden anhand einer Reihe von Eigenschaften automatisch erkannt. Die dabei bewusst in Kauf genommenen Redundanzen dürften in der Praxis bedeutungslos sein.

Aufgrund der hohen Fehleranfälligkeit durch das POSIXlt-Format in R mit seiner schwer kontrollierbaren automatischen Anpassung der Zeitzone wurden vorübergehend die Spalten *datum_char* und *posix* mitaufgenommen, die beim Anlegen der Tabellen miteingefügt werden. Damit ist eine Überprüfung der Spalte *datum* möglich. Beim Anlegen der zum Zeitpunkt der Übergabe der Software vorhandenen Subserver konnten hier allerdings keine Fehler mehr gefunden werden. Somit wird empfohlen, falls auch beim Anlegen einiger weiterer Subserver keine Hinweise auf diesbezügliche Fehler auftauchen, diese Spalten wieder zu löschen und die entsprechenden Teile aus der Software zu entfernen.

Eine simplere Möglichkeit für das Speichern der Daten ohne Datenbank wäre es gewesen, die Daten in einzelne Spalten aufzuteilen und diese einzeln und jeweils mit Datum und ggf. weiteren für die Aggregation etc. benötigten Spalten als CSV-Datei oder RData-/RDS-Datei zu speichern. Ein numeric-Vector der Länge 5.256.000 benötigt etwa 40 MB im RAM, somit könnten selbst bei einem 32-Bit-System einige dieser Spalten mit Zusatzspalten zugleich eingelesen werden. Die Vorteile einer CSV-Datei liegen darin, dass sie weniger Festplattenspeicherplatz zu benötigen scheint, als eine vergleichbare Tabelle in der Datenbank und theoretisch weitere Zeilen an die bereits gespeicherte Datei angehängt werden können. Die R-eigenen Dateiformate (RData bzw. RDA und RDS) benötigen noch einmal deutlich weniger Festplattenspeicherplatz als die CSV-Dateien, können schneller eingelesen werden und es müssen keine Typ-Umwandlungen der Vektoren nach dem Einlesen erfolgen, da

sie bereits in den R-eigenen Typen gespeichert sind. Das Speichern der Daten im CSV- oder RDA/RDS-Format geht deutlich schneller vonstatten, als in eine MySQL-Tabelle. Allerdings müssen diese bei jeder Ergänzung um neue Daten komplett neu gespeichert werden. Auch erlauben beide Formate (CSV und RDA/RDS) anders als Datenbank-Tabellen keine Aggregationen/Berechnungen vor dem Einlesen in R (Ausnahme: CSV über sqldf, allerdings sehr langsam) . Ein gewichtiger Nachteil der sich durch das Speichern der Daten in Formaten, für die die vollständigen Dateien eingelesen werden müssen, ergibt, sind die zusätzlich bei jedem Einlesen benötigten Joins, die bei einer großen Zeilenanzahl sehr teuer und somit langsam werden. Auch könnte die dadurch zustande kommende große Anzahl an Dateien schwer überschaubar werden.

3.2 FAMOS-Daten

In der Geäudemanagement-Software FAMOS von Keßler Solutions werden u.a. Gebäudedaten und manuell abgelesene Zähler erfaßt. Diese Zähler werden aktuell ca. einmal monatlich unregelmäßig abgelesen. Die Zählerstände können u.a. in Form einer XLSX-Datei exportiert werden und reichen in Einzelfällen bis ins Jahr 1995, i. d. R. aber bis etwa 2009 zurück. Aufgrund der langen Ableseintervalle ergeben sich hier keine besonders großen Datenmengen. Somit muss hier auch keine Speicherung in einer Datenbank erfolgen.

3.3 Wetterdaten des Meteorologischen Instituts der LMU

Ergänzend zu den von der GLT erfassten Außentemperaturmessungen werden weitere Wetterdaten vom Meteorologischen Institut bezogen. Das Meteorologische Institut ist Teil der Fakultät für Physik der Ludwig-Maximilians-Universität mit Sitz in der Theresienstraße 37, wo auch die bezogenen Messungen erfolgen. Diese sind ab 1.9.2014 verfügbar und beinhalten Außentemperatur, Feuchttemperatur, Globalstrahlung, Diffusstrahlung, Druck und Niederschlag im Minutentakt. Die historischen Wetterdaten sind ab 1.1.2013 im Stundentakt mit Werten für Diffusstrahlung, Globalstrahlung, Lufttemperatur, Niederschlag und relative Feuchte verfügbar. Die Daten werden nur einmal monatlich für den letzten Kalendermonat bezogen und für EnMoLMU in einer Datenbanktabelle gespeichert.

Da die **relative Feuchte** in den aktuellen Daten nicht enthalten ist, muss diese berechnet werden. Dies erfolgt nach Anleitung des Meteorologischen Institutes, die im Folgenden wiedergegeben werden soll.

Die psychrometrische Methode ist die wichtigste und verbreitetste Methode der Feuchtemessung und beruht auf der Abhängigkeit der Verdunstung von den Feuchteverhältnissen der umgebenden Luft. Ein Psychrometer besteht aus zwei Thermometern, von denen eines die Lufttemperatur ϑ_L messen soll. Das Quecksilbergefäß des anderen ist mit einem feuchten Strumpf überzogen und kühlt sich infolge der Verdunstung unter die Lufttemperatur ab. Mit ihm wird die Temperatur ϑ' gemessen.

Die relative Luftfeuchtigkeit f gibt das Verhältnis des aktuellen Wasserdampfdruckes e_L zum Sättigungsdampfdruck E' über Wasser bei der Lufttemperatur ϑ_L an.

$$\text{relative Feuchte } f = \frac{e_L}{E_L}$$

mit Sprungscher Psychrometerformel

$$\text{Dampfdruck } e_L = E' - 0,663 \cdot \frac{p}{1.000} \cdot \left(\vartheta_L - \vartheta'\right)$$

wobei
ϑ_L = Lufttemperatur
ϑ' = Temperatur des Quecksilbergefäßes
p = Luftdruck, üblicherweise $p = 1006,6$ hPa unabhängig von der Temperatur

E′ ist Sättigungsdampfdruck bei Feuchttemperatur und E Sättigungsdampfdruck bei Lufttemperatur

Sättigungsdampfdruck über Wasser (Gültigkeitsbereich 0°C - 100°C):

$$\text{Sättigungsdampfdruck } E' = 6,1078 \cdot e^{\frac{17,0809 \cdot \vartheta_L}{234,175 + \vartheta_L}}$$

mit
ϑ_L = Lufttemperatur

4 Anwendung der Software

4.1 Bedienung

Sind die Subserver, Berichtsvorlagen und Skripte bereits angelegt, so können viele der Aufgaben einfach per Doppelklick auf das entsprechende Batch-Skript über das Desktop-Symbol `BedienungEnMoLMU`, welches eine Desktop-Verknüpfung zum Ordner `Skripte/batch` ist, ausgeführt werden.

glt_aktualisieren Durch Doppelklick auf dieses Batch-Skript werden die neuen GLT-Tagesdateien in die entsprechenden Datenbanktabellen geschrieben.

wetter_aktualisieren Hiermit werden die aktuellen Daten des Meteorologischen Instituts in die Datenbanktabelle *theresienwetter* geschrieben.

berichte_erstellen Die Skripte zur Berichtserstellung generieren zu allen Vorlagen aus einem bestimmten oder mehreren Ordnern die fertigen Berichte.

datenbankdump Erstellt eine Sicherung der Tabellen in der Datenbank.

Die Abbildung 4.1 zeigt beispielhaft (mit leicht abweichenden Skript-Namen) diesen Ordner mit den einzelnen Batch-Skripten.

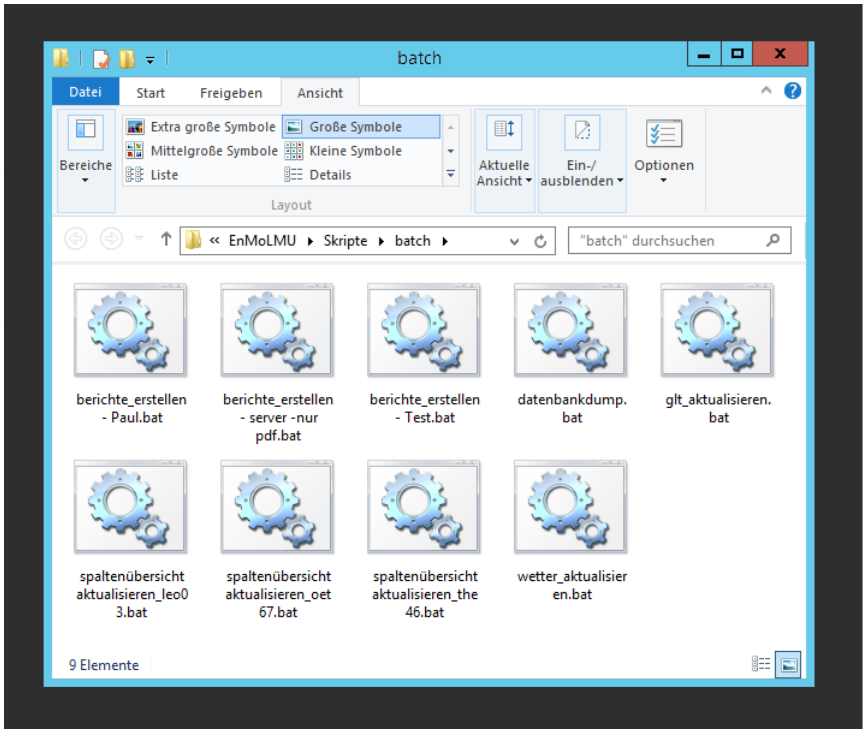


Abbildung 4.1: Vereinfachte Bedienung durch Ordner mit Batch-Skripten für die meisten Vorgänge.

4.2 Schritte Anwendung

4.2.1 Vorbereitung

Installieren der benötigten Software

Hier soll lediglich die benötigte Software aufgezählt werden, um einen Überblick bereitzustellen, was benötigt wird, um EnMoLMU auszuführen. Anleitungen dazu befinden sich in den entsprechenden Abschnitten.

- R
- RStudio (empfohlen)
- benötigte R-Pakete (evtl. über RStudio installieren, falls die Installation über R blockiert wird)

- Installieren der EnMoLMU R-Pakete
- Pandoc (oder Pfad zu in RStudio enthaltenem Pandoc in den Systemvariablen setzen, falls die Installation von Pandoc blockiert wird)
- MikTex, falls die Berichte im PDF-Format erstellt werden sollen (da mindestens ein in der Basisinstallation nicht enthaltenes Paket benötigt wird, kann über proTeXt (`https://www.tug.org/protext/`) offline eine vollständige Installation durchgeführt werden, falls das Installieren von Paketen über das Internet blockiert wird - was zum Zeitpunkt der Erstinstallation von EnMoLMU auf dem Server enmon1 der Fall war))
- MySQL
- ggf. Libreoffice und Pfad zu dessen Ordner `program` (z. B. `C:\ProgramFiles\LibreOffice4\program`) in den Systemvariablen setzen, falls eine Umwandlung von Dateiformaten über Libreoffice durchgeführt werden soll.

Installieren der benötigten CRAN-R-Pakete

Durch Ausführen der folgenden Befehle in der Console in RStudio können alle von CRAN benötigten R-Pakete heruntergeladen und installiert werden:

```
cran_pakete <- c("devtools", "roxygen2", "knitr",
                 "markdown ", "tools", "methods",
                 "RODBC", "plyr", "reshape2", "data.table",
                 "sqldf", "mgcv", "ggplot2",
                 "dplyr", "xlsx", "grid", "gridExtra",
                 "lubridate", "gtable", "quantreg")
install.packages(cran_pakete)
```

Falls dies für einzelne Pakete nicht funktioniert (was z. B. auf dem Server bei *httpuv* der Fall war, welches für *shiny* benötigt wird), so können diese Pakete über `https://cran.r-project.org/web/packages/available_packages_by_name.html` als zip-Datei heruntergeladen und von der Festplatte aus installiert werden. Dies kann in RStudio über **Tools - Install Packages - Package Archive File (.zip; .tar.gz)** ausgeführt werden.

Anlegen der Ordnerstruktur für EnMoLMU

Um die für EnMoLMU verwendeten Ordner und Dateien möglichst übersichtlich zu halten und bei jeder Installation einen möglichst identischen Aufbau zu haben, wurde eine empfohlene Ordnerstruktur erarbeitet.

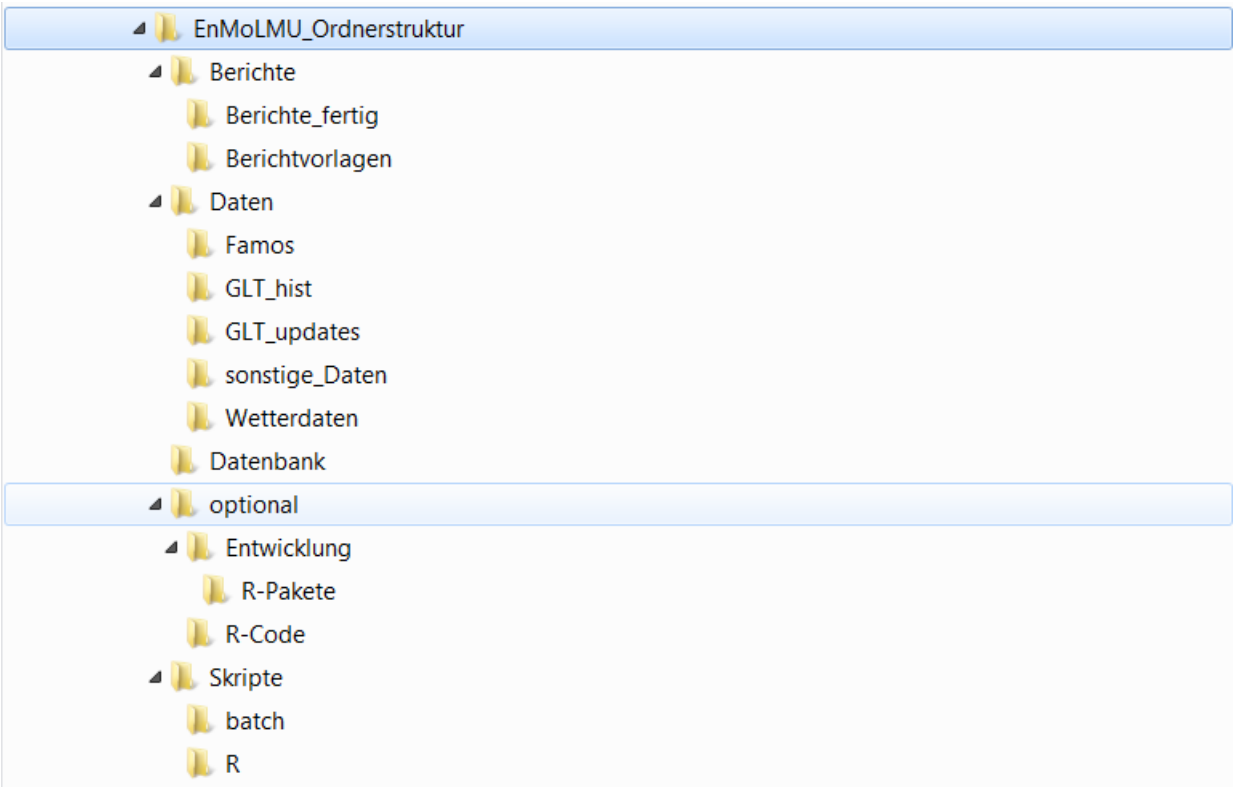


Abbildung 4.2: Die empfohlene Ordnerstruktur für EnMoLMU. Die Unterordner des Ordners optional sind nur unter bestimmten Umständen nötig.

Berichte In den Ordner `Berichte` kommen sowohl die Berichtsvorlagen im R Markdown-Format, als auch die fertigen Berichte im PDF- oder HTML-Format.

Daten Der Ordner `Daten` beinhaltet Unterordner für Famos-Daten (aus Famos exportierte Daten und aufbereitete Daten), sowohl historische und somit manuell exportierte GLT-Daten und deren aufbereitete Versionen, als auch automatisch abgelegte tägliche Updates und schließlich die Wetterdaten des Meteorologischen Institutes, die manuell abgelegt, entpackt, und benannt werden müssen.

Datenbank Hier kommt die Datenbank für die GLT- und Wetterdaten-Tabellen rein und die Datenbankdumps, sofern hierfür kein separates Laufwerk vorgesehen ist

Skripte Im Ordner `Skripte` werden alle verwendeten Skripte, wie Batch- und R-Skripte abgelegt

optional Die hier enthaltenen Unterordner sind nicht zwangsläufig nötig. Im Falle von Entwicklung nur, falls die R-Pakete auf demselben System weiterentwickelt werden; im Falle von R-Code, falls die R-Codes nicht in Form von Paketen verwendet werden sollen, sondern als "rohe" Code-Dateien über source geladen werden.

Diese Ordner können mit Hilfe des R-Pakets *enmoSonstiges* aus der EnMoLMU-Software angelegt werden:

```
library(enmoSonstiges)
enmolmu_ordnerstruktur2("D:/EnMoLMU")
```

Wobei der Pfad, der in der zweiten Zeile angegeben wurde durch den Pfad, in welchem die Software gespeichert werden soll, ersetzt werden muss. Die einzelnen Ordner im Pfad müssen durch ein "/"-Zeichen getrennt werden (Windows verwendet "\").

Hinzufügen von Daten

- Datei mit den aktuellen Feriendaten im CSV-Format und mit dem Namen `ferien.csv` im Ordner `sonstige_Daten` ablegen. Der Inhalt muss in folgendem Format sein:

  ```
  Ferienbeginn;Ferienende
  22.07.2012;14.10.2012
  10.02.2013;14.04.2013
  21.07.2013;13.10.2013
  09.02.2014;06.04.2014
  14.07.2014;05.10.2014
  01.02.2015;12.04.2015
  19.07.2015;11.10.2015
  07.02.2016;10.04.2016
  17.07.2016;16.10.2016
  ```

Die Datei kann mit Excel editiert werden, es muss aber darauf geachtet werden, dass sie wieder als .CSV-Datei abgespeichert wird und nicht als .XLS o.ä. Außerdem müssen die Spalten durch ein Semikolon getrennt sein (das Komma ist hier das Dezimaltrennzeichen).

- Die Wetterdaten des Meteorologischen Instituts gehören in den Ordner Wetterdaten. Sie müssen entpackt und so umbenannt werden, dass sie am Ende des Namens das Datum (Jahr und Monat) der Daten im Namen haben, z. B. `TheresienWetter2015_03` für die Daten vom März.
- Die aus GLT und Famos automatisch exportierten Daten werden im Idealfall direkt in den Ordner `GLT_updates` und `Famos` abgelegt. Falls dies nicht möglich ist, können diese per Skript automatisch abgeholt werden.
- In den Ordner `Skripte/R` kommen die R-Skripte zum Update der Datenbanktabelle für die Wetterdaten und die einzelnen Subserver und das Generieren der Berichte.
- Zu jedem dieser R-Skripte, welches automatisch ausgeführt werden soll, muss es im Ordner `Skripte/batch` ein Batch-Skript geben, welches dieses R-Skript ausführt.
- Sobald die Batch-Skripte angelegt worden sind, können einzelne Skripte als automatisch zu bestimmten Zeitpunkten auszuführende Aufgaben im Task Scheduler angelegt werden.
- Zur vereinfachten manuellen Steuerung empfiehlt es sich auch, eine Verknüpfung des Ordner `Skripte/batch` auf dem Desktop mit dem Namen `"BedienungEnMoLMU"` anzulegen, um die einzelnen Aufgaben bequem per Doppelklick auf die einzelnen Skript-Dateien ausführen zu können (siehe auch 4.1).

4.2.2 Verwendung der Gebäudeleittechnik-Daten

Die GLT-Daten (3.1 werden getrennt nach Subservern auf Tabellen aufgeteilt in die Datenbank geschrieben. Dazu müssen zuerst die historischen Daten des jeweiligen Subservers aus der ProGrafNT-Datenbank als .CSV-Datei exportiert werden. Damit kann dann die entsprechende Tabelle in der EnMoLMU-Datenbank angelegt und die exportierten Daten darin abgespeichert werden (siehe 4.2.3).

Nun können die Spaltenübersichtsdatei erstellt (4.2.6) und die Tabelle aktualisiert werden (4.2.3). In die Spaltenübersichtsdatei können manuell alternative Zählerbezeichnungen aufgenommen werden, welche zur Beschriftung der Grafiken verwendet werden. Weiterhin können die zu den Zählern gehörigen Einheiten ergänzt werden, welche zur Beschriftung der Grafiken und Berechnung von Werten verwendet werden. Wurden vor dem Export der Aktualisierungsdaten neue Zähler aufgenommen, so muss nach der Aktualisierung auch die Spaltenübersichtsdatei aktualisiert werden (4.2.7) und, falls vorhanden, die historischen Daten ergänzt werden (4.2.4).

Das Diagramm 4.3 zeigt den Ablauf für die GLT-Subserver.

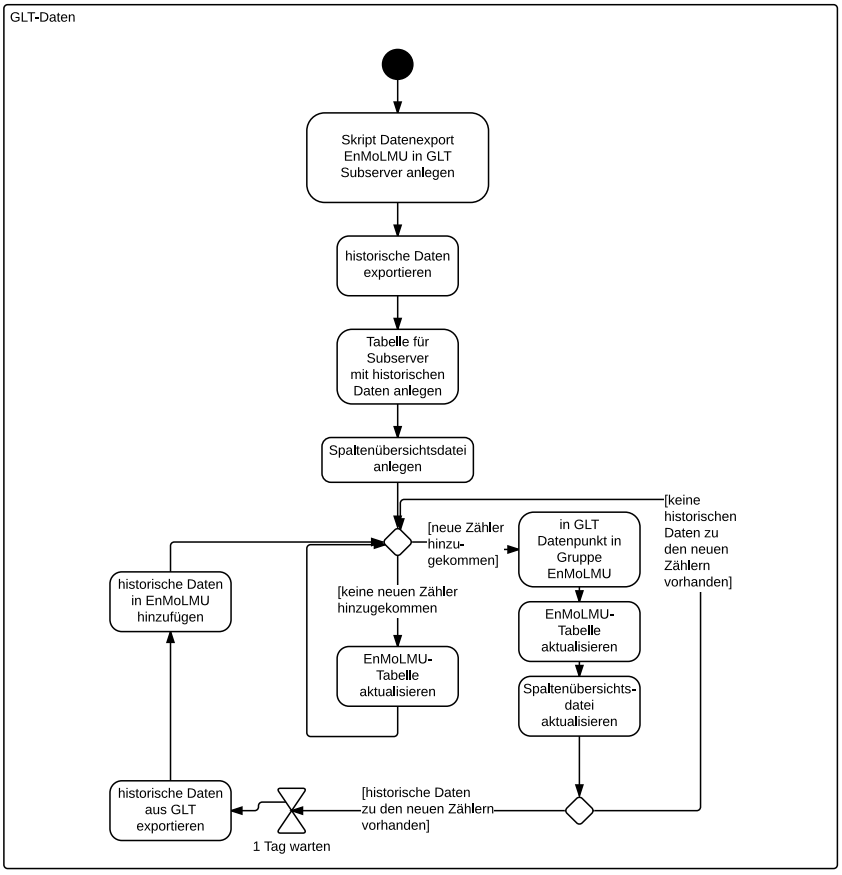


Abbildung 4.3: Anlegen der Subserver, hinzufügen neuer Zähler und aktualisieren.

4.2.3 Anlegen und Aktualisieren der Subserver zu den Gebäudeleittechnik-Daten

Damit EnMoLMU die Daten eines GLT-Subservers verwenden kann, muss dieser zuerst in der ProGrafNT-Software mit den benötigten Zählern angelegt werden, sodass sowohl die historischen Daten exportiert werden können, als auch die täglichen Aktualisierungen exportiert werden. Nun kann der Server auch in EnMoLMU mithilfe der exportierten historischen Daten angelegt werden.

Für das folgende Beispiel wurden die GLT-Daten für den Subserver Oettingenstraße 67 unter dem Namen `alles_h_Oet67EnMoLMU.csv` exportiert und im Ordner `D:\\EnMoLMU\\Daten\\GLT_hist` auf dem Server abgelegt. Die neue Tabelle in der Datenbank wird *oet67* genannt und die Datei mit den Anfangs- und Endzeitpunkten der Semesterferien liegt im Ordner `D:\\EnMoLMU\\Daten\\sonstige_Daten` (siehe 4.2.1).

```
library(RODBC)
library(EnMoLMU)

ch <- odbcConnect("ansi_benutzer", DBMSencoding = "utf-8")


glt_hist_vorbereiten(datei =
                       "D:\\EnMoLMU\\Daten\\GLT_hist\\alles_h_Oet67EnMoLMU.csv",
                     tabelle = "oet67", con = ch,
                     zwischenschritte_loeschen = TRUE,
                     ferien = "D:\\EnMoLMU\\Daten\\sonstige_Daten\\ferien.csv")
```

Falls für den Ordner, in welchem sich die Ursprungsdatei befindet keine Schreibrechte bestehen, kann mit dem Argument `zwischenschritte_ordner` ein Pfad zu einem anderen Ordner angegeben werden, in den dann die Zwischenschritte gespeichert werden. Eine Beschreibung aller Argumente der Funktion kann in der R-Hilfe oder unter A.1 (Paket *enmoDaten*) gefunden werden.

Nach Abschluss des Schreibens der Daten in die Datenbank gibt die Funktion `glt_hist_vorbereiten` einige Zusammenfassungen aus, mit denen die Daten in der Tabelle überprüft werden können. Anfangs- und Enddatum müssen dem Anfangs- und Enddatum des

exportierten Datensatzes entsprechen, die einzelnen Monate müssen mit dem ersten Tag des jeweiligen Monates um 0:00 Uhr beginnen (die Uhrzeit wird dann nicht mitangezeigt) und mit dem jeweils letzten Tag des Monates um 23:59 enden. Die Häufigkeiten der Werte 00, 15, 30 und 45 in der Spalte *cut15min* müssen gleich sein.

Falls die Daten aufgeteilt in mehrere Dateien exportiert werden, kann das Argument `datei` ein Vektor mit allen Dateien sein.

Ein R-Skript zum Anlegen der Subserver-Tabellen Theresienstraße 46, Oettingenstraße 67 und Leopoldstraße 03 befindet sich in Anhang A.3.1.

Ab jetzt kann die Tabelle mit den automatischen täglichen Exporten für den Subserver Oettingenstraße 67 mit Hilfe des Befehls `update_glt2` aktualisiert werden:

```
library(RODBC)
library(EnMoLMU)

ch <- odbcConnect("ansi_benutzer", DBMSencoding = "utf-8")


# Aktualisierung der Tabellen für den bereits angelegten Subserver Oet67

update_glt2(ch, tabelle = "oet67", name = "Oet67", name_allgemein = "EnMoLMU",
            ordner = "D:\\EnMoLMU\\Daten\\GLT_updates",
            ferien = "D:\\EnMoLMU\\Daten\\sonstige_Daten\\ferien.csv",
            einzeln = FALSE
)
```

`tabelle` muss dem Namen der vorher angelegten Tabelle entsprechen, `name` ist der Namensteil der exportierten Dateien, der die zum jeweiligen Subserver gehörenden von den anderen unterscheidet, `name_allgemein` der Namensteil, den alle zu irgendeinem der Subserver gehörenden Dateien im Ordner `ordner` gemeinsam haben und über `einzeln = FALSE` wird festgelegt, dass alle neuen Dateien im Ordner aufbereitet und auf einmal in die Datenbank geschrieben werden. Falls die Anzahl der neuen Dateien zu groß ist (benötigter RAM, Limit der verwendeten Funktion `rbind.fill`), kann das Argument `einzeln = TRUE` abhilfe schaffen. Eine Beschreibung aller Argumente der Funktion befindet sich wieder in der R-Hilfe oder unter A.1 (*enmoDaten*).

Ein R-Skript zum Aktualisieren der Subserver-Tabellen Theresienstraße 46, Oettingenstraße 67 und Leopoldstraße 03 befindet sich in Anhang A.3.1.

Dieses Skript kann z. B. durch durch Doppelklick auf die Batch-Datei `glt_aktualisieren.bat` in der Desktop-Verknüpfung `"BedienungEnMoLMU"` ausgeführt werden.

Soll eine neue Tabelle angelegt werden, ohne dass dafür zuerst die historischen Daten aus der ProGrafNT-Datenbank exportiert werden, so kann dies mit dem Befehl `glt_hist` und den täglichen Aktualisierung-Dateien geschehen:

```
glt_hist(con=ch, tabelle = "leo03_test", namensteil = "Leo03",
         quellordner = "D:\\EnMoLMU\\Daten\\Glt_updates fuer tests",
         ferien = "D:\\EnMoLMU\\Daten\\sonstige_Daten\\ferien.csv")
```

Wird durch `update_glt2` eine neue Spalte angelegt, können die historischen Daten mit `db_spalten_hinzufuegen` ergänzt werden, was in 4.2.4 beschrieben wird.

4.2.4 Nachträgliches Einfügen der Gebäudeleittechnik-Daten in die Datenbank-Tabellen

Werden neue Zähler in die GLT-Export-Gruppen aufgenommen, so werden zwar die entsprechenden Spalten in den Datenbanktabellen erstellt und die Daten ab dem Zeitpunkt der Aufnahme in die Export-Gruppen eingetragen, aber die historischen Werte fehlen in den Tabellen, da EnMoLMU auf diese keinen direkten Zugriff hat. Diese können aber manuell aus der GLT exportiert und mit dem Befehl `db_spalten_hinzufuegen` nachgetragen werden.

Die Vorgehensweise soll an einem Beispiel verdeutlicht werden, in welchem eine neue Tabelle erstellt, daraus eine Spalte gelöscht, und die Tabelle mit den Daten eines weiteren Tages aktualisiert wird, so dass die Spalten wieder vorhanden sind, aber alle Werte des ersten Tages fehlen. Diese können nun nachträglich wieder eingefügt werden.

Der erste Teil des R-Skriptes legt die Datenbanktabelle an und bereitet sie soweit vor, dass die Daten wieder eingefügt werden können. Dieser Teil ist natürlich nur um eine Tabelle für dieses Beispiel zu erzeugen, nötig, und muss in den richtigen Tabellen, in denen die

Daten ja gerade deshalb nachträglich eingefügt werden sollen, weil sie nicht vorhanden sind, nicht ausgeführt werden.

```
# Test/Beispiel nachträglich historische Daten hinzufügen in Datenbank-Tabelle

# nachtragen einzelner Spalten

# Vorgehen:
# Tabelle anlegen mit 1 Datei, Spalten löschen, weitere Datei hinzufügen (so dass
#  Spalten wieder da sind, aber für den 1. Tag leer), Spalten wieder befüllen.


library(EnMoLMU)
library(RODBC)
ch <- odbcConnect("ansi_benutzer")

# GLT-Datei für ersten Tag im Ordner quellordner:
glt_hist(con = ch, tabelle = "oet_nachtragen_tests", namensteil = "Oet67",
         quellordner = "D:/EnMoLMU/Daten/Glt_updates fuer tests",
         ferien = "D:/EnMoLMU/Daten/sonstige_Daten/ferien.csv")

# welche Spalten in der Tabelle
sqlColumns(ch, "oet_nachtragen_tests")$COLUMN_NAME

# Spalte x7070_01_u138hzg03tem0002mp01 entfernen:
sqlQuery(ch, "ALTER TABLE oet_nachtragen_tests DROP x7070_01_u138hzg03tem0002mp01")

# Spalte jetzt nicht mehr in der Tabelle
sqlColumns(ch, "oet_nachtragen_tests")$COLUMN_NAME

# GLT-Datei für zweiten Tag in den Ordner quellordner verschoben.
update_glt2(con = ch, tabelle = "oet_nachtragen_tests", name = "Oet67",
         ordner = "D:/EnMoLMU/Daten/Glt_updates fuer tests",
         ferien = "D:/EnMoLMU/Daten/sonstige_Daten/ferien.csv")
```

```
# Spalte x7070_01_u138hzg03tem0002mp01 wieder vorhanden:
sqlColumns(ch, "oet_nachtragen_tests")$COLUMN_NAME

sqlQuery(ch,
         "select x7070_01_u138hzg03tem0002mp01 from oet_nachtragen_tests limit 10")
# Werte des ersten Tags fehlend
sqlQuery(ch,
         "select x7070_01_u138hzg03tem0002mp01 from oet_nachtragen_tests
          WHERE datum >= '2015-02-17' limit 10")
# Werte des zweiten Tags vorhanden
```

Als nächstes müssen die einzufügenden Daten aus der CSV-Datei eingelesen und aufbereitet werden.

```
daten <- read.csv2("D:/EnMoLMU/Daten/Glt_updates fuer tests/150216Oet67EnMoLMU.csv")
# Datum muss von factor in POSIXct umgewandelt werden:
daten <- cbind(daten, datum2 = as.POSIXct(as.character(daten$Datum),
               format = "%d.%m.%Y %H:%M:%S", tz="GMT"))
# sollen weitere Spalten, wie Differenzen, hinzugefügt werden, so müssen diese
# an dieser Stelle an Daten angehängt werden
names(daten)
```

Jetzt können die Daten in die Tabelle nachgetragen werden.

```
db_spalten_hinzufuegen(ch, "oet_nachtragen_tests", df = daten,
                       spalten_tabelle = NULL, # automatische Umwandlung des df-Namens
                       spalten_df = "X7070.01_U138HZG03TEM0002MP01",
                       datum_df = "datum2") # umgewandeltes Datum aus daten verwenden

sqlQuery(ch,
         "select x7070_01_u138hzg03tem0002mp01 from oet_nachtragen_tests limit 10")
# jetzt auch wieder werte des ersten Tages vorhanden

#Test-Tabelle wieder löschen:
```

```
sqlTables(ch)
sqlDrop(ch, "oet_nachtragen_tests")
```

Ein abgewandeltes Skript für zwei Spalten befindet sich im Anhang unter A.2.1. Diese Funktion kann natürlich auch für andere Tabellen, als die der GLT-Daten verwendet werden.

Um aus der GLT exportierte historische Daten nachträglich einfügen zu können, müssen diese erst mit `glt_hist_aufteilen` bzw. der schnelleren Variante `glt_hist_aufteilen_fread` und `glt_umbauen` oder `glt_umbauen_ordner` in das Format der Tagesupdates gebracht werden.

4.2.5 Anlegen und Aktualisieren der Wetterdaten des Meteorologischen Instituts

Die Wetterdaten des Meteorologischen Instituts werden als Textdatei (ohne Dateiendung) geliefert in einer eigenen Tabelle mit dem Namen `theresienwetter` abgespeichert.

```
library(RODBC)
library(EnMoLMU)

ch <- odbcConnect("ansi_benutzer", DBMSencoding = "utf-8")

update_wetter(ch, ordner = "D:\\EnMoLMU\\Daten\\Wetterdaten",
              tabelle = "theresienwetter")
```

Der Befehl `update_wetter` kann hier sowohl zum Anlegen der Tabelle, als auch zu ihrer Aktualisierung verwendet werden. Es werden immer alle Dateien aus dem Ordner `ordner` verwendet, die von einem neueren Monat, als das neueste Datum in der Tabelle sind.

Ein R-Skript zum Anlegen/Aktualisieren der Wetterdaten befindet sich in Anhang A.3.1.

Für dieses Skript befindet sich auch wieder eine Batch-Datei in der Desktop-Verknüpfung `"BedienungEnMoLMU"`. Diese kann durch Doppelklick auf wetter_aktualisieren.bat ausgeführt werden. Bei dieser Art der Ausführung wird eine Datei `update_wetter.Rout` im Ordner `Skripte/R` erstellt, deren Inhalt weitgehend dem R-Consolen-Inhalt, der bei einer manuellen Ausführung des R-Skriptes entstanden wäre, entspricht.

4.2.6 Anlegen der Spaltenübersichten zu den einzelnen Tabellen

Sobald die GLT-Tabellen in der Datenbank angelegt und aktuell sind, kann für jede Tabelle eine CSV-Datei mit einer Übersicht zu den einzelnen Spalten angelegt werden, die Informationen zu Bezeichnungen, Einheiten etc. für die Berichterstellung enthält.

Dazu muss als erstes aus der aus Famos exportierten aktuellen Gebäudeliste eine Gebäudeübersichts-CSV-Datei mit Hilfe des Befehls `gebaeudeueberblick` aus dem Paket enmoFamos erzeugt werden.

```
library(enmoFamos)
gebaeudeueberblick("D:/EnMoLMU/Daten/sonstige_Daten/20141216 Gebäudeliste.csv",
                   "D:/EnMoLMU/Daten/sonstige_Daten/gebäudeübersicht.csv")
```

Nun kann mit Hilfe des Befehls `spaltenuebersicht`, einer aktuellen GLT-Tagesupdate-Datei (Argument CSV), der entsprechenden Datenbank-Tabelle (tabelle), der soeben erzeugten Gebäudeübersichts-Datei (gebaeude) die Übersichts-Datei über die Spaltennamen (GLT-Export und Datenbanktabelle), der Bauteil-Kurzbezeichnungen (Famos Gebäudeliste) und noch manuell einzutragenden weiteren Spalten (Einheit, etc.) die Spaltenübersicht erzeugt und als CSV-Datei (als_csv) gespeichert werden.

```
# Spaltenübersicht für Grafiken für The46 erstellen (erstes Mal):
spaltenuebersicht(
  csv = "D:/EnMoLMU/Daten/GLT_updates/150510The46EnMoLMU.csv", #aktuelle datei
  con = ch,
  tabelle = "the46",
  gebaeude = "D:/EnMoLMU/Daten/sonstige_Daten/gebäudeübersicht.csv",
  als_csv = "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46_alt_test.csv")
```

```
read.csv2("D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46_alt_test.csv",
          nrows=6) # die ersten 6 Zeilen einlesen und anzeigen
```

Die Datei kann nun mit z. B. Excel geöffnet und bearbeitet werden, wobei hierbei darauf geachtet werden sollte, dass diese wieder im CSV-Format gespeichert wird. In Excel können jetzt die Spalten *Nr., IP.Kommentar, Anlage, Einheit, Faktor.Umwandlung.in.Standardeinheiten*, und beliebige weitere Spalten hinzugefügt und entsprechende Werte eingetragen werden. Die Werte für *diff_*-Spalten sollten dabei bei den Zeilen für die ursprünglichen Spaltennamen eingetragen werden, da das Zusammenfügen der Zeilen bei einer späteren Aktualisierung über die Spalte *csv* erfolgt, in welcher es keine Einträge für die *diff_*-Spalten gibt.

4.2.7 Aktualisieren der Spaltenübersichten zu den einzelnen Tabellen

Da von Zeit zu Zeit neue Zähler in die GLT-Gruppen aufgenommen werden und Änderungen in der Famos-Gebäudeliste möglich sind, muss auch diese Spaltenübersicht immer wieder aktualisiert werden. Um zu vermeiden, dass die manuellen Einträge wiederholt von Hand eingetragen werden müssen, können über den Befehl `spaltenuebersicht` Spalten aus der alten Spaltenübersicht übernommen werden.

```
library(EnMoLMU)
library(RODBC)
ch <- odbcConnect("ansi_benutzer")
spaltenuebersicht(
  csv = "D:/EnMoLMU/Daten/GLT_updates/150510The46EnMoLMU.csv", #aktuelle datei
  con = ch,
  tabelle = "the46",
  uebernehmen = c("Nr.", "IP.Kommentar", "Anlage", "Einheit",
                  "Faktor.Umwandlung.in.Standardeinheiten",
                  "COP.Kältemaschine", "Rasterdiagramm.Energie",
                  "Jahresdauerlinie", "Betriebsart", "T_RL..T_VL",
                  "Energiesignatur", "Ranking"),
  aus = "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46_alt.csv",
  gebaeude = "D:/EnMoLMU/Daten/sonstige_Daten/gebäudeübersicht.csv",
```

```
  als_csv = "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46.csv")

read.csv2("D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46.csv",
          nrows=6)
```

Die Datei kann nun zur Beschriftung der Grafiken verwendet werden:

```
tagesverlauf_db_t(ch, tabelle = "the46", datum = "datum",
                  werte = "diff_x1120_01_021anhv01elz0001zw30",
                  aussentemp = "x1110_01_0032hzg00tem0001ma01",
                  tabelle2 = "theresienwetter", strahlung = "globalstrahlung",
                  abschnitte = "15 Minuten", between=letzter_monat(),
                  legende =
                   "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46.csv",
                  einheit =
                   "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46.csv")
```

Da der Text für die Legende in diesem Fall viel zu lang ist, empfiehlt es sich, in diesem Beispiel, eine manuelle Abkürzung anzugeben (bzw. diese bereits in die Tabelle einzutragen oder den vorhandene Wert mit Zeilenumbrüchen zu versehen).

```
tagesverlauf_db_t(ch, tabelle = "the46", datum = "datum",
                    werte = "diff_x1120_01_021anhv01elz0001zw30",
                    aussentemp = "x1110_01_0032hzg00tem0001ma01",
                    tabelle2 = "theresienwetter", strahlung = "globalstrahlung",
                    abschnitte = "15 Minuten", between=letzter_monat(),
                    legende = "Wirkarbeit\nSumme",
                    einheit =
                     "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46.csv")
```

4.2.8 Berichte erstellen

Zum Erstellen der Berichte müssen als erstes die Vorlagen im R Markdown-Format erstellt werden. Dazu kann eine neue R Markdown-Datei in RStudio geöffnet oder eine bereits

bestehende Vorlage kopiert werden. In diese Vorlage müssen nun die R-Codes zum Laden der benötigten R-Pakete, zum Öffnen einer Verbindung zur Datenbank und zum erstellen der Grafiken und die gewünschten Überschriften und Erklärungen eingefügt werden. Genauere Informationen zum Erstellen der R Markdown-Vorlagen können unter 6.5 gefunden werden. Der vorgesehene Ordner in der vorgeschlagenen Ordnerstruktur für diese Vorlagen ist `Berichte\Berichtvorlagen`.

Als nächstes wird ein R-Skript in `Skripte\R` benötigt, das aus den soeben erstellten Vorlagen die Berichte generiert. Dieses könnte z. B. folgendermaßen aussehen:

```
library(EnMoLMU)

pfad <- "D:\\EnMoLMU\\"

# Alle Vorlagen als PDF-Bericht erstellen
knit_all_format(ordner = paste(pfad, "Berichte\\Berichtvorlagen",sep=""),
                output_ordner = paste(pfad, "Berichte\\Berichte_fertig", sep=""),
                format = "pdf")
```

Das Skript mit dem Namen `Berichte erstellen.R` lädt das R-Paket EnMoLMU, legt den Pfad bis zum Beginn der Ordnerstruktur fest und ruft dann `knit_all_format` mit den entsprechenden Ordnern aus der Ordnerstruktur auf, was die Berichte im PDF-Format erzeugt. Die fertigen Berichte werden im Ordner `Berichte_fertig` abgelegt.

Um dieses Skript, und somit den Vorgang der Berichterstellung automatisiert über die Aufgabenplanung 6.3 oder ohne die Datei erst in RStudio öffnen zu müssen per Doppelklick, ausführen zu können, wird noch ein Batch-Skript benötigt (siehe 6.2), welches das gerade besprochene R-Skipt in R ausführt. Dies kommt in den Ordner `Skripte\batch` und soll hier `berichte_erstellen.bat` heißen. Es sieht so aus:

```
rem Aufruf Skript zur Berichterstellung
R CMD BATCH "D:\EnMoLMU\Skripte\R\Berichte erstellen.R"
```

Alles, was in einer Zeile auf `rem` folgt, ist ein Kommentar und wird nicht ausgeführt. Somit beinhaltet dieses Skript nur einen Befehl `R CMD BATCH <R-Skript>`, der das R-Skript

`Berichte-erstellen.R` in R ausführt. Der Pfad zu `RSkript.exe` muss dazu natürlich in den Systemvariablen gesetzt sein (siehe 6.2).

Ein Batch-Skript wird durch einen Doppelklick auf dessen Symbol anders als ein R-Skript nicht im Editor geöffnet, sondern ausgeführt. Um es zu bearbeiten kann es mit **Rechtsklick – Bearbeiten** geöffnet werden.

Jetzt können somit durch einen Doppelklick auf das Symbol der Batch-Datei `berichte_erstellen.bat` alle Berichte, für die es Vorlagen im Ordner gibt, erstellt werden, oder eine automatische Erstellung zu bestimmten Zeiten eingerichtet werden (6.3).

Diese beiden Skripte befinden sich auch im Anhang unter A.3.1 und A.3.2.

4.3 Beispielbericht

Der folgende Bericht im PDF-Format wurde mit EnMoLMU erstellt. Die Berichtsvorlage dazu befindet sich unter A.4.

Beispielbericht

Thomas

2015-07-30 14:12:53

Beispielbericht

Dies ist ein Beispielbericht vom 2015-07-30 14:12:53 mit einer Reihe von Grafiken.

Energiesignatur aufgeteilt nach Werktag/Wochenende und Vorlesungszeit/Semesterferien (energiesignatur_aufgeteilt_db_ferien)

Energiesignatur des Zählers diff_x7070_01_u138hzg03wmz0001zw01 und der Außentemperatur x7070_01_u138hzg03tem0003mp01 für die Zeiträume vom 1.1.2014 bis 1.4.2015 (Modelle schätzen) und 2.4.2015 bis 30.4.2015 (überprüfen).

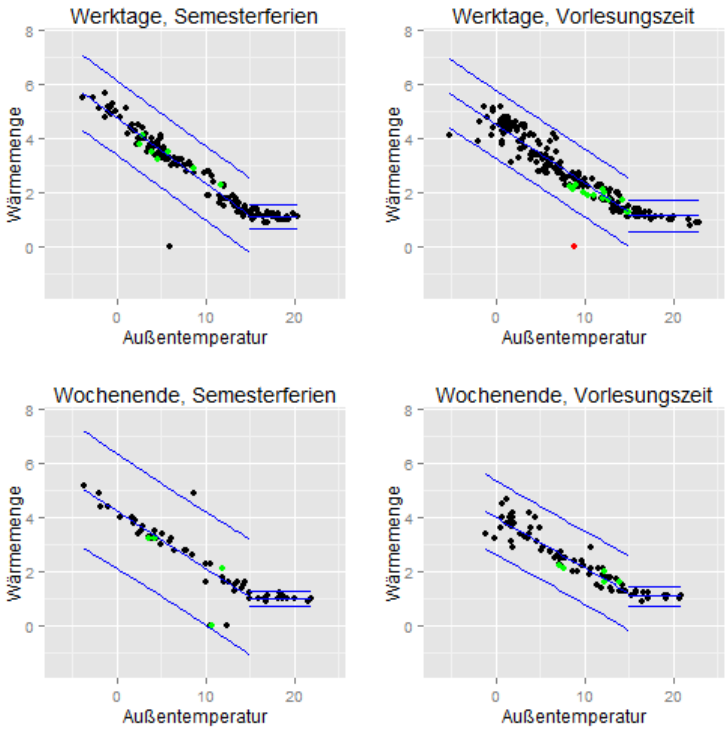


Figure 1: plot of chunk Energiesignatur aufgeteilt ferien

```
## Werktage, Semesterferien
## Ausreißer linearer Abschnitt:
## [1] waermemenge      aussentemperatur datum
## <0 rows> (or 0-length row.names)
## Ausreißer konstanter Abschnitt:
## [1] waermemenge      aussentemperatur datum
```

```
## <0 rows> (or 0-length row.names)
##
## Wochenende, Semesterferien
## Ausreißer linearer Abschnitt:
## [1] waermemenge      aussentemperatur datum
## <0 rows> (or 0-length row.names)
## Ausreißer konstanter Abschnitt:
## [1] waermemenge      aussentemperatur datum
## <0 rows> (or 0-length row.names)
##
## Werktage, Vorlesungszeit
## Ausreißer linearer Abschnitt:
##    waermemenge aussentemperatur      datum
## 14           0              8.8 2015-04-30
## Ausreißer konstanter Abschnitt:
## [1] waermemenge      aussentemperatur datum
## <0 rows> (or 0-length row.names)
##
## Wochenende, Vorlesungszeit
## Ausreißer linearer Abschnitt:
## [1] waermemenge      aussentemperatur datum
## <0 rows> (or 0-length row.names)
## Ausreißer konstanter Abschnitt:
## [1] waermemenge      aussentemperatur datum
## <0 rows> (or 0-length row.names)
```

Tagesverlauf

Oettingenstraße 67

```
## Warning in loop_apply(n, do.ply): Removed 572 rows containing missing
## values (geom_path).
```

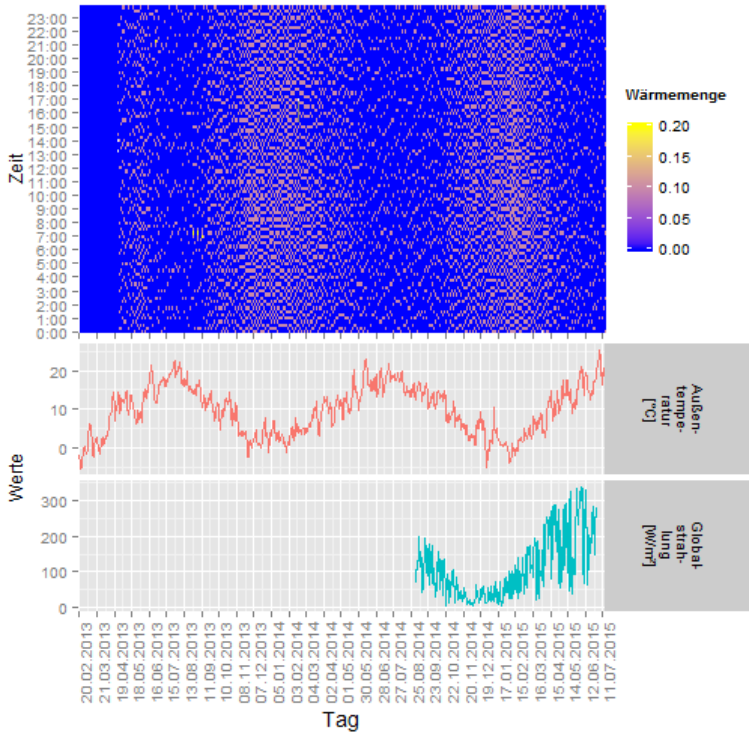


Figure 2: plot of chunk Tagesverlauf Oet67

Theresienstraße 46

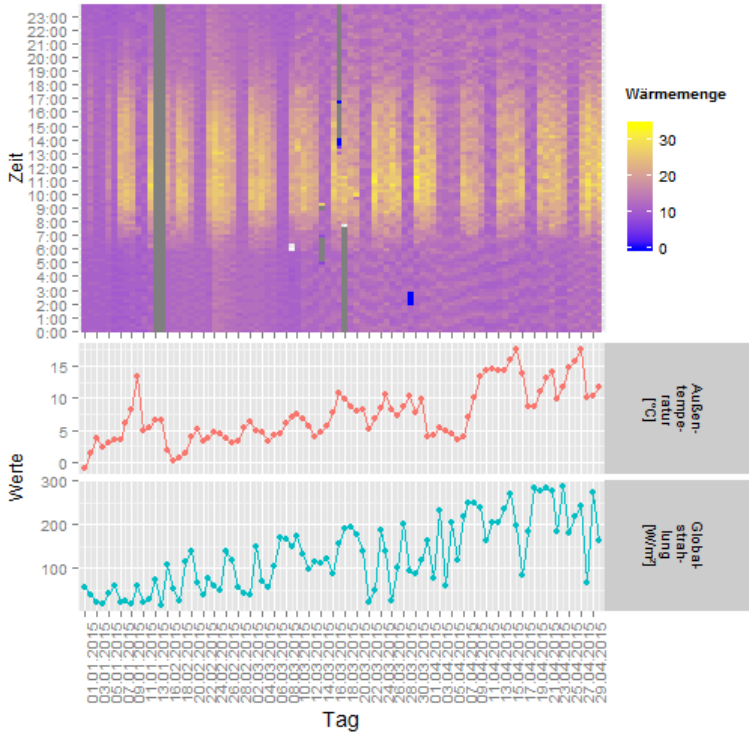


Figure 3: plot of chunk Tagesverlauf The46

Temperaturvergleich (H_Heizkreise Rücklauf)

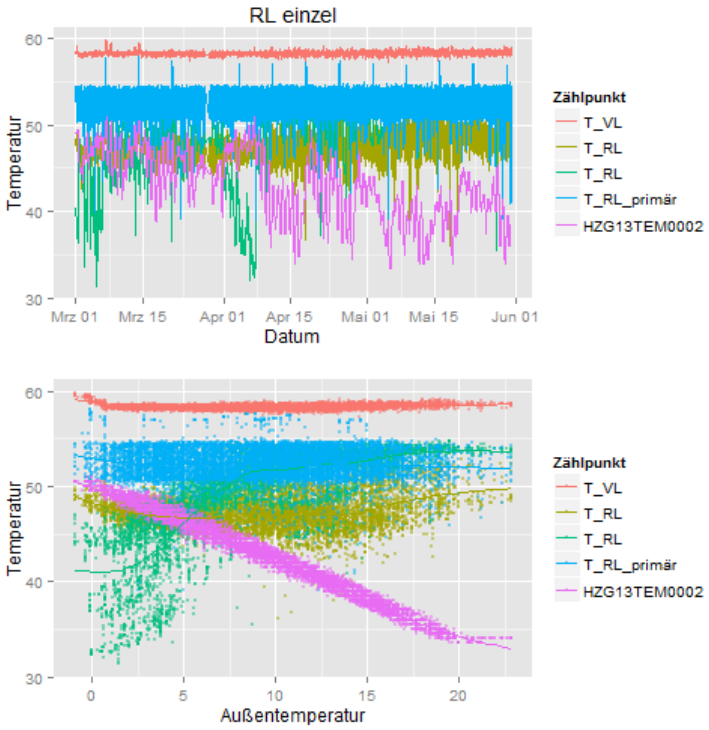


Figure 4: plot of chunk Temperaturvergleich

Jahresdauerlinie (Kälteleistung)

Aggregatsfunktion: avg (Stundendurchschnittswerte)

Jahresdauerlinie im Stundentakt über die letzten zwei Jahre vor der Berichtserstellung.

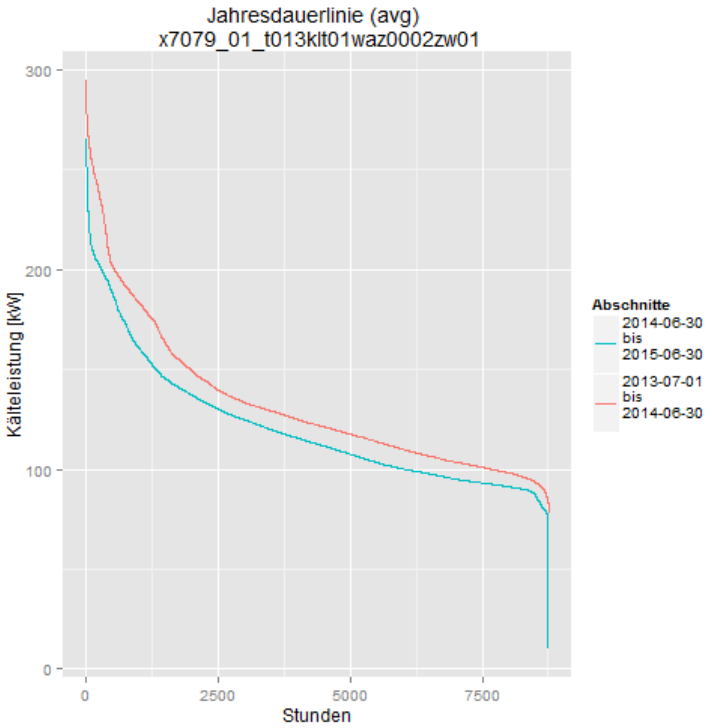


Figure 5: plot of chunk Jahresdauerlinie

Jahresdauerlinie (Aussentemperatur)

Aggregatsfunktion: avg

```
## Die Anzahl der unaggregierten Beobachtungen, die  x7070_01_u138hzg03tem0003mp01 -20 and 5
```

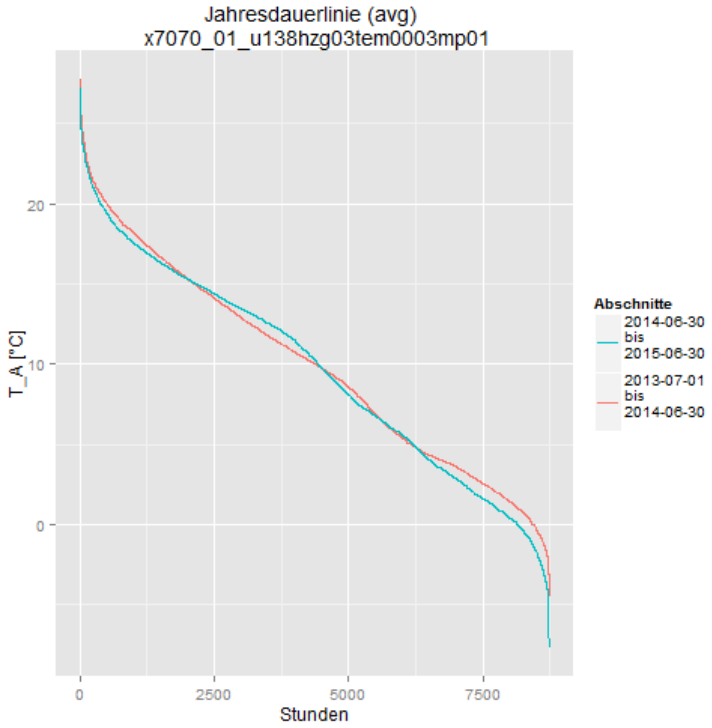


Figure 6: plot of chunk Jahresdauerlinie 2

Jahresdauerlinie (Aussentemperatur Wetterdaten)

Aggregatsfunktion: avg

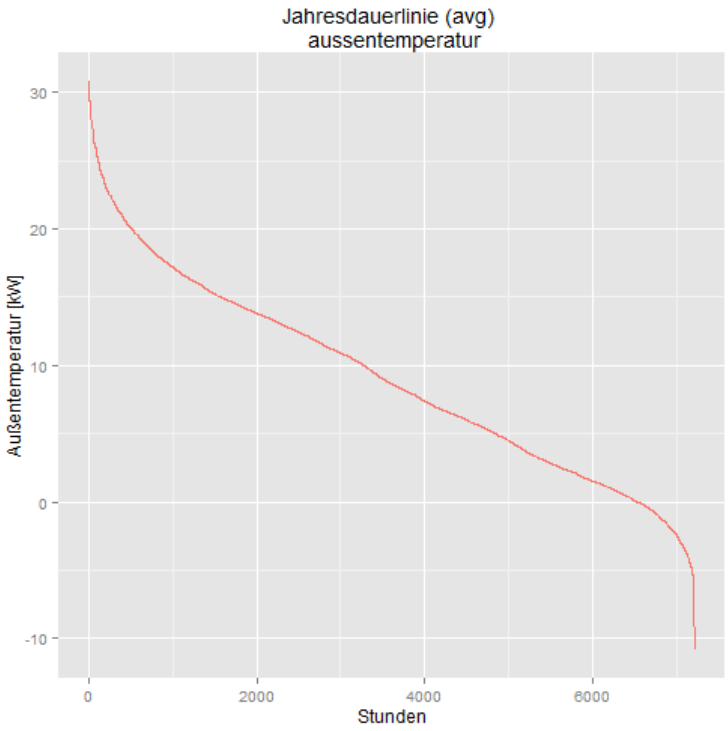


Figure 7: plot of chunk Jahresdauerlinie 3

4.4 Grafiken

Die Berichte enthalten eine Reihe von grafischen Darstellungen der Daten, um die darin enthaltenen Unregelmäßigkeiten schnell erkennen zu können. In diesem Abschnitt wird auf die bereits implementierten Grafiken eingegangen, es können aber jederzeit neue Funktionen für Auswertungen implementiert und in die Software und Berichte eingebunden werden.

Es existiert zu jeder Grafik mindestens eine Funktion, welche die Grafik mit fertig aggregierten und bereinigten Daten erstellt (z. B. `energiesignatur`), und eine weitere Funktion mit dem Zusatz `_db` (z. B. `energiesignatur_db`), die die Daten aus der Datenbank abfrägt, ggf. bereinigt und aufbereitet und sie dann an die Funktion zur Grafikerstellung übergibt. Damit die Berichterstellung im Falle eines Fehlers bei einer einzelnen Grafik nicht abgebrochen wird, gibt es zudem jeweils noch eine Funktion mit dem Zusatz `_db_t` (z. B. `energiesignatur_db_t`), die der Funktion mit dem Zusatz `_db` entspricht, aber zusätzlich eine Ausnahmebehandlung beinhaltet. Es wird empfohlen, diese Funktionen in den Berichten zu verwenden.

Zum Erstellen aller Grafiken müssen zuerst die benötigten R-Pakete eingelesen und eine Verbindung zur Datenbank geöffnet werden.

```
# Laden der benötigten Pakete
library(EnMoLMU)
library(RODBC)
# ODBC-Verbindung zu MySQL öffnen:
ch <- odbcConnect("ansi_benutzer",  DBMSencoding = "utf-8")
```

4.4.1 Zeitraumauswahl in den Grafikfunktionen

Um in den Grafikfunktion (z. B. `energiesignatur_db, hydr_abgleich_db, jahresdauerlinien_db und tagesverlauf_db`) die Zeiträume auswählen zu können, für die die Grafiken erstellt werden sollen (z. B. über das Argument `between`), müssen diese im Format `"'2015-03-01 00:00:00' and '2015-05-31 23:59:59'"` übergeben werden. Um in den Berichtsvorlagen automatisch immer einen aktuellen Zeitraum zu haben und die Eingabe zu vereinfachen, können die Funktionen `letzter_monat()` etc. verwendet

werden. Diese Funktionen erstellen einen Character-String eines Zeitraums der Dauer eines Monats/Jahres/etc., der in der letzten Minute (um 23:59:59 Uhr) des zum Zeitpunkt des Aufrufs vergangenen Monats endet. Die Funktion `bis_letzter_monat` beginnt am 1.1.2013 (also vor dem Zeitpunkt, zu dem die aktuell verfügbaren GLT-Daten beginnen) und endet am vorletzten Monat. An die Funktion `letzter_zeitraum` kann eine beliebige Dauer übergeben werden (siehe Beispiel), während die anderen Funktionen als Vereinfachung davon die wichtigsten Zeiträume erzeugen, ohne dass ein Argument übergeben werden muss.

Folgende Beispiele zeigen die Rückgabe der Funktionen bei einem Aufruf im Juni 2015.

```
> letzter_zeitraum("-3 months")
[1] "'2015-03-01 00:00:00' and '2015-05-31 23:59:59'"
> letzter_monat()
[1] "'2015-05-01 00:00:00' and '2015-05-31 23:59:59'"
> bis_letzter_monat()
[1] "'2013-01-01 00:00:00' and '2015-04-30 23:59:59'"
> letztes_jahr()
[1] "'2014-06-01 00:00:00' and '2015-05-31 23:59:59'"
> letzte_2jahre()
[1] "'2013-06-01 00:00:00' and '2015-05-31 23:59:59'"
> letzte_3jahre()
[1] "'2012-06-01 00:00:00' and '2015-05-31 23:59:59'"
```

4.4.2 Leistungszahl (Coefficient of Performance bzw. Energy Efficiency Ratio)

"Die Leistungszahl ε (abgekürzt LZ), bekannt auch unter den englischen Bezeichnungen Energy Efficiency Ratio (kurz EER) für mechanische Kälteanlagen bzw. Coefficient of Performance (kurz COP) für mechanische Wärmepumpen ist das Verhältnis von erzeugter Kälte- bzw. Wärmeleistung zur eingesetzten elektrischen Leistung. Sie ist abzugrenzen von dem Wärmeverhältnis β für thermische Wärmepumpen bzw. β_0 für thermische Kälteanlagen, welches sich nicht auf die eingesetzte mechanische Leistung, sondern auf den eingesetzten Antriebswärmestrom $\dot{Q}_B$ bezieht.

Bei elektrischen Wärmepumpen (WP) mit Kältemittel gibt die Leistungszahl bzw. COP das Verhältnis der abgegebenen Heizleistung einer Wärmepumpe zur aufgewendeten elektrischen Leistung des Verdichters an. Eine Leistungszahl von z. B. 4,2 bedeutet, dass von der eingesetzten elektrischen Leistung des Kompressors das 4,2- fache an Wärmeleistung bereitgestellt wird. Anders formuliert kann mit dieser Wärmepumpe aus einem Kilowatt elektrischer Leistung 4,2 kW Wärmeleistung zur Verfügung gestellt werden.

Für eine Wärmepumpe mit der Heizleistung $\dot{Q}_H$ ist die Leistungszahl definiert als:

$$\varepsilon_{WP} = \frac{\dot{Q}_H}{P_{el}}$$

"[10]

Die Funktionen `cop_plot`, `cop_plot_db` und `cop_plot_db_t` berechnen erst die Leistungszahl aus der elektrischen Leistung und der Heizleistung und erstellen dann eine Grafik, in der die Leistungszahl als (geglättete) Kurve (unten), und jeweils mit Standardabweichung als Monats- bzw. Wochendurchschnitte (oben links) bzw. als Durchschnitte der einzelnen Wochentage (oben rechts) dargestellt wird. Zur Glättung im unteren Bild wird eine Loess-Kurve verwendet.

Da zum Zeitpunkt der Erstellung der Funktion bzw. dieser Beschreibung keine für diese Darstellung geeigneten Daten zur Verfügung standen, wurde die Beispielgrafik (Abb. 4.4) mit willkürlichen gerade vorhandenen Spalten aus der Tabelle erzeugt.

In EnMoLMU können Grafiken zur Leistungszahl über folgende Funktionen erstellt werden:

`cop_plot_db_t`
`cop_plot_db`
`cop_plot`

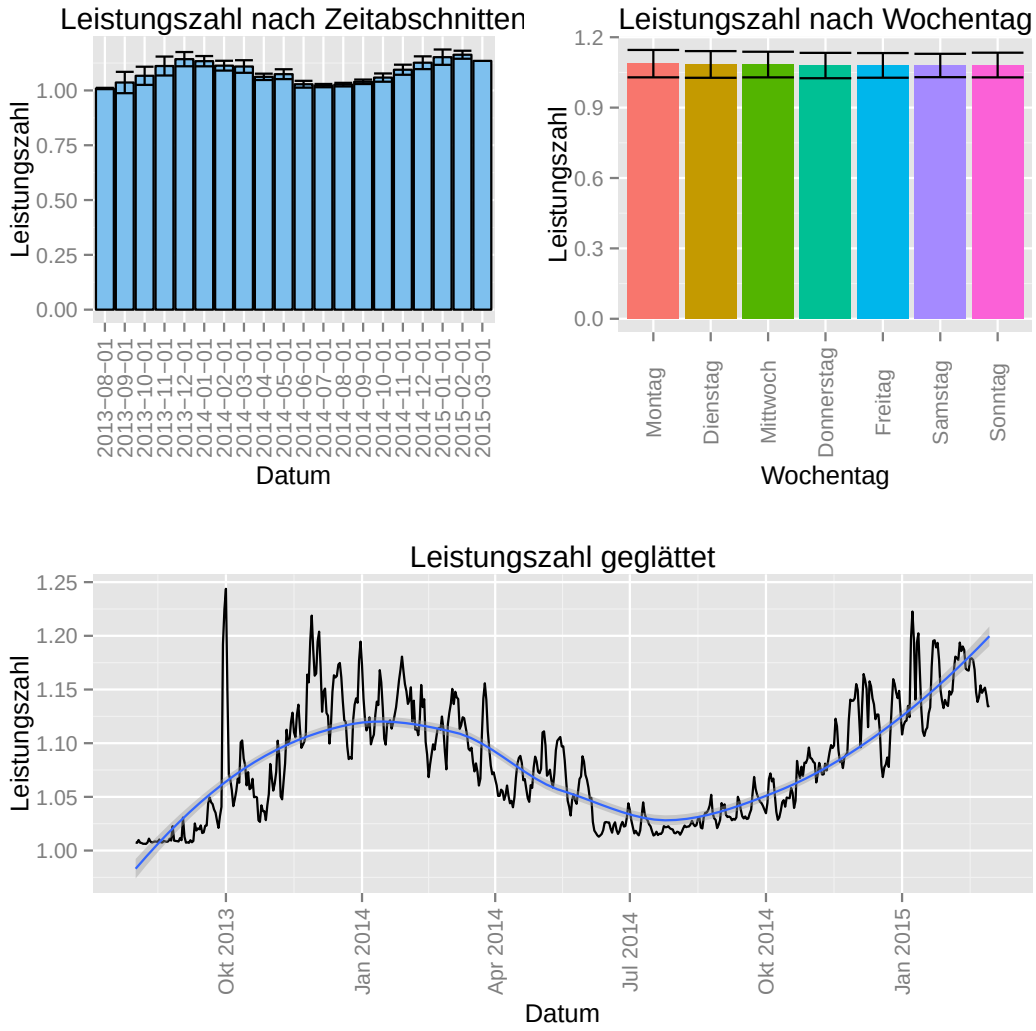


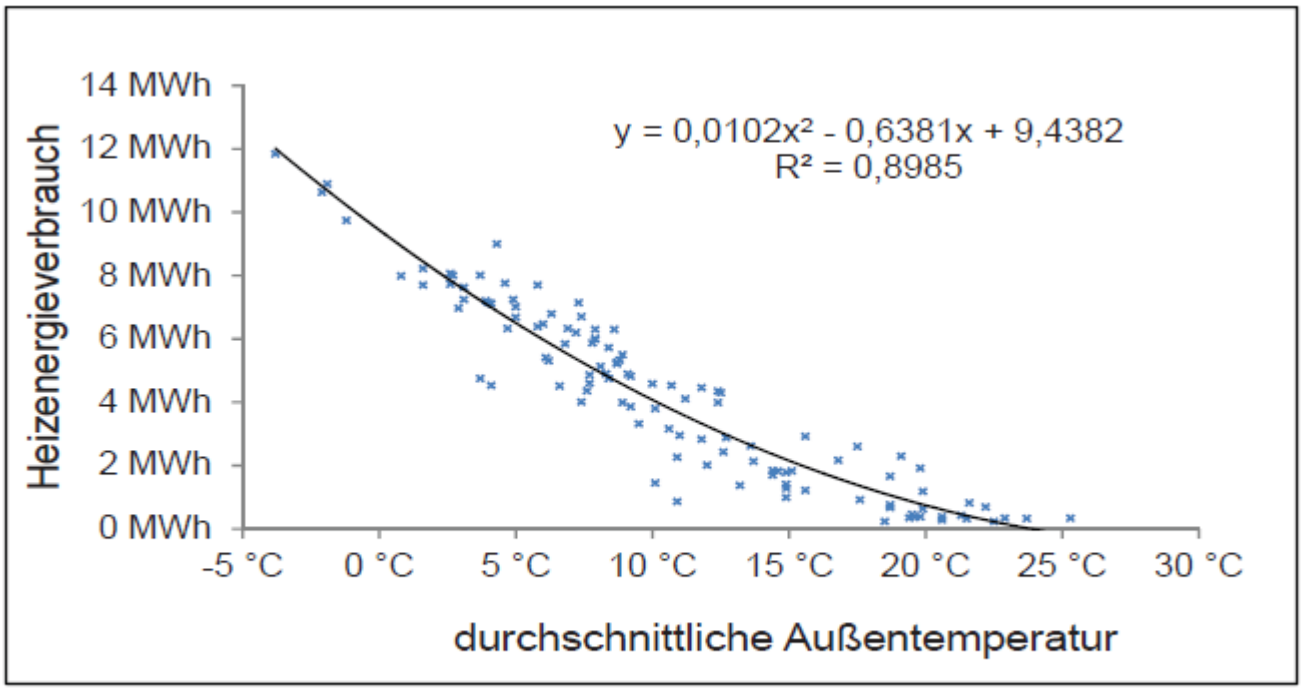
Abbildung 4.4: Leistungszahl (Coefficient of Performance bzw. Energy Efficiency Ratio).

```
cop_plot_db(ch, "glt_hist_oet67", "x7079_01_t013klt01waz0002zw01", "wochentag",
            waermeleistung = "x7070_01_u138hzg03wmz0001zw04",
            durchschnitt = "Monat",
            between = "'2013-08-01' and '2015-03-01'")
```

4.4.3 Energiesignatur

Die Heizenergieverbräuche unterschiedlicher Perioden und Standorte können mittels Gradtagzahlbereinigung gemäß VDI 3807 miteinander verglichen werden (siehe 4.4.8).

Da dazu die Werte für den kompletten jeweiligen Monat benötigt werden, kann diese Art der Auswertung nur mit einer Verzögerung durchgeführt werden. Um allerdings kurzfristig auf betriebsbedingte Verbrauchsanomalien reagieren zu können, ist eine relativ zeitnahe Auswertung und Bereinigung der Heizenergieverbräuche notwendig. Dazu wurde an der Universität Mainz das Konzept der Energiesignaturen in Anlehnung an ein vergleichbares Modell der Universität Ulm[17] entwickelt. Dabei wird ein funktionaler Zusammenhang zwischen dem täglichen Heizenergieverbrauch eines Gebäudes und der Tagesmitteltemperatur hergestellt. Nach Liers lässt sich dieser Zusammenhang am besten mit einem Polynom 2. Grades beschreiben, welches die gebäudespezifische Abhängigkeit des Heizenergieverbrauchs von der Außentemperatur darstellt. Mit Hilfe diese Modells lässt sich sehr einfach und zeitnah eine Prognose eines gebäudespezifischen Soll-Heizenergieverbrauchs erstellen. Die Funktion wird anhand von Werten aus der Vergangenheit geschätzt, für die ein normaler Betrieb angenommen wird. Mit Hilfe dieses Zusammenhangs werden dann tagesaktuelle Verbrauchsprognosen in Abhängigkeit von der Außentemperatur erstellt. Liegen die Ist-Verbräuche über einem definierten Schwellenwert der Soll-Verbräuche, wird eine Alarmierung ausgelöst.[12]



Abhängigkeit des täglichen Heizenergieverbrauchs vom Tagesmittel der Außentemperatur für das Gebäude der Rechts- und Wirtschaftswissenschaften der JGU

Abbildung 4.5: Energiesignatur JGU aus [12]

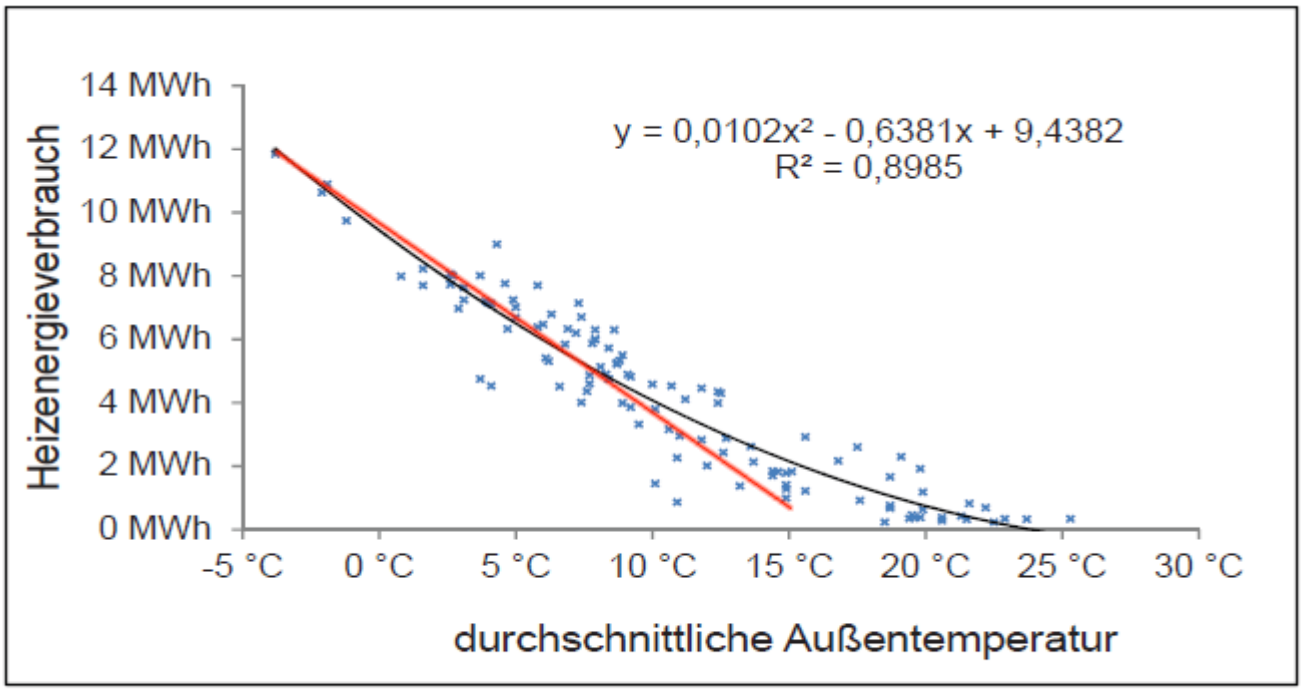


Abbildung 4.6: Energiesignatur JGU aus [12]. Die rote Gerade wurde nachträglich hinzugefügt.

Abb. 4.5 zeigt ein Beispiel für die von Liers vorgeschlagene Darstellung der Energiesignatur mit dem Polynom 2. Grades. Da weitgehend nur bei einer Außentemperatur von unter 15 Grad Celsius geheizt wird, lassen sich die Daten so in zwei getrennte Bereiche unterteilen. Es ist daher inhaltlich nicht sinnvoll, die Annahme zu treffen, dass die den Zusammenhang von Außentemperatur und Heizenergieverbrauch beschreibende Funktion an dieser Stelle immer stetig ohne Sprung und ohne Knick verbunden ist. Der Zusammenhang unterhalb der 15 Grad-Grenze lässt sich durch eine Gerade mindestens genauso gut beschreiben, wie durch ein Polynom 2. Grades, wie z. B. aus der Grafik von Liers (Abb. 4.6) und dem EnMoLMU-Beispiel (Abb. 4.7) ersichtlich ist. Somit verwendet die EnMoLMU-Implementierung diese getrennte Betrachtung und ein Lineares Modell (ohne Polynome) für den Bereich unter 15 Grad (Abb. 4.7). Durch ein Prädiktionsintervall, innerhalb welchem 99,9% der Beobachtungen zu erwarten sind, werden die Grenzen zwischen unauffälligen und auffälligen Werten gezogen. Für den Bereich über 15 Grad ist ein linearer oder konstanter Zusammenhang denkbar. Die Beobachtungen der angesprochenen Beispiele scheinen hier dazwischen zu liegen. Die EnMoLMU-Implementierung

schätzt hier eine Konstante und bei Normalverteilungsannahme das 0,05%- und 99,95%-Quantil als Grenzen. Da hier nicht angenommen wird, dass ein bestimmter Anteil der Messwerte auffällig ist, sondern möglichst automatisch Grenzen festgelegt werden sollen, die möglichst alle unauffälligen Werte einschließen, aber nicht zu weit davon entfernt liegen, und von einer annähernd symmetrischen Verteilung ausgegangen wird, ist es hierfür von untergeordneter Bedeutung, ob die Normalverteilungsannahme erfüllt ist. Durch das Level des Prädiktionsintervalls/der Quantile kann eine Anpassung der Intervallgrenzen erfolgen.

Die Erwartungswerte und Prädiktionsintervalle/Quantile werden aus alten und bereits bereinigten Daten geschätzt. Die neuen und noch nicht überprüften Daten werden dann auf Auffälligkeiten überprüft, indem überprüft wird, ob sie innerhalb der Prädiktionsintervalle/Quantile liegen. Neue Beobachtungen, welche außerhalb der Prädiktionsintervalle/Quantile liegen, werden sowohl farblich hervorgehoben, als zusammen mit dem dazugehörigen Datum auf der Console bzw. in den Berichten ausgegeben.

Sollten in der Praxis häufig falsche Intervalle durch eine unsymmetrische Verteilung auftreten, wäre es denkbar durch ein Generalisiertes Lineares Regressionsmodell andere stetige Verteilungen (z. B. Gamma) zu verwenden oder mit Hilfe der Quantilregression die Quantile direkt zu schätzen (siehe 4.4.3).

Da zur Schätzung der Modelle Daten aus größeren Zeiträumen verwendet werden und einzelne Beobachtungen mit sehr großen Abweichungen bzw. viele fehlerhafte Daten die Schätzergebnisse zu stark verändern können, beinhaltet der Default für die Datenbankvarianten der Funktion eine Datenbereinigung (siehe auch 4.4.3).

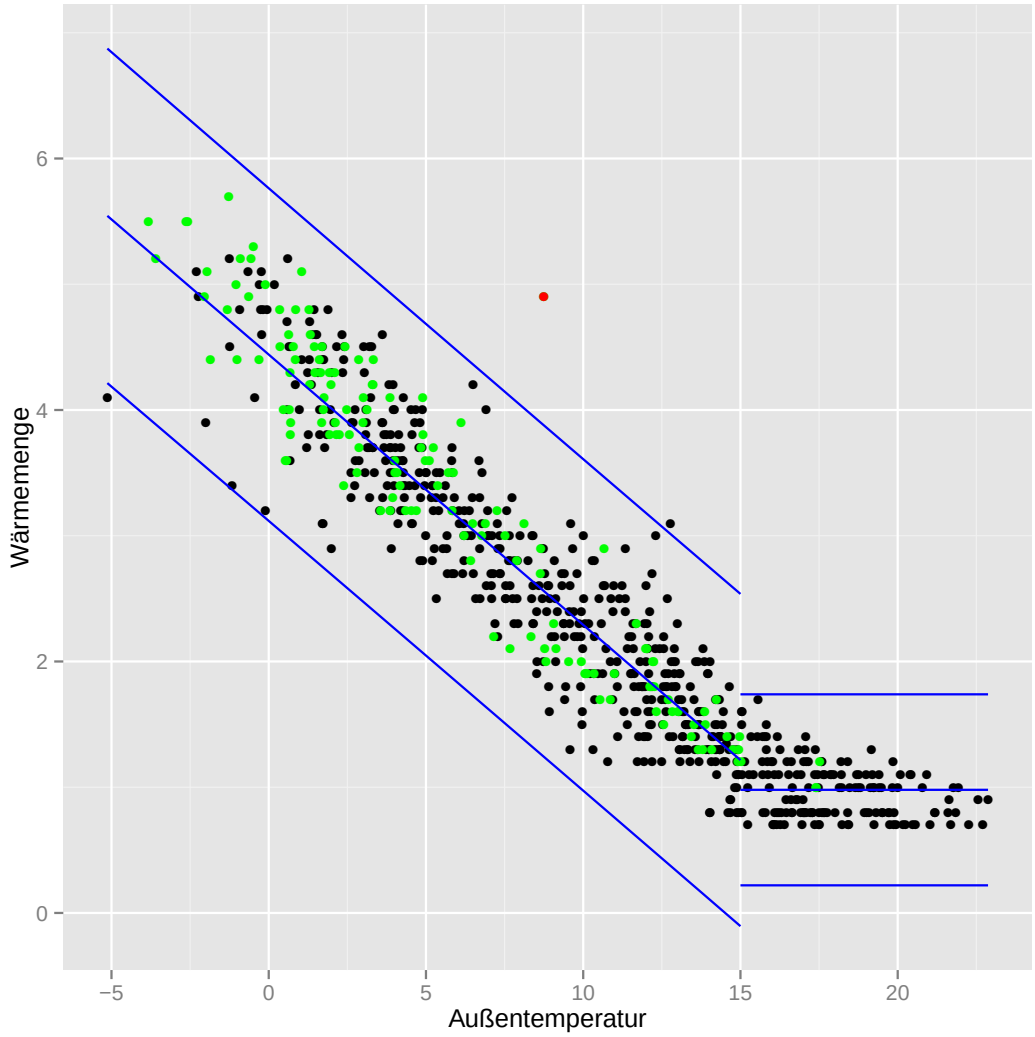


Abbildung 4.7: Energiesignatur. Die Beobachtungen des älteren Zeitraumes aus denen das Lineare Modell bzw. die Konstante geschätzt werden sind in schwarz eingezeichnet. Die mittleren blauen Linien zeigen die Erwartungswerte der beiden Abschnitte und die äußeren blauen Linien das Prädiktionsintervall bzw. das 0,05%- und 99,95%-Quantil. Beobachtungen des neuen Zeitraumes sind in grün eingezeichnet oder, falls sie außerhalb des Prädiktionsintervalls/der Quantile liegen, in rot.

Die Funktion gibt im Falle von Ausreißern, also Punkte des neuen Zeitraumes, die außerhalb der Prädiktionsintervalle liegen, getrennt nach Abschnitt deren Wert, zugehörige Außentemperatur und Datum (letzte Minute der unaggregierten Daten) aus. Im betrachteten Beispiel gab es einen Ausreißer (roter Punkt in Abb. 4.7, der im linearen Abschnitt, also bei einer Außentemperatur von weniger als 15 Grad Celsius vorkam, für den am 29.3.2015 bei einer Außentemperatur von 8,74 Grad eine durchschnittlicher Wärmemen-

genverbrauch von 4,9 gemessen wurde.

```
Ausreißer linearer Abschnitt:
    waermemenge  aussentemperatur                datum
86          4.9          8.743472  2015-03-29 23:59:00
Ausreißer konstanter Abschnitt:
    waermemenge  aussentemperatur                datum
<0 rows> (or 0-length row.names)
```

Der Aufruf lautet:

```
energiesignatur_db(ch, "oet67", "diff_x7070_01_u138hzg03wmz0001zw01",
                   "x7070_01_u138hzg03tem0003mp01",
                   "'2011-01-01' and '2015-01-01'",
                   "'2015-01-02' and '2016-01-01'",
                   bereinigung = list(untere_grenze = 0+1e-100,
                                      quantil_box = c(0.1, 0.9),
                                      faktor_oben = 10))
```

Allgemein kann die Energiesignatur kann über folgende Funktionen verwendet werden:

```
energiesignatur_aufgeteilt_db_ferien_t
energiesignatur_aufgeteilt_db_t
energiesignatur_db_t
energiesignatur_aufgeteilt_db_ferien
energiesignatur_aufgeteilt_db
energiesignatur_db
energiesignatur
energiesignatur_aufgeteilt
```

Abb. 4.8 zeigt die gegenseitigen Aufrufe der Funktionen zum Erstellen der Energiesignatur und deren Aufgaben. Die Funktion `energiesignatur` übernimmt dabei das tatsächliche Erstellen der einzelnen Energiesignatur-Grafiken und `energiesignatur_db` die Abfrage der Daten aus der Datenbank etc.. `energiesignatur_aufgeteilt_db` und

`energiesignatur_aufgeteilt_db_ferien` verwenden die Funktion `energiesignatur_db` für nach Werktag/Wochenende bzw. Semester/Ferien aufgeteilte Daten. Zu jeder dieser Funktionen existiert noch eine weitere Funktion, die diese mit einer Ausnahmebehandlung aufruft. Die Funktion `energiesignatur_aufgeteilt` zeichnet die Energiesignatur mit aufgeteilten Daten, welche sich im Hauptspeicher befinden.

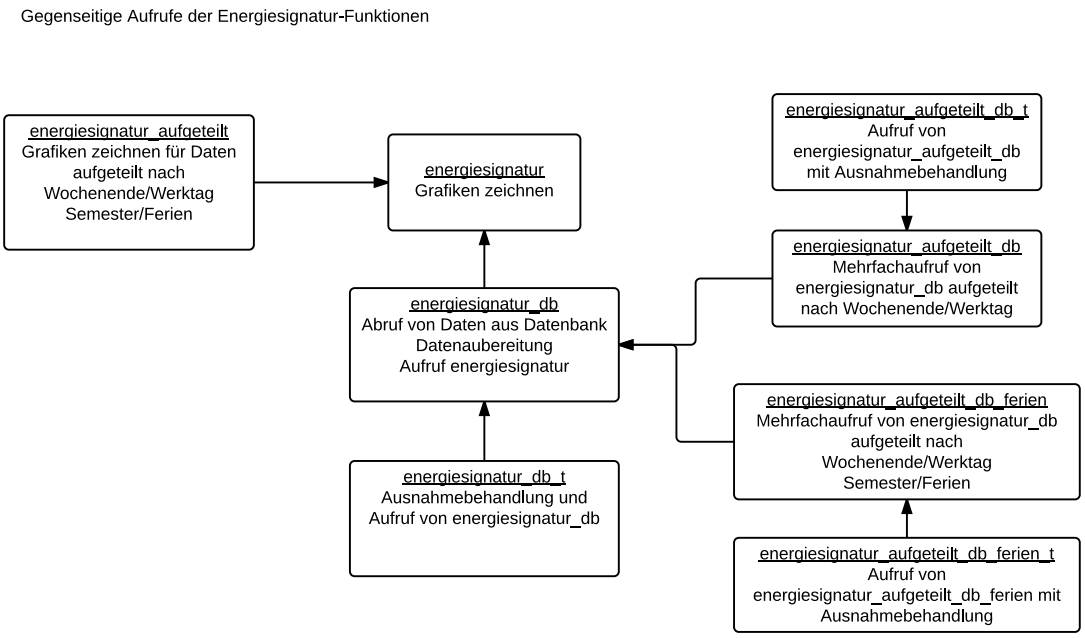


Abbildung 4.8: Funktionen mit denen Grafiken der Energiesignatur erstellt werden können und deren gegenseitige Aufrufe.

Weiter wäre es möglich das Modell so zu erweitern, dass es das Prädiktionsintervall aus allen Daten außer denen des letzten Tages schätzt und nur dann "Alarm schlägt", falls die Beobachtung des letzten Tages außerhalb der Intervallgrenzen liegt. Dies könnte täglich automatisch durch den Task Scheduler gestartet werden und der Alarm über eine Nachricht in einem Popup-Fenster oder durch Versendung eines E-Mails geschlagen werden. Allerdings müssten dazu die GLT-Daten auch bereits täglich automatisch in einen Ordner abgelegt werden, auf den EnMoLMU Zugriff hat.

Aufteilung der Daten nach Werktag/Wochenende und Vorlesungszeit/Semesterferien

Der Verbrauch wird neben der Außentemperatur möglicherweise auch davon beeinflusst, ob er an einem Werktag oder Wochenende und ob er in der Vorlesungszeit oder in den Semesterferien zustande kommt. Somit ist eine getrennte Betrachtung sinnvoll, was mit der EnMoLMU-Software entweder nur nach Wochenende (`energiesignatur_aufgeteilt_db`) oder zusätzlich nach Ferien/Semester (`energiesignatur_aufgeteilt_db_ferien`) möglich ist. Abb. 4.9 zeigt hierfür ein Beispiel aus dem Subserver Oettingenstr. 67. Der Wärmemengenverbrauch an den Werktagen liegt insbesondere in den Semesterferien bei niedrigen Außentemperaturen höher. Ansonsten unterscheiden sich in erster Linie die Streuungen in den vier Einzelbetrachtungen. Dies ist jedoch auf einzelne Ausreißer zurückzuführen, die mit den Default-Einstellungen für die Datenbereinigung nicht entfernt werden. Eine Erhöhung der unteren Grenze der als korrekt betrachteten Daten auf über 0 würde dieses Problem beheben.

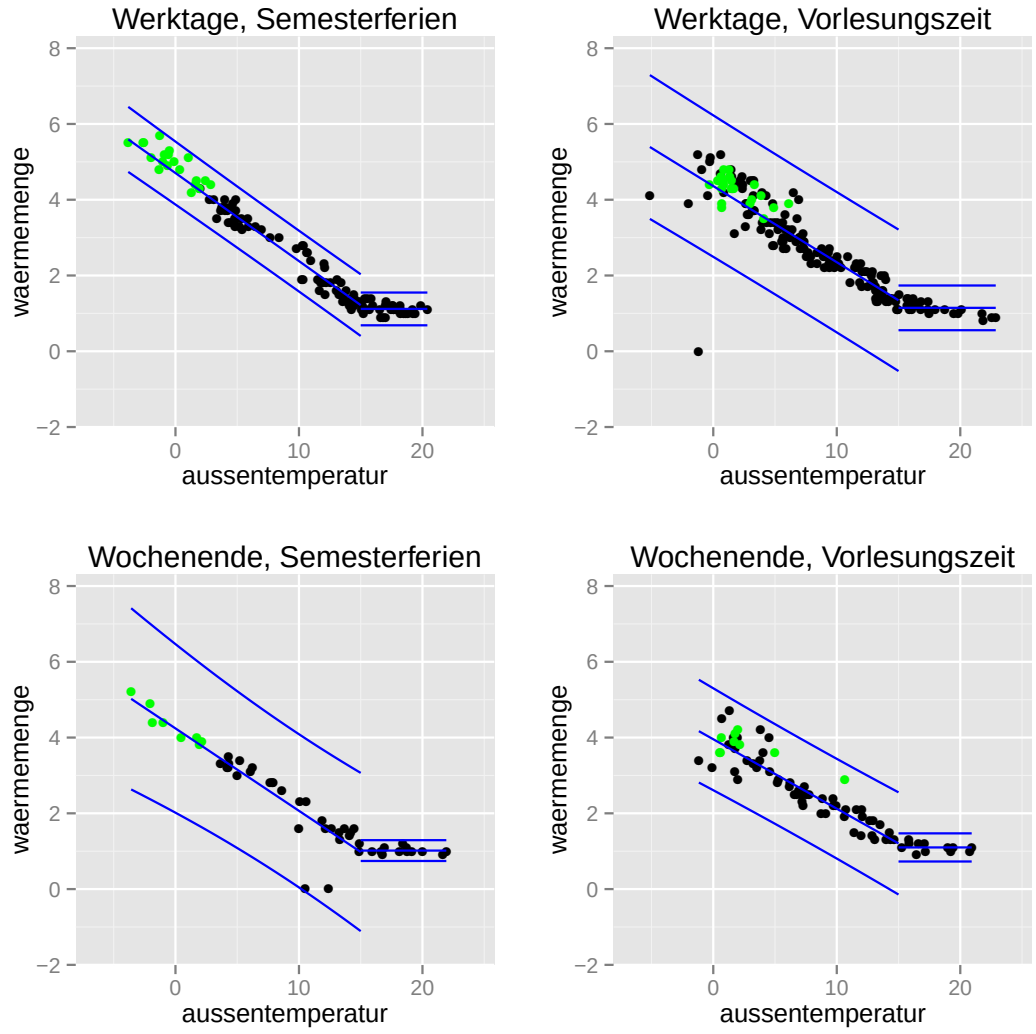


Abbildung 4.9: Energiesignatur. In die oberen beiden Grafiken gehen nur die Beobachtungen ein, die auf einen Werktag fallen, in die unteren beiden nur die, welche auf ein Wochenende fallen. In die linken beiden Grafiken gehen ausschließlich die Tage ein, die zusätzlich in den Semesterferien liegen und in die rechten beiden die, welche auf die Vorlesungszeit fallen.

```
energiesignatur_aufgeteilt_db_ferien(ch, "oet67",
                                     "diff_x7070_01_u138hzg03wmz0001zw01",
                                     "x7070_01_u138hzg03tem0003mp01",
                                     "'2014-01-01' and '2015-01-01'",
                                     "'2015-01-02' and '2015-02-30'")
```

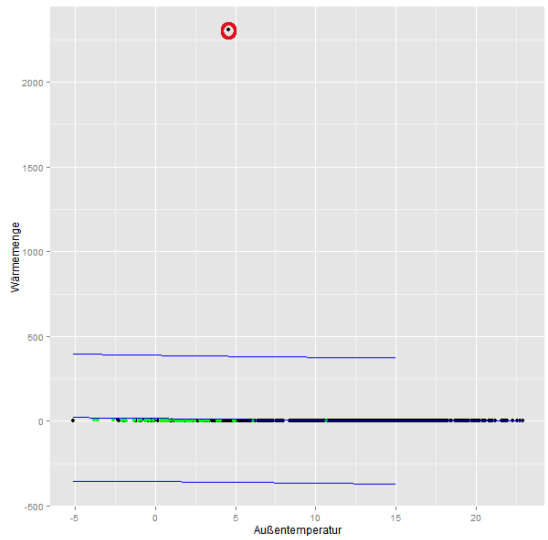
Lineares Modell vs. Quantilregression

Durch den Neueinbau, Austausch und Ausfall eines Zählers und andere Ereignisse ergeben sich häufig fehlerhafte Messwerte. Die hauptsächlich verwendete Methode dieser Software, um die damit verbundenen falschen Schätzungen und Grafiken, auf denen nichts mehr zu erkennen ist, zu vermeiden, ist die Bereinigung der Daten, also das Löschen der offensichtlich fehlerhaften Daten.

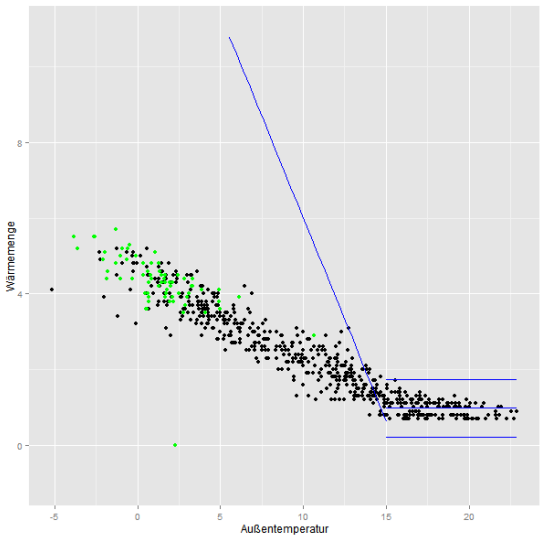
Eine Alternative dazu besteht darin, robustere Methoden, also Methoden, die sich durch die fehlerhaften Daten weniger stark beeinflussen lassen, zu verwenden. Für die Energiesignatur bietet sich hierfür an, statt dem Linearen Modell die Quantilregression zu verwenden. Dabei wird anstatt des bedingten Erwartungswerts der bedingte Median und statt des Prädiktionsintervalls weitere Quantile direkt geschätzt. Da der Median der Wert ist, unter welchem 50% der Beobachtungen liegen, macht es keinen Unterschied, welchen konkreten Wert einzelne extreme Beobachtungen annehmen, wohingegen in die Schätzung des Erwartungswertes alle Beobachtungen mit ihrem Wert eingehen. Allerdings gilt für sehr kleine und sehr große Quantile, dass hier wenige Beobachtungen die Schätzung sehr stark beeinflussen können. Um dieses Problem zu lösen, wurde in der EnMoLMU-Software nicht z. B. das 0,0005- und das 0,9995-Quantil geschätzt, was die Entsprechung zum 0,999-Prädiktionsintervall des Linearen Modells wäre, sondern das 0,25- und 0,75-Quantil (unteres und oberes Quartil) und deren Abstand zum Median mit einem Faktor multipliziert. Eine Alternative wäre es, den Median parallel um einen Faktor zu verschieben, der möglicherweise eine Funktion eines Streuungsmaßes ist, um die Notwendigkeit der manuellen Anpassung zu verringern.

Die Verwendungsmöglichkeiten und Grenzen dieser Methode sollen im Folgenden am Beispiel zweier unterschiedlicher durch fehlerhafte Daten verursachter Probleme verdeutlicht werden. Abb. 4.10 zeigt die Grafik zur Energiesignatur für Daten mit einem einzelnen fehlerhaften Messwert, der ein Vielfaches des Unterschiedes zwischen Minimum und zweitgrößten Wert von letzterem entfernt ist. Somit ist auch das Prädiktionsintervall des Linearen Modells sehr weit von den Daten entfernt (Abb. 4.10a, oben Mitte). Das zweite Bild (Abb. 4.10b) zeigt einen Ausschnitt der y-Achse des ersten Bildes, so dass der geschätzte Erwartungswert erkennbar ist. Dieser wird durch den einen Ausreißer so stark beeinflusst, dass alle anderen Daten darunter liegen. Der zu überprüfende neue Datenpunkt bei Wärmemenge 0 und Außentemperatur von ca. 2 Grad wird nicht als Ausreißer erkannt und rot markiert. Entfernt man den fehlerhaften Datenpunkt, so läuft

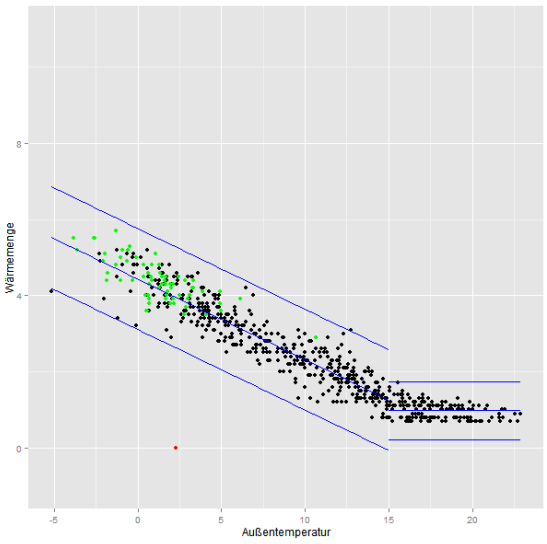
die Regressionsgerade relativ mittig durch die Datenpunkte und das Prädiktionsintervall schließt alle Punkte vollständig ein, ohne weit von ihnen entfernt zu sein (Abb. 4.10c). Damit wird der Ausreißer in den neuen Daten erkannt. Verwendet man die Quantilregression, so passt die Regressionsgerade auch ohne Datenbereinigung genauso gut. Bei dem unteren und oberen Quartil müsste nur der Faktor minimal angepasst werden (Abb. 4.10d). Der Ausreißer bei der Wärmemenge von 0 wird auch hier erkannt. Auch die Verteilung des Konstanten Abschnittes wird hier durch die Quantile wesentlich besser beschrieben, obwohl der Ausreißer hierauf keinen Einfluss hat. Dies liegt an der schiefen Verteilung der Werte.



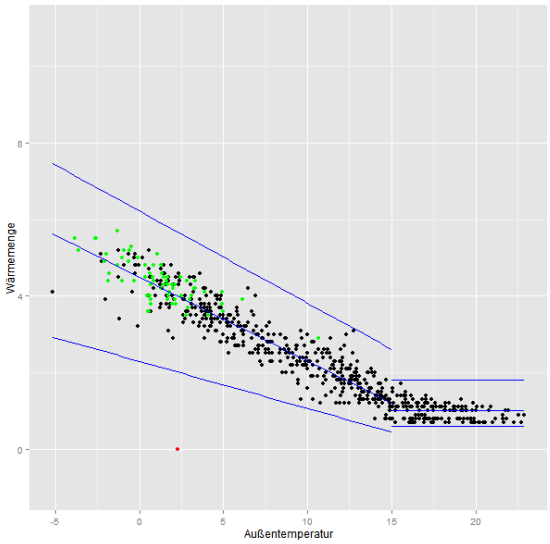
(a) Lineares Modell mit ursprünglichen Daten (Ausreißer siehe roter Kreis)



(b) Lineares Modell mit ursprünglichen Daten (Ausschnitt der y-Achse)



(c) Lineares Modell mit Datenbereinigung (Default; Ausschnitt der y-Achse)

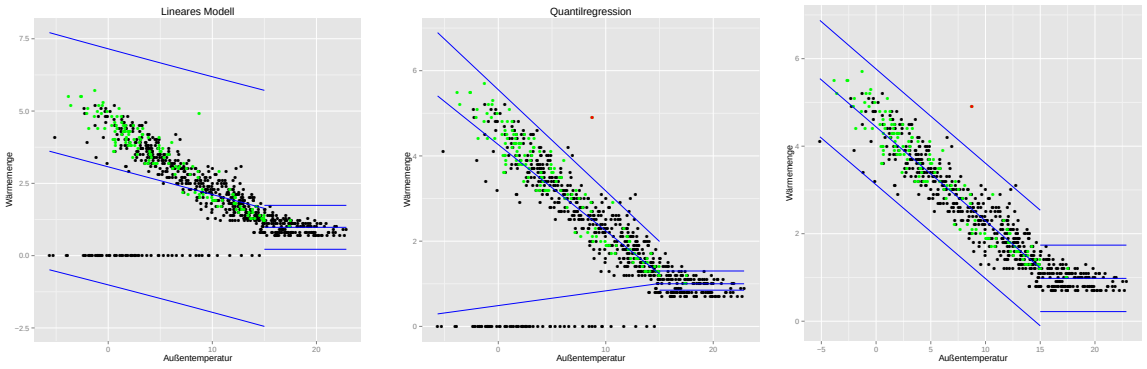


(d) Quantilregression (Ausschnitt der y-Achse)

Abbildung 4.10: Energiesignatur - ein einzelner extrem abweichender Wert

Für das zweite Beispiel wurden Daten mit vielen systematisch angeordneten fehlerhaften Messwerten, die dafür aber nicht so extrem wie im ersten Beispiel abweichen, verwendet (Abb. 4.11). Beim Linearen Modell ohne Datenbereinigung (Abb. 4.11a) schneidet die Regressionsgerade die Datenwolke zwar, hat aber eine viel zu geringe Steigung. Das Prädiktionsintervall ist viel zu breit. Mit Hilfe der Quantilregression (Abb. 4.11b) werden

zwar der Median und die obere Intervallgrenze gut geschätzt, aber die untere Grenze wird viel zu stark von den falschen Werten beeinflusst und hat dadurch sogar eine negative Steigung. Theoretisch müsste allerdings auch der Median nach unten verschoben sein, da die fehlerhaften Daten, hätten sie die Verteilung der restlichen Daten, zur Hälfte oberhalb des Medians liegen würden. Hier liefert also nur das Modell mit den bereinigten Daten eine gute Schätzung (Abb. 4.11c).



(a) Lineares Modell (Beispiel aus der Hilfe)

(b) Quantilregression (Beispiel aus der Hilfe)

(c) Lineares Modell mit bereinigten Daten (Beispiel aus der Hilfe)

Abbildung 4.11: Energiesignatur - viele falsche Werte

Die Energiesignatur-Variante mit der Quantilregression soll nur die prinzipielle Möglichkeit und den möglichen Nutzen aufzeigen und ist noch nicht vollständig ausgereift. Sollte sich die Datenbereinigung in der Praxis nicht bewähren, oder als zu aufwändig herausstellen, so könnte eine Variante der Quantilregression (z. B. bei der die Prädiktionsintervalle durch Verschiebung des Medians in Abhängigkeit von einem Streuungsmaßes berechnet werden) eine Lösung darstellen.

4.4.4 Hydraulischer Abgleich

Die Grafik zum Hydraulischen Abgleich (z. B. Abb. 4.12) zeigt die verschiedenen Temperaturen eines Heizungssystems als einzelne Kurven von Wassertemperaturen, die von unterschiedlichen Gebäudebereichen zurück zur Heizungszentrale laufen. Diese werden zeitlich (oberes Bild) und gegen die Außentemperatur als Streudiagramm mit einer geglätteten Kurve (Additives Modell oder Loesskurve) (unteres Bild) aufgetragen. Bei einem optimalen Heizungsnetz liegen alle Rücklauf-Temperaturen (grüne, tannengrüne,

blaue und lilane Kurve) unabhängig von der Außentemperatur und Zeit auf dem gleichen Niveau. Somit würde sich in beiden Bildern jeweils nur eine horizontale Gerade ergeben. Die Vorlauftemperatur (rote Kurve) sollte mit zunehmender Außentemperatur abnehmen, um dem dann sinkenden Wärmebedarf der Räume zu entsprechen. Dies ist in der Abbildung nicht der Fall, was auf eine fehlerhafte Regelung hinweist. Die mit zunehmender Außentemperatur abnehmende lila Kurve und verschobene grüne, tannengrüne und blaue Kurve deutet auf ein hydraulisches Problem hin (unterschiedlicher Volumenstrom).

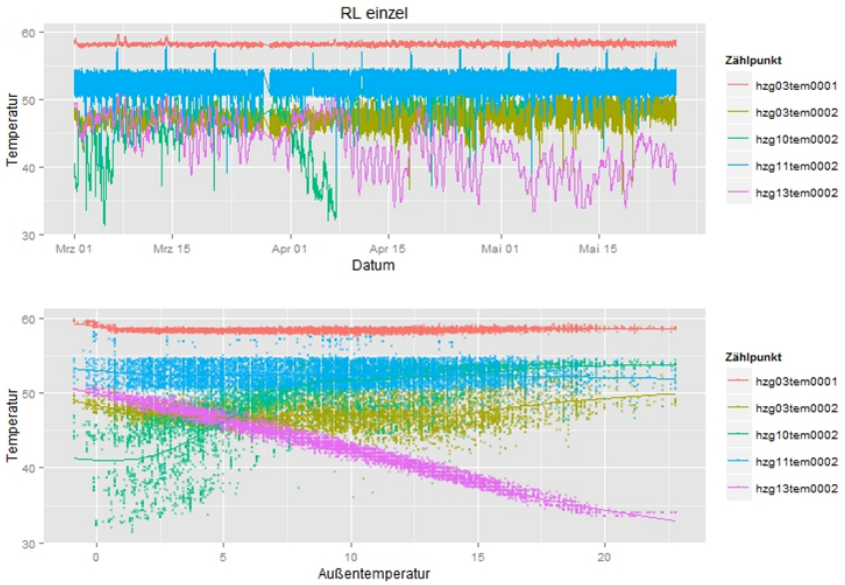


Abbildung 4.12: Hydraulischer Abgleich.

```
hydr_abgleich_db_t(ch, "oet67",
                   c("x7070_01_u138hzg03tem0001mp01",
                     "x7070_01_u138hzg03tem0002mp01",
                     "x7070_01_u138hzg10tem0002mp01",
                     "x7070_01_u138hzg11tem0002mp01",
                     "x7070_01_u138hzg13tem0002mp01"),
                     "x7070_01_u138hzg03tem0003mp01",takt = "15 min",
                    between = "'2015-03-01' and '2015-05-31'",
                    kurzbezeichnungen =
                    "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_oet67.csv")
```

In EnMoLMU können Grafiken zum hydraulischen Abgleich über folgende Funktionen erstellt werden:

```
hydr_abgleich_db_t
hydr_abgleich_db
hydr_abgleich
hydr_abgleich_variablen # Variableneingabehilfsfunktion
```

Um eine größere Anzahl Zähler wie in Abb. 4.13 übersichtlicher darzustellen, kann der Aggregationszeitraum von 15 Minuten auf z. B. Tagesmittelwerte zusammengefasst werden (siehe Abb. 4.14).

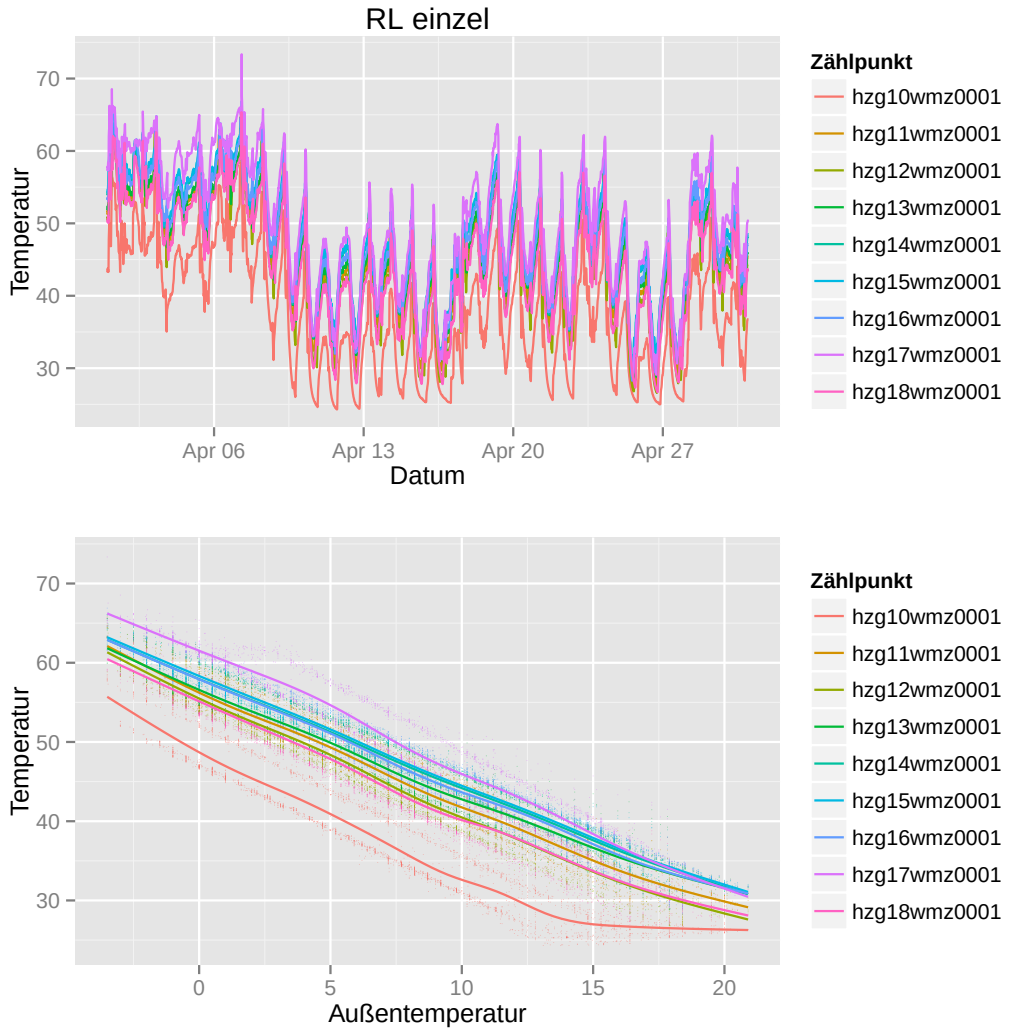


Abbildung 4.13: Hydraulischer Abgleich mit 10 verschiedenen Zählern und 15-Minuten-Takt.

```
hydr_abgleich_db_t(ch, "leo03",
                   c("x0407_01_061bhzg10wmz0001mp01",
                     "x0407_01_061bhzg11wmz0001mp01",
                     "x0407_01_061bhzg12wmz0001mp01",
                     "x0407_01_061bhzg13wmz0001mp01",
                     "x0407_01_061bhzg14wmz0001mp01",
                     "x0407_01_061bhzg15wmz0001mp01",
                     "x0407_01_061bhzg16wmz0001mp01",
                     "x0407_01_061bhzg17wmz0001mp01",
```

```
  "x0407_01_061bhzg18wmz0001mp01"),
"x0407_01_061bhzg00tem0001ma01",takt = "15 min",
between = "'2015-04-01' and '2015-04-30 23:59'",
kurzbezeichnungen =
"D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_leo03.csv")
```

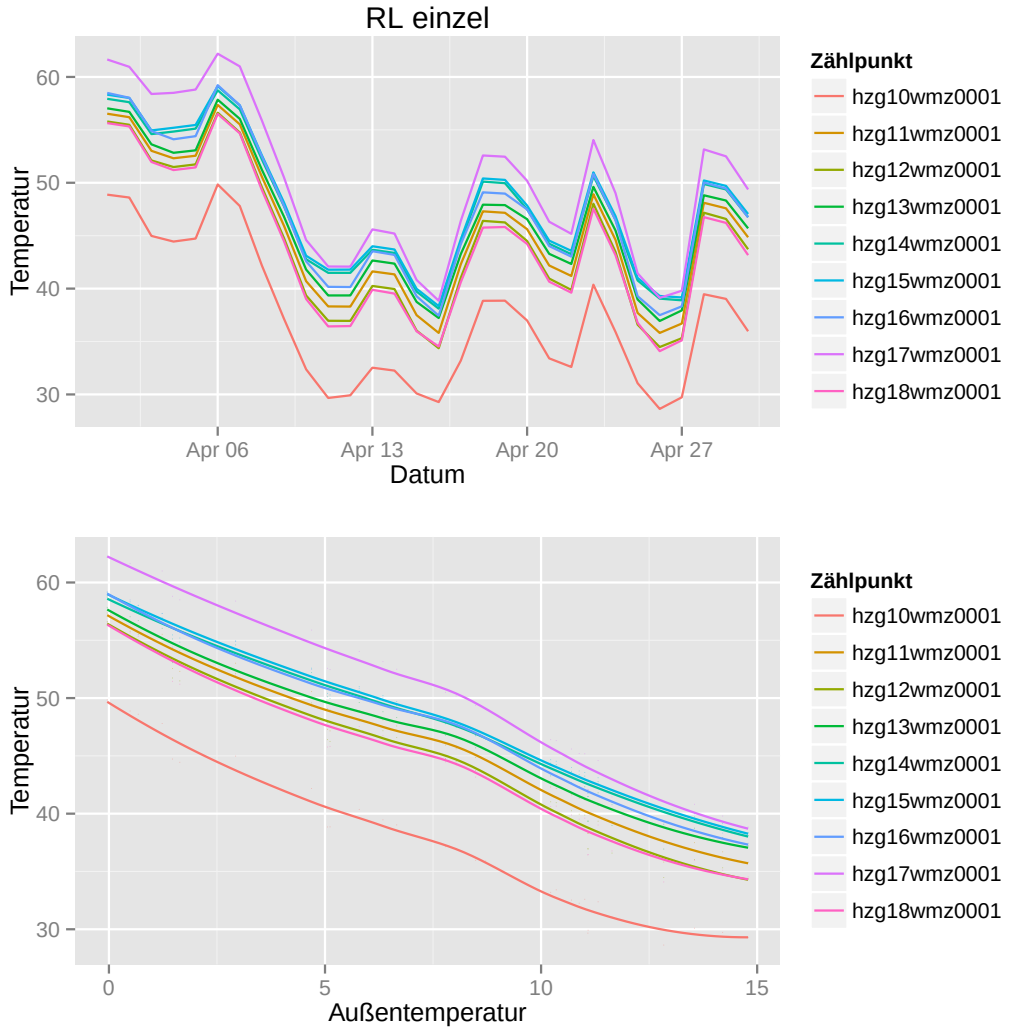


Abbildung 4.14: Hydraulischer Abgleich mit 10 verschiedenen Zählern und Tages-Takt.

```
hydr_abgleich_db_t(ch, "leo03",
                   c("x0407_01_061bhzg10wmz0001mp01",
                     "x0407_01_061bhzg11wmz0001mp01",
```

```
  "x0407_01_061bhzg12wmz0001mp01",
  "x0407_01_061bhzg13wmz0001mp01",
  "x0407_01_061bhzg14wmz0001mp01",
  "x0407_01_061bhzg15wmz0001mp01",
  "x0407_01_061bhzg16wmz0001mp01",
  "x0407_01_061bhzg17wmz0001mp01",
  "x0407_01_061bhzg18wmz0001mp01"),
"x0407_01_061bhzg00tem0001ma01",takt = "tag",
between = "'2015-04-01' and '2015-04-30 23:59'",
kurzbezeichnungen =
"D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_leo03.csv")
```

4.4.5 Jahresdauerlinien

Bei der Jahresdauerlinie handelt es sich um ein Diagramm, das z. B. in der Energiewirtschaft verwendet wird, um den Leistungsbedarf eines Versorgungsobjekts auf Basis der jeweiligen Nutzungszeit darzustellen. Die Jahresdauerlinie zeigt also an, wie viele Stunden im Jahr eine bestimmte Leistung nachgefragt wird. Dies wird z. B. zur Kapazitätsplanung und Wirtschaftlichkeitsbewertung verwendet, da Bedarfsspitzen, die nur wenige Stunden im Jahr auftreten anders abgedeckt werden, als ein nahezu ständig vorhandener Dauerbedarf. Weiterhin wird die Jahresdauerlinie verwendet, um die Auslastung eines Strom- oder Wärmeerzeugers darzustellen.[11]

"In der gebräuchlichsten Form eines Jahresdauerlinien-Diagramms wird die Stundenzahl – maximal 8.760 Jahresstunden – auf der Abszisse dargestellt und die Leistung auf der Ordinate. In dieser Darstellungsform wäre eine waagrecht liegende Jahresdauerlinie der günstigste Fall, weil dies einen jederzeit gleich hohen Bedarf bedeutet. Je steiler die Linie von der Waagrechten abweicht, desto häufiger bleibt eine auf den Maximalbedarf hin dimensionierte Anlage unausgelastet."[11]

In der EnMoLMU-Variante der Jahresdauerlinie können zudem die Jahresdauerlinien für mehrere Jahre nebeneinander innerhalb einer Grafik angezeigt und somit miteinander verglichen werden. Der übergebene Zeitraum wird dabei so aufgeteilt, dass er von der neuesten Datumsgrenze rückwärts in 12-Monats-Abschnitte unterteilt wird und der Rest für eine weitere Kurve verwendet wird (siehe Abb. 4.15).

Der Titel der Grafik (siehe z. B. Abb. 4.15 und Abb. 4.16) beinhaltet (in Klammern) die Aggregationsfunktion, z. B. avg für Durchschnittswerte (engl. average) der Intervalle und den Zählernamen, die x-Achsenbeschriftung die Intervalltaktung, also z. B. Tage und die y-Achse die Bezeichnung und Einheit des Zählers. In Abb. 4.15 wurden Bezeichnung und Einheit des Zählers aus der entsprechenden Spaltenübersicht ausgelesen. In Abb. 4.16 hingegen manuell übergeben.

Die rote Linie in Abb. 4.16 zeigt, dass die Stundendurchschnitte der Wärmeleistung im Zeitraum vom 1.3.2013 bis 27.2.2014 nie unter knapp 50 kW fallen, ca. 7.000 Stunden bei mindestens 100 kW liegen, 2.500 Stunden bei mindestens 130 kW liegen und in der höchsten Stunde innerhalb des Zeitraumes etwa 360 kW erreichen. Im 12-Monats-Zeitraum davor (blaue Linie) ist der Verlauf nahezu identisch, nur dass das Minimum bei etwa 75 kW und das Maximum bei 275 kW liegen.

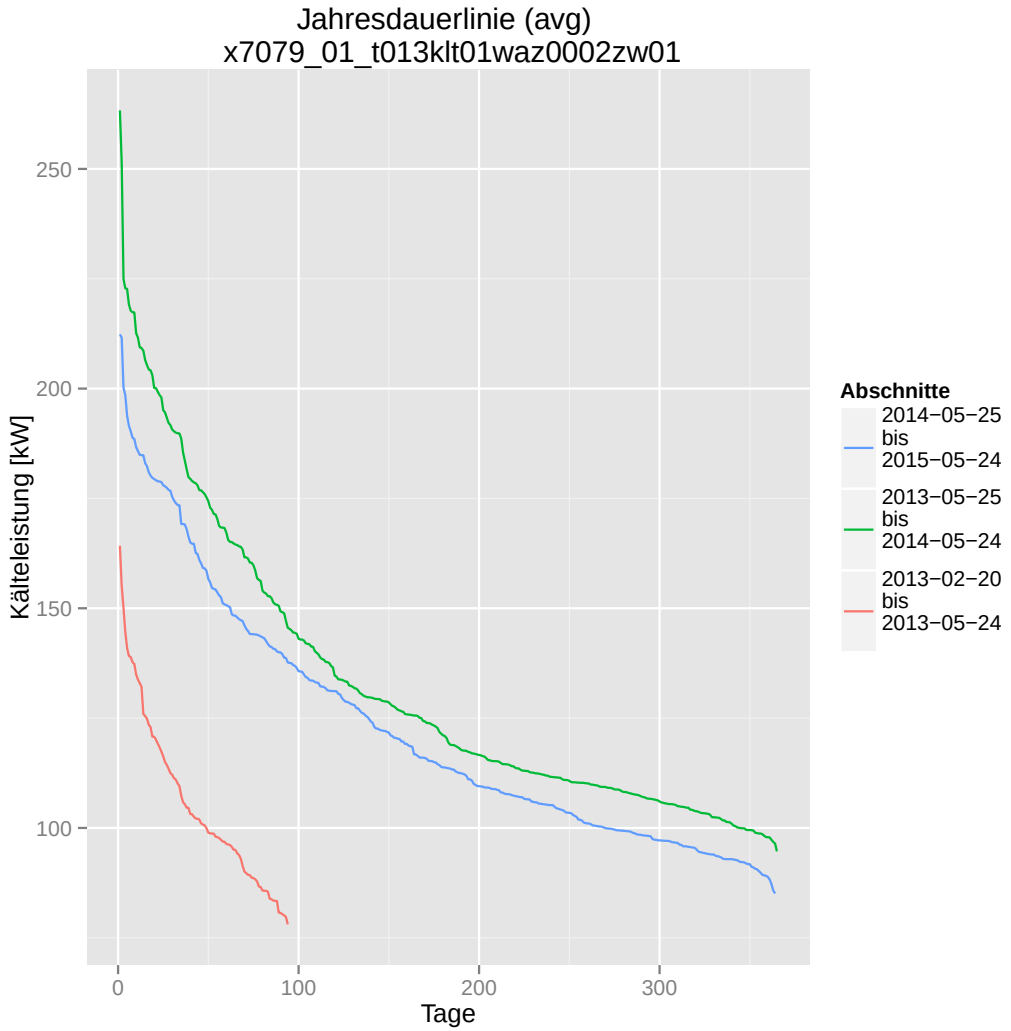


Abbildung 4.15: Jahresdauerlinien der Tagesdurchschnitte der Kälteleistung in kW für die Abschnitte 20.2.2013 bis 24.5.2013 (rot; kein vollständiges Jahr), 25.5.2013 bis 24.5.2014 (grün) und 25.5.2014 bis 24.5.2015 (blau).

```
jahresdauerlinien_db(ch, "oet67", "x7079_01_t013klt01waz0002zw01",
                     intervalle = "Tage",
                     between = "'2011-01-01' and '2016-01-01'",
                     ylab =
                     "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_oet67.csv",
                     einheit =
                     "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_oet67.csv")
```

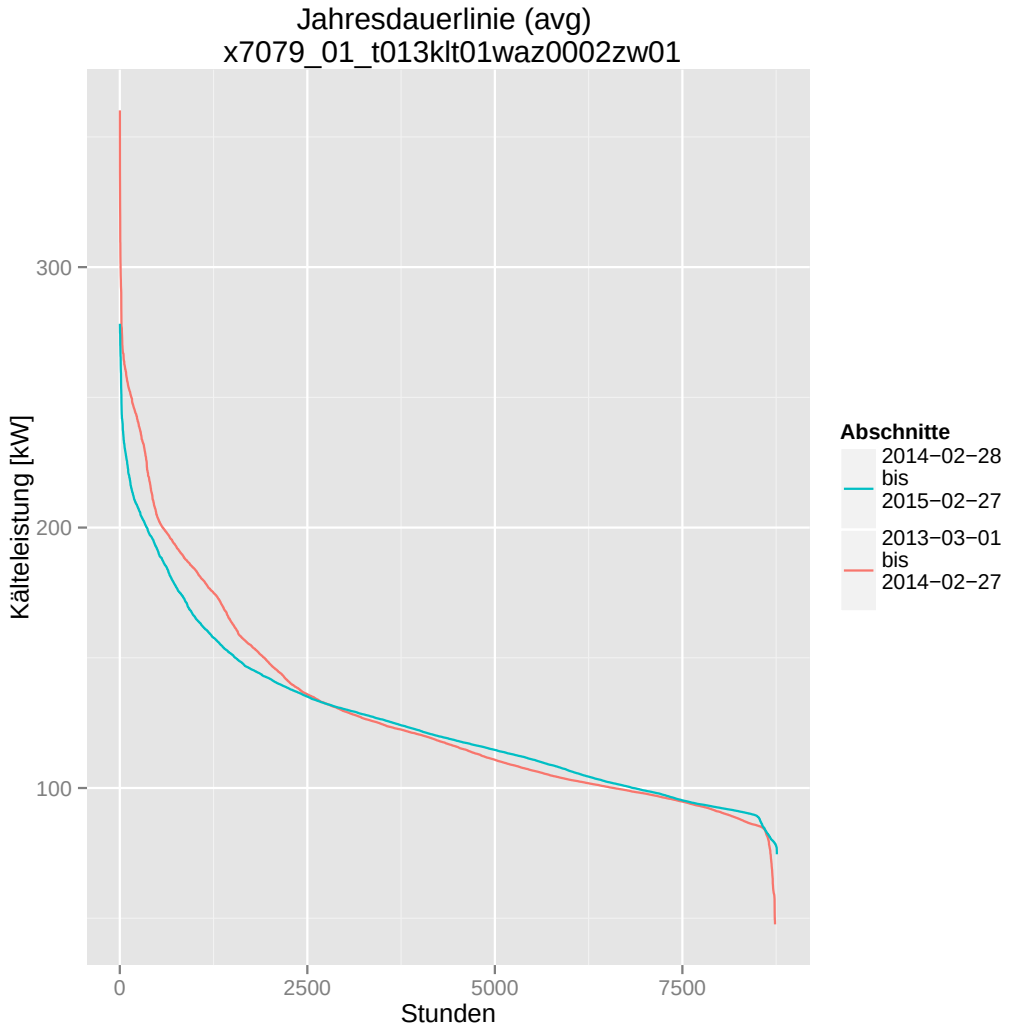


Abbildung 4.16: Jahresdauerlinien der Stundendurchschnitte der Kälteleistung in kW für die Abschnitte 1.3.2013 bis 27.2.2014 (blau) und 28.2.2014 bis 27.2.2015 (rot).

```
jahresdauerlinien_db(ch, "glt_hist_oet67", "x7079_01_t013klt01waz0002zw01",
                     intervalle = "Stunden",
                     between = "'2013-03-01' and '2015-02-28'",
                     ylab = "Kälteleistung", einheit = "kW")
```

Folgende Funktionen stehen zur Erstellung der Jahresdauerlinie zur Verfügung:

```
jahresdauerlinien_db_t
jahresdauerlinien_db
```

`jahresdauerlinien`
`(jahresdauerlinie)`

Abb. 4.17 gibt einen Überblick über die einzelnen Funktionen, mit denen eine Jahresdauerlinie erstellt werden kann und deren Verwendungszweck.

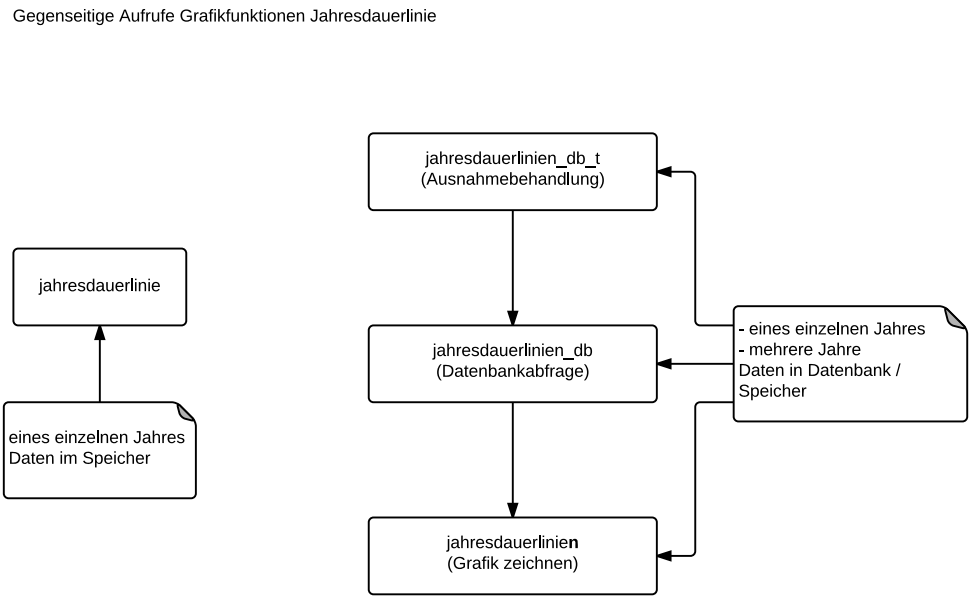


Abbildung 4.17: Funktionen zum Erstellen der Jahresdauerlinie und deren Verwendung.

4.4.6 Kälteerzeugung

Die in diesem Unterabschnitt beschriebene Grafik dient zur Überprüfung der Freikühlung. Zur Kühlung eines Raumes wird ein Trägermedium, wie z. B. ein Kältemittel oder Wasser verwendet. Die Wärme des Raumes wird dann auf dieses Medium übertragen. Das Trägermedium wird danach wieder mittels Ventilatoren über die Außenluft abgekühlt,

falls diese kalt genug ist (Freikühlung). Falls die Abkühlung über die Außenluft nicht ausreicht, erfolgt eine weitere Abkühlung mithilfe einer Kältemaschine. Da die Kältemaschine einen hohen Energiebedarf aufweist, sollte die ausschließlich über die Außenluft erfolgte Abkühlung den Energieverbrauch reduzieren, was allerdings durch die dafür schneller laufenden Ventilatoren relativiert wird, da die Ventilatoren im Freikühlmodus schneller drehen müssen, als im Kältemaschinenmodus.

Mithilfe dieser Grafiken soll nun überprüft werden können, ob durch die Freikühlung wirklich Strom gespart wird und ob die Umschaltung (Ventilstellung) korrekt funktioniert, etc.

Die hier implementierte Darstellung (Abb. 4.18) als Zeitreihen dient dazu, sobald Daten verfügbar sind, diese grob überblicken zu können und eine bessere Grundlage für weitere Auswertungen zu schaffen.

Eine direktere Darstellung der Abhängigkeit des Energieverbrauches von der Ventilstellung und Außentemperatur bzw. der Ventilstellung von der Außentemperatur mit automatischer Fehlererkennung, etc. könnte implementiert werden, sobald entsprechende Daten bzw. eine entsprechende Erfahrung mit der Interpretation dieser Daten vorliegen.

In EnMoLMU können Grafiken zur Kälteerzeugung über folgende Funktionen erstellt werden:

```
kaelteerzeugung_db_t
kaelteerzeugung_db
kaelteerzeugung
```

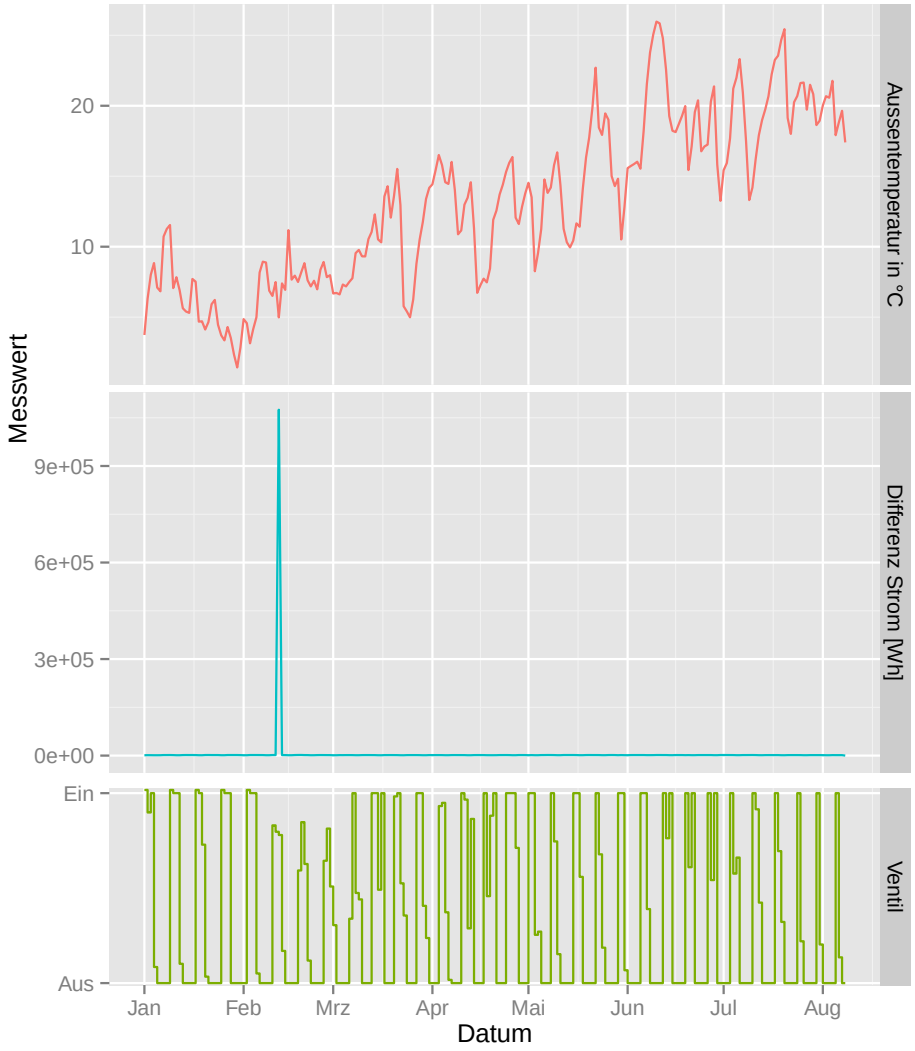


Abbildung 4.18: Kälteerzeugung.

```
kaelteerzeugung_db(ch, "the46",
                   "1110-01_0032HZG00TEM0001MA01",
                   "diff_x1120_01_021anhv01elz0001zw30",
                   "1120+04--529KLT01VEL0003S101",
                   between = "'2014-01-01' and '2014-08-08'",
                   takt = "tag",
                   einheit =
                   "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46.csv")
```

4.4.7 Tagesverlauf

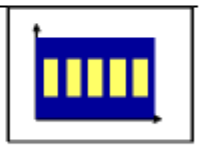
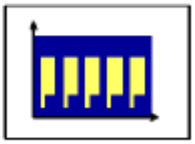
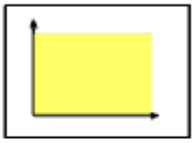
Der Tagesverlauf wird mit Hilfe eines Rasterdiagramms (auch Carpetplot) dargestellt.

Rasterdiagramme eignen sich gut zur Darstellung des zeitlichen Verlaufs einer Messgröße, bei der man die Muster kleinerer Zeitabschnitte, wie Tage, im langfristigen Verlauf, z. B. über ein Jahr, betrachten möchte. Der kleinere Abschnitt erstreckt sich dabei über die y-Achse, der größere über die x-Achse und die Messwerte werden mithilfe einer Farbskalierung angezeigt. Somit können über Rasterdiagramme regelmäßige zeitliche Muster, z. B. durch Betriebs- und Anwesenheitszeiten und ihrer gegenseitigen Abhängigkeiten analysiert werden. Sie werden z. B. zur Betrachtung der charakteristischen Betriebsmuster von Energieverbrauch, Heiz-/Kühlkreisen und RLT-Anlagen herangezogen und ermöglichen eine schnelle Fehlererkennung.[13]

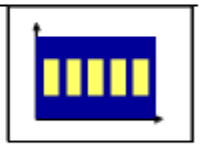
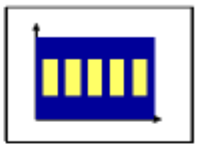
Die folgende Tabelle listet eine Reihe von Mustern für die unterschiedlichen Verbräuche auf und wie diese interpretiert werden können.

Tabelle 23 gibt einen Überblick über den generellen Aufbau der Carpetplots für die Verbrauchswerte.

Tabelle 23 Genereller Aufbau der Carpetplots für Energie- und Wasserverbrauch
AT = Außentemperatur

Plots	Hinweise zur Interpretation	Typisches Erscheinungsbild (vereinfacht)
Wetterdaten (AT und Solarstrahlung)	○ Diese Daten werden nur als Referenz dargestellt	
Fernwärme	○ **Typisches Erscheinungsbild:** In Abhängigkeit der Betriebszeiten und der Regelung können unterschiedliche Muster auftauchen. Falls der Betrieb lediglich über einen festen Zeitplan gesteuert wird, treten meist sehr regelmäßige tägliche und wöchentliche Muster auf (A), die den Absenkbetrieb in der Nacht und am Wochenende anzeigen. In manchen Gebäuden ist nach der Wochenendabsenkung eine spezielle Aufheizphase mit verlängerten Betriebszeiten vorgesehen, um die verstärkte Auskühlung zu kompensieren (B). In anderen Gebäuden wird in Abhängigkeit der AT bzw. bei niedriger AT die Betriebszeit der Heizung verlängert (C) Falls der Zeitplan an Werktagen und Wochenenden gleich ist, ergibt sich Muster (D), bei durchgehendem Betrieb ergibt sich (E). ○ **Saisonale Änderungen:** Typischerweise „verschwindet" das Muster für Fernwärme in den Sommermonaten, sofern die Grundlast gering ist. In der Übergangszeit kann n es vorkommen, dass das Muster nur in den Morgenstunden, nach dem Absenkbetrieb in der Nacht, deutlich zu erkennen ist. Falls an diesen Tagen an den Nachmittagen Kühlenergiebedarf auftritt, könnte dies ein Einsparpotenzial darstellen. Falls die Ferienzeiten im Zeitplan für den Betrieb berücksichtigt sind, sollten diese aus dem Muster ablesbar sein.	A)  B)  C)  D)  E) 
Fernkälte	○ **Typisches Erscheinungsbild:** analog zu Fernwärme ○ **Saisonale Änderungen:** analogo zu Fernwärme, das Muster „verschwindet" jedoch im Winter	
Brennstoffe	○ **Typisches Erscheinungsbild:**	

Fraunhofer ISE 05.07.2011 Seite 113

Plots	Hinweise zur Interpretation	Typisches Erscheinungsbild (vereinfacht)
	In Abhängigkeit der Nutzung des Brennstoffs kann das Muster ähnlich zu dem der Fernwärme oder Fernkälte sein. Wird der Brennstoff jedoch z.B. in einer Anlage zur Kraft-Wärme-Kopplung oder für andere thermische Prozesse eingesetzt, ist zur Interpretation des Musters eine genaue Betrachtung des Systems notwendig.	
Strom	○ **Typisches Erscheinungsbild:** Das typische Muster des Stromverbrauchs für Nichtwohngebäude ist ein klares Wochenmuster (A). Der Stromverbrauch außerhalb der Nutzungszeit ist deutlich geringer. Daher erlaubt der Carpetplot des Stromverbrauchs einen Rückschluss auf die Anwesenheit von Personen. Abweichungen von diesem Muster können ein Hinweis auf Einsparpotenziale sein. ○ **Saisonale Änderungen:** Saisonale Änderungen des Stromverbrauchsmusters treten meist auf, wenn es einen klimaabhängigen Anteil des Verbrauchs gibt. Meist ist das Muster über das Jahr sehr konstant. Ferienzeiten sollten klar ablesbar sein.	A) 
Wasser	○ **Typisches Erscheinungsbild:** Der Wasserverbrauch ist - noch mehr als der Stromverbrauch – ein Indikator für Anwesenheit von Personen (z.B. Toiletten, Kantine). Daher zeigen Nicht-Wohngebäude häufig ein sehr regelmäßiges Wasserverbrauchsmuster. Abweichungen vom Muster können ein Hinweis auf Einsparpotenziale sein (z.B. taucht eine klemmende Toilettenspülung als senkrechte Linie im Plot auf). ○ **Saisonale Änderungen:** Der Wasserverbrauch ist in den meisten Fällen lediglich nur abhängig von der Belegung des Gebäudes. Daher tauchen Ferienzeiten als Unterbrechungen des Musters auf. Saisonale Effekte können z.B. durch Bewässerung von Gärten oder Gründächern oder auch durch den Betrieb von nassen Kühltürmen im Sommer entstehen.	A) 
Gesamtplot	Aus einer vergleichenden Betrachtung der einzelnen Plots können folgende Informationen generiert werden: ○ **Angepasste Betriebszeiten** Ein Vergleich der Muster kann offenbaren, ob Heizung oder Kühlung unnötig betrieben	

Plots	Hinweise zur Interpretation	Typisches Erscheinungsbild (vereinfacht)
	werden. Häufig ist dies in Ferienperioden zu beobachten. Wasser- und Stromverbrauch zeigen zu diesen Zeiten eine Unterbrechung des regelmäßigen Musters. Falls Heizung oder Kühlung nicht dieselbe Unterbrechung aufweisen, stellt dies ein Einsparpotenzial dar.	

[13]

Abb. 4.19 zeigt einen Tagesverlauf der Wirkarbeit (Strom) des Subservers Theresienstraße 46 über einen Zeitraum von etwas mehr als zwei Jahren. Zusätzlich werden die Tagesmittelwerte der Außentemperatur und Globalstrahlung (Sonneneinstrahlung) als Zeitreihe angezeigt. Im Carpetplot bedeutet blau eine niedrige, lila eine mittlere und gelb eine hohe Wirkarbeit. Bei der dunkelgrauen Fläche dürfte es sich um fehlende Werte bei vorhandenen Beobachtungen und bei den (weiß erscheinenden) Flächen in der Hintergrundfarbe um komplett fehlende Beobachtungen (wie sie durch die Datenbereinigung zustande kommen, die extreme Ausreißer entfernt) handeln. Die Unterbrechung in der Außentemperaturkurve im März 2013 kommt ebenfalls durch die Default-Datenbereinigung zustande, die unplausible Temperatur-Messwerte außerhalb des Intervalls zwischen -40 und 45 Grad Celsius entfernt. Der Verbrauch ist täglich zwischen etwa 7.00 Uhr und 17.00 Uhr deutlich erhöht außer an den Wochenenden und der Weihnachtspause (24.12. bis 6.1.). Außerdem fällt eine jährliche Periodizität auf, die sich ungefähr umgekehrt zur Außentemperatur verhält. Der Verlauf der Globalstrahlung ist beim vorhandenen Ausschnitt und ebenfalls von der jährlichen Periodizität betrachtet dem Verlauf der Außentemperatur sehr ähnlich, könnte aber zeitlich etwas nach vorne verschoben sein. Der höchste Verbrauch im betrachteten Zeitraum liegt etwa Anfang Dezember 2014.

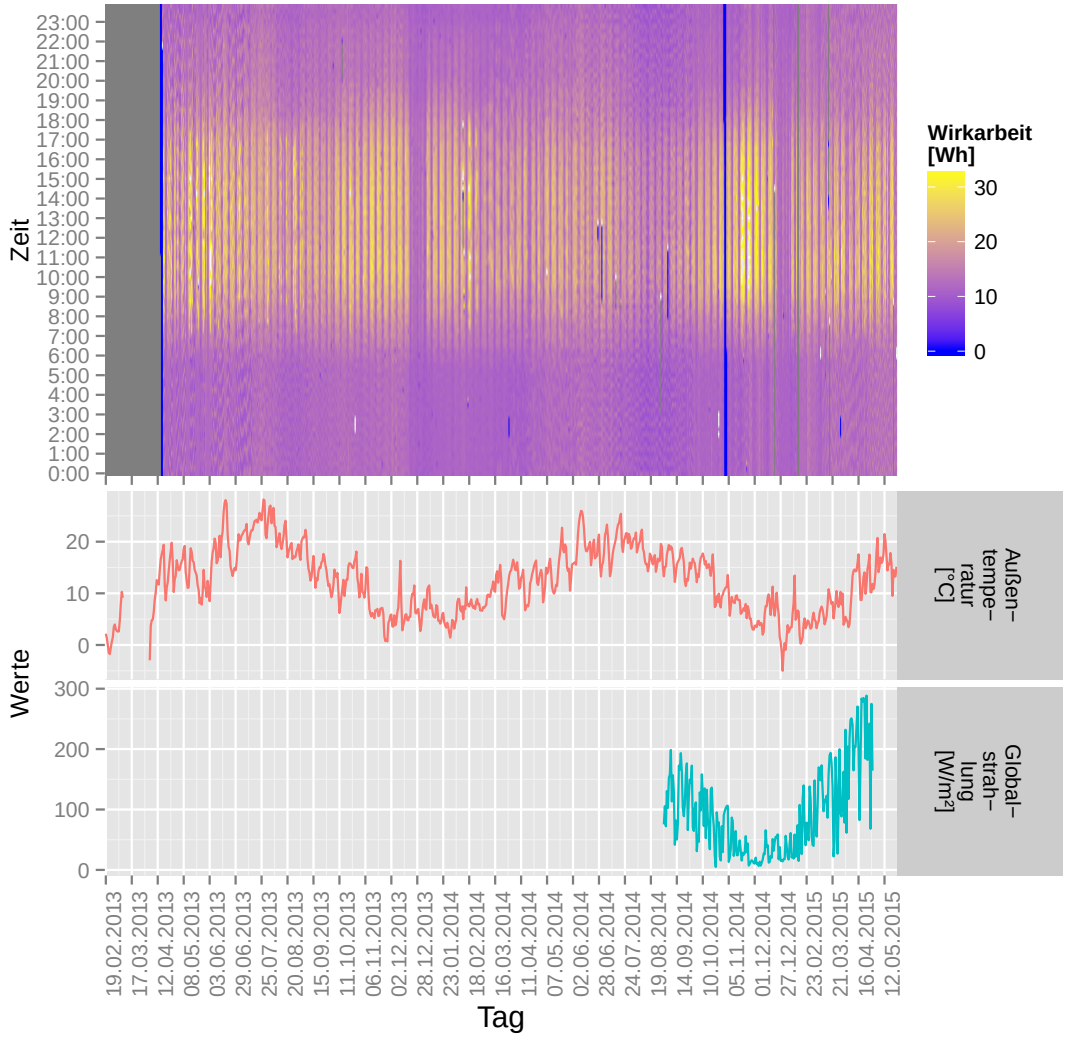


Abbildung 4.19: Tagesverlauf der Wirkarbeit in Wh über einen ca. Zwei-Jahres-Zeitraum im 15-Minuten-Takt als Rasterdiagramm mit den Tagesdurchschnitten der Außentemperatur und Globalstrahlung als Kurve.

```
tagesverlauf_db(ch, tabelle = "the46", datum = "datum",
                werte = "diff_x1120_01_021anhv01elz0001zw30",
                aussentemp = "x1110_01_0032hzg00tem0001ma01",
                tabelle2 = "theresienwetter",
                strahlung = "globalstrahlung",
                abschnitte = "15 Minuten",
                legende = "Wirkarbeit",
                einheit =
```

```
"D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46.csv")
```

Betrachtet man hingegen nur den Zeitraum von zwei Wochen (Abb. 4.20), so ist das Wochenmuster deutlich zu erkennen. Die Grafik beginnt am 6.4.2015, der ein gesetzlicher Feiertag ist (Ostermontag) und einen minimal höheren Verbrauch aufweist, als die Wochenendtage (11./12. und 18./19.4.). An den Werktagen ist der Verbrauch von etwa 7.00 Uhr bis 18.00 Uhr deutlich erhöht und nimmt dann deutlich ab, aber ist bis 24.00 Uhr erkennbar höher als an den Wochenenden. Vormittags scheint der Verbrauch auch meistens höher zu sein, wobei die zweite Woche deutlich unruhiger wirkt. Bis ca. 5.00 Uhr ist der Verbrauch kaum höher als am Wochenende. In der ersten Woche steigt die Außentemperatur an den Werktagen kontinuierlich an, während der Verbrauch im selben Zeitraum kontinuierlich abnimmt. In den ersten vier Tagen der zweiten Woche scheint eher das Gegenteil der Fall zu sein, aber von 16. auf 17.4. nimmt die Außentemperatur wieder deutlich ab und der Verbrauch (zumindest am Vormittag) erkennbar zu. Allerdings verhält sich der Verbrauch in der zweiten Woche sehr ausgeprägt umgekehrt zur Globalstrahlung.

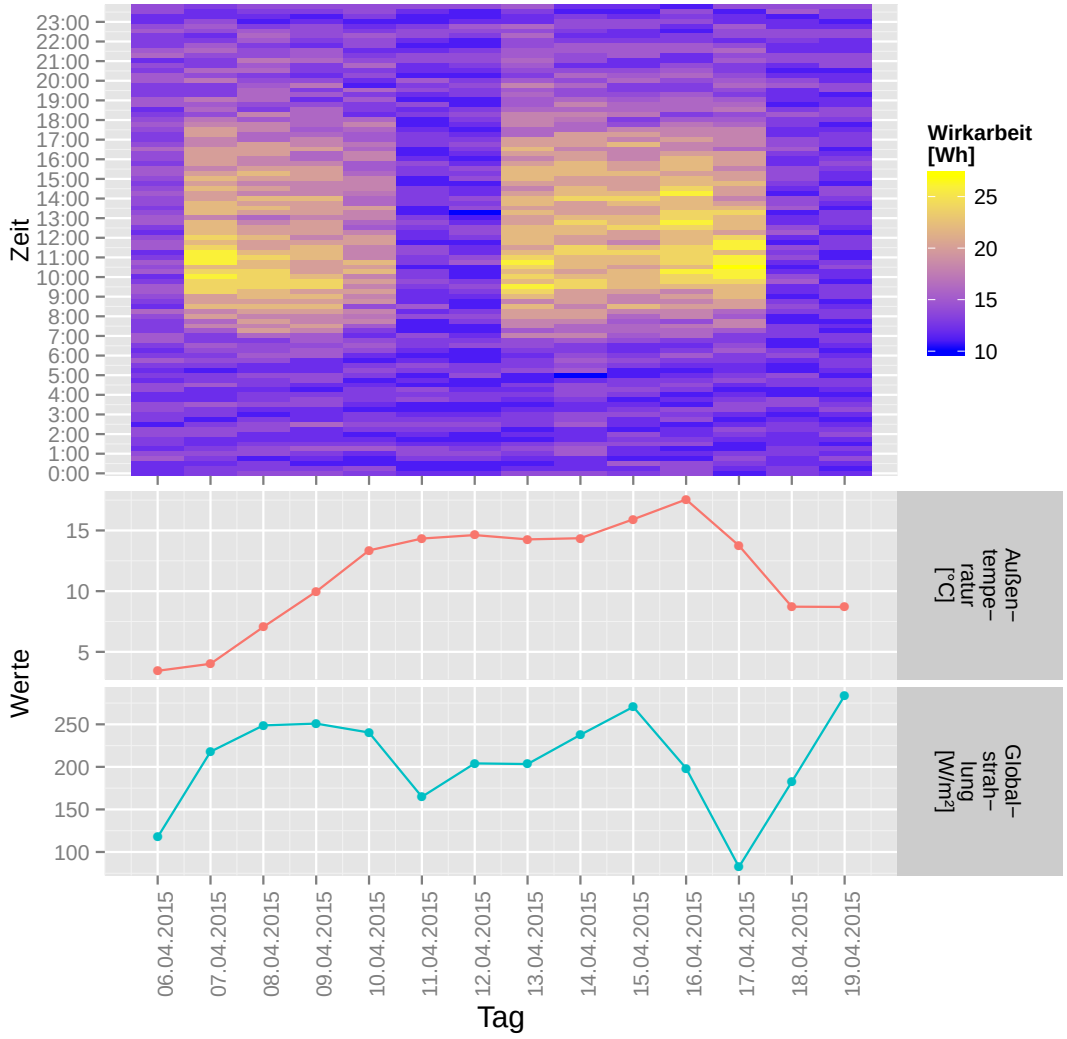


Abbildung 4.20: Tagesverlauf der Wirkarbeit in Wh im 15-Minuten-Takt als Rasterdiagramm mit den Tagesdurchschnitten der Außentemperatur und Globalstrahlung als Kurve. Der zweiwöchige Zeitraum beginnt mit einem Feiertag (Ostermontag).

```
tagesverlauf_db(ch, tabelle = "the46", datum = "datum",
                werte = "diff_x1120_01_021anhv01elz0001zw30",
                aussentemp = "x1110_01_0032hzg00tem0001ma01",
                tabelle2 = "theresienwetter",
                strahlung = "globalstrahlung",
                abschnitte = "15 Minuten",
                between = "'2015-04-06 00:00:00'  and '2015-04-19 23:59:00'",
                legende = "Wirkarbeit",
```

```
einheit =
"D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46.csv")
```

Die im EnMoLMU-Paket zur Erstellung des Tagesverlaufs zur Verfügung stehenden Funktionen lauten:

```
tagesverlauf_db_t
tagesverlauf_db
tagesverlauf
```

Minutentakt

Der Verbrauch kann auch im Minutentakt angezeigt werden. Abb. 4.21 und Abb. 4.22 zeigen die erste Woche aus Abb. 4.20 zum Vergleich im Viertelstunden bzw. Minutentakt. Jedoch muss beim Erstellen der Rasterdiagramme im Minutentakt beachtet werden, dass das Auslesen der unaggregierten Daten aus der Datenbank deutlich länger dauert.

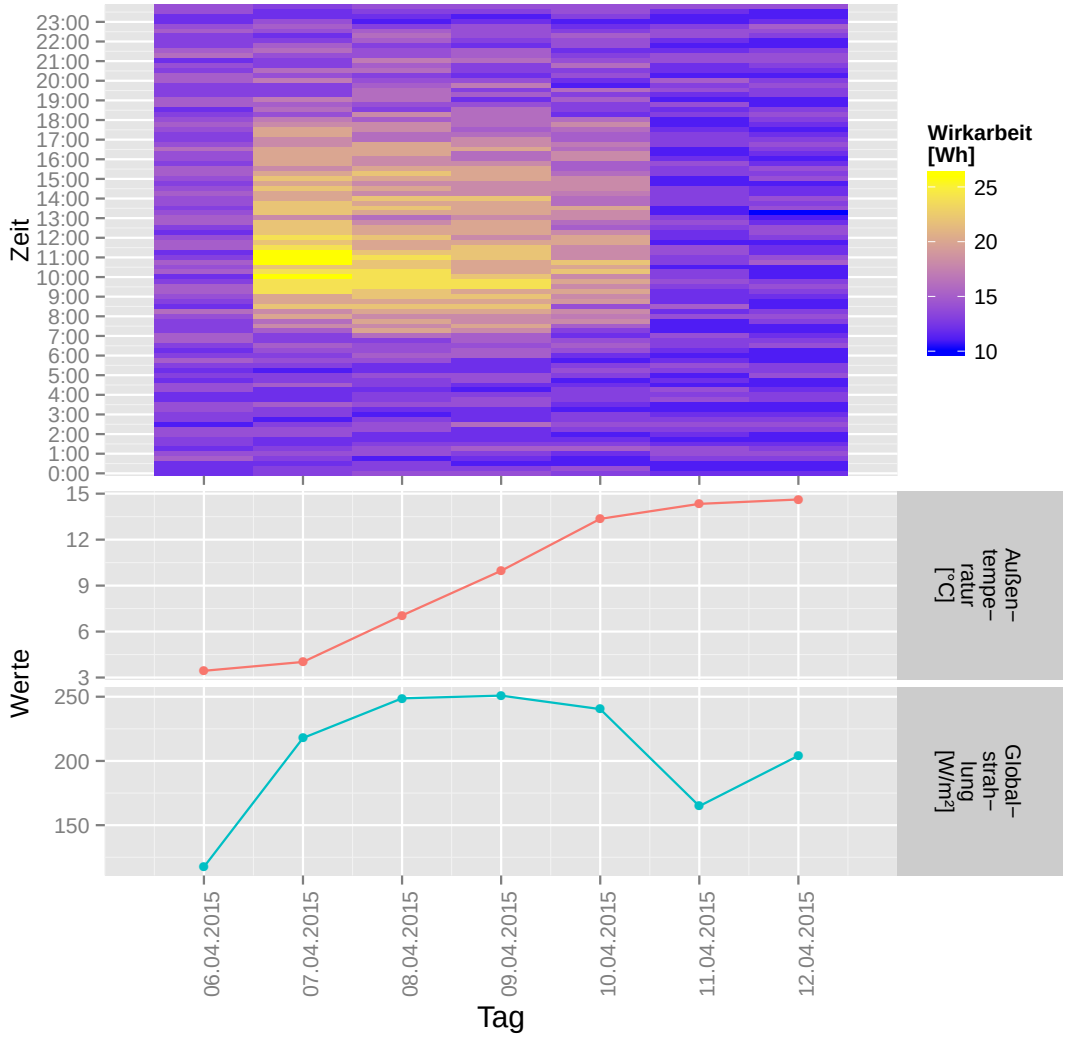


Abbildung 4.21: Tagesverlauf der Wirkarbeit in Wh im 15-Minuten-Takt als Rasterdiagramm mit den Tagesdurchschnitten der Außentemperatur und Globalstrahlung als Kurve. Der einwöchige Zeitraum beginnt mit einem Feiertag (Ostermontag).

```
tagesverlauf_db(ch, tabelle = "the46", datum = "datum",
                werte = "diff_x1120_01_021anhv01elz0001zw30",
                aussentemp = "x1110_01_0032hzg00tem0001ma01",
                tabelle2 = "theresienwetter",
                strahlung = "globalstrahlung",
                abschnitte = "15 Minuten",
                between = "'2015-04-06 00:00:00'  and '2015-04-12 23:59:00'",
                legende = "Wirkarbeit",
```

```
                einheit =
                "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46.csv")
```

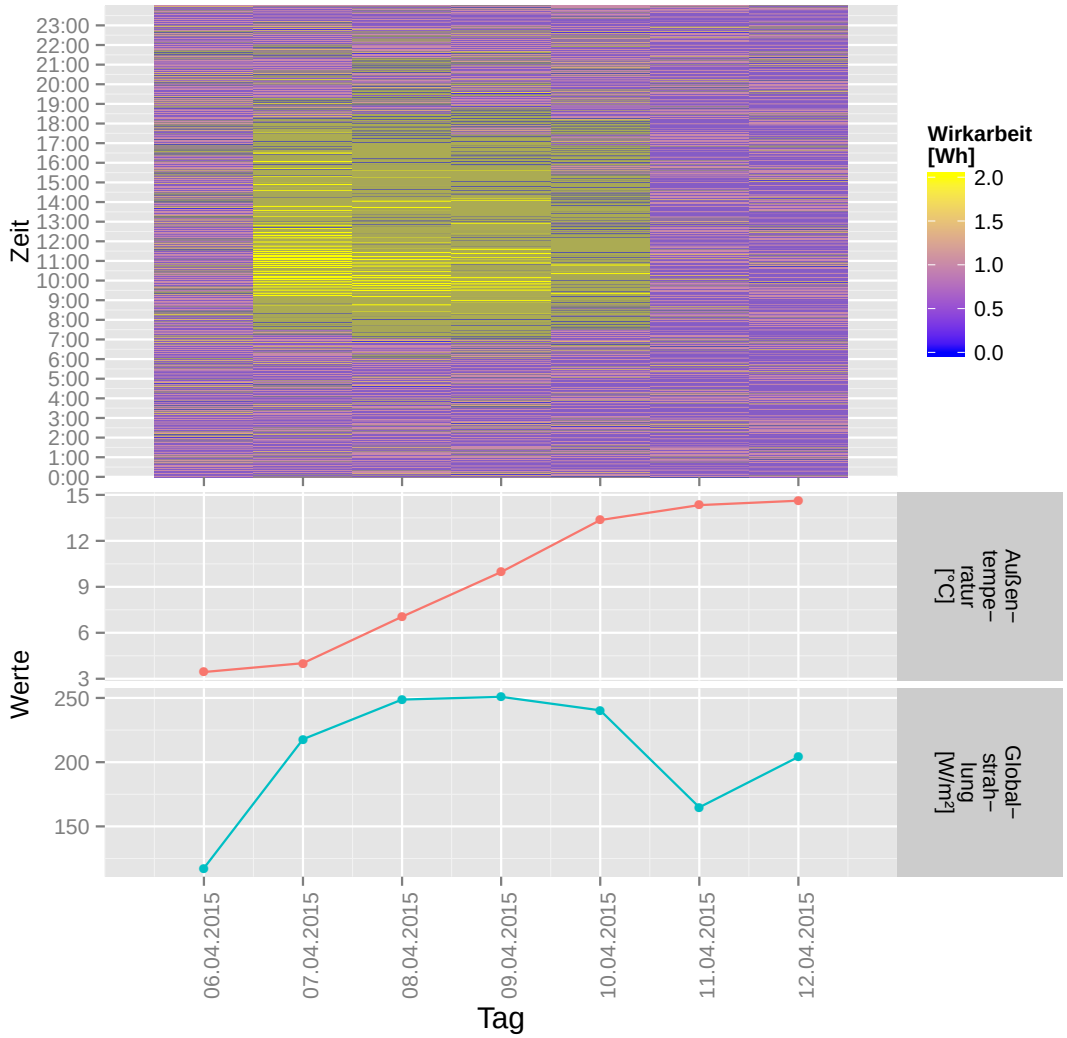


Abbildung 4.22: Tagesverlauf der Wirkarbeit in Wh im Minuten-Takt als Rasterdiagramm mit den Tagesdurchschnitten der Außentemperatur und Globalstrahlung als Kurve. Der zweiwöchige Zeitraum beginnt mit einem Feiertag (Ostermontag).

```
tagesverlauf_db(ch, tabelle = "the46", datum = "datum",
                werte = "diff_x1120_01_021anhv01elz0001zw30",
                aussentemp = "x1110_01_0032hzg00tem0001ma01",
                tabelle2 = "theresienwetter",
                strahlung = "globalstrahlung",
```

```
abschnitte = "Minuten",
between = "'2015-04-06 00:00:00'  and '2015-04-12 23:59:00'",
legende = "Wirkarbeit",
einheit =
"D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_the46.csv")
```

Die Ausführungsdauer dieser Funktion hängt hauptsächlich von der Dauer des Einlesens der Daten aus der Datenbank ab. Somit benötigt das Erstellen obiger Grafiken für den Minutentakt ca. 15-mal so lange wie für den 15-Minuten-Takt (864,99 bzw. 56,78 Sekunden).

Ein merklich anderes Bild liefert Abb. 4.23 mit einem Wärmemengenzähler aus dem Subserver Oettingenstr. 67. Hier ist die tägliche Periodizität kaum zu erkennen, dafür aber die ausgeprägte jährliche Periodizität bzw. der Zusammenhang mit der Außentemperatur. Darüber hinaus wird die Genauigkeit der Darstellung durch die begrenzte Genauigkeit des Zählers eingeschränkt, sodass deutliche Zählersprünge zu erkennen sind. Hierfür wäre möglicherweise eine Darstellung im Stundentakt sinnvoll.

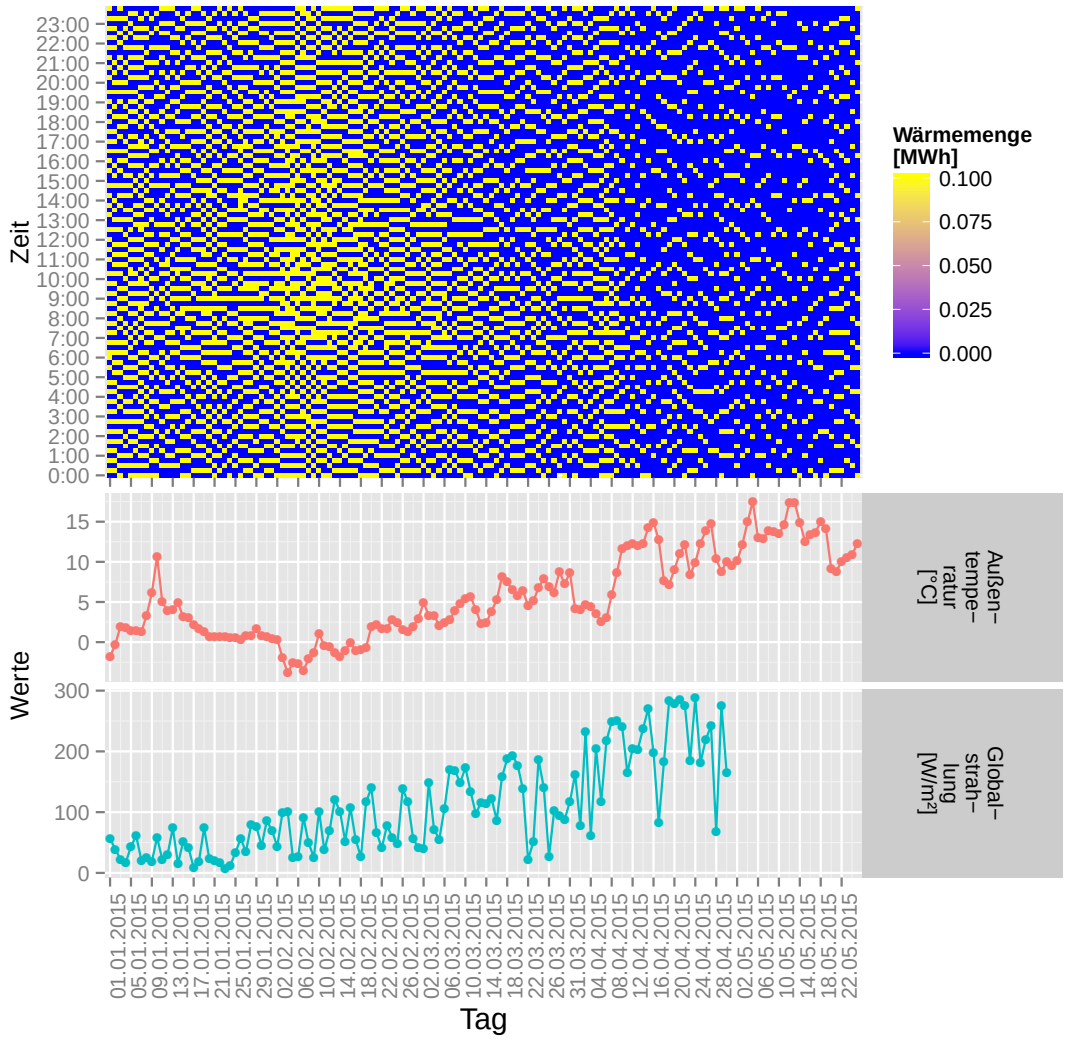


Abbildung 4.23: Tagesverlauf der Wärmemenge in MWh aus dem Subserver Oettingenstr. 67 über die erste Hälfte von 2015.

Vergleich Ferien Oettingenstr.pdf

```
tagesverlauf_db(ch, tabelle = "oet67", datum = "datum",
                werte = "diff_x7070_01_u138hzg03wmz0001zw01",
                aussentemp = "x7070_01_u138hzg03tem0003mp01",
                abschnitte = "15 Minuten",
                tabelle2 = "theresienwetter",
                strahlung = "globalstrahlung",
                between = "'2015-01-01 00:00:00'  and '2015-12-30 00:00:00'",
```

```
legende =
"D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_oet67.csv",
einheit =
"D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_oet67.csv")
```

4.4.8 Zählervergleich (Benchmark)

Die Grafik zeigt den Energiebedarf unterschiedlicher Gebäude(teile) im Vergleich untereinander und im Vergleich unterschiedlicher Zeiträume. Die Bedeutung des Zählervergleichs liegt insbesondere darin, dass damit Daten aus unterschiedlichen Quellen zusammengeführt und gemeinsam ausgewertet werden können. Der Wärmeverlust und damit Wärmeverbrauch steigt mit sinkender Außentemperatur. Um die Verbräuche unterschiedlicher Zeiträume trotz unterschiedlicher Außentemperaturen miteinander vergleichbar zu machen, ist eine Gradtagzahlbereinigung gemäß VDI 3807 integriert. Diese Balkendiagramme eignen sich dazu, den generellen Trend des Energieverbrauchs eines Gebäudes oder der Uni zu erkennen und um Gebäude mit dem größten Optimierungspotential zu identifizieren.

Viele Werte für Zähler, die über den Zählervergleich verglichen werden sollen, liegen nicht direkt vor, sondern müssen mithilfe einer Formel aus anderen Zählern berechnet werden. Die Formel für diese virtuellen Zähler können aus der Gebäudemanagementsoftware exportiert werden, sodass eine CSV-Datei mit Zählern, Quelle (virtuell, Famos oder GLT) und ggf. Formel vorliegt (Testbeispiel siehe 4.1).

Tabelle 4.1: CSV-Datei für das Argument hierarchie (zaehlerebenen_benchmark4.csv)

Ebene	Zähler	Quelle	Formel
1	zähler a	virtuell	ZPL0000937+ZPL000487E
1	zähler b	virtuell	ZPL0000959
2	ZPL0000937	famos	
2	ZPL000487E	famos	
2	ZPL0000959	famos	
1	zähler d	virtuell	diff_x7079_01_t013klt01waz0001zw01+diff_x0407_01_061bhzg03wmz0001zw01-zähler e
2	diff_x0407_01_061bhzg10wmz0001zw01	glt	
2	diff_x0407_01_061bhzg03wmz0001zw01	glt	
2	zähler e	virtuell	diff_x0407_01_061bhzg10wmz0001zw01+diff_x7070_01_u138hzg03wmz0001zw01
3	diff_x7079_01_t013klt01waz0001zw01	glt	
3	diff_x7070_01_u138hzg03wmz0001zw01	glt	

Das einfache Beispiel für einen Zählervergleich mit Gradtagzahlbereinigung (4.24) und ohne Gradtagzahlbereinigung (4.25) zeigt den Vergleich für die (erfundenen) Zähler a, b und d für September (rot) und Oktober (blau). Dabei besteht Zähler a aus der Summe

zweier mit der Anteilsmethode geschätzter Famos-Zähler, Zähler b hingegen nur aus einem Famos-Zähler, der mithilfe des Labels virtuell nur in zähler b umbenannt wird, und Zähler d aus einer Summe von GLT-Zählern und einem weiteren virtuellen Zähler, der wiederum aus einer Summe aus GLT-Zählern besteht. Die Einheiten der unterschiedlichen Zähler werden angeglichen, wobei hier MWh als gemeinsame Einheit gewählt wurde. Für den Kommentar, der zu jedem Zählernamen als zweite Zeile hinzugefügt werden kann, wurde hier die Quelle (Famos oder GLT) gewählt, da in diesem Beispiel alle virtuellen Zähler nur aus ausschließlich Famos- oder ausschließlich GLT-Zählern berechnet werden. Prinzipiell sind hierfür aber alle Kombinationen aus Famoszählern und GLT-Zählern aus allen Subservern möglich.

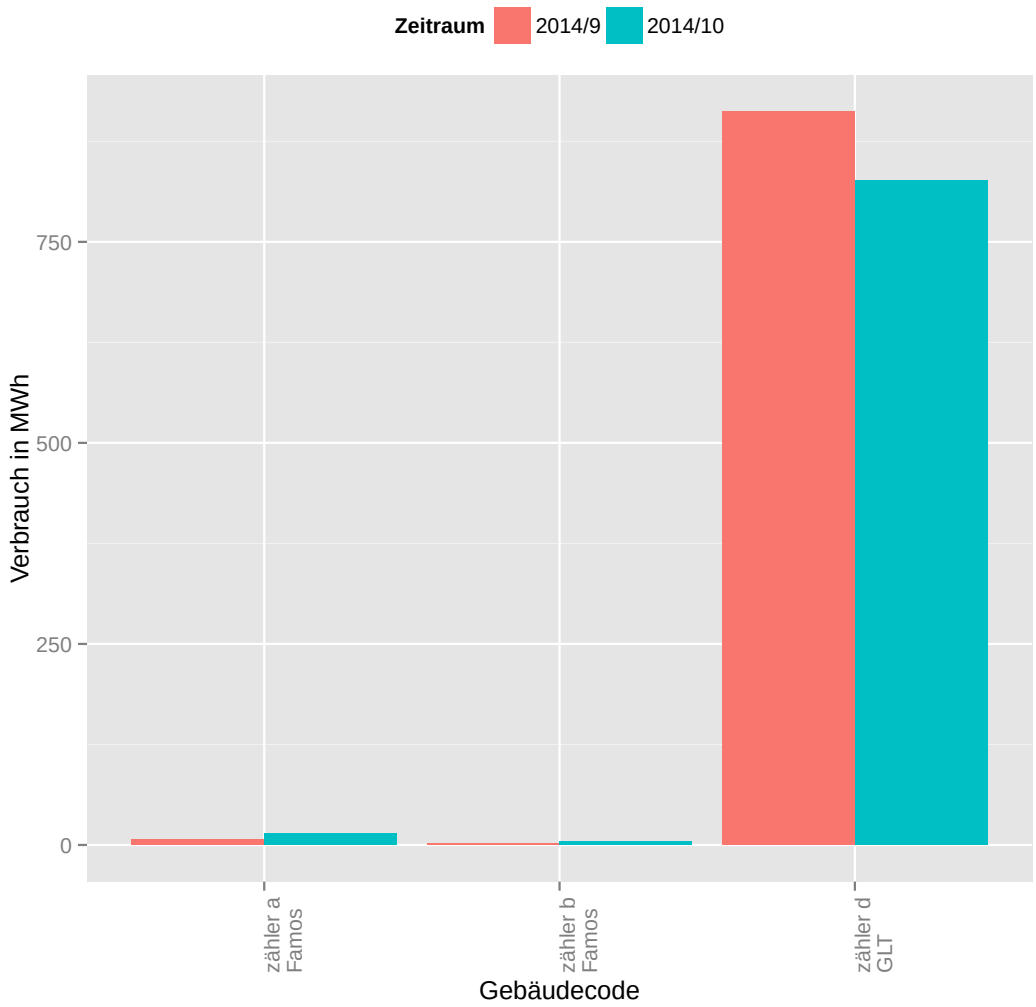


Abbildung 4.24: Zählervergleich mit Famos- und GLT-Daten

```
hierarchie2 <- read.csv2("zaehlerebenen_benchmark4.csv", as.is = TRUE)


zaehlervergleich(c("zähler d", "zähler a", "zähler b"),
                 zaehlerbeschreibung=c("GLT","Famos","Famos"),
                 hierarchie = hierarchie2,
                 con = ch, zeitraum = c("2014/9","2014/10"),
                 spaltenuebersichten =
                      c(glt_hist_leo03 = "spaltenuebersicht_leo03.csv",
                        glt_hist_oet67 = "spaltenuebersicht_oet67.csv",
                        glt_hist_the46 = "spaltenuebersicht_the46.csv"),
                 einheit = "MWh",
                 datei =
                   "Daten/Famos_daten/20150128 Zählerstand Stammgelände Komplett.csv",
                   rueckgabewerte = TRUE, gradtagzahlbereinigung = FALSE)
```

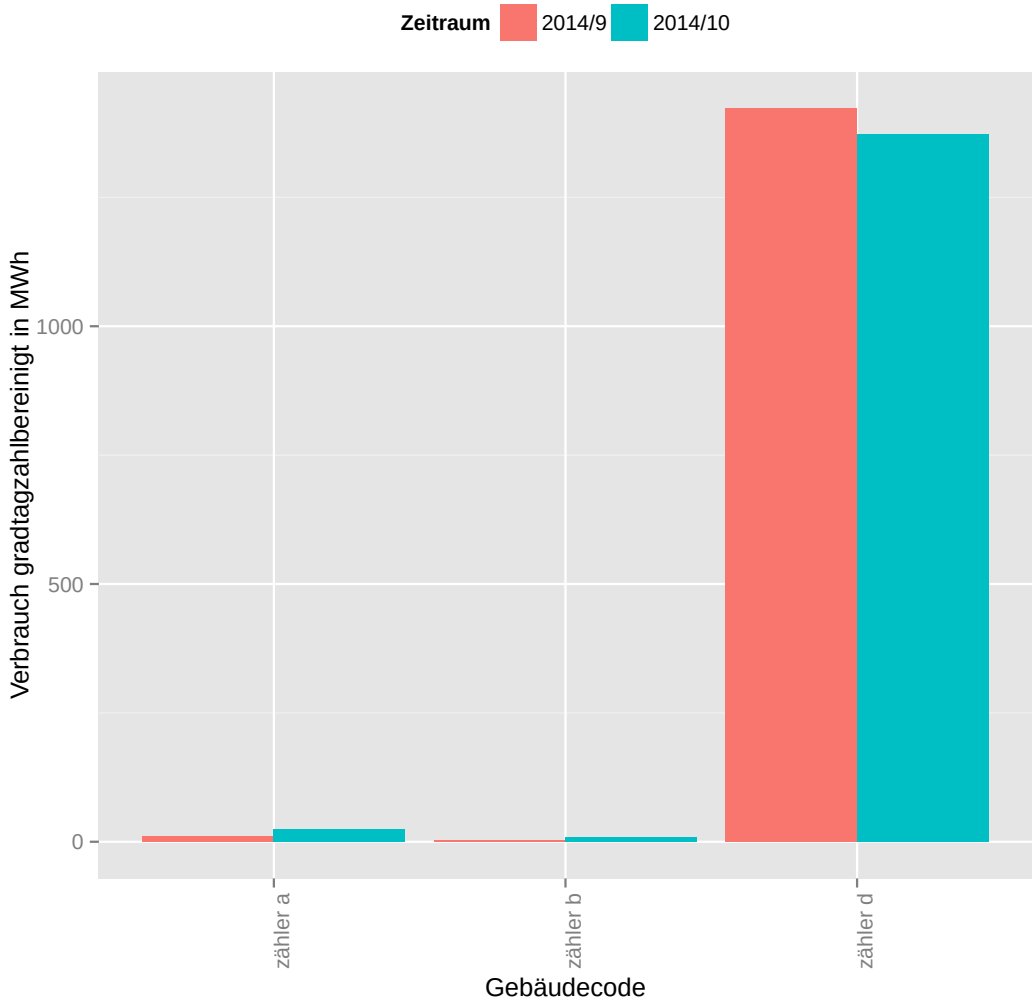


Abbildung 4.25: Zählervergleich gradtagzahlbereinigt.

```
zaehlervergleich(c("zähler a", "zähler b", "zähler d"), hierarchie = hierarchie2,
                 con = ch, zeitraum = c("2014/9","2014/10"),
                 spaltenuebersichten =
                      c(glt_hist_leo03 = "spaltenuebersicht_leo03.csv",
                        glt_hist_oet67 = "spaltenuebersicht_oet67.csv",
                        glt_hist_the46 = "spaltenuebersicht_the46.csv"),
                 einheit = "MWh",
                 datei =
                   "Daten/Famos_daten/20150128 Zählerstand Stammgelände Komplett.csv",
                   rueckgabewerte = TRUE)
```

Da die in der Gebäudemanagementsoftware Famos gespeicherten Daten in unregelmäßigen Abständen von ca. einem Monat oder mehr abgelesen werden und die Gebäudeleittechnik-Messungen im Minutentakt erfolgen und nach Subservern getrennt gespeichert werden, müssen die Daten auch individuell eingelesen und aufbereitet werden.

Schätzung der Famos-Zähler

Die Famos-Zähler werden in unregelmäßigen Abständen alle ein bis zwei Monate manuell abgelesen. Da man aber an Zeiträumen interessiert ist, die i. d. R. nicht dem Ablese-Intervall entsprechen, müssen die tatsächlichen Werte für diese Zeiträume geschätzt werden.

Um die Schätzmethoden zu überprüfen, wurden zwei Zähler der Gebäudeleittechnik verwendet, für die tägliche Werte verfügbar waren und von denen angenommen wurde, dass sie dieselben Eigenschaften haben, wie die manuell abgelesenen Zähler. Einer der Zähler stammt aus dem Subserver Oettingenstraße, der andere aus dem Subserver Schellingstraße. Anhand der Ablesezeitpunkte eines willkürlich ausgewählten Famos-Zählers wurde dann durch das Bilden von Mittelwerten für die Zeiträume, die bei einem Famos-Zähler vorhanden gewesen wären, ein Famos-Zähler simuliert. Da die Ablesezeitpunkte des Famos-Zählers nur bis einschließlich 2014 verfügbar waren, wurden für 2015 die Ablesezeitpunkte von 2012, für die in den GLT-Zählern keine Werte verfügbar waren, um drei Jahre verschoben und als Ablesezeitpunkte für 2015 verwendet. Abb. 4.26 und 4.27 zeigt sowohl die gemessenen Tageswerte (schwarze Punkte), als auch die simulierten Durchschnittswerte, die bei einem Famos-Zähler vorhanden gewesen wären (rote Kurve) und die (wahren) Durschnittswerte für die gewünschten Zeiträume (blaue Kurve), die es zu schätzen gilt.

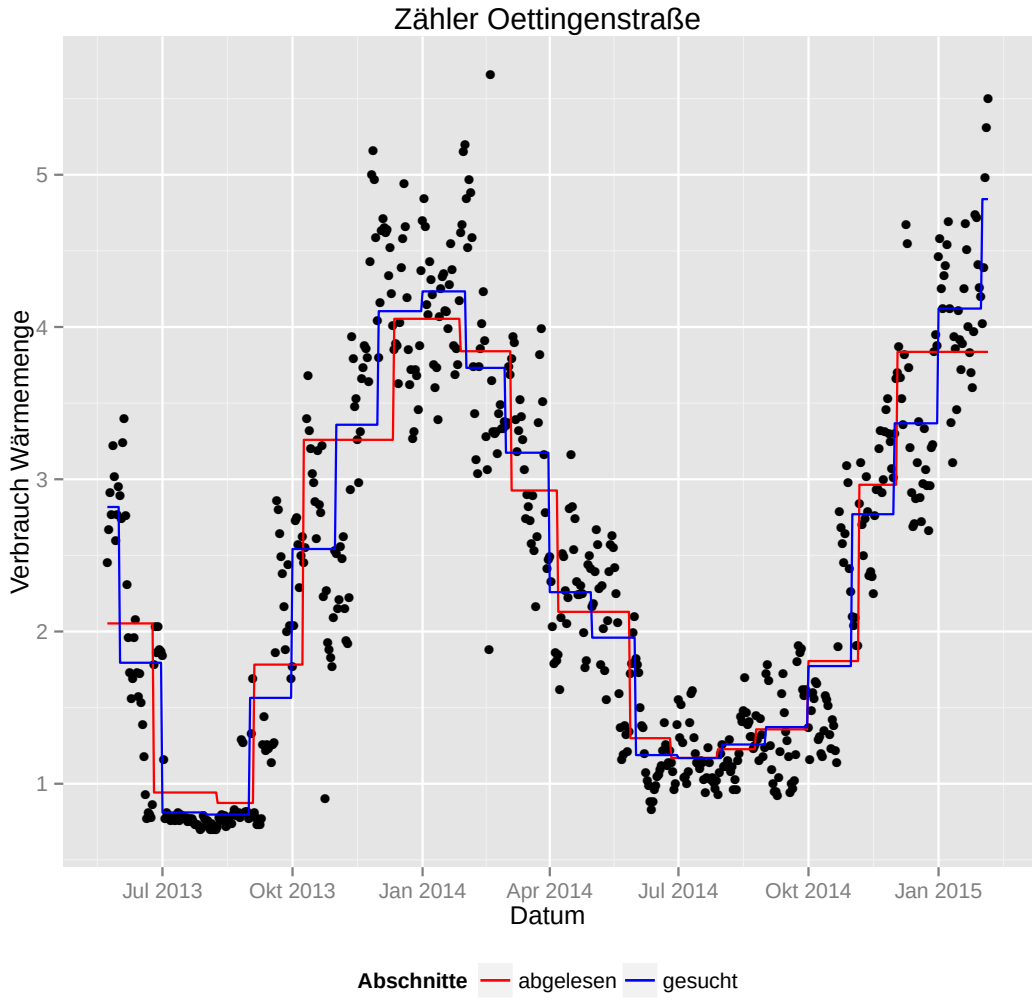


Abbildung 4.26: Die wahren Verbräuche (schwarze Punkte), die durchschnittlichen Tagesverbräuche für die Ablesezeiträume (rote Kurve) und die gesuchten Zeiträume (blaue Kurve) für einen Zähler des Subservers Oettingenstraße.

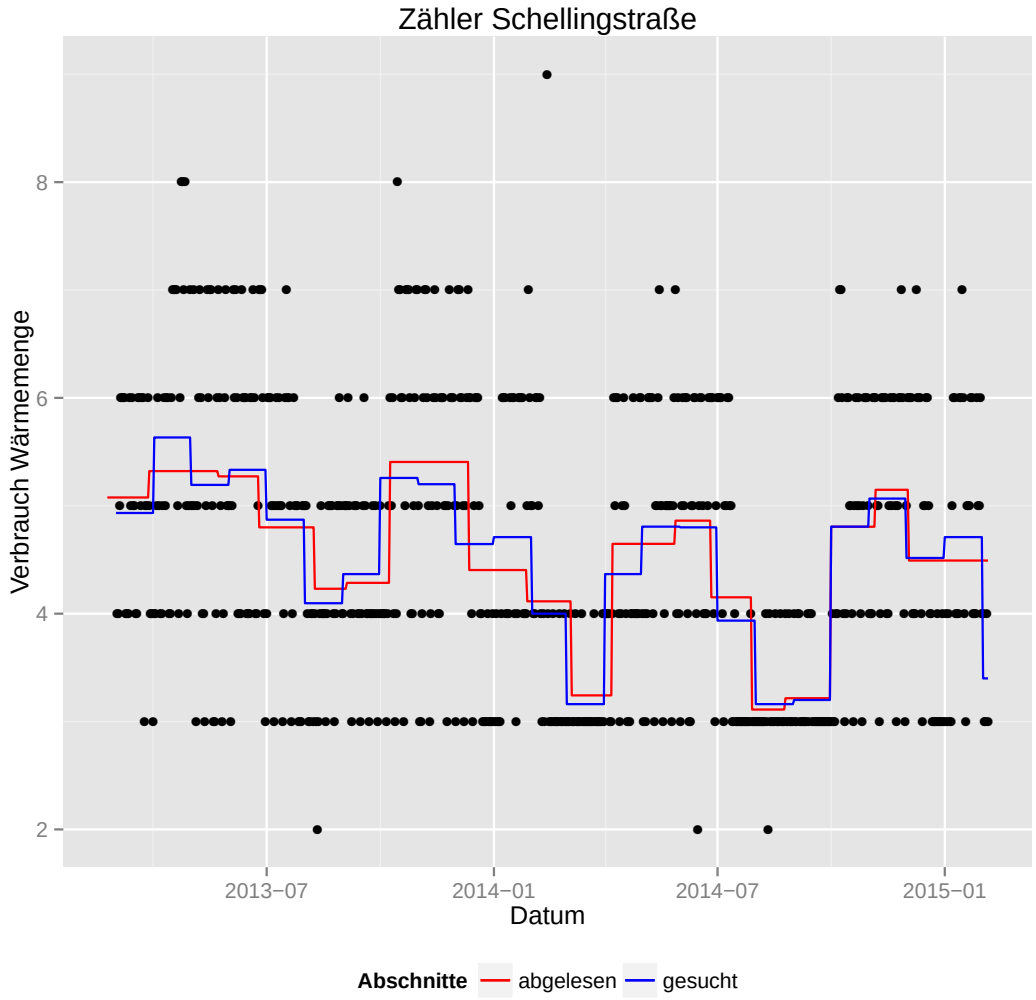


Abbildung 4.27: Die wahren Verbräuche (schwarze Punkte), die durchschnittlichen Tagesverbräuche für die Ablesezeiträume (rote Kurve) und die gesuchten Zeiträume (blaue Kurve) für einen Zähler des Subservers Schellingstraße.

Für die Schätzung werden die bekannten (abgelesenen) Intervalle, welche sich mit dem gewünschten Intervall überschneiden anteilig verwendet. D.h. wurde ein Zähler zum 20.1. (Zählerstand a: 1), 10.2. (Zählerstand b: 5) und 30.3. (Zählerstand c: 10) abgelesen und möchte man den Wert für Februar wissen, so wird die Differenz zwischen Zählerstand b und Zählerstand a durch die Anzahl der Tage dazwischen geteilt und mit mit der Anzahl der Tage, die sich im Februar befinden multipliziert. Das selbe wird mit dem zweiten sich mit dem gewünschten Zeitraum überschneidenden Intervall gemacht: $\frac{(5-1)\cdot 10}{21} + \frac{(10-5)\cdot 18}{48}$.

Die Allgemeine Formel lautet:

$$\hat{y}_{\text{gesucht}} = \sum_i \left(\frac{y_i}{t_i} \cdot \left(t_i \cap \hat{t}_{\text{gesucht}} \right) \right)$$

wobei
$\hat{y}_{\text{gesucht}}$ = geschätzter Verbrauch für den gesuchten Zeitraum
$\hat{t}_{\text{gesucht}}$ = Dauer des gesuchten Zeitraumes
y_i = Verbrauch eines Ableseintervalls, das sich mit dem gesuchten Intervall überschneidet
t_i = Dauer eines Ableseintervalls, das sich mit dem gesuchten Intervall überschneidet

Eine andere Methode versucht die Informationen der vergangenen Zeiträume zu verwenden, um die Aufteilung des Verbrauchs auf die einzelnen Tage innerhalb der gemessenen sich mit den gewünschten Zeiträumen überschneidenden Intervalle zu bestimmen. Hierfür wird ein additives Modell mit allen vorhandenen abgelesenen Zeiträumen geschätzt und damit sowohl die abgelesenen sich mit dem gewünschten Zeitraum überschneidenden Verbräuche, als auch der Verbrauch für den gewünschten Zeitraum vorhergesagt. Dabei sind die einzelnen Tage die Beobachtungen und die Durchschnittswerte für die Tage innerhalb der abgelesenen Tage die Zielgröße, wobei davon ausgegangen wird, dass die geschätzten Werte aufgrund der Glättung korrekter als die zur Schätzung verwendeten Werte sind. Die Summe der abgelesenen Verbräuche, die für den vorhandene Zeitraum der korrekte Wert ist, wird mit einem Faktor, der sich durch die Division des vorhergesagten gewünschten durch die vorhergesagten vorhandenen Zeitraumes errechnet, multipliziert, was schließlich die Schätzung ergibt. Es gehen also sowohl die aus allen vorhandenen Daten geschätzten Werte innerhalb der betrachteten Zeiträume, als auch der bekannte abgelesene Wert der sich mit dem gewünschten Zeitraum überschneidenden Ableseintervalle in die Schätzung ein.

Die Modellformel hierfür lautet:

$$\hat{y}_{modell} = \beta_0 + \beta_{Wochenende} x_{Wochenende} + \beta_{Semesterferien} x_{Semesterferien}$$
$$+ f(Tag\ des\ Jahres) + f(Außentemperatur)$$

wobei der Tag des Jahres zyklisch eingeht, also der 365. Tag nahtlos in den 1. Tag des Jahres übergeht. Genauer gesagt wurden penalisierte zyklische kubische Regressionssplines

verwendet, deren Enden bis zur zweiten Ableitung übereinstimmen (siehe `https://cran.r-project.org/web/packages/mgcv/mgcv.pdf` S.206). Etwas korrekter wäre es, die Endknoten auf auf 0,5 und 366,5 festzulegen (von einem Schaltjahr ausgegangen, da dies die maximale Anzahl an Tagen im Jahr besitzt), da nicht der erste Tag gleich dem letzten sein soll, sondern der gemeinsame Wert eigentlich zwischen diesen beiden Tagen liegen müsste. Eine Anleitung, dies mit dem mgcv-Paket umzusetzen findet sich unter `http://web.mit.edu/~r/r_v3.1.1/lib/R/library/mgcv/html/smooth.construct.cr.smooth.spec.html`.

Mit diesem Modell wird dann die Vorhersage für folgende Formel gemacht:

$$\hat{y}_{gesucht} = y_{vorhanden} \cdot \frac{\hat{y}_{modell,gesucht}}{\hat{y}_{modell,vorhanden}}$$

$\hat{y}_{gesucht}$ = tatsächliche Schätzung für den gewünschten Zeitraum,
$y_{vorhanden}$ = abgelesener Wert für Überschneidungszeiträume,
$\hat{y}_{modell,gesucht}$ = durch das Additive Modell vorhergesagter gesuchter Zeitraum,
$\hat{y}_{modell,vorhanden}$ = durch das Additive Modell vorhergesagte abgelesene Überschneidungszeiträume

Oder als R-Code (nur Schätzung des additiven Modells):

```
mgcv::gam(diff_w ~ Wochenende + as.factor(Semesterferien) +
            s(tag_jahres, bs="cc", k=20)+
            s(aussentemperatur, k=20),
          data=sch_w6, family=gaussian())
```

Der Verlauf der Verbräuche als gemessene Werte und deren geglättete Variante (Loess-Kurve) aufgeteilt nach Werktage und Wochenende ist in Abb. 4.28 und Abb. 4.29 zu sehen. Die Semesterferien sind als blaue Hintergründe eingezeichnet. Es ist auf den Grafiken zu erkennen, dass die Loess-Kurven für die Werktage nicht parallel zu denen für die Wochenenden verlaufen, also das Modell wahrscheinlich um Interaktionen erweitert werden sollte. Weiterhin ist zu sehen, dass der Zähler aus der Oettingenstraße (Abb. 4.28) zwar eine sehr ausgeprägte jährliche Periodizität aufweist, aber der Zusammenhang zwischen Verbrauch und Ferien kaum zu bestehen scheint. Dieser ist für den Zähler aus der Schellingstraße dafür klar zu erkennen (Abb. 4.29).

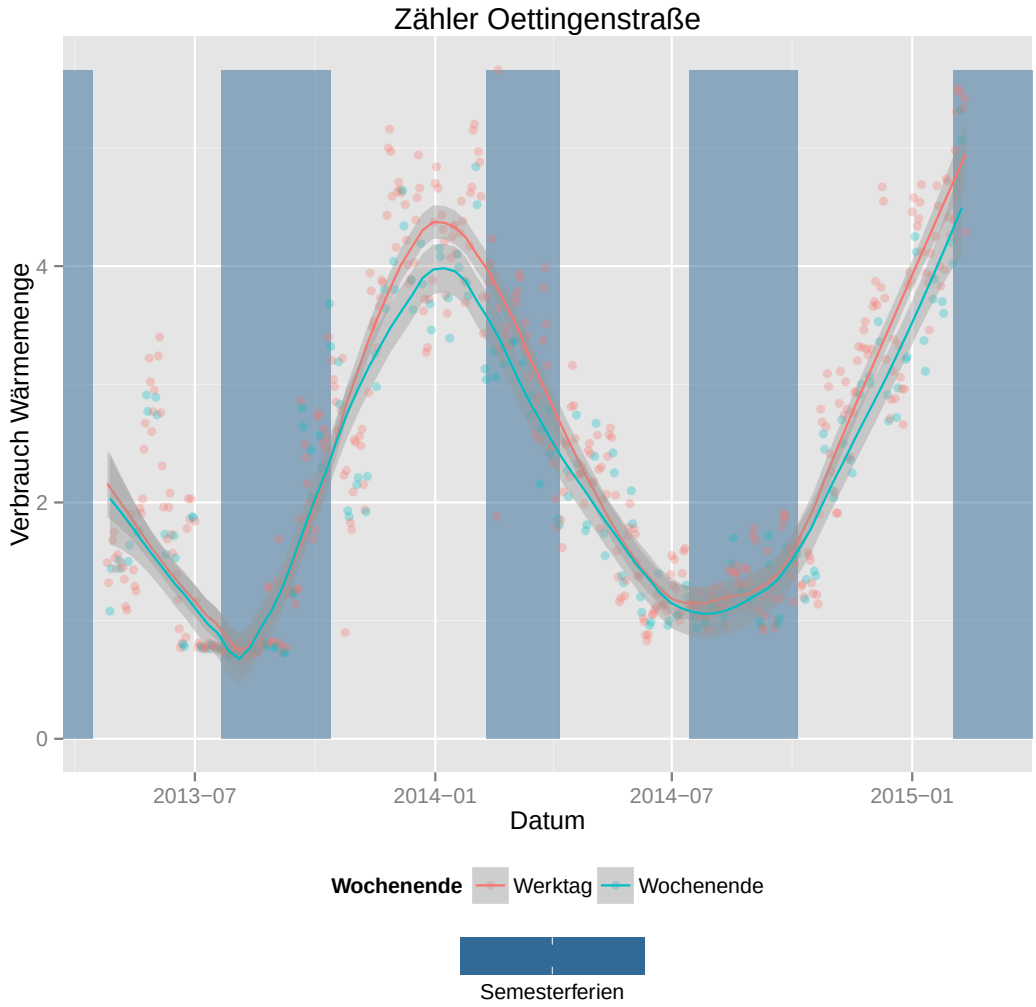


Abbildung 4.28: Tägliche Werte eines Zählers des Subservers Oettingenstraße als Punkte und geglättete Kurve (Loess-Kurve) für Wochenenden (blau) und Werktage (rot). Die vorlesungsfreien Zeiten sind durch die blauen Hintergründe gekennzeichnet.

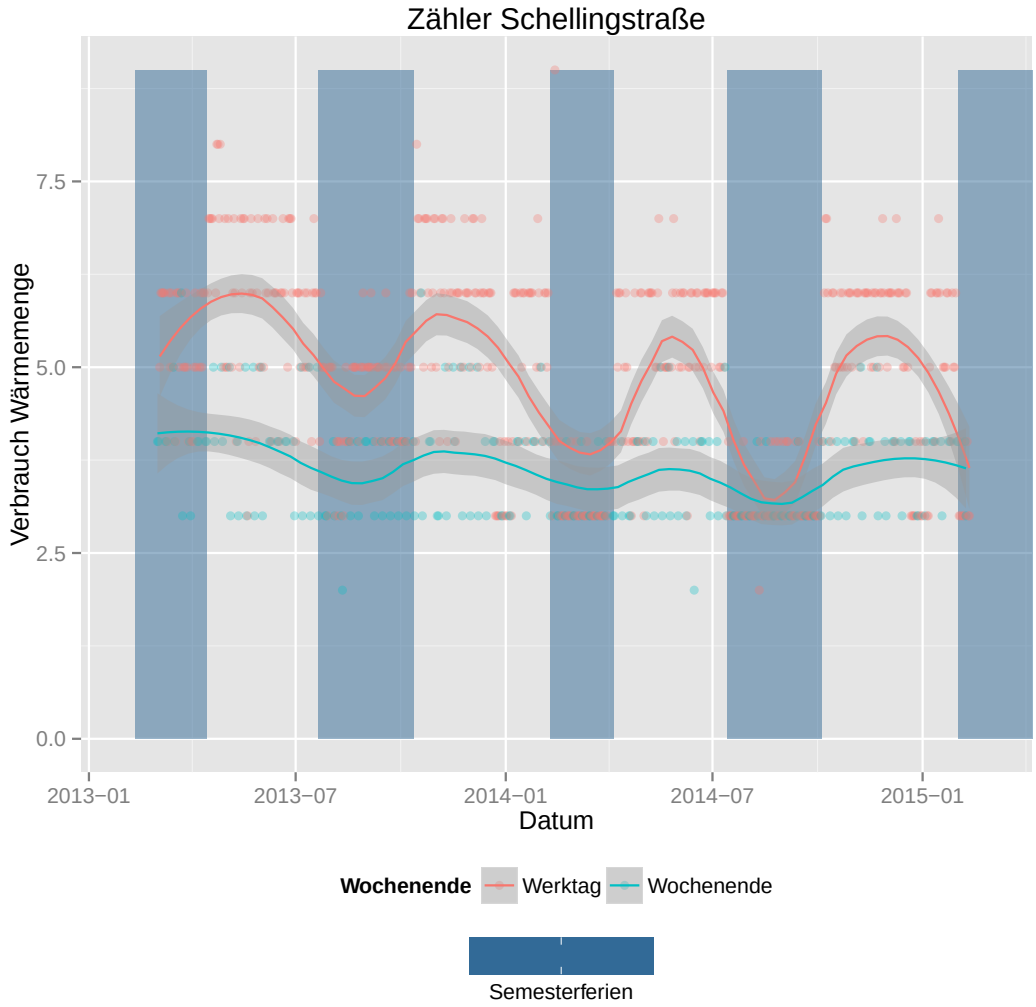


Abbildung 4.29: Tägliche Werte eines Zählers des Subservers Schellingstraße als Punkte und geglättete Kurve (Loess-Kurve) für Wochenenden (blau) und Werktage (rot). Die vorlesungsfreien Zeiten sind durch die blauen Hintergründe gekennzeichnet.

Die Ergebnisse der beiden Schätzmethoden können in Abb. 4.30 und Abb. 4.31 grafisch verglichen werden.

Die durchschnittliche absolute Abweichung der geschätzten von den wahren Werten $(\frac{1}{n}\sum_i|\frac{\hat{y}_{gesucht,i}-y_{gesucht,i}}{y_{gesucht,i}}|)$ beträgt für den Zähler der Schellingstraße für die Anteilsmethode 0,0417 und für die GAM-Methode 0,03290, also eine durchschnittliche Abweichung von 4,17 bzw. 3,29 Prozent. Bei der mittleren quadratischen Abweichung ergibt sich für die Anteilsmethode 0,0059 und für die GAM-Methode 0,0028. Mittelt man die Schätzung für

die beiden Methoden, absolute Abweichung 0,0343 und quadratische Abweichung 0,0038. Das nur wenige Tage dauernde erste und letzte gesuchte Intervall wurden dabei in diese Berechnungen nicht miteinbezogen.

Für den Zähler der Oettingenstraße ergibt sich bei den absoluten Abweichungen 0,0545 (Anteile), 0,0362 (GAM) und 0,0367 (gemittelt) und für die quadratischen Abweichungen 0,0052 (Anteile), 0,0021 (GAM) und 0,0023 (gemittelt).

Somit ergibt die GAM-Methode eine bessere Schätzung als die Anteils-Methode oder die gemittelten Schätzungen.

Andere Modellgütemaße etc. für die Additiven Modelle dürften hier schwierig zu interpretieren sein, da die Zielgröße des Modell bereits aus Durchschnittswerten für die einzelnen Ablesezeiträume besteht und die Vorhersage ausschließlich für Zeiträume erfolgt, die auch in die Modellschätzung eingehen (also auch keine Verallgemeinerung für eine Grundgesamtheit, die über die vorhandenen Beobachtungen hinausgeht, angestrebt wird). Weiter erfolgt die Schätzung der tatsächlichen Werte nicht mit Hilfe des GAMS, sondern nur die Aufteilung der Verbräuche auf die einzelnen Tage innerhalb der abgelesenen Zeiträume.

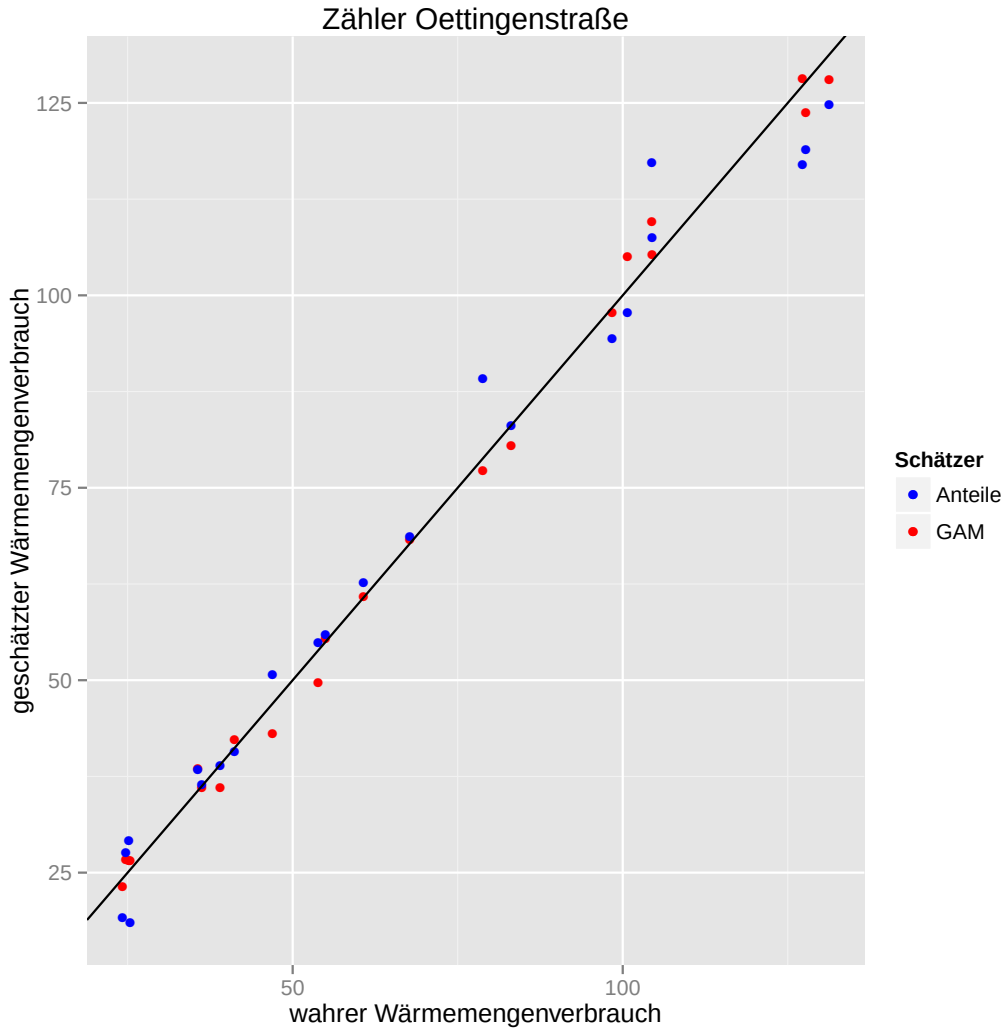


Abbildung 4.30: Die mit der Anteilsmethode (blau) und dem Additiven Modell (rot) geschätzten Verbräuche für die einzelnen Kalendermonate in Abhängigkeit von den wahren Verbräuchen für den Zähler der Oettingenstraße. Bei einer perfekten Schätzung wären die geschätzten Werte gleich den wahren Werten und würden sich alle auf der schwarzen Linie befinden

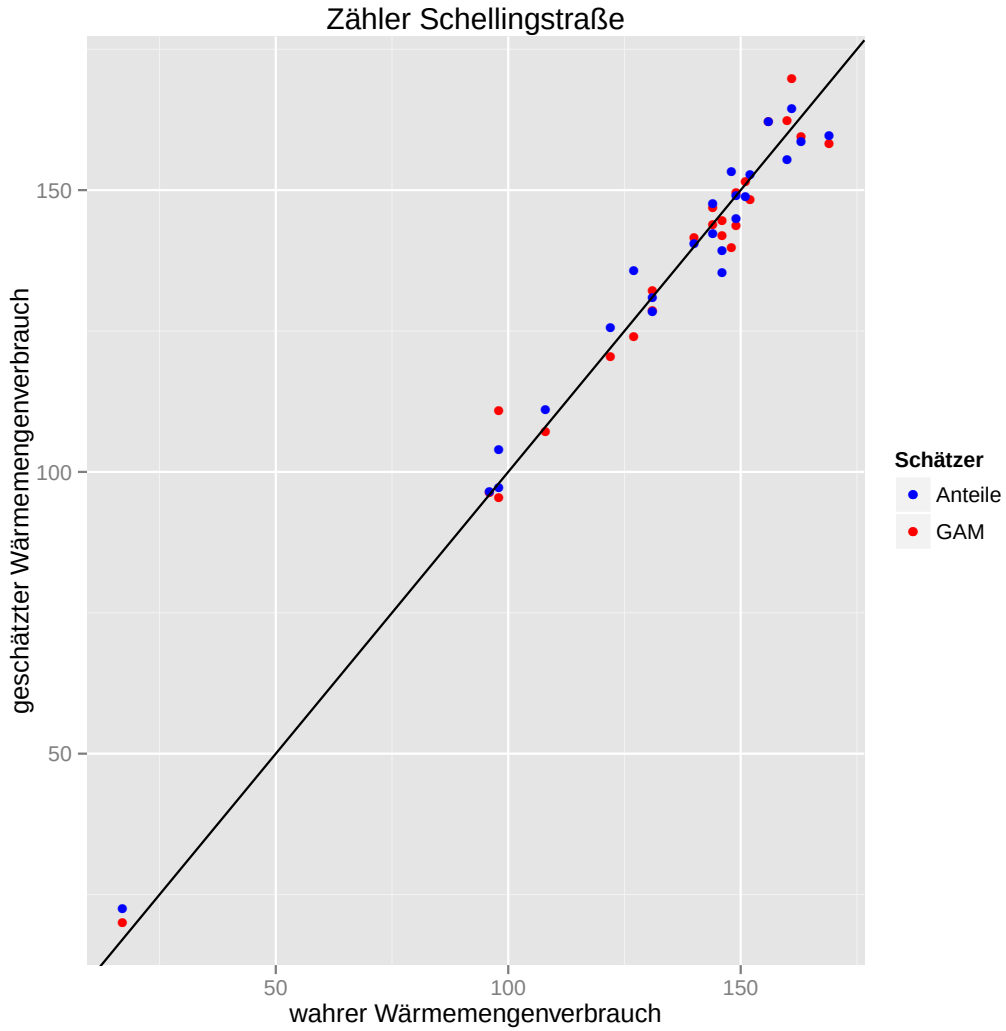


Abbildung 4.31: Die mit der Anteilsmethode (blau) und dem Additiven Modell (rot) geschätzten Verbräuche für die einzelnen Kalendermonate in Abhängigkeit von den wahren Verbräuchen für den Zähler der Schellingstraße. Bei einer perfekten Schätzung wären die geschätzten Werte gleich den wahren Werten und würden sich alle auf der schwarzen Linie befinden

EnMoLMU beinhaltet in der Version 1.0 eine vollständige Implementation der Anteilsmethode, da diese leichter interpretierbar und somit vermittelbar ist. Eine Implementierung der GAM-Methode ist vorhanden, muss aber noch korrigiert und vervollständigt werden. Insbesondere wären hier eine systematische Suche nach dem bestmöglichen Modell (weitere Einflussgrößen möglich? Verbesserte Schätzung durch Interaktionen? Andere Schätzalgorithmen?) und eine automatische Auswahl der Variablen und Knotenanzahl anhand der Anzahl der vorhandenen Beobachtungen und vorhandenen Einflussgrößen

bzw. automatische Verwendung der Anteilsmethode, falls zu wenige Daten vorhanden sind, zu nennen. Überdies ist eine spezielle Datenbereinigung nötig, da z. B. auch sehr lange Ableseintervalle (ca. 1 Jahr) in den Famos-Daten vorkommen, die die Schätzung der jährlichen Periodizität verschlechtern. Auch fehlende Werte können sich als sehr problematisch erweisen, falls sie innerhalb des Zeitraumes liegen, für den die Vorhersagen gemacht werden soll (Überschneidungszeiträume und gesuchter Zeitraum), einen bestimmten Zusammenhang verdecken, oder einen längeren Abschnitt umfassen. Auch hierfür müsste mit einer automatischen Entfernung der entsprechenden Variable oder einem Imputationsverfahren eine Lösung eingebaut werden.

5 Verwendete statistische Methoden

Da in dieser Arbeit erst eine Verwendungsfähige Basis der Software geschaffen werden musste, wurden die Möglichkeiten, die statistische Methoden für das Energiemonitoring bieten, nicht annähernd ausgeschöpft. In diesem Kapitel sollen die Methoden, die bereits in diesem Stadium in EnMoLMU eingeflossen sind, soweit beschrieben werden, dass der Anwender eine Vorstellung davon hat, wie die Schätzungen zustande kommen und wie die Ergebnisse zu interpretieren sind. Auf – insbesondere mathematische – Details wurde aus Gründen der Lesbarkeit und Übersichtlichkeit allerdings verzichtet.

5.1 Normalverteilung

Eine Zufallsvariable X ist eine Variable, deren Werte die Ergebnisse eines Zufallsvorganges sind. Der konkrete Wert $x \in \mathbb{R}$, der für X bei einer Durchführung dieses Zufallsvorganges herauskommt, heißt Realisierung. Die Wahrscheinlichkeitsverteilung einer stetigen Zufallsvariable kann mithilfe einer Dichtekurve beschrieben werden[24]. Dabei ergibt die Fläche unter einem bestimmten Abschnitt der möglichen Werte von X der Dichtekurve die Wahrscheinlichkeit, dass das Ergebnis des Zufallsvorganges in diesen Abschnitt fällt[24]. Die von einer Dichtekurve überdeckte Gesamtfläche ist immer gleich 1. Eine besonders wichtige Klasse dieser Dichtekurven bilden Normalverteilungen oder Gauß-Verteilungen. Sie zeichnen sich dadurch aus, dass sie symmetrisch, unimodal und glockenförmig sind. [23]

"Die Normalverteilung ist die bekannteste und wichtigste Verteilung. Zwei wesentliche Gründe dafür sind: In vielen Anwendungen läßt sich die empirische Verteilung von Daten, die zu einem stetigen oder quasi-stetigen, d.h. feinabgestuften diskreten, Merkmal X erhoben werden, durch eine Normalverteilung ausreichend gut approximieren,

zumindest dann wenn die Originaldaten bzw. das ursprüngliche Merkmal geeignet transformiert wurden. [...] Die glockenförmige Gestalt dieser Dichte ist insbesondere dann ein gutes Modell für die Verteilung einer Variable X, wenn diese durch das Zusammenwirken einer größeren Zahl von zufälligen Einflüssen entsteht, etwa bei Messfehlern, bei Abweichungen von einem Soll- oder Durchschnittswert bei der Produktion von Geräten, bei physikalischen Größen wie Gewicht, Länge, Volumen, bei Punktezahlen in Tests usw. Die theoretische Rechtfertigung für dieses empirisch beobachtbare Phänomen liefert der zentrale Grenzwertsatz[...]. Für die induktive Statistik noch entscheidender ist, dass sich viele andere Verteilungen, insbesondere solche, die bei Schätz- und Testprozeduren mit größerem Stichprobenumfang auftreten, durch die Normalverteilung gut approximieren lassen. Auch hierfür liefert der zentrale Grenzwertsatz die theoretische Rechtfertigung."[23]

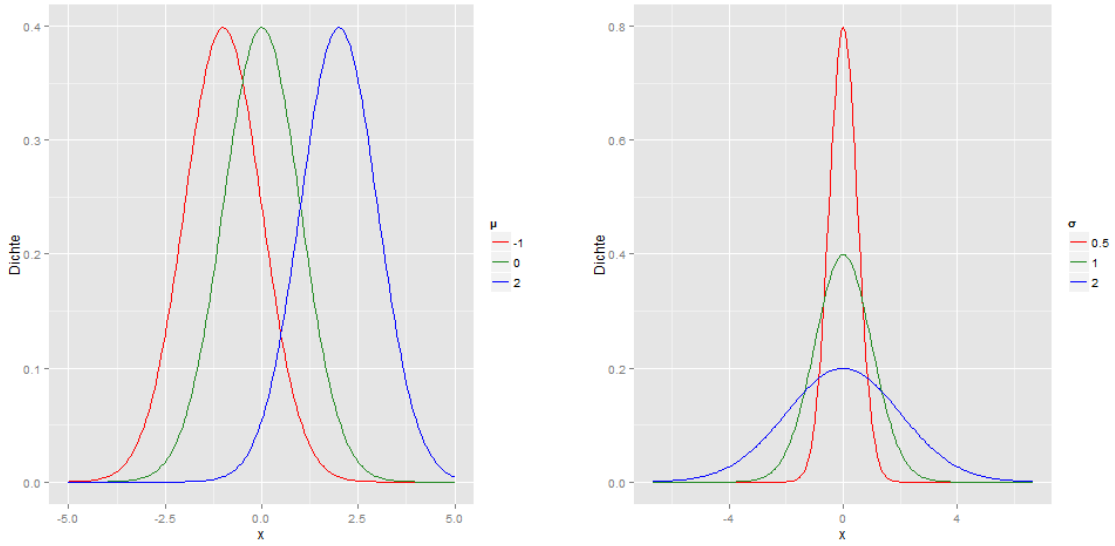
Normalverteilungen sind durch eine spezielle Formel für die Dichtekurven definiert:

Die Dichte ist für jedes $x \in \mathbb{R}$ durch

$$f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

definiert.[23]

Somit kann die Dichte einer normalverteilten Zufallsvariable durch die zwei Parameter Mittelwert $\mu \in \mathbb{R}$ und Standardabweichung $\sigma > 0$ eindeutig festgelegt werden. Abb. 5.1 zeigt, wie die beiden Parameter die Dichtekurve beeinflussen. Diese können z. B. mithilfe des arithmetischen Mittels und der empirischen Standardabweichung $\tilde{s}^2 = \sqrt{\frac{1}{n}\sum_1^n (x_i - \bar{x})}$ (auch mit Division durch $n-1$ anstatt durch n gebräuchlich) aus der Stichprobe geschätzt werden.[23]



(a) Unterschiedliche Werte für μ. **(b)** Unterschiedliche Werte für σ.

Abbildung 5.1: Dichtekurve in Abhängigkeit von den Parametern μ und σ.

Abb. 5.2 illustriert den Zusammenhang zwischen der Fläche unter der Dichtekurve und der Wahrscheinlichkeit, dass eine Realisation x in ein bestimmtes Intervall fällt.

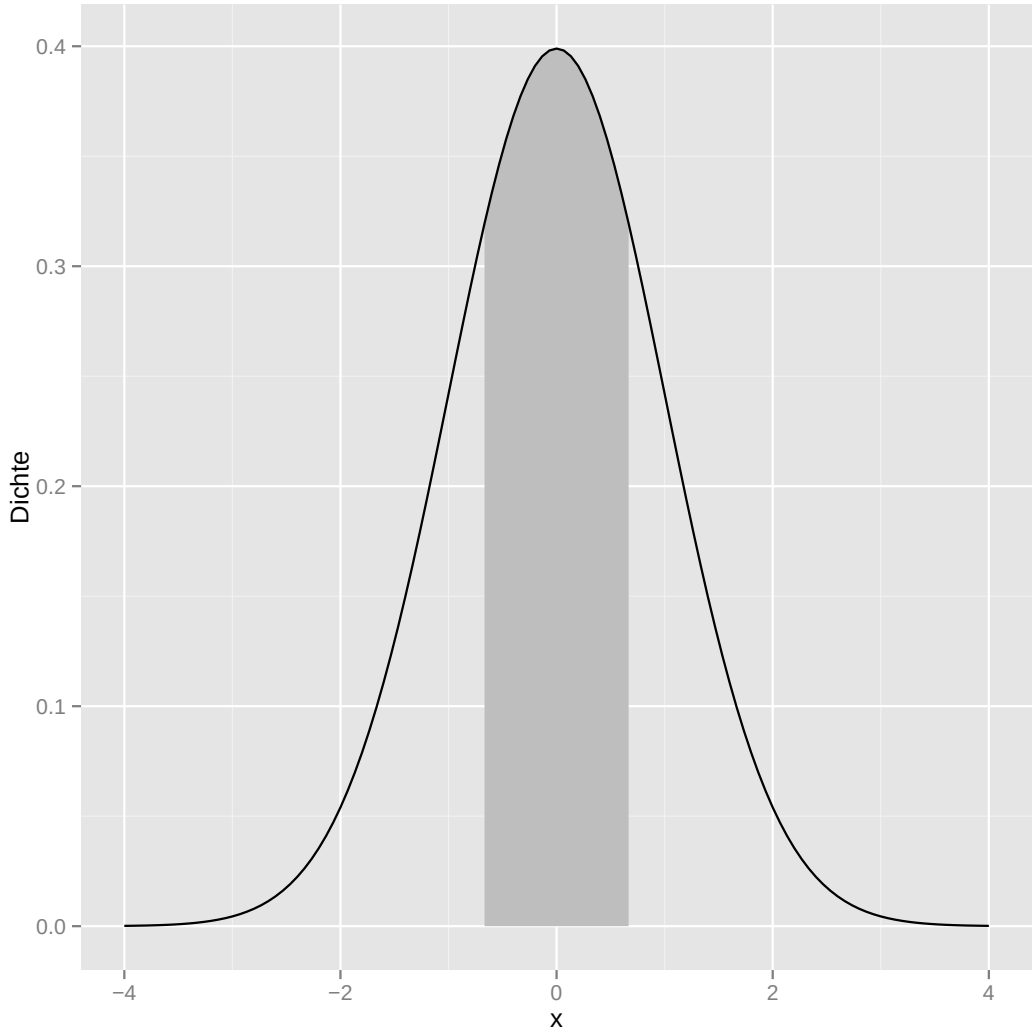


Abbildung 5.2: Die Fläche des grauen Bereichs ist 0,5. Dies bedeutet, dass die Wahrscheinlichkeit bei 0,5 liegt, dass der Wert des Zufallsexperiments in diesen Abschnitt der x-Achse fällt, falls die Zufallsvariable normalverteilt mit $\mu = 0$ und $\sigma = 1$ ist.

5.2 Lineares Regressionsmodell

Die in den Wirtschafts-, Sozial- und Lebenswissenschaften am häufigsten eingesetzte statistische Methodik zur Analyse empirischer Fragestellungen dürfte die Regression darstellen [23]. Mithilfe der Regression wird der Zusammenhang zwischen einer oder mehreren Einflussvariablen auf eine Zielvariable untersucht, wobei davon ausgegangen wird, dass der Zusammenhang zwischen den Variablen nicht exakt gilt, sondern durch zufällige Einflüsse

überlagert ist [23]. D.h. der tatsächliche Wert der Zielvariable y beinhaltet immer eine zufällige Abweichung vom deterministischen Erwartungswert.

5.2.1 Einfaches lineares Modell

Die einfachste Regressionsvariante ist das einfache Lineare Regressionsmodell. Hierbei wird davon ausgegangen, dass der Zusammenhang im Falle von einer einzigen Einflussvariable mit der Formel

$$\underbrace{y_i}_{\text{Zielvariable}} = \underbrace{\beta_0 + \beta_1 x_i}_{\text{systematischer Einfluss}} + \underbrace{\epsilon_i}_{\text{Störgröße}}$$

mit
$\beta_0 =$ Intercept
$\beta_1 =$ Koeffizient zur Einflussvariable oder erklärenden Variable x
$x_i =$ i. Beobachtung der Einflussgröße

dargestellt werden kann. D.h. der systematische Einfluss $\beta_0 + \beta_1 x$ ist linear und besteht aus einem Wert, der den Erwartungswert für die Zielvariable angibt, falls die Einflussvariable 0 ist und einem Wert, um den sich der Erwartungswert erhöht, falls sich die Einflussvariable um 1 erhöht. Der Erwartungswert $E(y|x)$ ist der Wert der Zielgröße y ohne die zufällige Abweichung.

Betrachtet man das Modell am Beispiel der Tageswärmemengenverbräuche (diff_x7070_01_u138hzg03wmz0001zw01) und der durchschnittlichen Außentemperatur (x7070_01_u138hzg03tem0003mp01) der Oettingenstraße 67 für Außentemperaturen unter 15°C, so ergeben sich die Koeffizienten $\hat{\beta}_0 = 4,50$ und $\hat{\beta}_1 = -0,22$ und damit die Regressionsgerade für $\hat{y|x_1}$ aus Abb 5.3. Dabei sind $\hat{y|x_1}$, $\hat{\beta}_0$, $\hat{\beta}_1$ die mithilfe des Modells geschätzten Werte für $E(y|x_1)$, β_0 und β_1. Somit liegt der geschätzte Erwartungswert für den Wärmemengenverbrauch bei einer Außentemperatur von 0°C bei 4,50 und nimmt mit jedem Grad, das die Außentemperatur höher liegt um 0,22 ab. Für eine Außentemperatur von 10°C liegt er damit bei $4,50 + 10 \cdot (-0,22) = 2,30$.

Da mithilfe des Modells der Zusammenhang aus der Stichprobe für die Grundgesamtheit geschätzt werden soll, gilt der Zusammenhang auch für die für die Schätzung nicht

verwendeten Beobachtungen. Die Grundgesamtheit besteht in diesem Fall aus allen Messungen mit den verwendeten Zählern, solange sich keine anderen Faktoren, die den Verbrauch beeinflussen, ändern.

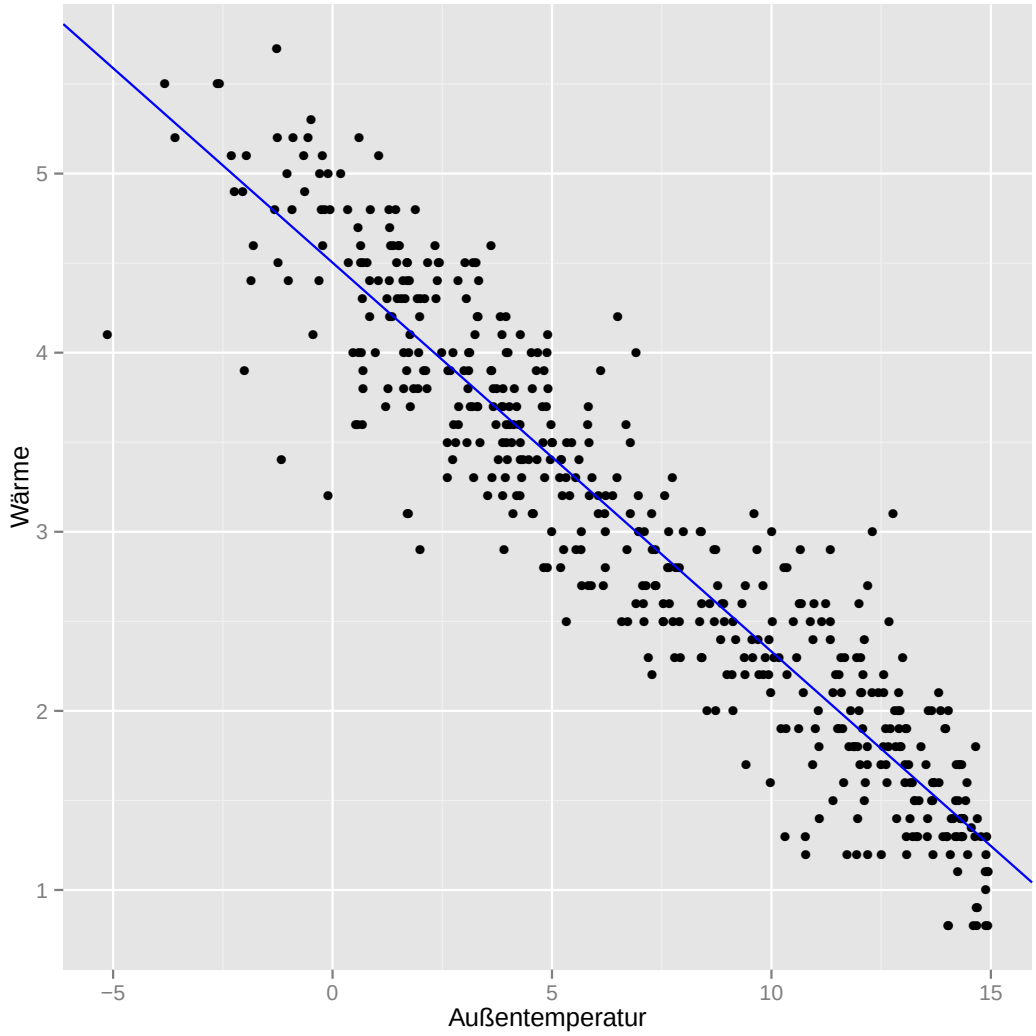


Abbildung 5.3: Wärmemengenverbrauch in Abhängigkeit von der Außentemperatur für Temperaturen unter 15°C mit Regressionsgerade eines Linearen Modells.

5.2.2 Klassisches lineares Modell

Will man mehrere Einflussvariablen zugleich betrachten, so lässt sich das Modell zum klassischen linearen Modell mit

$$y = X\beta + \epsilon$$

erweitern. Dabei ist $\boldsymbol{y}$ der Vektor mit den einzelnen Beobachtungen der Zielvariablen, $\boldsymbol{X}$ die Designmatrix mit

$$\boldsymbol{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1k} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \dots & x_{nk} \end{pmatrix}$$

ist, also mit einer Einsen-Spalte und je einer Spalte für jede Einflussvariable und je einer Zeile für jede Beobachtung. $\boldsymbol{\beta}$ ist der Vektor mit Parametern des Modells und $\boldsymbol{\epsilon}$ der Vektor mit den Störgrößen.

Weiterhin müssen die folgenden Annahmen gelten:

1. $E(\boldsymbol{\epsilon}) = \mathbf{0}$
2. $Cov(\boldsymbol{\epsilon}) = E(\boldsymbol{\epsilon}\boldsymbol{\epsilon'}) = \sigma^2 \boldsymbol{I}$
3. Die Designmatrix $\boldsymbol{X}$ besitzt vollen Spaltenrang, d.h. $rg(\boldsymbol{X}) = k + 1 = p$
4. bei klassischer Normalregression zusätzlich $\epsilon \sim N(\mathbf{0}, \sigma^2 \boldsymbol{I})$

Für Details dazu siehe [23].

Die Schätzung der Koeffizienten $\boldsymbol{\beta}$ erfolgt über die Minimierung der quadratischen Abstände zwischen den geschätzten Erwartungswerten und den wahren y.

5.2.3 Prädiktionsintervall

Um Grenzen angeben zu können, innerhalb derer Beobachtungen mit hoher Wahrscheinlichkeit dem Normalbetrieb entsprechen, und außerhalb derer die Beobachtungen mit hoher Wahrscheinlichkeit vom Normalbetrieb abweichen, können Prädiktions- oder Prognoseintervalle verwendet werden. Mithilfe dieser Intervalle können Grenzen konstruiert werden, innerhalb derer die neuen Beobachtungen mit einer bestimmten Wahrscheinlichkeit liegen, falls der Zusammenhang der zur Schätzung verwendeten Daten auch für die neuen Beobachtungen gilt. Es ist z. B. zu erwarten, dass 99% der neuen Beobachtungen innerhalb des 99%-Prognoseintervalls liegen.[23]

Ein Prognoseintervall für eine zukünftige Beobachtung y_0 an der Stelle $\boldsymbol{x_0}$ zum Niveau $1 - \alpha$, also sodass zukünftige Beobachtungen mit der Wahrscheinlichkeit $1 - \alpha$ in das

Intervall fallen, ist gegeben durch

$$x_0'\hat{\beta} \pm t_{n-p}\left(1-\alpha/2\right)\hat{\sigma}\left(1+x_0'\left(X'X\right)^{-1}x_0\right)^{1/2}.$$

Dabei ist $t_{n-p}\left(1-\alpha/2\right)$ das $1-\alpha/2$-Quantil der Students t-Verteilung für n Freiheitsgrade und $\hat{\sigma}$ die geschätzte Standardabweichung der Residuen ϵ.[23]

Abb. 5.4 zeigt den geschätzten Erwartungswerte mit 95%-Prognoseintervall für das in 5.2 vorgestellte Lineare Modell.

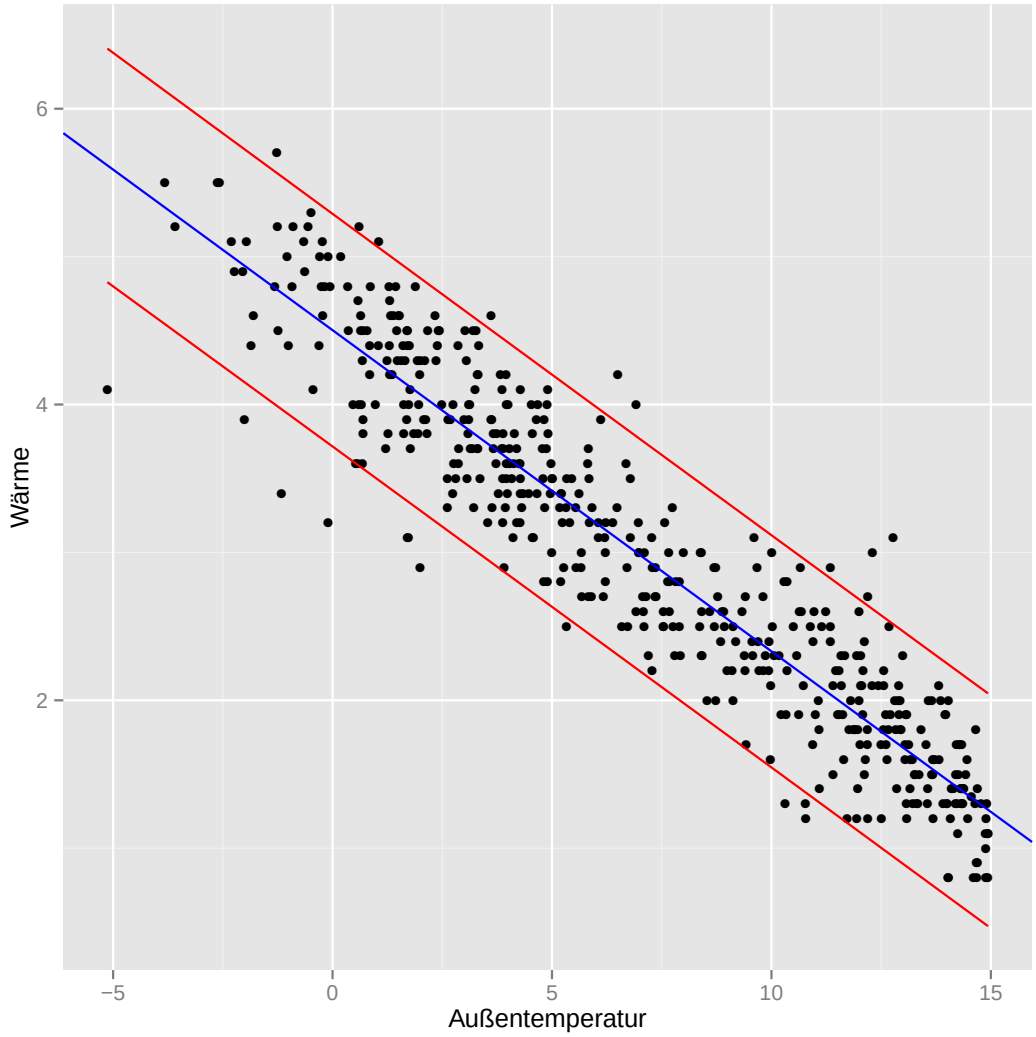


Abbildung 5.4: Wärmemengenverbrauch in Abhängigkeit von der Außentemperatur für Temperaturen unter 15°C mit Regressionsgerade (blau) und 95%-Prognoseintervall (rot) eines Linearen Modells.

5.3 Additives Regressionsmodell

Häufig sind die Zusammenhänge zwischen einer oder mehreren Einflussgrößen und der Zielgröße nicht linear. Mithilfe der additiven Regression können stetige Zusammenhänge sehr flexibel geschätzt werden. Das Standardmodell der additiven Regression lautet

$$y_i = \underbrace{f_1(z_{i1}) + ... + f_q(z_{iq})}_{\text{nichtlineare Effekte}} + \beta_0 + \underbrace{\beta_1 x_{i1} + ... + \beta_k x_{ik}}_{\text{lineare Effekte}} + \epsilon_i.$$

Dabei entsprechen die linearen Effekte für die Kovariablen $x_1, \ldots, x_k$ denen der linearen Regression und die Funktionen $f_1(z_1), \ldots, f_q(z_q)$ beschreiben die nichtlinearen Effekte der stetigen oder metrischen Kovariablen $z_1, \ldots, z_q$. Sie werden auch als "glatt" bezeichnet. Für die Fehlervariablen ϵ_i werden die gleichen Annahmen getroffen wie im klassischen linearen Regressionsmodell.[23]

Abb. 5.5 zeigt die Verbrauchswerte des Zählers `diff_x1120_01_021anhv01elz0001zw30` zu 15-Minutenabschnitten aggregiert in Abhängigkeit von der Tageszeit (in Sekunden seit Tagesbeginn) für Werktage mit dem mithilfe eines additiven Modells geschätzten Erwartungswert.

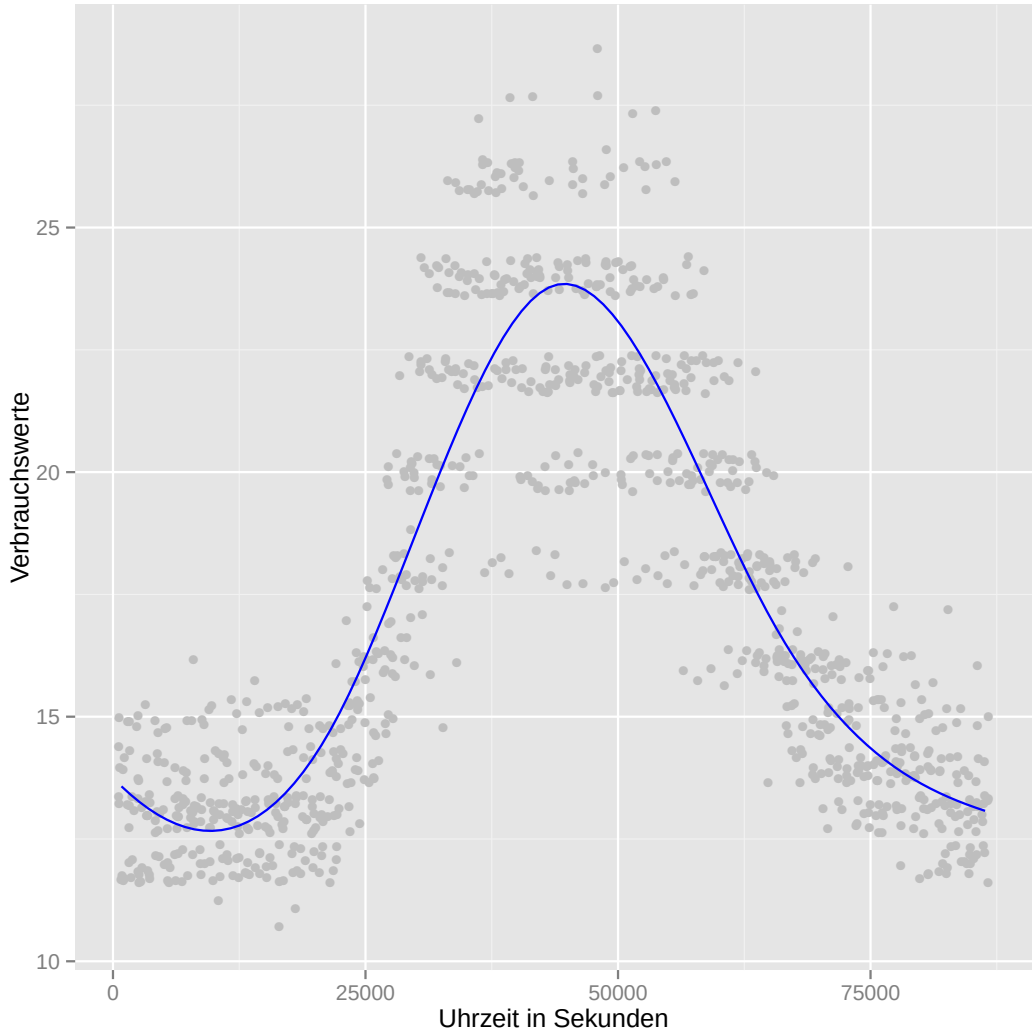


Abbildung 5.5: Verbrauch in Abhängigkeit der Tageszeit (in Sekunden seit Tagesbeginn) mit Erwartungswert eines additiven Modells.

Diese glatten Funktionen können mithilfe von sogenannten Basisfunktionen geschätzt werden. Hierfür wird der Bereich, den die Kovariable abdeckt, in Abschnitte zerlegt, transformiert und zu jedem Abschnitt ein Koeffizient geschätzt. Die Summe der mit dem jeweiligen Koeffizienten multiplizierten Basen ergibt dann $f(x)$. Ein Beispiel für diese Basen bilden P-Splines. Für Details dazu siehe [23]. Abb. 5.6 zeigt diese Splinebasen (eine B-Spline-Basis mit einem diskreten Strafterm auf die Basiskoeffizienten nach Eilers und Marx[18]) und den konstanten Wert von 1 für den Intercept (rot) für die Einflussgröße Tageszeit und in Abb. 5.7 sind diese Basen und die Interceptkonstante bereits mit den entsprechenden Koeffizienten multipliziert. Die Summe aller Basen und des Intercepts

ergibt dann die fertige Schätzung aus Abb. 5.5.

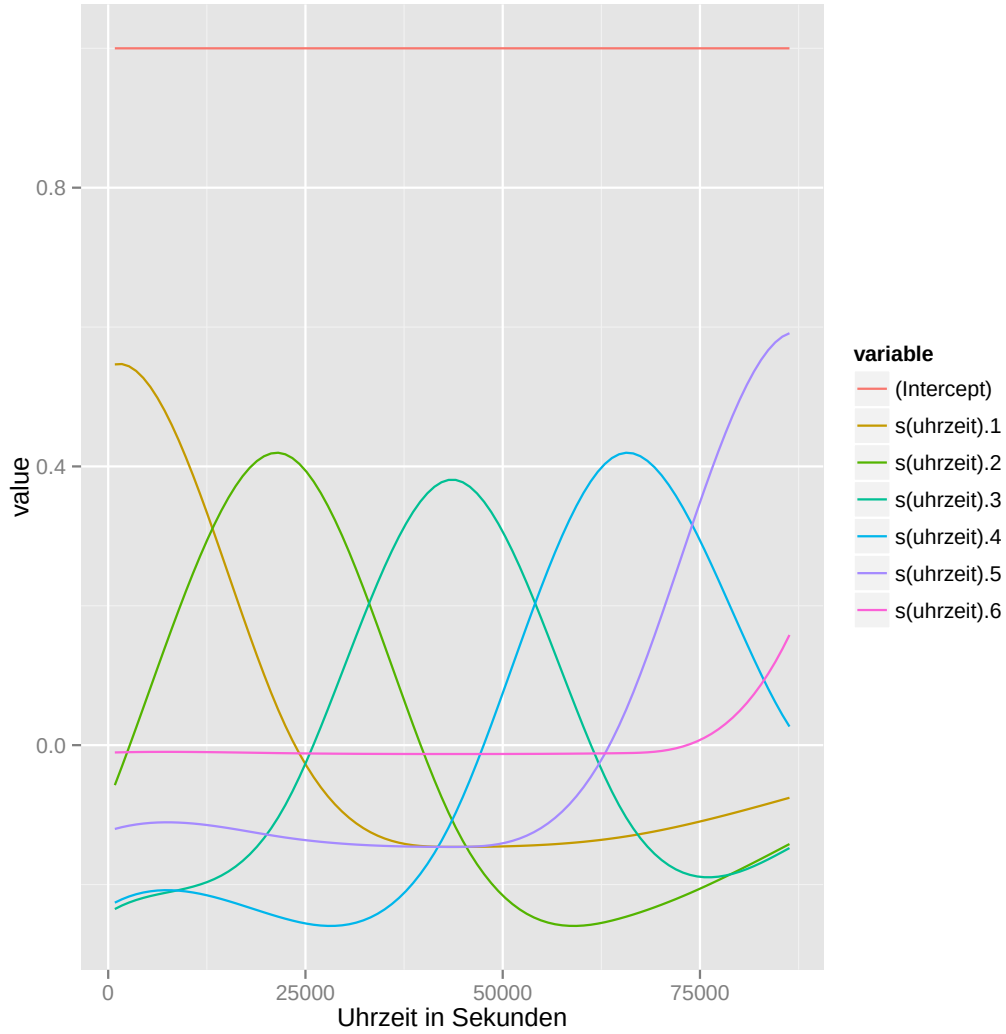


Abbildung 5.6: P-Spline-Basen des Modells aus Abb. 5.5.

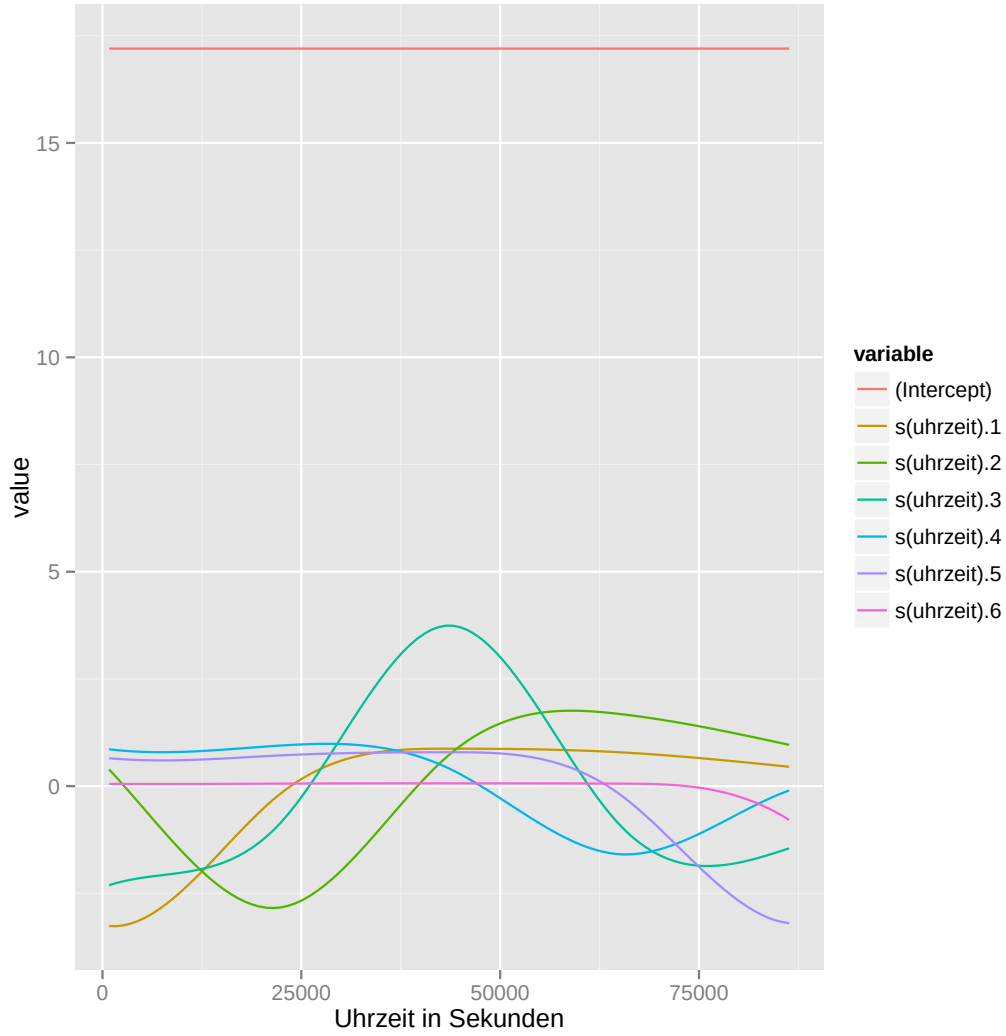


Abbildung 5.7: Mit ihren Koeffizienten multiplizierte P-Spline-Basen des Modells aus Abb. 5.5.

5.4 Quantilregression

Der für das lineare Regressionsmodell verwendete Kleinste-Quadrate-Schätzer reagiert äußerst sensibel auf eine kleine Menge von Ausreißern. Deshalb ist er in vielen Fällen, in denen keine Normalverteilung vorhanden ist, sehr schlecht geeignet. Dies gilt insbesondere für long-tailed-Verteilungen, d.h. wenn weiter vom Erwartungswert entfernte Beobachtungen häufiger auftreten als es bei einer Normalverteilung der Fall wäre. Die Annahme normalverteilter Fehler ist in der Realität jedoch häufig Spekulation. Eine Lösung

hierfür bietet die Quantilregression.[21]

Während bei der linearen Regression der bedingte Erwartungswert $\mathbb{E}(Y|x)$ als Funktion von Kovariablen x modelliert wird, werden bei der Quantilregression bedingte Quantile τ, also Quantile in Abhängigkeit von der Kovariablen x, modelliert.[20]

Ein Quantil ist ein Schwellenwert, der größer als ein bestimmter Anteil der Beobachtungen ist. So ist z. B. das 25%-Quantil der Wert, für den gilt, dass 25% aller Werte kleiner sind als dieser Wert.[19]

Die Modellformel für die lineare Quantilregression lautet

$$y_i = \boldsymbol{x}_i^T \boldsymbol{\beta}_\tau + \epsilon_{\tau i}$$

und damit ergibt sich für ein Quantil τ, $0 < \tau < 1$

$$Q_\tau(y_i|\boldsymbol{X} = \boldsymbol{x}_i) = \boldsymbol{x}_i^T \boldsymbol{\beta}_\tau$$

Die Störgrößen $\epsilon_{\tau i}$ dürfen heteroskedastisch sein, aber müssen unabhängig sein und $F_{\epsilon_{\tau i}}(0) = \int_{-\inf}^{0} f(\epsilon_{\tau i}) d\epsilon_{\tau i} = \tau$ muss erfüllt sein.[20]

Die Quantilregression ist eine robuste Methode, d.h. die Schätzer sind unempfindlich gegenüber Ausreißern. Die Methode verzichtet auf Annahmen zur Verteilung der Störgrößen und erlaubt die Untersuchung des Einflusses der Kovariablen auf unterschiedliche Quantile der Zielvariablen-Verteilung[22]. Allerdings kann es zu Quantilüberschneidungen (Quantile Crossing) kommen, d.h. zwei Quantilregressionskurven können sich überschneiden, sodass für einen bestimmte Kovariablenkombination z. B. das geschätzte 25%-Quantil fälschlicherweise über dem 50%-Quantil liegt.[20]

Würde man analog zur linearen Regression anstatt der Summe der quadratischen Abstände die Summe der absoluten Abstände zwischen den beobachteten y_i und den geschätzen $\hat{y}_i$ minimieren, würde sich als Sonderfall der Quantilregression eine Schätzung des bedingten Medians (50%-Quantil) ergeben.[23]

Beispiele für die Anwendung der Quantilregression im Energiemonitoring und deren Ergebnisse befinden sich in 4.4.3.

5.5 Loess-Kurve

Will man den Zusammenhang zweier Variablen als glatte Kurve darstellen, ohne vorausgehende Annahmen zur Art dieses Zusammenhangs zu treffen, eignen sich bivariate Glätter.

Der in der Praxis am häufigsten verwendete bivariate Glätter wird Lowess- oder Loess-Kurve genannt. Mit einem bivariaten Glätter kann eine glatte Kurve durch ein Streudiagramm gezeichnet werden. Die beiden Akronyme stehen dabei für das Prinzip der lokal gewichteten Regression, d.h. der y-Wert zu einem bestimmten Punkt entlang der x-Achse wird nur durch die Punkte in dessen Nähe bestimmt. Die Methode macht keine Annahmen zur Form des Zusammenhangs und erlaubt, dass diese Form aus den Daten selbst entsteht.[25]

Zur Schätzung der Kurve wird ein Polynom niedrigen Grades mit der Teilmenge der Daten gefittet, deren x-Werte in der Nähe der Beobachtung liegen, deren y-Wert gerade geschätzt wird. Dazu wird ein lineares Modell mit zu einem Polynom transformierten x-Werten und gewichteten Beobachtungen verwendet. Die Gewichte der Punkte, deren x-Werte in der Nähe der Beobachtung mit dem zu schätzenden y-Wert liegen, sind dabei höher als die der anderen Beobachtungen. Der mit diesem geschätzten Modell für den jeweiligen x-Wert vorhergesagte Wert ist dann der Wert der Regressionsfunktion. Dies wird für jede Beobachtung einzeln durchgeführt.[26]

Abb. 5.8 zeigt ein Beispiel für eine Loess-Kurve.

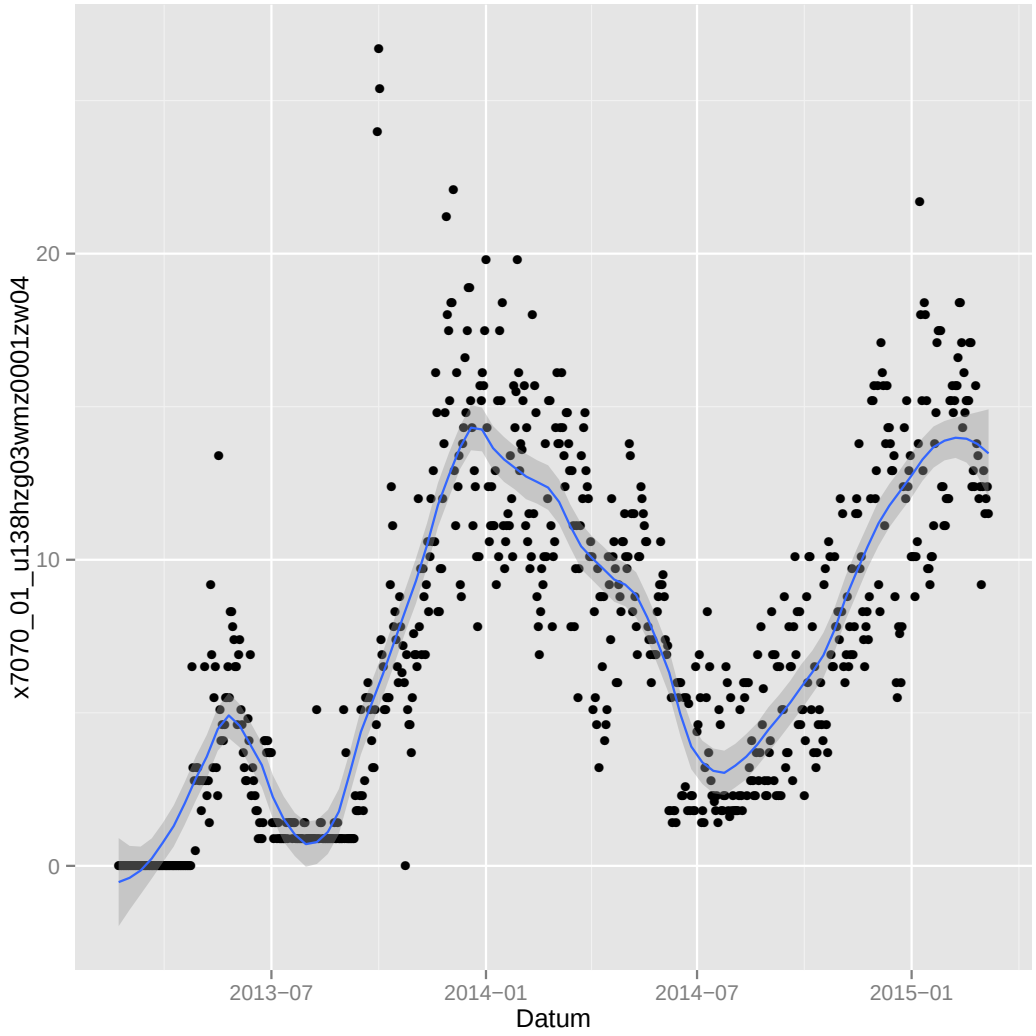


Abbildung 5.8: Bivariate Glättung einer Zeitreihe eines täglichen Messwertes aus der Oettingenstraße mithilfe einer Loess-Kurve (mit Standardfehler).

6 Verwendete Werkzeuge (Kurzanleitung)

In diesem Kapitel wird eine kurze Einführung in die verwendeten Werkzeuge gegeben, wobei insbesondere auf die für EnMoLMU relevanten Dinge eingegangen wird, um die Einarbeitungszeit für eine Weiterentwicklung zu verkürzen.

6.1 MySQL

Es sollen langfristig bis zu 10 Jahre an Daten im Minutentakt verwendet werden, was – berücksichtigt man keine Schaltjahre in der Berechnung – 5.256.000 Datensätzen (hier in der Bedeutung von Zeilen) je Tabelle entspricht. Mittelfristig sind elf Subserver mit teilweise je mehreren hundert Spalten vorgesehen. Durch lineare Hochrechnung kleiner Dateien (die Größe von Dateien im CSV-Format dürfte sich annähernd linear zu deren Inhalt verändern) ergab sich als grobe Abschätzung 180 GB als Gesamtgröße aller GLT-Dateien im CSV-Format (siehe 3.1). Hinzu kommen noch die Wetterdaten des Meteorologischen Instituts. Weiter besteht die Möglichkeit, dass weitere Subserver hinzukommen und die Daten, nachdem sie älter als 10 Jahre sind, nicht sofort bzw. überhaupt nicht gelöscht werden. Auch ist damit zu rechnen, dass gegenwärtig monatlich abgelesene Zähler im Laufe der Zeit auf automatische minütliche Erfassung umgestellt werden und darüber hinaus variiert die Größe der Daten um ein Vielfaches in Anhängigkeit von ihrem Format. Somit muss ein Vielfaches der 180 GB verarbeitet werden können und die Obergrenze eher im Terabyte-Bereich angesiedelt werden. Bereits einzelne Subserver mit Daten über 2 Jahre können als CSV-Datei bei einem 32-Bit Windows 7 Professional mit 8 GB RAM, wovon 3,39 GB verwendbar sind, weder in Excel noch in R geöffnet werden. Bei einem 64-Bit Windows entstehen diese Probleme natürlich erst bei größeren Dateien.

Relationale Datenbanksysteme, wie MySQL bieten die Möglichkeit, aus einer Tabelle nur die benötigten und bereits zusammengefassten Daten in den Hauptspeicher bzw. R einzulesen. Die maximale Größe einer MySQL-Tabelle wird zwar auch durch das Betriebssystem bestimmt, liegt aber selbst bei einem Win32-System mit NTFS bei mindestens 2 TB[4].

Die maximale Spaltenanzahl pro Tabelle für MySQL liegt bei 4.096, es stehen aber maximal 65.535 Bytes pro Zeile zur Verfügung[7]. Die meisten Spalten in den für EnMoLMU verwendeten Tabellen sind entweder vom Typ INT (integer) oder DOUBLE und benötigen somit jeweils 8 bzw. 4 Bytes (DATETIME benötigt 8 Bytes und der Speicherbedarf von VARCHAR ist abhängig von der tatsächlichen Länge)[8]. Dies würde $\frac{65.535}{8} = 8.191,875$ Spalten mit je 8 Bytes ergeben. Somit können die 4.096 Spalten also mit keiner Kombination aus INT, DOUBLE, und DATETIME unterschritten werden und es bleibt dennoch Platz für einige Spalten vom Type VARCHAR. Dies dürfte für alle Subserver, die jeweils eine eigene Tabelle bilden, ausreichen. Da Joins mehrerer Tabellen bei vielen Werten, wie sie z. B. bei Messwerten im Minutentakt über einen längeren Zeitraum vorhanden sind, relativ lange dauern (zumindest falls sie vor dem Join nicht zu längeren Abschnitten aggregiert werden), erscheint eine Aufteilung in zu viele Tabellen nicht zielführend.

Ursprünglich war die Verwendung von SQLite geplant, welches leichter zu installieren, einzurichten und zu portieren ist und alle benötigten Funktionen bereitstellt. Da später doch entschieden wurde, EnMoLMU auf einem virtuellen Server auszuführen und auf diesem nur MySQL und Microsoft SQL Server unterstützt werden, fiel die Wahl auf MySQL.

MySQL wird unter anderen von Google und Facebook verwendet[6] und ist laut DB-Engines Ranking nach Oracle das zweitpopulärste Datenbankmanagementsystem (Stand März 2015)[5] und laut MySQL-Website die populärste Open-Source-Datenbank der Welt[3]. Somit dürfte es leicht fallen, Studenten und Mitarbeiter mit Kenntnissen darin zur Erweiterung von EnMoLMU zu finden.

Der Unterabschnitt 6.1.3 behandelt die Installation und Einrichtung von MySQL. **Passwörter und Port** sind in der vorliegenden Arbeit geschwärzt ("****").

6.1.1 Datenbanksprache SQL

In relationalen Datenbanksystemen wird je nach Datenbankmanagementsystem ein bestimmter Dialekt der Datenbanksprache SQL verwendet. Für eine MySQL-Datenbank kann diese z. B. im "MySQL 5.6 Command Line Client" ausgeführt werden. Im Folgenden sollen einige einfache und für die vorliegende Arbeit relevante Abfragen vorgestellt werden, für einen umfangreichere Anleitung sei hier auf eines der unzähligen Online-Tutorials verwiesen.

Bei den SQL-Befehlen wird nicht zwischen Groß- und Kleinschreibung unterschieden, zur einfacheren Unterscheidung zwischen Befehl und Namen von Datenbanken, Tabellen und Spalten werden hier alle Befehle in Großbuchstaben geschrieben. Jede Anweisung muss mit einem ";" abgeschlossen werden.

Alle vorhandenen Datenbanken können mit folgender Anweisung angezeigt werden:

```
SHOW DATABASES;
```

Auswählen der Datenbank *enmolmu* in der sich die Tabellen befinden, aus denen eine Abfrage gemacht werden soll:

```
USE enmolmu;
```

Anzeigen aller Tabellen, welche sich in der ausgewählten Datenbank befinden:

```
SHOW TABLES;
```

Eine neue Datenbank mit dem Namen *datenbank1* kann mit

```
CREATE DATABASE datenbank1;
```

angelegt werden, und eine neue Tabelle mit

```
CREATE TABLE tabelle1 (spalte1 INT, spalte2 VARCHAR(255))
             DATA DIRECTORY = 'D:/Tabellenordner';
```

wobei *tabelle1* der Tabellenname, *spalte1* und *spalte2* die Spaltennamen, INT und VARCHAR(255) deren Datentypen und `'D:/Tabellenordner'` der Ordner, in dem die Daten der Tabelle gespeichert werden, sind. DATA DIRECTORY muss nicht angegeben werden, wenn der voreingestellte Ordner verwendet werden soll.

Wieder gelöscht werden können Tabellen mit

```
DROP TABLE tabelle1, tabelle2;
```

und Datenbanken mit

```
DROP DATABASE datenbank1;
```

Wir gehen im Folgenden von einer Tabelle mit dem Namen *oet67* aus, welche die Spalten *datum* mit dem kompletten Datum und die Spalten *jahr, monat, tag, stunde* und *minute* als Integer beinhaltet. Jede Zeile der Tabelle entspricht einer Minute.

Ein Minimalbeispiel für eine Abfrage ist

```
SELECT * FROM oet67 limit 6;
```

wobei nach SELECT die gewünschten Spalten ausgewählt werden und nach FROM die Tabelle, in der sich die Daten befinden. Mit "*" nach SELECT werden alle Spalten der Tabelle ausgewählt. Dieses Beispiel zeigt also die komplette Tabelle *oet67* an. Fügt man am Schluss der Abfrage LIMIT 6 an, so werden nur die ersten 6 Zeilen der abgefragten Tabelle angezeigt.

Um nur einen Teil der Tabelle anzuzeigen, können nach SELECT die Spalten und nach WHERE die Zeilen in Form von Bedingungen ausgewählt werden

```
SELECT jahr, monat, tag FROM oet67 WHERE jahr = 2015 and monat >= 11;
```

Hier werden also von den Spalten *jahr, monat* und *tag* der Tabelle *oet67* die Zeilen ausgewählt, bei denen in der Spalte *jahr* 2015 und in der Spalte *monat* ein Wert größergleich 11 steht, also November und Dezember.

Das folgende Beispiel zeigt eine Abfrage, die die meisten grundlegenden Bausteine enthält (von denen zwar nicht alle vorhanden sein müssen, aber die vorhandenen in dieser Reihenfolge stehen müssen).

```
SELECT COUNT(*) AS anzahl, MAX(datum) FROM oet67 WHERE jahr = 2014
GROUP BY monat HAVING count(*) <> 4460;
```

Es wird wieder die Tabelle oet67 verwendet und daraus alle Zeilen (welche in diesem Zusammenhang als "Datensätze" bezeichnet werden) ausgewählt, in denen in der Spalte jahr 2014 steht. Diese Zeilen werden dann durch GROUP BY monatsweise zusammengefasst, d.h. für jeden Monat gibt es nur eine Zeile im Ergebnis. da hierfür die ca. $30 \times 24 \times 60 = 4.320$ Zeilen (Anzahl der Minuten des jeweiligen Monats) zu einer zusammengefasst werden müssen, müssen nach dem SELECT-Statement Funktionen von Spalten stehen, die die Art der Zusammenfassung angeben. Z. B. COUNT(*) zählt die Zeilen des jeweiligen Monates, MAX(datum) gibt den Maximalwert des Datums im jeweiligen Monat an und auf AS folgt der gewünschte Name für die errechnete Spalte. Aus diesen zusammengefassten Zeilen kann dann wiederum eine Teilmenge nach bestimmten Kriterien über HAVING ausgewählt werden. Im obigen Beispiel werden also nur die Monate angezeigt, deren Gesamtanzahl an Zeilen (also Anzahl der Minuten) ungleich 4.460 ist.

Für die Bedingungen nach WHERE oder HAVING gilt:

Operator	Bedeutung	Beispiel
=	gleich	tag = 1
<	kleiner	tag < 10
>	größer	tag > 10
<=	kleinergleich	tag <= 15
>=	großergleich	jahr >= 2014
BETWEEN	zwischen zwei Werten	monat BETWEEN 3 AND 7
AND	Verküpfung mehrerer Bedingungen durch und zugleich	jahr = 2015 AND monat = 1
OR	Verknüpfung mehrere Bedingungen durch oder	jahr = 2015 OR monat <> 5

Tabelle 6.1: SQL Verknüpfungen von Bedinungen

Einige Funktionen zum Zusammenfassen von Zeilen:

Funktion	Bedeutung	Beispiel
AVG	Durchschnitt	AVG(preis)
MAX	Maximum	MAX(datum)
MIN	Minimum	MIN(datum)
SUM	Summe	SUM(preis)
COUNT	Anzahl	COUNT(*)

Tabelle 6.2: SQL Funktionen zur Datenzusammenfassung

Befinden sich die Spalten in verschiedenen Tabellen, so müssen diese über JOIN verknüpft werden.

```
SELECT oet67.x7070_01_u138hzg03tem0003mp01, theresienwetter.aussentemperatur
FROM oet67 JOIN theresienwetter USING(datum)
WHERE oet67.jahr = 2015 and oet67.monat = 1 and oet67.tag = 3
      and oet67.stunde = 15;
```

Hier werden in SELECT der Tabellenname, in dem sich die jeweilige Spalte befindet und der Spaltenname durch einen Punkt getrennt angegeben. Falls es eine Spalte mit diesem Namen nur in einer Tabelle gibt, ist es nicht nötig, den Tabellennamen vor den Spaltennamen zu schreiben. Die Tabellen, aus denen Spalten, die verwendet werden sollen stammen, werden wieder nach FROM angegeben und mit JOIN verknüpft. Über USING wird festgelegt, anhand welcher Spalte (die in beiden Tabellen vorhanden sein muss) die Tabellen verknüpft werden sollen (d.h. welche Zeile der einen Spalte zu welcher Zeile der anderen gehört).

Über die WHERE-Klausel wird die Stunde 15 des Tages 3 vom Monat 1 des Jahres 2015 ausgewählt, also insgesamt 1 Stunde, was 60 Zeilen entspricht (alle Minuten zwischen 15.00 Uhr und 15.59 des 3.1.2015). Dies geschieht anhand von Spalten, welche aus der Tabelle oet67 kommen.

Statt USING(datum) kann ON(oet67.datum = theresienwetter.datum) verwendet werden, was als einzigen Unterschied zur Folge hat, dass für die Ergebnistabelle beide Datumsspalten zur Verfügung stehen (die aber ohnehin gleich sein müssen). Notwendig ist ON

jedoch, falls die Datumsspalten in den beiden Tabellen unterschiedliche Namen haben, z. B. ON(oet67.date = theresienwetter.datum).

Es gibt unterschiedliche Varianten, wie die beiden Tabellen verknüpft werden können. JOIN, CROSS JOIN und INNER JOIN sind bei MySQL äquivalent[9] und sorgen dafür, dass nur die Zeilen im Ergebnis vorkommen, für die es in beiden Tabellen eine Entsprechung gibt. LEFT JOIN erzeugt eine Tabelle, in der alle Zeilen, welche in der ersten Tabelle vorhanden sind vorkommen und nur die aus der zweiten, für die es eine entsprechende Zeile in der ersten gibt. RIGHT JOIN verwendet alle Zeilen der zweiten Tabelle. Ein Beispiel mit zwei übersichtlichen Minitabellen kann unter A.2.2 gefunden werden. Für Details zum Thema JOIN-Operationen siehe `https://dev.mysql.com/doc/refman/5.6/en/join.html`

Falls aggregierte Daten aus den Tabellen benötigt werden, so verkürzt sich die Abfragedauer deutlich, wenn die beiden Tabellen, wie im folgenden Beispiel, bereits vor dem JOIN aggregiert werden.

```
SELECT *
FROM (SELECT AVG(x7070_01_u138hzg03tem0003mp01), MAX(datum) AS datum
FROM oet67
GROUP BY jahr, monat) X
JOIN
(SELECT AVG(aussentemperatur), MAX(datum) AS datum FROM theresienwetter
GROUP BY jahr, monat) Y
USING(datum);
```

In diesem Beispiel werden erst die einzelnen Tabellen nach den Spalten *jahr* und *monat* gruppiert, wobei aus den einzelnen Monaten immer das jüngste Datum und die Durchschnittstemperatur ausgewählt wird. Das neu errechnete jüngste Datum bekommt als Spaltennamen jeweils wieder *datum*, womit die beiden Tabellen verbunden werden. *X* und *Y* sind Aliase, also Namen für die aggregierten Tabellen. Diese werden zwar in diesem Beispiel nicht verwendet, müssen aber immer festgelegt werden.

Sollen die abgefragten Daten verändert ausgegeben werden, zum Beispiel, um bestimmte Werte oder Bereiche auf NULL zu setzen, falls diese als falsch betrachtet werden, kann IF verwendet werden:

```
SELECT minute, IF(minute between 1 and 3, NULL, minute) AS minute_neu
FROM the46 LIMIT 6;
```

Es werden zwei Spalten ausgegeben: die ersten 6 Werte der ursprünglichen Spalte *minute* und die veränderte Spalte *minute*, bei der alle Werte zwischen 0 (ausgeschlossen) und 3 (eingeschlossen) auf NULL gesetzt wurden. Die Anweisung IF entspricht ungefähr der `ifelse`-Anweisung in R. Das erste Argument beinhaltet die Bedingung, das zweite, was zurückgegeben werden soll, falls diese erfüllt ist und das dritte die Rückgabe für den Fall, dass sie nicht erfüllt ist. Findet die Abfrage aus R über `sqlQuery` {RODBC} statt, wird NULL in NA umgewandelt.

Für Zählervariablen werden den Tabellen von EnMoLMU zusätzliche Spalten mit den Differenzen, also den Verbräuchen hinzugefügt. Diese Differenzen können aber auch bei der Abfrage der Daten aus der Datenbank berechnet werden:

```
SET @diff = NULL;
SET @prevval = -1;
SELECT datum,
       (@diff:=IF(@prevval=-1, NULL, x1120_01_021anhv01elz0001zw30 - @prevval))
       difference,
       (@prevval:=x1120_01_021anhv01elz0001zw30) prevval,
       diff_x1120_01_021anhv01elz0001zw30
       FROM glt_hist_the46
       WHERE datum BETWEEN '2015-02-06 14:00:00' AND '2015-02-06 14:10:00';
```

datum	difference	prevval	diff_x1120_01_021anhv01elz0001zw30
2015-02-06 14:00:00	NULL	1574805	2
2015-02-06 14:01:00	0	1574805	0
2015-02-06 14:02:00	2	1574807	2
2015-02-06 14:03:00	2	1574809	2
2015-02-06 14:04:00	0	1574809	0
2015-02-06 14:05:00	2	1574811	2

```
| 2015-02-06 14:06:00 |          2 | 1574813 |                                    2 |
| 2015-02-06 14:07:00 |          2 | 1574815 |                                    2 |
| 2015-02-06 14:08:00 |          0 | 1574815 |                                    0 |
| 2015-02-06 14:09:00 |          2 | 1574817 |                                    2 |
| 2015-02-06 14:10:00 |          2 | 1574819 |                                    2 |
+---------------------+------------+---------+--------------------------------------+
11 rows in set (0.00 sec)
```

Wie an der Ausgabe zu erkennen ist, ist die dabei erzeugte Spalte *difference* identisch mit der durch EnMoLMU berechneten *diff*-Spalte, außer dass der erste Wert fehlt. Es muss somit eine Zeile mehr abgefragt werden.

Falls nicht sichergestellt ist, dass die Tabelle nach Datum geordnet ist, muss erst eine sortierte Version der Tabelle erstellt werden. Es wird nur der Zeitabschnitt der Tabelle sortiert und weiterverwendet, welcher auch benötigt wird, was die Abfragedauer deutlich verkürzen kann:

```
SET @diff = NULL;
SET @prevval = -1;
SELECT datum,
       (@diff:=IF(@prevval=-1, NULL, x1120_01_021anhv01elz0001zw30 - @prevval))
       difference,
       (@prevval:=x1120_01_021anhv01elz0001zw30) prevval,
       diff_x1120_01_021anhv01elz0001zw30
       FROM (SELECT * FROM glt_hist_the46
       WHERE datum BETWEEN '2015-02-06 14:00:00' AND '2015-02-06 14:10:00'
       ORDER BY datum) the46sortiert;
```

Allerdings ist es für beide Varianten notwendig, dass der vorherige Wert auch tatsächlich vorhanden ist. D.h. es wird nicht überprüft, ob einzelne Minuten fehlen. Dies kann mit einem Join der Tabelle mit sich selber um eine Zeiteinheit verschoben bewerkstelligt werden. Dabei verlangsamt sich allerdings die Ausführungsgeschwindigkeit deutlich:

```
SELECT x.datum, x.x1120_01_021anhv01elz0001zw30 AS alt,
       y.x1120_01_021anhv01elz0001zw30 AS neu,
  y.x1120_01_021anhv01elz0001zw30 - x.x1120_01_021anhv01elz0001zw30 AS differenz,
  y.diff_x1120_01_021anhv01elz0001zw30 AS diff_variable
FROM glt_hist_the46 x
     LEFT OUTER JOIN glt_hist_the46 y
     ON x.datum = date_add(y.datum, INTERVAL -1 MINUTE)
     WHERE x.datum BETWEEN '2015-01-01 14:00:00' AND '2015-01-01 14:10:00' ;
```

```
+---------------------+---------+---------+-----------+---------------+
| datum               | alt     | neu     | differenz | diff_variable |
+---------------------+---------+---------+-----------+---------------+
| 2015-01-01 14:00:00 | 1521710 | 1521710 |         0 |             0 |
| 2015-01-01 14:01:00 | 1521710 | 1521711 |         1 |             1 |
| 2015-01-01 14:02:00 | 1521711 | 1521712 |         1 |             1 |
| 2015-01-01 14:03:00 | 1521712 | 1521713 |         1 |             1 |
| 2015-01-01 14:04:00 | 1521713 | 1521714 |         1 |             1 |
| 2015-01-01 14:05:00 | 1521714 | 1521714 |         0 |             0 |
| 2015-01-01 14:06:00 | 1521714 | 1521715 |         1 |             1 |
| 2015-01-01 14:07:00 | 1521715 | 1521716 |         1 |             1 |
| 2015-01-01 14:08:00 | 1521716 | 1521717 |         1 |             1 |
| 2015-01-01 14:09:00 | 1521717 | 1521718 |         1 |             1 |
| 2015-01-01 14:10:00 | 1521718 | 1521718 |         0 |             0 |
+---------------------+---------+---------+-----------+---------------+
11 rows in set (6.52 sec)
```

Es ist auch möglich, einfache Berechnungen im "MySQL 5.6 Command Line Client" durchzuführen. Folgendes Beispiel berechnet die Anzahl der Minuten eines Tages:

```
SELECT 24*60;
```

Aktuell laufende Prozesse können mit

```
show processlist;
+-----+------+-----------+---------+---------+-------+-------+------------------+
| Id  | User | Host      | db      | Command | Time  | State | Info             |
+-----+------+-----------+---------+---------+-------+-------+------------------+
| 385 | prog | localhost | NULL    | Query   |     0 | NULL  | show processlist |
| 957 | prog | localhost | test    | Sleep   | 21619 |       | NULL             |
+-----+------+-----------+---------+---------+-------+-------+------------------+
```

aufgelistet und mit kill Prozessnummer (Id) beendet werden:

```
kill 957
```

[14]

Eine nachträgliche Veränderung einer Tabelle kann z. B. mit INSERT INTO, UPDATE oder ALTER TABLE durchgeführt werden. Beispielsweise können weitere Zeilen (INSERT INTO) oder Spalten (ALTER TABLE) hinzugefügt werden oder einzelne Einträge (UPDATE) verändert werden.

```
# neue Tabelle anlegen:
CREATE TABLE tabelle1 (zaehler1 INT, datum DATETIME);
# eine Zeile einfügen:
INSERT INTO tabelle1 SET zaehler1 = 8, datum = '2015-01-01 10:00:00';
# neue Spalte hinzufügen:
ALTER TABLE tabelle1 ADD COLUMN zaehler2 INT;
# einen Werte in die neue Spalte schreiben:
UPDATE tabelle1 SET zaehler2 = 10 WHERE datum = '2015-01-01 10:00:00';
```

Das Festlegen einer Spalte als Schlüssel (ein für jede Zeile eindeutiger Index, für die Tabellen in EnMoLMU i. d. R. die Spalte *datum*) kann eine Reihe von Vorgängen, z. B. das nachträgliche Einfügen größerer Datenmengen für nachträglich hinzugefügte Spalten, drastisch beschleunigen:

```
ALTER TABLE tabelle1 ADD PRIMARY KEY (datum);
```

6.1.2 Speicherort der Datenbank

Die Daten der Datenbank müssten sich entweder `C:\ProgramFiles\MySQL\MySQLServer5.6\data` oder im versteckten Ordner `C:\ProgramData` (kann durch direkte Eingabe des Pfads im Explorer angezeigt werden) befinden. Die Dateien im Ordner haben bei InnoDB als Storage Engine die Dateiendungen .ibd und .frm und sind jeweils in einem Ordner mit dem Namen der Datenbank.

Per SQL-Anweisung kann das Verzeichnis folgendermaßen abgefragt werden:

```
SHOW VARIABLES WHERE Variable_Name LIKE "datadir";
```

6.1.3 Installieren und Einrichten

Installieren von MySQL Server 5.6

Für die MySQL-Server-Installationen zur Entwicklung von EnMoLMU wurden entweder die einzelnen Komponenten Server, Verbindungen und Doku ausgewählt oder der Developer-Default. Der Haken bei "start mysql server at windows startup" sollte gesetzt bleiben. Sollte es zur Meldung "Failing Requirements" kommen, da Visual Studio und Python nicht installiert sind, kann diese meistens ignoriert werden. Auf einem PC (Laptop) musste allerdings Visual Studio installiert werden, um die ODBC-Konnektoren installieren zu können, außerdem wurde das .NET-Framework benötigt. Eine Installationsanweisung befindet sich unter `http://dev.mysql.com/doc/refman/5.7/en/installing.html`, der Download unter `http://dev.mysql.com/downloads/mysql/`.

Nach der Installation sollten noch die Systemumgebungsvariablen gesetzt werden (siehe 6.2).

ODBC-Verbindung einrichten

Um von R aus auf MySQL über das R-Paket RODBC zugreifen zu können, muss eine ODBC-Verbindung eingerichtet werden. Dies wird ausführlich auf [http://dev.mysql.com/doc/connector-odbc/en/connector-odbc-configuration-dsn-windows-5-2.html] beschrieben und soll an dieser Stelle nur kurz zusammengefasst werden.

Unter **Systemsteuerung - System und Sicherheit - Verwaltung** bzw. unter **Systemsteuerung - alle Systemsteuerungselemente - Verwaltung** befindet sich Datenquellen (ODBC) bzw. ODBC-Datenquellen (64-Bit). Dies muss als Administrator ausgeführt werden (also entweder, indem man bei Windows als Administrator angemeldet ist, oder über **Rechtsklick - als Administrator ausführen**). Sollte das Ausführen als Administrator nicht möglich sein, so kann im Folgenden Benutzer-DSN verwendet werden.

Im nun geöffneten ODBC-Datenquellen-Administrator können unter anderem die Reiter System-DSN (für alle Nutzer) und Benutzer-DSN (nur für den aktuellen Nutzer) ausgewählt werden. Solange jeder Nutzer vollen Zugang zur Datenbank haben soll, wird hier System-DSN empfohlen, um nicht für jeden Nutzer einen neuen Zugang einrichten zu müssen, der dann auch im Code jeweils angepasst werden muss.

Nach dem Klick auf Hinzufügen erscheint das Fenster "Neue Datenquelle erstellen" in welchem unter anderem MySQL ODBC 5.3 ANSI Driver und MySQL ODBC 5.3 Unicode Driver ausgewählt werden können. Bei der Entwicklung von EnMoLMU wurde der ANSI-Treiber verwendet.

Hier müssen folgende "Connection Parameters" eingegeben werden:

Data Source Name:	Irgendein zulässiger Name für die Datenquelle, auf die zugegriffen werden soll. Bei der EnMoLMU-Entwicklung wurde der Name ansi_system gewählt.
Description:	Beliebiger Text zur Beschreibung der Verbindung, z. B. "System-DSN und ANSI-Treiber"
TCP/IP Server:	Name des MySQL server host, auf den zugegriffen werden soll. Default und EnMoLMU: localhost
Port:	****
User:	MySQL-User mit den entsprechenden Rechten. Bei der EnMoLMU-Entwicklung wurde root als Administrator mit allen Rechten verwendet.
Password:	Passwort für diesen User. Bei der EnMoLMU-Entwicklung war dieses **** für den User root.
Database:	Hier sollte automatisch eine Liste der Datenbanken erscheinen, auf die der Nutzer eine Zugriffserlaubnis hat. Für die EnMoLMU-Entwicklung wurde die Datenbank *enmolmu* verwendet.

Tabelle 6.3: ODBC-Verbindung einrichten

6.1.4 RODBC

Um von R aus auf die MySQL-Datenbank zuzugreifen, wurde das R-Paket *RODBC* verwendet, welches über die in 6.3 eingerichtete ODBC-Verbindung auf die Datenbank zugreift. Als Alternative wären z. B. die R-Pakete *RMySQL*, welches nicht über die ODBC-Konnektoren, sondern direkt auf MySQL zugreift (und somit leichter einzurichten und theoretisch schneller ist) und *dplyr*, welches eine eigene Syntax bietet infrage gekommen. Aus der Sicht des Verfassers scheint keines der drei Pakete vollkommen zu sein, so funktionierte die Version von *dplyr*, welche zum Zeitpunkt des ersten MySQL-Tests installiert war mit MySQL überhaupt nicht, jedoch in der aktuellen Version könnte es wieder für die MySQL-Datenbank von EnMoLMU verwendet werden (Siehe dazu `https://github.com/rstats-db/RMySQL/issues/27`). Da es sich beim Schreiben in die und Lesen aus der Datenbank durchaus um von der Ausführungsgeschwindigkeit her ins Gewicht fallende Vorgänge handelt, wären mittelfristig Vergleiche mit anderen Paketen

und ggf. ein Umstieg zu erwägen, was aber im Rahmen dieser Arbeit aufgrund anderer Prioritäten nicht mehr möglich war.

Weitere Nachteile von *RODBC* sind, dass Tabellen- und Spaltennamen nur Kleinbuchstaben enthalten dürfen und keine Sonderzeichen, wie Umlaute und ß verwendet werden können. Als Vorteile wäre zu nennen, dass die automatische Umwandlung zwischen den Datentypen in R und denen in MySQL etwas ausgereifter erscheint (z. B. bei POSIX und DATETIME), aber auch hier mussten Anpassungen vorgenommen werden.

Die Dokumentation zu RODBC findet sich unter `http://cran.r-project.org/web/packages/RODBC/RODBC.pdf` und im Folgenden werden einige für EnMoLMU wichtige Befehle und Vorgänge vorgestellt.

Zum Laden des R-Pakets RODBC und zum Aufbau einer ODBC-Verbindung zu einer wird Folgendes in R ausgeführt:

```
library(RODBC)
ch <- odbcConnect("ansi_system",  DBMSencoding = "utf-8")
```

"ansi_system" ist der Name der unter 6.1.3 eingerichteten ODBC-Verbindung und die "utf-8"-Kodierung funktioniert, der Verfasser der vorliegenden Arbeit konnte allerdings nicht herausfinden, ob hier etwas anderes Vorteile bieten würde (der Default-Wert, also wenn das Argument nicht angegeben wird, ist die Kodierung des Betriebssystems, womit auch kein anderes Verhalten festgestellt werden konnte).

Wieder geschlossen werden kann die Verbindung mit folgendem Befehl, wobei mehrere Verbindungen gleichzeitig geöffnet sein können.

```
odbcClose(ch)
```

Mit sqlTables werden alle Tabellen ausgegeben, die in der Datenbank von "ansi_system" vorhanden sind.

```
sqlTables(ch)
```

Um die in 6.1.1 vorgestellte SQL-Syntax von R aus zu verwenden, kann der Befehl `sqlQuery` verwendet werden, z. B.

```
sqlQuery(ch, "SELECT datum, aussentemperatur FROM theresienwetter LIMIT 6")
```

Über diese Funktion können im Übrigen nicht nur SQL-Abfragen an die Datenbank gesendet werden, es können damit auch andere SQL-Befehle, wie SHOW TABLES oder CREATE TABLE etc. ausgeführt werden. Der Strichpunkt am Ende wird hier nicht benötigt.

```
sqlQuery(ch, "SELECT datum, aussentemperatur FROM theresienwetter LIMIT 6")
```

Da die Funktion `sqlQuery` bei der Umwandlung von Spalten, welche in der Datenbank im DATETIME-Format gespeichert sind, in das POSIXct-Format, welches R verwendet, die global eingestellte Zeitzone mit Sommer-/Winterzeit (CEST bzw. CET) verwendet und deren Verhalten bei Vergleichen und durch automatische Zeitverschiebungen bei bestimmten Operationen schwer zu kontrollieren ist, wurde die Funktion *sqlQueryGMT* geschrieben. Hier werden alle Variablen, die beim Einlesen in das POSIXct-Format konvertiert wurden wieder auf "GMT" (siehe as.POSIXct-Argument tz) gesetzt, was der UTC (Universal Time, Coordinated), also der koordinierten Weltzeit entspricht. Hier ist das Verhalten besser vorhersehbar, man muss aber darauf achten, dass bei einigen Operationen wieder eine Umwandlung in die gegenwärtige Zeitzone stattfindet. Ansonsten kann diese Funktion analog zu *sqlQuery* verwendet werden.

```
sqlQueryGMT(ch, "SELECT datum, aussentemperatur FROM theresienwetter LIMIT 6")
```

```
# Vergleich des Datumsformates:
source('Code_Daten/datenbank_funktionen.R')
sqlQuery(ch, "SELECT datum FROM theresienwetter LIMIT 6")[,]
sqlQueryGMT(ch, "SELECT datum FROM theresienwetter LIMIT 6")[,]
```

Durch `[,]` wird der einspaltige data.frame in einen Vektor umgewandelt und damit die Zeitzone angezeigt.

Eine Alternative dazu ist eine angepasste Variante von `sqlQuery` bzw. der davon verwendeten Funktion `sqlGetResults`. Hier können zusätzlich die Argumente `format` und `tz` angegeben werden, welche an die Funktion `sqlGetResults2` übergeben werden und direkt bei der Umwandlung in das POSICXct-Format die Zeitzone und das Datumsformat

festlegen. Die Default-Werte sind `"GMT"` (koordinierte Weltzeit) und `"%Y-%m-%d %H:%M:%S"`. Somit wird zusätzlich auch sichergestellt, dass beim Umwandeln das Datum nicht abgeschnitten wird, was manchmal bei `sqlQuery` passiert. Die Funktion `sqlQuery2` dürfte die bessere Lösung darstellen und wird an dieser Stelle empfohlen, `sqlQueryGMT` ist allerdings noch in einigen Funktionen enthalten.

```
source('Code_Daten/sqlQuery2_vgl_RODBC.R')
sqlQuery2(ch, "SELECT datum, aussentemperatur FROM theresienwetter LIMIT 6")
sqlQuery2(ch, "SELECT datum FROM theresienwetter LIMIT 6")[,]
```

Um einen data.frame aus dem RAM in eine MySQL-Tabelle zu schreiben, kann der Befehl `sqlSave` verwendet werden.

```
df <- data.frame(a = 1:10, b = letters[1:10])
sqlSave(ch, dat = df, tablename = "tabelle_df")
```

Hiermit wird der erzeugte Dataframe *df* in die Tabelle *tabelle_df* geschrieben. Für EnMoLMU sind weiterhin die Argumente `append, rownames, safer` und `varTypes` (siehe Hilfe) von Bedeutung.

Wieder gelöscht werden kann die Tabelle schließlich mit

```
sqlDrop(ch, sqtable = "tabelle_df")
```

6.1.5 Sicherungskopie einer Tabelle erstellen (Datenbankdump)

Um eine Datensicherung einer Datenbank oder einzelner darin enthaltener Tabellen zur erstellen, oder um diese Tabellen in eine andere Datenbank zu kopieren, kann ein sogenannter Datenbankdump erstellt werden.

Um einen solchen Datenbankdump zu erstellen, öffnet man die Windows Eingabeaufforderung (nicht MySQL 5.6 Command Line Client!) unter **Start - Alle Programme - Zubehör - Eingabeaufforderung**, bzw. indem man unter **Start - Programme/Dateien durchsuchen** `"Eingabeaufforderung"` eintippt, und gibt dort den nach eigenen Bedürfnissen angepassten folgenden Code ein.

Hinweis: In die Zwischenablage kopierter Text kann hier nur über das Kontextmenü bzw. **Rechtsklick** auf die Titelleiste und **Bearbeiten**, nicht aber über die Tastenkombination `Strg+V` eingefügt werden.

```
"C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqldump.exe" -u root -p""
testdb glt_hist_leo03 >
"D:\Arbeitsverzeichnis\Masterarbeit\Datentests\datensicherungen\korrekt
2015_03_12\glt_hist_leo03update1_dump.sql"

"C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqldump.exe" -u root -p****
enmolmu leo03h2 oet67h2 >
"C:\Users\Rottner\Documents\Datentests\datensicherungen\
20150303_dump_leo03h2_oet67h2.sql"
```

In den ersten Anführungszeichen befindet sich der Pfad zu mysqldump.exe (dürfte immer so wie oben sein, es sei denn, es handelt sich um eine andere Version von MySQL Server), root ist der Benutzername und nach -p kommt (ohne Leerzeichen!) das dazugehörige Passwort (wenn das Passwort aus 0 Zeichen besteht, so kann dies durch "" angegeben werden). *testdb* ist der Name der Datenbank, in welcher sich die Tabelle *glt_hist_leo03* befindet und in den letzten Anführungszeichen ist schließlich der Pfad und Dateiname des zu schreibenden Dumps. Die Dateiendung eines Datenbankdumps schließlich lautet ".sql".

Sollen mehrere Tabellen gesichert werden, so können diese nacheinander durch ein Leerzeichen getrennt (kein Komma) aufgezählt werden. Diese werden dann in einer einzigen Dump-Datei gesichert. Wird keine Tabelle angegeben, so wird die gesamte Datenbank in den Datenbankdump geschrieben. Um später einzelne Tabellen möglichst einfach auswählen und in die Datenbank zurückkopieren zu können, ist es für die EnMoLMU-Datenbank vermutlich am sinnvollsten die Tabellen einzeln, also in je eine Dumpdatei pro Tabelle zu sichern.

Das Zurückkopieren (restore) des Dumps in die Datenbank geht, indem folgendes in die Eingabeaufforderung getippt wird (und kann mit Strg+C abgebrochen werden):

```
"C:\Program Files\MySQL\MySQL Server 5.6\bin\mysql.exe" -u root -p**** enmolmu <
"C:\Users\Rottner\Documents\Datentests\datensicherungen\von
```

```
Daheim\20150303_dump_the46_alle_zusatzspalten.sql"
```

In den ersten Anführungszeichen steht hier der Pfad zu `mysql.exe` (nicht `mysqldump.exe`!), root ist wieder der Benutzername und auf das -p folgt wieder ohne Leerzeichen das Passwort dafür. Auf die Datenbank folgt diesmal ein "Kleiner-als"-Zeichen (also die Spitze zeigt zur Datenbank) und das Ende bildet wieder der Pfad zur Dump-Datei. Die angegebene Datenbank muss nicht dieselbe sein aus welcher der Datenbankdump ursprünglich erstellt wurde.

Ist der Pfad zu MySQL Server in den Systemumgebungsvariablen eingetragen (siehe 6.2), so kann in der Eingabeaufforderung und in Batch-Skripten statt des Pfades zu `mysqldump.exe` auch der Befehl `mysqldump` verwendet werden. Das folgende Beispiel zeigt ein Batch-Skript der Serverinstallation von EnMoLMU, welches eine jeweils eine Sicherung der Tabellen *leo03, oet67, the46* und *theresienwetter* aus der Datenbank *enmo1* im Ordner `D:\\EnMoLMU\\Daten\\Sicherungen` erstellt. Dabei wird auch der vom localhost abweichende Port **** angegeben und das aktuelle Datum an den Dateinamen der Kopien angehängt.

```
rem Dieses Skript erstellt eine Sicherung der Datenbanktabellen (dump)

mysqldump -u root -p**** --port=**** enmo1 leo03 >
"D:\EnMoLMU\Daten\Sicherungen\leo03_%date%.sql"
mysqldump -u root -p**** --port=**** enmo1 oet67 >
"D:\EnMoLMU\Daten\Sicherungen\oet67_%date%.sql"
mysqldump -u root -p**** --port=**** enmo1 the46 >
"D:\EnMoLMU\Daten\Sicherungen\the46_%date%.sql"
mysqldump -u root -p**** --port=**** enmo1 theresienwetter >
"D:\EnMoLMU\Daten\Sicherungen\theresienwetter_%date%.sql"
pause
```

6.2 Batch (.bat)

Ein Batch-Skript ist eine Code-Text-Datei mit der Endung .bat und die darin enthaltenen Befehle werden beim Öffnen (z. B. durch Doppelklick) ausgeführt. Zum Ausprobieren

der einzelnen Befehle können diese auch in die Eingabeaufforderung eingetippt werden.

```
"C:\Program Files\R\R-3.1.2\bin\Rscript.exe" "C:\Users\Rottner\Documents\EnMoLMU\
Code_Berichte\test.R"
```

In den ersten Anführungszeichen steht der Pfad zur Software, also in diesem Fall R (bzw. Rscript), mit der der Code aus dem zweiten Pfad ausgeführt werden soll. Befindet sich der erste Pfad `"C:\Program Files\R\R-3.1.2\bin"` (ohne `Rscript.exe` am Ende) in den Systemvariablen unter `PATH`, so kann dieser Teil im Batch-Skript durch `R CMD BATCH` ersetzt werden, also z. B.:

```
R CMD BATCH "C:\Users\Rottner\Documents\EnMoLMU\Code_Berichte\test.R"
```

Die **Systemvariablen** können unter **Systemsteuerung - Alle Systemsteuerungselemente - System - Erweiterte Systemeinstellungen - Umgebungsvariablen** bei Systemvariablen durch Markieren der Variable `Path` und Klick auf Bearbeiten ergänzt werden. Die einzelnen Einträge werden durch ein Semikolon getrennt und es darf kein Leerzeichen zwischen dem Semikolon und dem nächsten Pfad stehen.

Um Fehler zu finden, kann es sinnvoll sein, am Ende eines Batch-Skriptes den Befehl `pause` einzufügen, damit das Fenster nicht nach der Ausführung automatisch geschlossen wird.

6.3 Aufgabenplanung (Task Scheduler)

Um in Windows ein R-Skript zu einem festgelegten Zeitpunkt einmalig oder regelmäßig automatisch zu starten, kann der Task Scheduler verwendet werden, der dann eine .bat-Datei ausführt, welche wiederum das R-Skript aufruft (siehe 6.2).

Um ihn zu öffen, klickt man auf das **Windows-Startmenü-Symbol** und gibt in die Suchleiste `"Aufgabenplanung"` (bzw. falls ein englisches Betriebssystem verwendet wird `"Task Scheduler"`) ein; oder über **Systemsteuerung > System und Sicherheit > Verwaltung > Aufgabenplanung**.

Über "Einfache Aufgabe Erstellen" kann dann ein Name und eine Beschreibung für die Aufgaben eingegeben werden, Ausführungshäufigkeit und -zeit angegeben werden und mit "Programm starten" die auszuführenden .bat-Datei festgelegt werden.

Eine detailliertere Anleitung befindet sich z. B. unter
`http://praxistipps.chip.de/windows-task-scheduler-aufgaben-in-windows-erstellen_28308`.

6.4 R

Der Hauptteil von EnMoLMU ist in der Programmiersprache R[27] (auch GNU_R oder R-Project) implementiert.

Eine allgemeine Einführung kann im Rahmen dieser Arbeit nicht gegeben werden, allerdings gibt es dazu eine Reihe von entsprechenden Kursen im Internet. An dieser Stelle sei hier "Einführung in das statistische Programmpaket R" (Christian Heumann, 2006, `http://www.statistik.lmu.de/~chris/Rkurs/Rkurs.pdf`) als eine Einführung, die relative umfassend ist, aber dennoch zügig durchgearbeitet werden kann, empfohlen.

6.4.1 Zeitzone für das POSIXct-Format

Die in R verfügbaren Klassen für Datum mit Uhrzeit heißen POSIXct und POSIXlt, wobei die in dieser Arbeit verwendete Klasse POSIXct die (auch negative) Abweichung in Sekunden seit dem 1. Januar 1970 angibt. Somit entspricht der 1. Januar 1970 0:00:00 Uhr dem numerische Wert 0.

```
as.numeric(as.POSIXct("1970-01-01 0:00:00", tz="GMT"))
[1] 0
```

Die POSIXct-Klasse enthält zudem eine Zeitzone, die über eine Verschiebung der Zeitzone von der koordinierten Weltzeit und Sommer-/Winterzeit enthalten ist.

"Die koordinierte Weltzeit (englisch Coordinated Universal Time, französisch Temps universel coordonné), kurz UTC, ist die heute gültige Weltzeit. Eingeführt wurde sie 1972. Aus einer Zeitangabe in UTC ergibt sich die entsprechende, in Deutschland, Österreich

und anderen mitteleuropäischen Staaten geltende Mitteleuropäische Zeit (MEZ), indem man eine Stunde, und die im Sommer geltende Mitteleuropäische Sommerzeit (MESZ), indem man zwei Stunden addiert."[15]

Um Komplikationen und schwer vorhersehbare automatische Umwandlungen und doppelt bzw. nicht vorkommende Uhrzeiten (z. B. CEST-CET 25.10.2015 zweimal 2.00 Uhr und keinmal 29.03.2015 2.00 Uhr) zu vermeiden, wurde für diese Arbeit ausschließlich die Zeitzone koordinierte Weltzeit verwendet. Sie kann über das Argument `tz = "GMT"` festgelegt werden.

Wird keine Zeitzone angegeben, wird die vom System vorgegebene Zeitzone, also CET (Central European Time) bzw. CEST (Sommerzeit) verwendet. Darüber hinaus ist größte Vorsicht geboten, da viele Befehle zu einer automatischen Anpassung der Zeitzone führen, z. B.

```
a <-as.POSIXct("1970-01-01 0:00:00", tz="GMT")
a
[1] "1970-01-01 GMT"
c(a)
[1] "1970-01-01 01:00:00 CET"
```

6.4.2 Kodierung der R-Dateien

Die für alle R-Dateien verwendete Kodierung ist UTF-8. Sie wurde in R-Studio unter **Tools - Project Options - Code Editing - Text** encoding: UTF-8 voreingestellt. Siehe auch **File - Save with Encoding** bzw. **Reopen with Endcoding**. Diese Kodierung schien insbesondere für die Verwendung von R Markdown geeignet.

Da Batch-Dateien die von ihnen aufgerufenen R-Skripte als ISO-8859-1 öffnen, sind alle R-Skripte, die von diesen Dateien aufgerufen werden, sollen in diesem Format gespeichert. Diese können in RStudio über **File - Reopen with Encoding** mit intakten Umlauten geöffnet werden.

6.4.3 Pakete und Dokumentation als PDF erstellen

Der Hauptgrund, warum für EnMoLMU der Prozess der R-Paketerstellung verwendet wurde, ist die dabei enthaltene Möglichkeit, die Paketdokumentation im PDF automatisch aus der Funktionsdokumentation in den Code-Dateien zu erstellen, wodurch das zeitaufwendige manuelle Abtippen und Formatieren in einer separaten Datei und die damit verbundenen Synchronisationsschwierigkeiten vermieden werden können.

Das Erstellen von Paketen bietet darüber hinaus die Vorteile des leichteren Ladens der Funktionen und der Hilfefunktion über die Console. Der vorgegebene Aufbau der Pakete erleichtert es schließlich, den Überblick über die einzelnen Funktionen zu bewahren.

Die folgende Kurzanleitung zum Erstellen von R-Paketen geht insbesondere auf die Dinge ein, die der Verfasser für EnMoLMU für am wichtigsten hält. Eine detailliertere Anleitung kann unter `http://r-pkgs.had.co.nz/man.html` gefunden werden.

Ein R-Paket beginnen

Ein R-Paket ist im Grunde ein Ordner mit dem Namen, den dieses Paket hat, und einem Mindestinhalt. Um ein verwendbares Paket erstellen zu können, müssen in diesem Ordner die Textdateien DESCRIPTION und NAMESPACE (ohne Dateiendung) und die Ordner man und R enthalten sein. Dieser Aufbau wird automatisch durch die Funktion `create` erstellt:

```
library(devtools)

create("Pakete/testpaket")
```

R-Code einfügen und dokumentieren

Nun können in den Ordner R die R-Code-Dateien kopiert werden. Deren Kodierung sollte UTF-8 sein, falls Umlaute etc. verwendet werden. Da die Dokumentation hier mithilfe des Paketes *roxygen2* erstellt werden soll, muss die Dokumentation der Funktionen in diesen Dateien entsprechend formatiert sein, z. B.:

```
#' Entfernen von Spalten mit nur NA
#'
#' Entfernt aus data.frame oder matrix alle Spalten mit
#' ausschließlich fehlenden Werten.
#'
#' @param x data.frame (oder matrix), der überprüft werden soll.
#' @param nur_index Falls TRUE gibt die Funktion nur den Index (logical)
#' der Spalten, die nicht ausschließlich NA sind zurück,
#' ansonsten den gesamten Datensatz mit nur diesen Spalten.
#' @return Logical-Vektor oder data.frame/matrix.
#' @examples
#' na_spalten_entfernen(data.frame(a = c(NA, NA), b = 1:2))
#' na_spalten_entfernen(data.frame(a = c(NA, NA), b = 1:2), nur_index = TRUE)
#' @export

na_spalten_entfernen <- function(x, nur_index = FALSE){
  nur_na <- apply(x, 2, FUN = function(y)all(is.na(y)))
  if(nur_index == TRUE) return(!nur_na)
  x[, !nur_na, drop = FALSE]
}
```

Die Funktionsdokumentation steht vor der Funktion und alle dazugehörigen Zeilen beginnen mit #'. Die erste Zeile ist der Titel. Dieser darf keine Umlaute etc. enthalten (also u.a. kein ä, ö, ü, ß), da er sonst in der PDF-Dokumentation nur als NA angezeigt wird. Darauf muss eine leere Zeile folgen und im nächsten Absatz steht die Beschreibung der Funktion, welche Umlaute enthalten darf. Mithilfe des Tags `@param` werden die Funktionsargumente in der Form `@param Argumentname Dies Argument ist das.` beschrieben, wobei der Argumentname durch ein Leerzeichen von der Argument-Beschreibung getrennt wird. Falls mehrere Argumente auf einmal beschrieben werden, so können sie durch Komma getrennt und ohne Leerzeichen aufgezählt werden (z. B. `@param x,y Koordinaten von irgendwas.`) Zeilenumbrüche werden ignoriert und können somit nach belieben eingefügt werden. @return kennzeichnet die Beschreibung des Rückgabewertes und `@examples` Beispielaufrufe. Um zu verhindern, dass die Beispiele z. B. bei einem check des Paketes automatisch ausgeführt werden, kann nach `@examples \dontrun{<Beispielaufruf>}` verwendet werden, was insbesondere dann empfehlenswert ist, wenn die Funktion Daten

auf der Festplatte verändert oder die Ausführung des Beispiels lange dauert. `@export` schließlich sorgt dafür, dass die Funktion nach dem Laden des Paketes direkt vom Anwender aufgerufen werden kann (anstatt nur durch Funktionen des Paketes bzw. über `Paketname:::Funktionsname`). Der Funktionscode darf wieder keine Nicht-ASCII-Zeichen enthalten.

Mit dem `@importFrom`-Tag können Funktionen aus anderen Paketen (in der Form `@importFrom Paketname Funktionsname`) angegeben werden, die von der eigenen Funktion aufgerufen werden, z. B.

```
...
@importFrom sqldf sqldf

addiere_mit_sql <- function(x,y){
sqldf(paste("select",x,"+",y))
}
```

In diesem Beispiel wird aus dem Paket *sqldf* (erstes Wort nach `@importFrom`) der Befehl `sqldf` (zweites Wort) importiert, um es in der eigenen Funktion `addiere_mit_sql` verwenden zu können. Alternativ kann das Paket direkt beim Aufruf der Funktion angegeben werden (`Paketname::Funktionsname`):

```
...

addiere_mit_sql <- function(x,y){
sqldf::sqldf(paste("select",x,"+",y))
}
```

Die Verwendung von `::` wird vom Verfasser der vorliegenden Arbeit empfohlen.

Die verwendeten Pakete müssen außerdem im der Datei DESCRIPTION unter Imports: angegeben werden (siehe 6.4.3).

Weitere Informationen zur Dokumentation gibt es unter `http://cran.r-project.org/web/packages/roxygen2/vignettes/rd.html` und zur Formatierung unter `http://cran.r-project.org/web/packages/roxygen2/vignettes/formatting.html`.

Die Datei DESCRIPTION

Die Datei `DESCRIPTION` wird automatisch durch den Aufruf von `create` erstellt und muss von Hand ergänzt werden, wie anhand des folgenden Beispiels gezeigt werden soll.

```
Package: testpaket
Title: Funktionen zum Testen des Paketerstellungs-Vorganges
Version: 0.1
Authors@R: "Thomas Rottner <Thomas.Rottner@Verwaltung.Uni-Muenchen.DE> [aut, cre]"
Description: Funktionen, mit denen das Erstellen von R-Paketen ausprobiert wurde.
    Diese Beschreibung ist einen Absatz lang und ab der 2. Zeile 4 Zeichen
    eingerückt.
Depends: R (>= 3.1.2)
License: What license is it under?
LazyData: true
Encoding: UTF-8
Imports:
  RODBC,
  plyr,
  reshape2
```

`Title`, `Version`, `Authors`, `Description` und `License` wird bereits als Vorlage eingefügt und muss noch manuell ausgefüllt werden. `Encoding` (dieselbe Kodierung, wie die R-Code-Dateien, i. d. R. UTF-8 für deutschsprachige Inhalte) und `Imports` (alle Pakete, die von den Funktionen im R-Code verwendet werden) wurden von Hand ergänzt. Nach der letzten Zeile muss eine leere Zeile stehen.

Imports eignet sich, um die Pakete anzugeben, die von einem R-Paket verwendet werden. **Depends** wird z. B. vom R-Paket *EnMoLMU* verwendet, um die anderen EnMoLMU-Pakete zu laden und alle darin enthaltenen Funktionen für den Nutzer verfügbar zu machen. Einen Startpunkt für die Suche nach weiteren Informationen zum Unterschied zwischen `Depends` und `Imports` bietet die Diskussion `http://stackoverflow.com/questions/8637993/better-explanation-of-when-to-use-imports-depends`.

Da der Paketordner nun die notwendigen Inhalte besitzt, können die Dokumentation und das Paket erstellt werden.

Die Paket-Dokumentations-PDF-Datei erstellen

Die einfachste Möglichkeit, das Dokumentations-PDF zu erstellen, ist mithilfe des Befehls `docu_pdf` aus dem Paket *enmoSonstiges* (welches Teil der EnMoLMU-Software ist):

```
library(enmoSonstiges)
docu_pdf("R Pakete/testpaket")
```

Zuerst wird das R-Paket *enmoSonstiges* geladen (Installieren des Paketes siehe 6.4.3). Der Befehl `docu_pdf` erstellt die Dokumentationsdatei des Pakets *testpaket*, das sich im Ordner R Pakete befindet. Wenn die Dokumentation ein weiteres Mal erstellt wird, muss die Datei vorher geschlossen werden, da sie sonst nicht überschrieben werden kann. Die alte Datei wird von `docu_pdf` automatisch gelöscht.

Wenn das Paket *enmoSonstiges* gerade nicht verfügbar ist, bleibt noch die Alternative, die einzelnen Schritte von Hand auszuführen, indem in die R-Console

```
library(roxygen2)
document("R Pakete/testpaket")
```

eingegeben wird, wodurch im Paket-Ordner man .Rd-Dateien erstellt werden, und dann in der **Eingabeaufforderung** von Windows Folgendes ausgeführt wird, wodurch daraus die PDF-Datei im Ordner R Pakete (durch `cd` festgelegtes Arbeitsverzeichnis in der Eingabeaufforderung) erstellt wird:

```
cd "C:\R Pakete"
R CMD Rd2pdf --encoding=UTF-8 testpaket
```

Hier muss die PDF-Datei vor jedem weiteren Erstellen manuell gelöscht werden.

R-Pakete bauen und installieren

Um ein R-Paket zu testen, kann es mit dem Befehl `load_all` aus dem Paket `devtools` geladen werden, ohne dass es vorher gebaut wurde, also eine Simulation des Bauens und Ladens des Pakets.

```
library(devtools)
load_all("R Pakete/testpaket")

# überprüfen:
dev_help("Funktion1") # Hilfe zur Funktion Funktion1
dev_example("Funktion1") # Beispiel zur Funktion Funktion1
help(package = "testpaket") # Hilfe zum Paket testpaket
unload("R Pakete/testpaket") # Paket wieder "entladen"

check("R Pakete/testpaket1") # Überprüfung auf viele Fehler und Konventionen
```

Soll das Paket dann schließlich gebaut werden, sodass es installiert werden kann, geht man wie folgt vor, wobei `document` (bzw. `docu_pdf`) immer vorher ausgeführt werden sollte, da dadurch die Datei `NAMESPACE` aktualisiert wird.

```
# bauen
build("R Pakete/testpaket")

# installieren:
install.packages("R Pakete/testpaket1_0.0.0.9000.tar.gz", repos = NULL,
                 type = "source")
# in RStudio alternativ über Tools - Install Packages -
# Install From: Package Archive File (.zip; .tar.gz) - Browse ...

# laden des installierten Pakets:
library(testpaket)
help(package="testpaket")
# Auflistung aller Funktionen im Paket mit Title und Link zur Hilfe
?Funktion1
```

```
# Hilfe zu Funktion1

# Paket wieder "entladen":
detach(package:enmoDaten)

# Paket deinstallieren:
remove.packages("testpaket")
```

Weitere Informationen zum Bauen eines Pakets gibt es unter `https://support.rstudio.com/hc/en-us/articles/200486508-Building-Testing-and-Distributing-Packages`.

Fehlersuche

Sollte es beim Erstellen der Dokumentation oder beim Bauen des Pakets zu einem Fehler kommen und die Fehlermeldung keinen besseren Hinweis geben, so kann damit begonnen werden, die folgenden Punkte zu überprüfen:

- \und % in der Funktionsdokumentation als Escape-Sequenzen schreiben \und \%, @ mit @@ (auch in \code etc.)
- { muss mit Escape-Sequenz \{ geschrieben werden (gleiches gilt natürlich auch für }).
- kein ä,ö,ü,ß, etc. im ersten Absatz (Titel) der Funktionsdokumentation
- in die `DESCRIPTION`-Datei `Encoding: UTF-8` schreiben
- eventuell in der Funktionsdokumentation das Tag `@encoding UTF-8` verwenden
- entweder z. B. `@importFrom devtools document` in die Funktionsdokumentation oder `devtools::document()` beim Aufruf im Funktionscode, falls eine Funktion aus einem anderen Paket im Code verwendet wird (natürlich müssen hier die Namen des Pakets und der Funktion verwenden, die verwendet werden. `devtools` und `document` ist nur ein Beispiel)
- besser `@examples` und das Beispiel in der Funktionsdokumentation in eine neue Zeile als `@example`
- `\dontrun{}` nach `@examples` in eine neue Zeile

- keine ä, ö, ü, ß im Funktionscode verwenden, auch nicht in den Argument-Defaults. Falls nötig, können diese über ASCII-Escapes (z. B. `\u00A3`) eingebunden werden (siehe 6.4.3)
- in der `DESCRIPTION`-Datei muss in der letzten Zeile eine Leerzeile stehen
- manchmal benötigt es auch einfach nur einen Neustart von R

Umlaute im Funktionscode

Werden Umlaute und andere Sonderzeichen in den Funktionscode eingebunden, so können diese zwar verwendet werden, wenn der Code gesourced wird, aber in einem R-Paket werden sie durch andere Zeichen ersetzt (z. B. ÄusreiÃŸer"). Um dies zu verhindern, können sie zumindest innerhalb von Strings durch ASCII-Escapes ersetzt werden. Ein kurzes Beispiel soll dies verdeutlichen. Weitere Informationen dazu gibt es unter `http://cran.r-project.org/doc/manuals/R-exts.html#Encoding-issues`.

```
# Escape abfragen:
charToRaw("ß")
[1] df
> charToRaw("ä")
[1] e4
# oder mit iconv:
> iconv("ß", from = "UTF-8", to= "ASCII", "byte")
[1] "<df>"

# In die Funktion einbauen:
umlaut1 <- function(){print("Ausrei\u00dfer")}
umlaut1()
[1] "Ausreißer"

# andere Möglichkeit:
umlaut2 <- function(){cat(paste("gl", paste0("\u00e4"), "nzend", sep=""))}
umlaut2()
glänzend
```

Dem Paket Daten beifügen

Jedem R-Paket können auch Daten beigefügt werden. Dazu sollen an dieser Stelle zwei Möglichkeiten gezeigt werden, eine detailliertere Abhandlung zu diesem Thema kann unter `http://r-pkgs.had.co.nz/data.html` gefunden werden.

Daten in den Code integrieren Die einfachste und insbesondere für sehr kleine Datenobjekte geeignete Methode ist es, einfach die Daten als R-Code in die Source-Dateien zu integrieren. Dies kann entweder manuell geschehen, z. B. `daten <- c(1,5,3)` oder bei komplizierteren Objekten mit Hilfe des Befehls `dput`. Im folgenden Beispiel werden Hierzu die historischen Wetterdaten verwendet, welche in *enmoFamos* enthalten sind.

```
# Laden des Pakets mit den Daten
library(enmoFamos)
head(wetter_hist)

# erzeugen des Objektes im Textformat (die ersten 10 Zeilen des Datensatzes)
dput(wetter_hist[1:10,])

structure(list(Datum = structure(c(15706, 15706, 15706, 15706,
15706, 15706, 15706, 15706, 15706, 15706), class = "Date"), Stunde = 1:10,
    Tag = c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L), Monat = c(1L,
    1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L), Jahr = c(2013L, 2013L,
    2013L, 2013L, 2013L, 2013L, 2013L, 2013L, 2013L, 2013L),
    DIFFUSSTRAHLUNG = c(0, 0, 0, 0, 0, 0, 0, 0, 29.6, 97.4),
    GLOBALSTRAHLUNG = c(0, 0, 0, 0, 0, 0, 0, 0, 33.3, 101.9),
    LUFTTEMPERATUR = c(1.9, 1.9, 1.9, 1.7, 1.3, 1.2, 0.7, 0.7,
    1.7, 2.5), NIEDERSCHLAG = c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
    RELATIVE_FEUCHTE = c(70.4, 70.4, 68.9, 68.6, 71.1, 71, 73.6,
    73.6, 68.6, 66.7), Datum2 = structure(c(1357002000, 1357005600,
    1357009200, 1357012800, 1357016400, 1357020000, 1357023600,
    1357027200, 1357030800, 1357034400), class = c("POSIXct",
    "POSIXt"), tzone = "GMT")), .Names = c("Datum", "Stunde",
```

```
"Tag", "Monat", "Jahr", "DIFFUSSTRAHLUNG", "GLOBALSTRAHLUNG",
"LUFTTEMPERATUR", "NIEDERSCHLAG", "RELATIVE_FEUCHTE", "Datum2"
), row.names = c("1", "2", "3", "4", "5", "6", "7", "8", "9",
"10"), class = "data.frame")
```

Binäre Daten in data/ Dies wird für größere Datensätze, wie z. B. den kompletten wetter_hist-Datensatz schnell sehr unübersichtlich. Hierfür bietet es sich an, im Ordner des entsprechenden R-Pakets einen Unterordner data anzulegen und den Datensatz darin im RDA-Format zu speichern. RDA-Dateien sind binäre Dateien, die alle R-Objekte enhalten können, sehr kompakt sind und sehr schnell eingelesen werden können. Der bereits im Workspace befindliche data.frame `wetter_hist` kann z. B. folgendermaßen als RDA-Datei im Ordner data des R-Pakets *enmoFamos* gespeichert werden. Eine RDA-Datei kann zwar mehrere R-Objekte enthalten, für diesen Zweck sollte sich jedoch immer nur ein Objekt in einer RDA-Datei befinden.

```
devtools::use_data(wetter_hist, pkg="R Pakete/enmoFamos")
```

Die Beschreibung zu den Daten kommt in den Ordner R des R-Pakets als R-Datei, z. B. wetter_hist.R:

```
#' historische Wetterdaten des Meteorologischen Instituts Theresienstr 2013 und 2014
#'
#' Historische Wetterdaten des Meteorologischen Instituts Theresienstr 2013 und
#' 2014 (1.1.2013 bis 1.9.2014) für die Gradtagzahlbereinigung
#' in \code{zaehlervergleich}. Stündliche Werte.
#'
#' @format Ein data.frame mit 14582 Zeilen und 11 Variablen:
#' \describe{
#'   \item{Datum}{Datum als Date (nur ganze Tage)}
#'   \item{Stunde,Tag,Monat,Jahr}{Datumsbestandteile als einzelne Integerwerte}
#'   \item{DIFFUSSTRAHLUNG}{DIFFUSSTRAHLUNG}
#'   \item{GLOBALSTRAHLUNG}{GLOBALSTRAHLUNG}
#'   \item{LUFTTEMPERATUR}{LUFTTEMPERATUR}
#'   \item{NIEDERSCHLAG}{NIEDERSCHLAG}
```

```
#'   \item{RELATIVE_FEUCHTE}{RELATIVE_FEUCHTE}
#'   \item{Datum2}{Datum als POSIXct (inkl. Uhrzeit)}
#' }
#' @source Meteorologisches Institut der LMU.
"wetter_hist"
```

Der Inhalt endet mit dem Namen des Datensatzes in Anführungszeichen. Das Beispiel stammt aus *enmoFamos*.

Sobald das Paket geladen ist, kann der Datensatz verwendet werden. Für den Fall, dass sich ein anderes Objekt mit demselben Namen im Speicher befinden kann, ist es möglich, mit Paketname::Datensatzname (z. B. `enmoDaten::wetter_hist`) sicherzustellen, dass auf den Datensatz aus dem Paket zugegriffen wird.

6.5 R Markdown

Die Berichte werden aus Vorlagen erstellt, in denen die Aufrufe für die einzelnen Grafiken und Text enthalten sind. Diese Vorlagen sind Dateien im R Markdown-Format. Unter `http://rmarkdown.rstudio.com/authoring_basics.html` befindet sich eine Übersicht über die wichtigsten Syntax-Elemente von R Markdown und unter `http://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf` ein "R Markdown Cheat Sheet". Neben den Überschriften wird für die Verwendung von EnMoLMU vor allem das Einbetten von R-Code benötigt. Dieser beginnt immer mit ```` ```{r} ```` und endet mit ```` ``` ````. Der erste R-Code-Block, der in der Vorlage enthalten sein muss, dient dem Laden der benötigten R-Pakete (i. d. R. *EnMoLMU* und *RODBC*) und dem Öffnen einer RODBC-Verbindung, damit die Grafikfunktionen die Daten aus der Datenbank abfragen können.

````
```{r, echo=FALSE, error=FALSE, warning=FALSE, include=FALSE}
library(EnMoLMU)
library(RODBC)

# ODBC-Verbindung zu MySQL öffnen:
ch <- odbcConnect("ansi_system",  DBMSencoding = "utf-8")
```
````

Die weiteren Code-Blöcke beinhalten dann die Grafiken und sonstigen Auswertungen, die in dem jeweiligen Bericht enthalten sein sollen, z. B.

```
```{r, echo=FALSE}
abschnittsweises_modell_db(ch, "glt_hist_oet67",
   "diff_x7070_01_u138hzg03wmz0001zw01",
   "x7070_01_u138hzg03tem0003mp01", "'2011-01-01' and '2015-01-01'",
   "'2015-01-02' and '2016-01-01'",
   bereinigung = list(untere_grenze = 0+1e-100,
                      quantil_box = c(0.1, 0.9),
                      faktor_oben = 10))

```
```

Vor, nach und zwischen den R-Code-Blöcken können Titel, Überschriften und Erklärungen zu den grafischen Darstellungsarten etc. eingefügt werden.

Das aktuelle Datum mit Uhrzeit der Berichterstellung kann über den innerhalb einer Zeile (inline) eingebetteten R-Code `` `r Sys.time()` `` werden. Soll das Datum der Berichterstellung anstatt dem Datum der Vorlagenerstellung im Titelabschnitt eingefügt werden, so muss der inline-Code zusätzlich in Anführungszeichen stehen:

```
---
title: "bericht1"
author: "Thomas"
date: "`r Sys.time()`"
output: pdf_document
---

# bericht1
```

Über eine oder mehrere Raute(n) (#) kann eine (Unter-)Überschrift erstellt werden.

In RStudio kann über **File - New File - R Markdown...** eine R Markdown-Datei erstellt werden, die bereits eine Vorlage enthält. Über die Schaltflächen Knit PDF etc. kann aus der Vorlage ein PDF-, HTML- oder Word-Dokument erstellt werden.

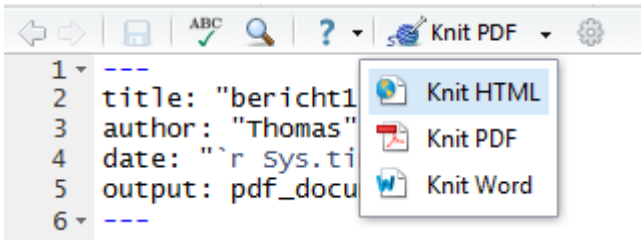


Abbildung 6.1: HTML Datei aus der Vorlage mithilfe von RStudio erstellen.

6.5.1 Besonderheiten bei PDF-Berichten

Sollen die Berichte im PDF-Format erstellt werden, so gilt es eine Reihe von Besonderheiten zu beachten.

Da die Vorlagen erst in das tex-Format übersetzt werden, bevor daraus Berichte im PDF-Format erzeugt werden, wird auch die LaTex-Eigenart, dass Bilder nicht an der Stelle eingefügt werden, wo sie hingehören, sondern, wo gerade Platz ist, bei der Umwandlung umgesetzt. Die Übergabe von `fig.pos="H"` in den R-Code-Block-Argumenten scheint nicht zu funktionieren, somit bleibt nur die Möglichkeit, nach jedem Abschnitt einen Seitenumbruch mit `\pagebreak` einzufügen, so dass zumindest die einzelnen Abschnitte mit den darin enthaltenen Grafiken voneinander getrennt werden.

Bei den R-Code-Block-Bezeichnungen in ```` ```{r Ich_bin_die_Bezeichnung, ... ```` muss darauf geachtet werden, dass sie keine Umlaute enthalten, da sonst kein PDF-Bericht erzeugt wird.

6.6 Pandoc

Zur Umwandlung der Berichtsvorlagen im R Markdown-Format in das Format PDF, DOCX, usw. wird der freie Parser Pandoc verwendet. Weiter Informationen dazu können unter der Webadresse `http://pandoc.org` und eine Anleitung zur Installation unter `http://pandoc.org/installing.html` gefunden werden.

6.7 LibreOffice

Mithilfe des Office-Pakets LibreOffice können zum Beispiel Dateien im ODT- oder DOC-Format in PDF-Dateien konvertiert werden, was unter Umständen nützlich sein kann, da sich manche Berichte am besten im ODT- oder DOC-Format erzeugen lassen. Dies kann auch über die Eingabeaufforderung und somit aus einem Batch- oder R-Skript heraus geschehen. Dazu muss zuerst LibreOffice installiert werden (`https://de.libreoffice.org/download/libreoffice-fresh/?type=win-x86&version=4.3.6&lang=de`) und dann der Pfad zum LibreOffice-Ordner `program` in den Systemvariablen gesetzt werden. Dieser ist normalerweise unter Programme zu finden und kann z. B.
`C:\Program Files\LibreOffice 4\program` lauten.

Danach kann man mit dieser Anweisung in der Eingabeaufforderung die Datei bericht1.odf in eine PDF-Datei konvertieren:

```
rem Eine Datei mit dem aktuellen Arbeitsverzeichnis als Zielordner:
start /wait soffice --headless --convert-to pdf "D:\Arbeitsverzeich
nis\Masterarbeit\enmolmu\schritt fuer schritt testen\enmolmu\Berichte\Berichte_f
ertig\bericht1.odt"

rem Wechsel des Arbeitsverzeichnisses (/d wird benötigt, wenn sich das neue
rem Verzeichnis auf einer anderen Festplatte befindet.
cd /d "D:\Arbeitsverzeichnis\Masterarbeit\enmolmu\schritt fuer schritt testen\
enmolmu\Berichte\Berichte_fertig"
```

7 Ausblick

Mit EnMoLMU in der Version 1.0 wurde eine einsatzfähige Energiemonitoring-Software geschaffen, die alle gegenwärtig interessierenden Daten verarbeiten kann und für größere Datenmengen als die zu erwartende Datenmengenobergrenze ausgelegt ist. Insbesondere ist es mit der Software möglich, die Daten aus den unterschiedlichen Quellen gemeinsam auszuwerten und viele Vorgänge zu automatisieren.

Im nächsten Schritt müssen die übrigen Subserver eingebunden werden, sodass möglichst viele Daten verfügbar sind, um die Software sowohl ausgiebig verwenden (und damit testen) zu können, als auch um für die Weiterentwicklung insbesondere im Hinblick auf eine möglichst ausgereifte und automatische Datenbereinigung über ausreichend Testmöglichkeiten zu verfügen.

Außerdem sollten auch baldmöglichst die Zugriffsrechte der Software auf die Daten am Ablageort der exportierenden Anwendungen eingerichtet werden. Somit kann der Vorgang der Datenaktualisierungen für die eingerichteten Subserver und damit auch die Generierung und Verteilung der Berichte weitgehend automatisiert werden. Von Seiten der EnMoLMU-Software ist dafür bereits alles vorbereitet.

Da noch weitere Anwender hinzukommen werden und bei der Arbeit mit den bereits vorhandenen Analysemöglichkeiten Wünsche nach weiteren oder überarbeiteten Alternativen zu erwarten sind, werden im Laufe der Verwendung noch eine Reihe weiterer Grafiken oder andere Auswertungen einprogrammiert werden müssen.

Wenngleich sie niedrigere Priorität haben, können weitere Möglichkeiten getestet werden, um die Bedienung zu vereinfachen. Z. B. könnten interaktive Grafiken implementiert werden, um die Auswertungen komfortabler verwenden zu können bzw. zur bequemeren Erzeugung der Berichtsvorlagen. So könnten beispielsweise mit dem R-Paket *shiny* die implementierten Grafiken in einer interaktiven Version umgesetzt werden, mit welcher man die Parameter zusammenklicken kann und am Ende den entsprechenden Aufruf

ebenfalls per Klick in die Berichtsvorlagen einfügen kann. Abb. 7.1 und im Anhang Abb. A.1 und Abb. A.2 zeigen einen ersten Test mithilfe von *shiny* implementierten interaktiven Grafiken.

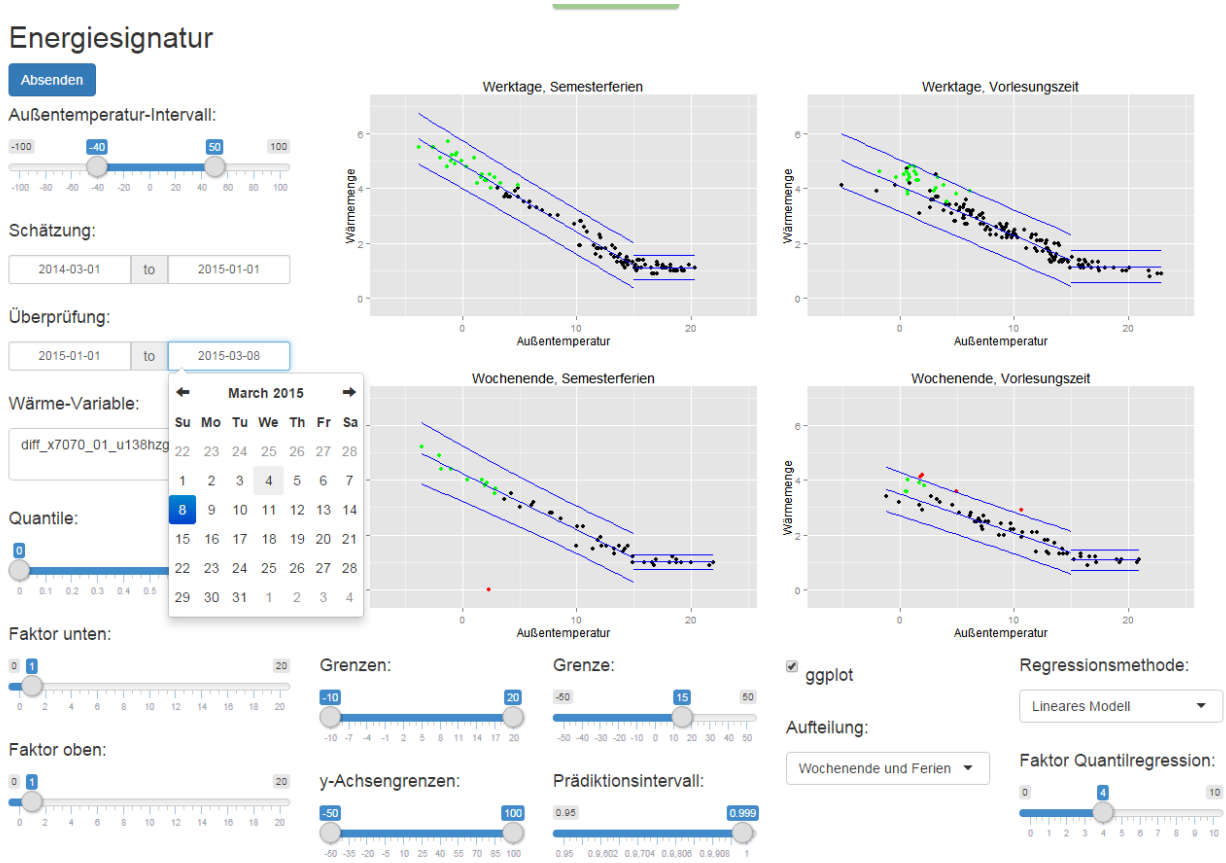


Abbildung 7.1: Interaktive Grafik mit *shiny*: Energiesignatur

Langfristig dürfte es sinnvoll sein, die dafür geeigneten Auswertungen automatisch ablaufen zu lassen und nur im Falle von verdächtigen Daten eine Meldung an die Nutzer senden zu lassen. Dies könnte z. B. per automatisch versandtem E-Mail geschehen. Der E-Mail-Versand könnte z. B. mithilfe des R-Pakets *sendmailR* durchgeführt werden. Die grafische Auswertung Energiesignatur bietet die dazu benötigten Bausteine bereits jetzt weitgehend. So könnten die Modelle täglich per Aufgabenplanung mit den Daten bis einschließlich des Vortages geschätzt werden und nur für den aktuellen Tag überprüft werden, ob sich der Wert innerhalb des Prädiktionsintervalls befindet. Ein sehr ähnliches Prinzip könnte z. B. für den Tagesverlauf umgesetzt werden, wobei allerdings die Globalstrahlung als Einflussgröße gegenwärtig nicht tagesaktuell verfügbar ist. Liegt ein Wert des aktuellen

Tages außerhalb des Prädiktionsintervalls oder weist ein anderes als kritisch definiertes Muster auf, so könnte von der Software die Grafik erstellt werden und per E-Mail an den zuständigen Nutzer zur Überprüfung versandt werden.

A Anhang

A.1 EnMoLMU R-Paket-Manuals

A.1.1 EnMoLMU

Package 'EnMoLMU'

September 7, 2015

Title Energie Monitoring Software der Ludwig-Maximilians-Universitaet Muenchen

Version 1.0

Description Lädt die zur EnMoLMU gehörigen R-Pakete enmoSonstiges, enmoDaten, enmoGrafiken, enmoFamos, enmoBerichte.

Depends enmoSonstiges,
enmoDaten,
enmoGrafiken,
enmoBerichte,
enmoFamos

License What license is it under?

LazyData true

Encoding UTF-8

R topics documented:

`EnMoLMU` *Energiemonitoring der Ludwig-Maximilians-Universitaet Muenchen.*

Description

Das EnMoLMU-Paket enhält selber keine Funktionen, sondern bündelt und lädt die Funktionen aus den R-Paketen enmoDaten, enmoGrafiken, enmoFamos, enmoBerichte und enmoSonstiges, mit denen Berichte zur Überwachung der Verbräuche generiert werden können.

Index

A.1.2 enmoSonstiges

Package 'enmoSonstiges'

September 7, 2015

Title Funktionen, um einige allgemeine Aufgaben zu erleichtern.

Version 1.0

Description Enhält r_source_dir, eine Funktion, um automatisch source-Befehle zu allen R-Dateien (Dateien, die eine bestimmte Zeichenkette, wie ``.R"enthalten) zu erzeugen; docu_pdf, um das Erstellen des Pdf-Manuals zu einem R-Paket zu erleichtern, pfad_ohne_datei um Pfad und Dateiname zu trennen und finde_pakete, um Pakete, die in einem Source-Code geladen werden, aufzulisten. Weiter sind Funktionen zum Erstellen der EnMoLMU-Ordnerstruktur enthalten.

Depends R (>= 3.1.2)

License Keine Ahnung!

LazyData true

Encoding UTF-8

Imports devtools, roxygen2

R topics documented:

`docu_pdf` *Erzeugen von Pdf-Manual zu einem Paket in einem Schritt*

Description

Die Funktion `docu_pdf` erzeugt ein Pdf-Manual aus einem Paketordner in einem Schritt.

Usage

```
docu_pdf(paket, altes_pdf_loeschen = TRUE)
```

Arguments

paket	Vollständiger Pfad (oder ab Arbeitsverzeichnis) und Name des Paketes, für das die Dokumentationspdf erzeugt werden soll.
altes_pdf_loeschen	Logical, soll eine bereits vorhandene pdf-Datei vorher gelöscht werden?

Details

Die R-Code-Dateien müssen UTF-8 kodiert sein.

Value

Schreibt und öffnet Paket-Manual als pdf.

Examples

```
## Not run: docu_pdf("C:/Users/Rottner/Documents/EnMoLMU/R Pakete/testpaket1", altes_pdf_loeschen = TRUE)
```

---

enmolmu_ordnerstruktur

Anlegen der EnMoLMU-Ordnerstruktur

Description

Die Funktion legt am angegebenen Ort die vorgeschlagene Ordnerstruktur an, also erzeugt die zum Ausführen von EnMoLMU benötigten Ordner und Unterordner.

Usage

```
enmolmu_ordnerstruktur(start)
```

Arguments

start	Vollständiger Pfad und Name des die Ordnerstruktur beinhaltenden Ordners.

Details

Für eine leichter an Veränderungen anpassbare Variante siehe enmolmu_ordnerstruktur2. Beim Gegenwärtigen Stand sind beide Funktionen gleichwertig. Für die Weiterentwicklung wird enmolmu_ordnerstruktur2 empfohlen.

Value

Die Ordner und Unterordner werden im Verzeichnis start angelegt und auf der Console die erstellten Ordner aufgelistet.

Examples

```
## Not run:
enmolmu_ordnerstruktur(start = "C:/Users/Rottner/Documents/EnMoLMU/EnMoLMU_Ordnerstruktur2")

## End(Not run)
```

`enmolmu_ordnerstruktur2`

Anlegen der EnMoLMU-Ordnerstruktur

Description

Die Funktion legt am angegebenen Ort die vorgeschlagene Ordnerstruktur an, also erzeugt die zum Ausführen von EnMoLMU benötigten Ordner und Unterordner.

Usage

```
enmolmu_ordnerstruktur2(start)
```

Arguments

`start`	Vollständiger Pfad und Name des die Ordnerstruktur beinhaltenden Ordners.

Details

Erzeugt dieselbe Ordnerstruktur wie `enmolmu_ordnerstruktur`, verwendet aber intern die Funktion `komponente`, welche weit flexibler ist und eine einfachere und flexiblere Anpassung der Ordnerstruktur in der künftigen Weiterentwicklung von EnMoLMU bzw. anderer Anwendungen für diese Funktion ermöglicht.

Value

Die Ordner und Unterordner werden im Verzeichnis start angelegt und auf der Console die erstellten Ordner aufgelistet.

Examples

```
## Not run:
enmolmu_ordnerstruktur2(start = "C:/Users/Rottner/Documents/EnMoLMU/EnMoLMU_Ordnerstruktur3")

## End(Not run)
```

enmoSonstiges *Ein Paket mit einer Reihe von Zusatzfunktionen.*

Description

Enhält r_source_dir, eine Funktion, um automatisch source-Befehle zu allen R-Dateien (Dateien, die eine bestimmte Zeichenkette, wie ".R"enthalten) zu erzeugen; docu_pdf, um das Erstellen des Pdf-Manuals zu einem R-Paket zu erleichtern, pfad_ohne_datei, um Pfad und Dateiname zu trennen und finde_pakete, um Pakete, die in einem Source-Code geladen werden, aufzulisten. Weiter sind Funktionen zum Anlegen der EnMoLMU Ordnerstruktur enthalten.

enmoSonstiges-Funktionen

siehe Hilfe zu docu_pdf, r_source_dir, pfad_ohne_datei und finde_pakete.

finde_pakete *Alle in einem Source-Code verwendeten Pakete finden*

Description

Die Funktion `finde_pakete` sucht in einer R-Code-Datei nach allen Aufrufen von `require` und `library` und gibt eine Liste von den damit geladenen Paketen aus.

Usage

```
finde_pakete(datei, kommentiert = TRUE)
```

Arguments

`datei`	Pfade und Name der Datei, die durchsucht werden soll.
`kommentiert`	Sollen die auskommentierten Zeilen auch verwendet werden.

Value

Liste mit den geladenen Paketen ggf. mit auskommentierten Aufrufen.

Examples

```
## Not run: finde_pakete("Code_Daten/datenbank_funktionen.R")
## Not run: unique(unlist(lapply(dir(recursive = TRUE, pattern = ".R"), finde_pakete, kommentiert = FALSE)))
```

komponente *Erzeugen einer Ordnerstruktur aus einer Liste*

Description

Die Funktion erzeugt eine Ordnerstruktur, deren Aufbau über eine Liste festgelegt wird. Hauptsächlich für die Verwendung durch die Funktion enmolmu_ordnerstruktur2 gedacht.

Usage

```
komponente(element, name)
```

Arguments

element	Eine Liste, in welcher weitere Listen oder character-Vektoren sind.
name	Name des Ordners, in welchem diese Ordner angelegt werden sollen.

Details

Der Ordner name muss nicht existieren, da er von komponente angelegt wird. Die Elemente der Liste müssen ein character-Vektor sein, falls es zu den einzelnen Einträgen des Vektors keine weiteren Unterordner mehr geben soll und eine Liste (list), falls die einzelnen Einträge wiederum Unterordner beinhalten sollen. Kommt beides vor (Ordner mit weiteren Unterordnern und Ordner ohne), so muss eine Liste verwendet werden und die Ordner ohne Unterordner in der Form `"Ordner3"=""` angegeben werden (also als Vektor mit einem Leeren String als Inhalt und als Namen der Name des Ordners).

Value

Legt die angegebenen Ordner an.

Examples

```
## Not run:
# Liste mit Struktur erstellen:
ebenen <- list("Berichte" = c("Berichte_fertig", "Berichtvorlagen"),
"Daten" = c("Famos", "GLT_hist", "GLT_updates", "sonstige_Daten", "Wetterdaten"),
"optional" = list("Entwicklung" = c("R-Pakete"), "R-Code"=""),
"Skripte" = c("batch", "R"))
# Ordner anlegen:
komponente(ebenen, "EnMoLMU_ordnerstruktur3")

## End(Not run)
```

pfad_ohne_datei *Datei und Ordner aus Pfad extrahieren*

Description

Extrahiert Pfad aus Pfad mit Datei, Pfad ohne letzten Ordner bzw. letzten Ordner/Datei ohne Pfad.

Usage

```
pfad_ohne_datei(pfad, datei = FALSE)
```

Arguments

pfad	Character-Vektor aus Pfaden bzw. Pfaden mit Dateien
datei	Logical (TRUE oder FALSE). Soll die Datei oder der letzte Ordner zurückgegeben werden oder (TRUE) oder der Pfad dahin (FALSE). Default-Wert: FALSE.

Details

Sollte der Pfad mit einem Ordner enden, dürfen am Schluss keine "/" oder "\" mehr sein.

Value

Character-Vektor mit Pfaden oder Dateien/Ordnern

Examples

```
pfad_ohne_datei(c("ordner1/ordner2/datei.R", "ordner1\\ordner2\\datei.docx",
                "datei.csv"))
pfad_ohne_datei(c("ordner1/ordner2/datei.R", "ordner1\\ordner2\\datei.docx",
                "ordner1/order2","datei.csv"), datei=T)
pfad_ohne_datei(c("ordner1/ordner2/datei.R", "ordner1\\ordner2\\datei.docx",
                "ordner1/order2","datei.csv"), datei=F)
```

r_source_dir *Source-Befehle erzeugen*

Description

Funktion, um automatisch source-Befehle zu allen R-Dateien (Dateien, die eine bestimmte Zeichenkette, wie ".R" enthalten) zu erzeugen

Usage

```
r_source_dir(pfad, pattern = ".R")
```

Arguments

pfad	Ordner, in dem sich die Dateien befinden.
pattern	Zeichenkette, die in den Namen aller Dateien vorhanden ist, für welche die source-Befehle erzeugt werden sollen (Default ist ".R").

Value

Gibt die Befehlszeilen in der Console aus, so dass sie in die Code-Datei kopiert werden können.

Examples

```
## Not run: r_source_dir("Code_Daten")
```

Index

A.1.3 enmoDaten

Package 'enmoDaten'

September 7, 2015

Title Funktionen zum Aufbereiten der Daten und Schreiben/Lesen in/aus die/der MySQL-Datenbank

Version 1.0

Description Funktionen zum Aufbereiten der Daten und Schreiben/Lesen in/aus die/der MySQL-Datenbank für EnMoLMU unter Verwendung von RODBC.

Depends R (>= 3.1.2)

License What license is it under?

LazyData true

Encoding UTF-8

Imports RODBC,
plyr,
reshape2,
data.table,
sqldf,
enmoSonstiges

R topics documented:

`between` *Liegt x im Intervall [min, max]?*

Description

Überprüfen, ob x im Intervall [min, max] bzw. (min, max) liegt.

Usage

```
between(x, min, max, incl_equal = TRUE)
```

Arguments

`x`	Zu überprüfender Wert (Skalar oder Vektor).
`min`	Untere Grenze des Intervalls.
`max`	Obere Grenze des Intervalls.
`incl_equal`	Logical, Sollen die Intervallgrenzen eingeschlossen werden.

Details

Überprüfen, ob x im intervall [min, max] (incl_equal = TRUE) bzw (min, max) (falls incl_equal = FALSE) liegt.

Value

Wahrheitswert (TRUE/FALSE)

Examples

```
between(2,2,5)
between(1:2, 1:2, 5:6, incl_equal = FALSE)
between(1, 10, 20)
```

`dampfdruck` *Dampfdruck e*

Description

Berechnen des Dampfdruckes e nach Anleitung des Meteorologischen Instituts.

Usage

```
dampfdruck(theta_L, theta_strich, p = 1006.6)
```

Arguments

`theta_L`	Lufttemperatur in °C
`theta_strich`	Feuchttemperatur
`p`	Druck in hPa (1006.6hPa = 755 Torr). Üblich: unabhängig von Temperatur und Feuchte gesetzt
`E_strich`	Sättigungsdampfdruck bei Feuchttemperatur

Value

Dampfdruck

`datentyp_waehlen` *Datentypkonverter von R nach MySQL*

Description

Findet die Datentyp-Entsprechung in einer MySQL-Datenbank zu R-Objekten.

Usage

```
datentyp_waehlen(typ)
```

Arguments

`typ`	Name des R-Datentyps (class) als character.

Details

Falls die Länge des Arguments typ größer als 1 ist, wird der nur der erste Wert genommen. Dies ermöglicht auch die Umwandlung von Rückgabewerten des Befehls `class`, falls dieser auf ein Objekt mit mehreren Klassen angewendet wird (z.B. class(datum_als_POSIXct) ergibt "POSIXct" "POSIXt"). Somit kann kein Vekor mit mehreren Typen, die alle konvertiert werden sollen, übergeben werden. Für alle Typen außer numeric, integer, POSIXct, POSIXlt und Date wird varchar zurückgegeben.

Value

Name des entsprechenden Variablentyps einer MySQL-Datenbank.

Examples

```
datentyp_waehlen("POSIXct")
```

`datentyp_waehlen_df` *Datentypkonverter zu MySQL fuer einen data.frame*

Description

Gibt die Datentypennamen für alle Spalten eines data.frames an, die in einer MySQL-Datenbank verwendet werden können.

Usage

```
datentyp_waehlen_df(df)
```

Arguments

`df`	data.frame.

Details

Verwendet `datentyp_waehlen`.

Value

Vektor mit den Datentypnamen.

Examples

```
df <- data.frame(a = 1:3, b = letters[1:3], c = as.POSIXct("2014-03-01"))
datentyp_waehlen_df(df)
```

`datum_konvertieren` *Datum in einzelne Spalten umwandeln*

Description

wandelt Datum im POSIX- oder Date-Format in data.frame mit 6 bzw. 3 Spalten für Jahr, Monat, Tag, Stunde... um und umgekehrt

Usage

```
datum_konvertieren(datum = NULL, spalten = NULL, klasse = c("auto",
  "POSIXct", "Date"), spaltennamen = c("Jahr", "Monat", "Tag", "Stunde",
  "Minute", "Sekunde"), tz = "GMT", spaltenformat = c("integer",
  "character", "factor"))
```

Arguments

`datum`	Vektor mit Datum im POSIXct oder Date-Format.
`spalten`	data.frame, matrix oder vektor (für 1 Datum) mit Spalten für Jahr, Monat, Tag und ggf. Stunde, Minute, Sekunde.
`klasse`	eins von "auto", "POSIXct", "Date". Auto wählt je nach Spaltenanzahl Posix bzw. Date aus, die anderen beiden schreiben das Format vor (Posix ist mit, Date ohne Uhrzeit).
`spaltennamen`	die Spaltennamen des data.frames/matrix bzw. Namen des Vektors spalten. Default ist c("Jahr", "Monat", "Tag", "Stunde", "Minute", "Sekunde").
`stringsAsFactors`	sollen bei einer umwandlung in Spalten die Werte in factor umgewandelt werden (anstatt character). Default: FALSE.

Details

Wenn spalten angegeben werden, wird ein Vektor mit Datum (und Uhrzeit) erzeugt; wenn datum angegeben wird, wird ein data.frame mit für die Bestandteile von Datum und Uhrzeit erzeugt. Für ersteres müssen mindestens Jahr, Monat und Tage vorhanden sein (dann wird ein Date-Objekt erzeugt, falls klasse = "auto"). Stunde, Minute und Sekunde können optional angegeben werden (falls sie angegeben werden, es aber nicht alle 3 sind, muss spaltennamen dennoch alle 6 Werte haben; dann wird ein POSIXct-Objekt erzeugt, falls klasse = auto). Durch klasse = "Date" oder klasse = "POSIXct" kann das Format des Ergebnisses unabhängig von den vorhandenen Spalten vorgegeben werden (bei 3 Spalten und Posix sind alle Elemente der Uhrzeit 0, bei 6 Spalten und Date wird die Uhrzeit entfernt). Die Zuordnung der Spalten erfolgt anhand deren Namen, es sei denn, es wird ein Vektor ohne Namen angegeben, dann wird die Reihenfolge von spaltennamen angenommen.

Value

Vektor mit Daten oder data.frame mit 3 oder 6 Spalten für Datum und Uhrzeit.

Examples

```
datum_konvertieren(datum = as.POSIXct("2014-03-04 10:04:00"))
datum_konvertieren(datum = as.Date("2014-03-04"))
datum_konvertieren(spalten = data.frame(Jahr = 2014, Monat=3, Tag = 15))
class(.Last.value)
datum_konvertieren(spalten = data.frame(Jahr = 2014, Monat=3, Tag = 15,
                                        Minute=4, Stunde = 3, Sekunde = 0))
class(.Last.value)
datum_konvertieren(spalten = data.frame(Jahr = 2014, Monat=3, Tag = 15),
                   klasse = "POS") #' POSIXct kann abgekürzt werden
class(.Last.value)
datum_konvertieren(spalten = data.frame(Jahr = 2014, Monat=3, Tag = 15,
                                        Stunde = 3, Minute=4, Sekunde = 0),
                                        klasse = "Date")
class(.Last.value)
datum_konvertieren(spalten = data.frame(Jahr = 2014:2015, Mon=3:4, Tg = 15:16,
                                        h = 3:4, Min=4:5, Sek = 0:1),
                   spaltennamen = c("Jahr", "Mon", "Tg", "h",
                                                  "Min", "Sek"))
class(.Last.value)
datum_konvertieren(spalten = c(2011, 3, 1, 0, 0, 0))
class(.Last.value)
datum_konvertieren(spalten = c(2011, 3, 1))
class(.Last.value)
datum_konvertieren(spalten = c(2011, 4, 4, 4))
datum_konvertieren(spalten = data.frame(Jahr = 2014, Monat=3, Tag = 15,
                                        Stunde = 3, Minute=4))
datumsspalten <- c(2015, 1,20, 12)
names(datumsspalten) <- c("Jahr", "Monat", "Tag", "Stunde")
datum_konvertieren(spalten = datumsspalten)
```

---

db_namen *Spaltennamen in einem data.frame Datenbank-sicher machen*

Description

Wandelt die Spaltennamen eines data.frames so um, dass sie in einer Datenbank-Tabelle verwendet werden können.

Usage

```
db_namen(x, ...)
```

Arguments

x	data.frame.
kleinbuchstaben	Logical (TRUE/FALSE). Sollen die Namen ausschließlich aus Kleinbuchstaben bestehen.

Value

data.frame mit neuen Spaltennamen.

Examples

```
db_namen(data.frame(A.3 = 1:4))
```

`db_spalten_hinzufuegen` *Spalten aus einem data.frame in bereits bestehende Zeilen einer Tabelle schreiben.*

Description

Mit der Funktion können ausgewählte Spalten aus einem data.frame in bereits existierende Zeilen einer MySQL-Tabelle geschrieben werden. Da sich `db_spalten_hinzufuegen2` in der Praxis als deutlich schneller erwiesen hat, wird diese Funktion empfohlen.

Usage

```
db_spalten_hinzufuegen(con, tabelle, df, spalten_tabelle = NULL,
  spalten_df = NULL, datum_tabelle = "datum", datum_df = "Datum",
  type_datum = TRUE)
```

Arguments

`con`	RODBC-Verbindung.
`tabelle`	Name der Tabelle in der Datenbank in die die Daten geschrieben werden sollen.
`df`	data.frame in dem sich die Daten befinden.
`spalten_tabelle`	In welche Spalten der Tabelle die Daten geschrieben werden sollen. Default ist umgewandelte Namen aus spalten_df (NULL).
`spalten_df`	Welche Spalten aus dem data.frame in die Tabelle geschrieben werden sollen. Default ist alle (NULL).
`datum_tabelle`	Name der Datumsspalte in tabelle. Default ist "datum".
`datum_df`	Name der Datumsspalte im data.frame. Default ist "Datum".
`type_datum`	Ist datum_df ein Datum (idR. TRUE)?

Details

Der data.frame wird zuerst temporär als neue Tabelle in die Datenbank geschrieben, und dann von dort mit update in die angegebene Tabelle kopiert. Die Zeilen und Spalten, in die Werte geschrieben werden sollen, müssen in der Datenbank-Tabelle bereits vorhanden sein. Die Spalten datum_tabelle und datum_df müssen kein Datum sein.

Value

Daten werden in die Tabelle in der Datenbank geschrieben.

See Also

`db_spalten_hinzufuegen2` für eine Variante, die keine Daten in der Datenbank zwischenspeichert und möglicherweise schneller ist.

Examples

```
## Not run:
daten_abd <- data.frame(a=1:5, b=letters[1:5], d=NA)
daten_neu <- data.frame(A=5:1, D = c(0,1,1,2,0)) #umgekehrte Reihenfolge
library(RODBC)
ch <- odbcConnect("ansi_system")
sqlSave(ch, daten_abd)
db_spalten_hinzufuegen(ch, "daten_abd", daten_neu, spalten_tabelle = "d",
spalten_df = "D", datum_tabelle = "a", datum_df="A", type_datum = FALSE)
sqlFetch(ch, "daten_abd")
sqlDrop(ch, "daten_abd")

## End(Not run)
```

`db_spalten_hinzufuegen2` *Spalten aus einem data.frame in bereits bestehende Zeilen einer Tabelle schreiben.*

Description

Mit der Funktion können ausgewählte Spalten aus einem data.frame in bereits existierende Zeilen einer MySQL-Tabelle geschrieben werden. Anders als `db_spalten_hinzufuegen` sendet `db_spalten_hinzufuegen2` die zu aktualisierenden Daten Zeile für Zeile an die Datenbank.

Usage

```
db_spalten_hinzufuegen2(con, tabelle, df, spalten_tabelle = NULL,
  spalten_df = NULL, datum_tabelle = "datum", datum_df = "Datum")
```

Arguments

`con`	RODBC-Verbindung.
`tabelle`	Name der Tabelle in der Datenbank in die die Daten geschrieben werden sollen.
`df`	data.frame in dem sich die Daten befinden.
`spalten_tabelle`	In welche Spalten der Tabelle die Daten geschrieben werden sollen. Default ist umgewandelte Namen aus spalten_df (NULL).
`spalten_df`	Welche Spalten aus dem data.frame in die Tabelle geschrieben werden sollen. Default ist alle (NULL).
`datum_tabelle`	Name der Datumsspalte in tabelle. Default ist "datum".
`datum_df`	Name der Datumsspalte im data.frame. Default ist "Datum".

Details

Es werden SQL-Abfragen zur Aktualisierung jeder einzelnen Zeile erzeugt und diese nacheinander an die Datenbank gesendet.

Value

Daten werden in die Tabelle in der Datenbank geschrieben.

Examples

```
## Not run:
daten_abd <- data.frame(a=1:5, b=letters[1:5], d=NA, e=NA)
daten_neu <- data.frame(A=5:1, D = c(0,1,1,2,0), e=1:5) #umgekehrte Reihenfolge
library(RODBC)
ch <- odbcConnect("ansi_system")
sqlSave(ch, daten_abd)
db_spalten_hinzufuegen2(ch, "daten_abd", daten_neu, spalten_tabelle = c("d","e"),
spalten_df = c("D", "e"), datum_tabelle = "a", datum_df="A")#, type_datum = FALSE)
sqlFetch(ch, "daten_abd")
sqlDrop(ch, "daten_abd")

## End(Not run)
```

---

df_aufteilen_rodbc *data.frame anhand von Tabelle aufteilen*

Description

data.frame danach aufteilen, ob die Spalten bereits in einer Datenbank-Tabelle vorhanden oder neu sind.

Usage

```
df_aufteilen_rodbc(con, tabelle, df, zweite_tabelle_mit_datum = TRUE,
  datum = "datum")
```

Arguments

con	Verbindung zu RODBC-Datenbank.
tabelle	Tabellenname in der Datenbank.
df	data.frame der für das Einfügen in die Tabelle vorbereitet werden soll.
zweite_tabelle_mit_datum	Soll in die zweite Tabelle, welche die Variablen beinhaltet, die in der ursprünglichen nicht vorkommen die Variable datum hinzugefügt werden?
datum	Der Name der Datumsvariablen muss in der Tabelle in der Datenbank genauso lauten wie im data.frame mit Ausnahme der Anpassungen die durch mache_db_namen vorgenommen werden ("." wird durch "_" und Großbuchstaben durch Kleinbuchstaben ersetzt).

Value

Liste mit zwei data.frames, wobei wie_db die Spalten, welche auch in der Tabelle vorhanden sind beinhaltet und neu_df die restlichen.

ds_hinzufuegen *Erweitern eines Datensatzes*

Description

Fügt zu einem Datensatz je eine Spalte mit Ferien, Wochentag, Jahr, etc. hinzu.

Usage

```
ds_hinzufuegen(x, Wochentag = FALSE, Wochenende = FALSE, Jahr = FALSE,
  Monat = FALSE, Tag = FALSE, semesterferien = NULL,
  ferienspalten = c(1, 2), ferienformat = NULL, differenz = NULL,
  datumsspalte = 1, datumsformat = NULL, tz = "GMT")
```

Arguments

x	Datensatz (data.frame) mit Datumsspalte.
Wochentag,Jahr,etc.	soll eine Spalte mit Wochentag, Jahr, etc. hinzugefügt werden
semesterferien	data.frame mit zwei Spalten für Semesterferien (Beginn und Ende), falls diese hinzugefügt werden sollen.
ferienspalten	In welchen Spalten (Spaltennummern oder Namen) befinden sich Beginn und Ende.
ferienformat	Format des Feriendatums (POSIXct), siehe tz
differenz	character-Vektor oder Spaltenindizes mit Variablen zu denen die Differenz zur jeweils letzten Beobachtung gebildet werden soll (als zusätzliche Variable mit diff_Variablenname, erster Wert=NA). Variable wird vorher nach datumsspalte sortiert
datumsspalte	in welcher Spalte (Index oder Name) steht das Datum (muss POSIXct, POSIXlt oder Date sein)
datumsformat	Format des Datums (POSIXct), falls es in POSIXct umgewandelt werden soll
tz	Zeitzone (time zone) des Datums, default = "GMT" (koordinierte Weltzeit; gilt auch für ferienformat).

Examples

```
t1 <- data.frame(a=as.Date(1:4, "2014-01-01"), e=letters[1:4+10])
ds_hinzufuegen(t1, Wochentag=TRUE)
ds_hinzufuegen(t1, Wochentag=FALSE, Jahr=TRUE, Monat=TRUE, Tag=TRUE, datumsspalte = "a")
t5 <- data.frame(a=as.Date(1:26, "2014-01-01"), e=letters[1:26])
semferien <- data.frame(Beginn = as.Date(c("2014-01-05", "2014-01-15", "2014-01-25")),
                        Ende = as.Date(c("2014-01-08", "2014-01-21", "2014-03-01")))
ds_hinzufuegen(t5, semesterferien=semferien, ferienspalten=c("Beginn", "Ende"))

bla <- data.frame(a=1:6, b=c(1,2,4,7,10,11))
ds_hinzufuegen(bla, differenz=2)
```

`glt_hist` *Tagesdateien in Tabelle schreiben*

Description

Liest die einzelnen Tagesdateien ein und schreibt sie (als 1 data.frame) in die Tabelle.

Usage

```
glt_hist(quellordner, namensteil, tabelle, con, datum = "Datum",
  ferien = "Daten/sonstige_Daten/ferien.csv", variablenauswahl = NULL)
```

Arguments

`quellordner`	Ordner in welchem sich die einzelnen Tagesdateien befinden.
`namensteil`	Teil des Dateinamens der einzelnen Tagesdateien, der sie von allen anderen Dateien im Ordner unterscheidet.
`tabelle`	Name der MySQL-Tabelle, in welche die Daten geschrieben werden soll.
`con`	RODBC-Verbindung zu der Datenbank in welche die Tabelle kommen soll.
`datum`	Name der Datumsspalte in den csv-Dateien (muss "datum" mit beliebiger Gross-/Kleinschreibung sein).
`ferien`	Datei im csv-Format, welche die Ferienzeiten enthält.
`variablenauswahl`	Character-Vektor mit Spaltennamen, falls nur bestimmte Spalten in die Datenbank-Tabelle geschrieben werden sollen.

Details

Die Spaltennamen in variablenauswahl müssen bereits im Datenbankformat sein, wobei die neu erzeugten Datums-/Zeitvariablen automatisch hinzugefügt werden, die diff-Variablen aber explizit angegeben werden müssen. Das Datum wird grundsätzlich wiederholt in mehreren Formaten in die Tabelle geschrieben, um sie leichter auf Fehler überprüfen zu können. Die Datumsvariable datum muss 16 oder 19 Zeichen haben ("%d.%m.%Y %H:%M:%S" mit bzw. ohne Sekunden).

Value

Nichts. Es werden Informationen aus der Datenbank zur Überprüfung in der Console ausgegeben: Auf der Console werden zur Überprüfung Anzahl der Zeilen, die in die Tabelle geschrieben wurden, ältestes und jüngstes Datum dieser Zeilen, dieselben Daten für jeden einzelnen Monat und die Häufigkeitstabelle der 15-Minuten-Takt- Variable cut15min ausgegeben. Mindestdatum sollte dem ersten Tag der historischen Daten und das Maximaldatum dem letzten Tag entsprechen. Weiter sollte jedes Mindestdatum die Uhrzeit 0:00 haben (wo die Uhrzeit nicht mitangezeigt wird) und das Maximaldatum jeweils 23:59:00 sein. Dies gilt auch für die Monatsweisen "Datume". Die einzelnen Häufigkeiten der Häufigkeitstabelle von cut15min sollten gleich sein und in ihrer Summe der Gesamtanzahl Zeilen entsprechen. Diese Übereinstimmungen garantieren natürlich nicht, dass die Daten in jeder Hinsicht richtig in die Tabelle geschrieben wurden, deckt aber viele bei der Programmierung aufgetretenen Fehler auf und ist ein Hinweis darauf, dass alles richtig ist.

Examples

```
## Not run:
  glt_hist(quellordner = "D:/historische daten glt 2015_02_20/umgebaut_zusammen",
           namensteil = "The46", tabelle = "glt_hist_the46", con = ch,
           variablenauswahl = c("x1110_01_0032hzg00tem0001ma01",
                                "x1120_01_021anhv01elz0001zw13",
                                "x1120_01_021anhv01elz0001zw30",
                                "x1120_04__529klt01vel0003s101",
                                "x1110_01_0032hzg03wmz0001zw04"))

  glt_hist(quellordner = "D:/historische daten glt 2015_02_20/umgebaut_zusammen",
           namensteil = "Oet67", tabelle = "glt_hist_oet67", con = ch)

  glt_hist(quellordner = "D:/historische daten glt 2015_02_20/umgebaut_zusammen",
           namensteil = "Leo03", tabelle = "glt_hist_leo03", con = ch)

## End(Not run)
```

`glt_hist_aufteilen` *Grosse GLT-Datei einlesen und in kleinere Dateien aufgeteilt speichern*

Description

Einlesen einer größeren GLT-Datei und Abspeichern des Inhalts in Dateien für die einzelnen Tage mit sqldf (Datei muss nicht in den RAM passen).

Usage

```
glt_hist_aufteilen(datei, in_verzeichnis, name, ...)
```

Arguments

`datei`	Pfad und Name zur Datei, welche aufgeteilt werden soll.
`in_verzeichnis`	Pfad zu dem Ordner, in welchen die aufgeteilten Dateien geschrieben werden sollen (wird erzeugt, falls nicht vorhanden).
`name`	Namensbestandteil der Zieldateien (wird zu paste(%y%m%d, name, ".csv", sep="")).

Details

`glt_hist_aufteilen_fread` ist schneller und wird somit bei genügend großem RAM empfohlen.

Value

Ziel-Dateien werden in Ordner in_verzeichnis geschrieben.

Examples

```
## Not run:
system.time(glt_hist_aufteilen(datei = "Daten/glt_test_hist_neu/The46EnMoLMU.csv",
              in_verzeichnis = "Daten/glt_test_hist_neu/einzelne_tage_muster",
              name = "The46EnMoLMU"))

glt_hist_aufteilen(datei = "Daten/glt_test_hist_neu/The46EnMoLMU.csv",
              in_verzeichnis = "Daten/glt_test_hist_neu/einzelne_tage_muster",
              name = "The46EnMoLMU",
              dbname = NULL)

## End(Not run)
```

---

glt_hist_aufteilen_fread

Einlesen von grosser GLT-Datei mit fread und als einzelne Tage speichern.

Description

Einlesen einer größeren GLT-Datei und Abspeichern des Inhalts in Dateien für die einzelnen Tage mit fread{data.table}. Hier muss die komplette Datei in den RAM passen.

Usage

```
glt_hist_aufteilen_fread(datei, in_verzeichnis, name, ...)
```

Arguments

datei	Pfad und Name zur Datei, welche aufgeteilt werden soll.
in_verzeichnis	Pfad zu dem Ordner, in welchen die aufgeteilten Dateien geschrieben werden sollen (wird erzeugt, falls nicht vorhanden).
name	Namensbestandteil der Zieldateien (wird zu paste(%y%m%d, name, ".csv", sep="")).

Details

Liest Dezimalzahlen mit Komma als Dezimaltrennzeichen als Charakter ein. Die Daten werden allerdings nur nach Datum aufgeteilt und dann wieder als csv geschrieben, somit dürfte dies keinen Unterschied machen. In der aktuellen development-Version 1.9.5 (Stand 1.4.2015) des Pakets data.table ist ein dec-Argument eingebaut. Sobald dies die "stable"-Version ist, kann nach einem Update auch dec="," in fread gesetzt werden.

Value

Ziel-Dateien werden in Ordner in_verzeichnis geschrieben.

Examples

```
## Not run:
glt_hist_aufteilen_fread(datei = "Daten/glt_test_hist_neu/2014_02_19_h_The46EnMoLMU.csv",
                         in_verzeichnis = "Daten/glt_test_hist_neu/einzelne_tage_muster",
                          name = "The46EnMoLMU")

## End(Not run)
```

`glt_hist_vorbereiten` *Vorbereiten und Einpflegen der historischen GLT-Daten in eine neue Tabelle der Datenbank.*

Description

Vorbereiten und Einpflegen der historischen GLT-Daten in eine neue Tabelle der Datenbank.

Usage

```
glt_hist_vorbereiten(datei, unterscheidungsname = "", tabelle, con,
  zwischenschritte_loeschen = FALSE, zwischenschritte_ordner = NULL, ...)
```

Arguments

`datei`	Welche Datei(en) (Pfad und Name) beinhaltet die historischen Daten (als character-Vektor). Dateien, die in dieselbe Tabelle kommen sollten in einem Aufruf verarbeitet werden.
`unterscheidungsname`	Teil des Dateinamens, der diese Datei(en) von anderen im selben Ordner unterscheidet. Falls keine anderen vorhanden sind, oder diese Funktion noch nicht auf diese angewendet worden ist bzw. die von der Funktion erzeugten Unterordner (deren Inhalt) wieder gelöscht wurde, ist dieses Argument nicht nötig.
`tabelle`	Name der Tabelle, in die diese Daten geschrieben werden soll (wird angelegt).
`con`	RODBC-Verbindung.
`zwischenschritte_loeschen`	Logical (TRUE/FALSE). Sollen die Ordner mit den Zwischenschritten wieder gelöscht werden.
`zwischenschritte_ordner`	Falls die Ordner, in welche die Zwischenschritte bei der Umwandlung gespeichert werden nicht in den selben Ordner wie datei geschrieben werden sollen, kann hier ein Pfad zu einem alternativen Ordner angegeben werden, z.B. wenn für den Ordner von datei keine Schreibrechte bestehen.
`...`	Weitere Argumente, die an glt_hist weitergereicht werden (ferien, variablenauswahl).

Details

Die Zwischenschritte werden in den Ordner der ersten Datei in datei geschrieben.

Value

Nichts, Aufbereitungsschritte werden in die neu erzeugten Ordner geschrieben und die Tabelle tabelle erzeugt und die Daten hineingeschrieben. Auf der Console werden zur Überprüfung Anzahl der Zeilen, die in die Tabelle geschrieben wurden, ältestes und jüngstes Datum dieser Zeilen, dieselben Daten für jeden einzelnen Monat und die Häufigkeitstabelle der 15-Minuten-Takt- Variable cut15min ausgegeben. Mindestdatum sollte dem ersten Tag der historischen Daten und das Maximaldatum dem letzten Tag entsprechen. Weiter sollte jedes Mindestdatum die Uhrzeit 0:00 haben (wo die Uhrzeit nicht mitangezeigt wird) und das Maximaldatum jeweils 23:59:00 sein. Dies gilt auch für die monatsweisen "Datume". Die einzelnen Häufigkeiten der Häufigkeitstabelle von cut15min sollten gleich sein und in Ihrer Summe der Gesamtanzahl Zeilen entsprechen. Diese Übereinstimmungen garantieren natürlich nicht, dass die Daten in jeder Hinsicht richtig in die Tabelle geschrieben wurden, deckt aber viele bei der Programmierung aufgetretenen Fehler auf und ist ein Hinweis darauf, dass alles richtig ist.

Examples

```
## Not run:
source('Code_Daten/DatenLesenAufbereiten.R')
source('Code_Daten/datenbank_funktionen.R')
source('Code_Daten/daten_aufbereiten_skript_fkt.R')
require(RODBC)
ch <- odbcConnect("ansi_system",  DBMSencoding = "utf-8")
#"ansi_system ist Name der eingerichteten ODBC-Verbindung, encoding evtl iso...
glt_hist_vorbereiten(datei = "130220_h_Oet67EnMoLMU_gekuerzt.csv",
                     unterscheidungsname = "Oet67EnMoLMU",
                  tabelle = "glt_hist2_oet67", con = ch, zwischenschritte_loeschen = FALSE)

glt_hist_vorbereiten(datei = "D:\\historische daten glt 2015_02_20/130220_h_Oet67EnMoLMU.csv",
                     unterscheidungsname = "Oet67EnMoLMU",
                     tabelle = "glt_hist2_oet67", con = ch,
                     zwischenschritte_loeschen = FALSE)
es kommen zwar seltsamerweise Warnungen wegen der UhrzeitlÃ¤nge, es wird aber
sowohl der richtige Teil davon verwendet, als auch ein richtiges Ergebnis in
die Dateien geschrieben.

## End(Not run)
```

`glt_umbauen` *Voraufbereiten der historischen GLT-Daten mit reshape2*

Description

Voraufbereiten der historischen GLT-Daten mit reshape2 (bei größeren Datenmengen und 32-Bit-OS Fehler: can't allocate...)

Usage

```
glt_umbauen(x)
```

Arguments

`x`	Pfade zur csv-Datei im neuen Format (ab 17.2.2015)

Value

data.frame mit Datei im Format der täglichen Aktualisierungen

Examples

```
## Not run:
head(glt_umbauen("Daten/glt_test_hist_neu/The46EnMoLMU - Muster mit Ãœberschrift.csv"))
system.time(test2 <- glt_umbauen("Daten/glt_test_hist_neu/The46EnMoLMU.csv"))
system.time(test2 <- glt_umbauen("Daten/glt_test_hist_neu/2014_02_19_h_The46EnMoLMU.csv"))
system.time(write.csv(file="Daten/glt_test_hist_neu/2014_02_19_hu_The46EnMoLMU.csv", x = test2))

## End(Not run)
```

`glt_umbauen2` *Voraufbereiten der historischen GLT-Daten mit for-Schleife (nach Skript von Paul Eichel).*

Description

Alternative zu glt_umbauen, falls 32-bit OS. Funktioniert auch bei 160 MB großen csv-Dateien auf 32-Bit OS.

Usage

```
glt_umbauen2(x)
```

Arguments

x	Pfade zur csv-Datei im neuen Format (ab 17.2.2015)

Value

data.frame mit Datei im Format der täglichen Aktualisierungen

Examples

```
## Not run:
head(glt_umbauen2("Daten/glt_test_hist_neu/The46EnMoLMU - Muster mit Ãœberschrift.csv"))
system.time(test2 <- glt_umbauen2("Daten/glt_test_hist_neu/The46EnMoLMU.csv"))
system.time(test2 <- glt_umbauen2("Daten/glt_test_hist_neu/2014_02_19_h_The46EnMoLMU.csv"))

## End(Not run)
```

`glt_umbauen_ordner` *umstellen eines gesamten Ornders mit GLT-Dateien vom Format historischer Daten in das Tagesupdate-Format*

Description

glt_umbauen (umstellen der GLT-Datensätze vom Format historischer Daten in das Tagesupdate-Format) aller Dateien eines Ordners und speichern als csv-Datei in einen anderen Ordner.

Usage

```
glt_umbauen_ordner(ursprung, ziel, nicht.ueberschreiben = TRUE,
  pattern = ".csv")
```

Arguments

`ursprung`	Ordner, in welchem sich die Ursprungsdateien befinden.
`ziel`	Ordner, in welchem sich die Zieldateien befinden.
`nicht.ueberschreiben`	Sollen Dateien, welche bereits im Zielordner sind überschrieben werden?
`pattern`	Teil des Namens der Dateien im Ursprungsordner, der sie als umzustellende Dateien kennzeichnet (default: ".csv")

Value

Nichts. Dateien werden in den Zielordner geschrieben.

Examples

```
## Not run:
glt_umbauen_ordner(ursprung =  "Daten/glt_test_hist_neu/einzelne_tage_muster",
                   ziel =  "Daten/glt_test_hist_neu/einzelne_tage_muster/umgebaut",
                   nicht.ueberschreiben = TRUE)
glt_umbauen_ordner(ursprung =  "Daten/glt_test_hist_neu/einzelne_tage_muster",
                   ziel =  "Daten/glt_test_hist_neu/einzelne_tage_muster/umgebaut",
                   nicht.ueberschreiben = FALSE)
#zum test wieder einlesen:
umgebaute_testdatei <- read.csv2(paste("Daten/glt_test_hist_neu/einzelne_tage_muster/umgebaut",
                                       "/", dir(paste0("Daten/glt_test_hist_neu/
                                                einzelne_tage_muster/umgebaut"))[1],
                                       sep = ""))
str(umgebaute_testdatei) # nichts, was auf Probleme hindeutet (1441 obs. = 24*60 = 1 Tag)

## End(Not run)
```

gradtagzahl *Gradtagzahl*

Description

Berechnen der Gradtagzahl (nach VDI 2067) http://de.wikipedia.org/wiki/Gradtagzahl

Usage

```
gradtagzahl(aussentemperatur, raumtemperatur = 20, heizgrenze = 15)
```

Arguments

aussentemperatur	Mittlere Tagesaußentemperatur.
raumtemperatur	Mittlere Raumtemperatur.
heizgrenze	Heizgrenze.

Details

Minutendaten müssen erst aggregiert werden, bevor sie an die Funktion übergeben werden können.

Value

Gradtagzahl.

grosser_datensatz *Einzelne GLT-Dateien zu einem zusammenfassen*

Description

Die Funktion fügt alle neuen Datensätze in einem Ordner zu einem großen Datensatz zusammen.

Usage

```
grosser_datensatz(con, tabelle = NULL, ordner, namensteil = "",
  ordner_neu = "", datum = "datum", csv_schreiben = TRUE)
```

Arguments

con	RODBC-Verbindung zur Datenbank.
tabelle	Tabelle in die die Daten eingefügt werden sollen und anhand derer überprüft werden soll, welche Datensätze neu sind; falls nichts angegeben wird (NULL), werden alle Datensätze verwendet.
ordner	Ordner, in dem sich die Datensätze befinden.
namensteil	Namensteil der Dateien im Ordner, der die Datensätze, welche zusammen- gefügt werden sollen identifiziert.
ordner_neu	Ordner, in den die fertige Datei kommen soll.

datum	Name der Datumsspalte in der tabelle anhand derer überprüft werden soll, welche Datensätze neu sind.
csv_schreiben	Soll der neue Datensatz als csv-Datei gespeichert werden oder als data.frame zurückgegeben werden.

Value

Schreibt zusammengefügte Daten als csv-Datei mit dem Namen der ersten Datei (mit dem ältesten Datum) in den Ordner ordner_neu und gibt Name und Pfad der Datei unsichtbar (invisible) zurück falls csv_schreiben = TRUE. Falls csv_schreiben = FALSE gibt die Funktion den neuen Datensatz zurück.

Examples

```
## Not run:
grosser_datensatz(ch, "oet67h2", ordner = "Daten/daten_glt_tagesupdates",
namensteil = "Oet67", ordner_neu = "Daten/daten_glt_tagesupdates_zusammengefÃ¼gt")
grosser_datensatz(ch, "leo03h2", ordner = "Daten/daten_glt_tagesupdates",
namensteil = "Leo03", ordner_neu = "Daten/daten_glt_tagesupdates_zusammengefÃ¼gt")
grosser_datensatz(ordner = "bla", ordner_neu = "bla")
grosser_datensatz(ch, ordner = "Daten/daten_glt_tagesupdates",
namensteil = "Leo03", csv_schreiben = FALSE) # Datensatz zurÃ¼ckgeben

## End(Not run)
```

---

historische_zaehler *Differenz-Variablen in Tabelle nachschlagen*

Description

Die Funktion sieht in den alten Daten in der Tabelle aus der Datenbank nach, welche Variablen auch als Differenz-Variablen vorhanden sind.

Usage

```
historische_zaehler(con, tabelle)
```

Arguments

con	RODBC-Verbindung zur Datenbank.
tabelle	Tabellenname.

Value

Alle in der Tabelle vorhandenen Variablennamen, die mit "diff_" beginnen (Name ohne "diff_", also z.B. "x7070_01_u138hzg13tem0001vi02" statt "diff_x707...")

Examples

```
## Not run: historische_zaehler(ch, "oet67")
```

`insert_wetter_mysql` *Einlesen der Wetterdaten und Speichern in der SQL-Datenbank*

Description

Die Funktion `insert_wetter_mysql` liest die Wetterdaten des Meteorologischen Instituts mit `read_wetter` ein, erweitert sie mithilfe von `wetter_erweitern` um zusätzliche Variablen und fügt sie einer Datenbank-Tabelle hinzu.

Usage

```
insert_wetter_mysql(con, monat, ..., tablename = "theresienwetter",
  relative_feuchte = TRUE, na = NULL)
```

Arguments

`con`	RODBC-Verbindung (siehe odbcConnect RODBC).
`monat`	Einzulesender Monat (z.B. "2015_01").
`tablename`	Name der Tabelle in der Datenbank, in die die Daten geschrieben werden sollen.
`na`	Welche Werte fehlenden Werten entsprechen und auf NA gesetzt werden sollen (Vektor mit einem oder mehreren Werten). Beim den bisher erhaltenen Wetterdaten des Meteorologischen Instituts der LMU werden fehlende Messwerte als 999.9 dargestellt und müssen somit durch NA (in R) bzw. NULL in der Datenbanktabelle ersetzt werden.
`name`	Name der Datei (z.B. "TheresienWetter").
`ordner`	Pfad zu dem Ordner, in welchem sich die Wetterdaten befinden.

Details

Siehe read_wetter().

Value

nichts, Daten werden in die Datenbank-Tabelle geschrieben.

Examples

```
## Not run:
sqlDrop(ch, "testwetter") # falls schon vorhanden
insert_wetter_mysql(ch, "2014_11", tablename = "testwetter")
insert_wetter_mysql(ch, c("2014_10", "2014_12", "2015_01"), tablename = "testwetter")
str(sqlFetch(ch, "testwetter", stringsAsFactor = FALSE))

## End(Not run)
```

ist_zaehler *Ueberpruefen, ob ein Vektor ein Zaehlervektor ist.*

Description

Diese Funktion überprüft anhand bestimmter Eigenschaften, ob ein Vektor zu einem Zähler gehört.

Usage

```
ist_zaehler(x, anteil_grenze = 0.999)
```

Arguments

x	Vektor der Zähler sein könnte.
anteil_grenze	Wie hoch muss der Anteil der Differenzen, die größergleich 0 sind, sein, damit die Variable als Zähler betrachtet werden soll.

Details

Spalten mit ausschließlich fehlenden Werten und Jahr, Monat, Datum als Zahl sollten vorher entfernt werden bzw. erst nachher angefügt werden. Kriterien im Namen der Zählers können hier nicht berücksichtigt werden, da sie nicht zwangsläufig an die Funktion übergeben werden (Spaltennamen, der von sapply weitergegeben wird, ist z.B. "X[[1L]]"). Siehe ist_zaehler_df

Value

TRUE für ist Zähler, FALSE für ist keiner

Examples

```
datensatz <- data.frame(a= letters[1:20], b = 1:20, c=rnorm(10))
zaehler <- sapply(datensatz, ist_zaehler)
# geht schneller (35 sek in einem Beispiel), aber nur data.frame
apply(datensatz, 2, ist_zaehler)
# dauert viel länger (4.5 min erster_datensatz_namen_na),
# funktioniert aber auch mit einer Matrix
```

ist_zaehler_df *Zaehlervariablen in data.frame suchen*

Description

Überprüft einen data.frame, bei welchen der Variablen es sich um einen Zähler handeln kann. Bezieht auch den Namen der Spalten in die Überprüfung mit ein.

Usage

```
ist_zaehler_df(x, name = "zw", anteil_grenze = 0.999,
  kleinbuchstaben = TRUE)
```

Arguments

x	data.frame mit Zähler-Variablen.
name	Namensbestandteil der Zähler-Variablen (4.- und 3.-letztes Zeichen).
anteil_grenze	Wie hoch muss der Anteil der Differenz-Werte sein, die größergleich 0 sind.
kleinbuchstaben	Namensbestandteil der Zählervariablen in x wird vor Vergleich in Kleinbuchstaben umgewandelt.

Value

Logical-Vektor für Variable im data.frame kann Zähler sein (TRUE) bzw. kann keiner sein (FALSE).

Examples

```
ist_zaehler_df(data.frame(xzw01 = c(1:10000), xzw02 = letters[1:10000],
                          xzw03 = c(10000, 1:9999), xxw04 = 1:10000,
                          xzw5 = 1:10000, xwz06 = 10000:1))# T,F,T,F,F,F
```

---

mache_db_namen *data.frame-Spaltennamen Datenbanksicher machen*

Description

einen Vektor mit data.frame-Spaltennamen so umwandeln, dass die Namen in einer Datenbank-Tabelle verwendet werden können ("." durch "_" ersetzen).

Usage

```
mache_db_namen(x, kleinbuchstaben = TRUE)
```

Arguments

x	Vektor mit data.frame-Spaltennamen.
kleinbuchstaben	Sollen die Spaltennamen nur aus Kleinbuchstaben bestehen (default ist TRUE).

Value

Spaltennamen für Datenbank.

Examples

```
mache_db_namen(c("variable.1", "VAR.2"))
mache_db_namen(c("variable.1", "VAR.2"), kleinbuchstaben = FALSE)
```

mache_db_namen2 *Einen vektor mit original Spaltennamen data.frame- und Datenbank-sicher machen*

Description

einen Vektor mit ursprünglichen Spaltennamen so umwandeln, dass die Namen in einem data.frame und einer Datenbank-Tabelle verwendet werden können (Buchstabe am Anfang, "+" etc. entfernen, "." durch "_" ersetzen...).

Usage

```
mache_db_namen2(x, ...)
```

Arguments

x	character-Vektor mit Spaltennamen.
kleinbuchstaben	Logical (TRUE/FALSE), sollen die Namen nur Kleinbuchstaben haben.

Details

Verwendet make.names und danach mache_db_namen.

Value

Spaltennamen für data.frame oder Datenbank.

Examples

```
mache_db_namen2(c("variable.1", "VAR.2"))
mache_db_namen2(c("variable.1", "VAR.2"), kleinbuchstaben = FALSE)
```

N *Anzahl Beobachtungen bei data.frame und vector mit einem Befehl*

Description

Kann die Anzahl Beobachtungen sowohl bei data.frame/matrix, als auch bei vector mit demselben Befehl anzeigen.

Usage

```
N(x)
```

Arguments

x	data.frame/matrix oder vector.

Value

Anzahl der Beobachtungen von x.

Examples

```
sapply(list(data.frame(1:3, letters[1:3]), matrix(1:12, nrow=4), 1:50), N)
```

---

na_spalten_entfernen *Entfernen von Spalten mit nur NA*

Description

Entfernt aus data.frame oder matrix alle Spalten mit ausschließlich fehlenden Werten.

Usage

```
na_spalten_entfernen(x, nur_index = FALSE)
```

Arguments

x	data.frame (oder matrix), der überprüft werden soll.
nur_index	Falls TRUE gibt die Funktion nur den Index (logical) der Spalten, die nicht ausschließlich NA sind zurück, ansonsten den gesamten Datensatz mit nur diesen Spalten.

Value

Logical-Vektor oder data.frame/matrix.

Examples

```
na_spalten_entfernen(data.frame(a = c(NA, NA), b = 1:2))
na_spalten_entfernen(data.frame(a = c(NA, NA), b = 1:2), nur_index = TRUE)
```

---

neue_spalten_in_tabelle_anlegen *Anlegen neuer Spalten aus data.frame in einer MySQL-Tabelle*

Description

Fügt die Spalten eines data.frame einer MySQL-Tabelle (ohne Inhalte) hinzu.

Usage

```
neue_spalten_in_tabelle_anlegen(con, tabelle, df)
```

Arguments

con	RODBC-Verbindung.
tabelle	Tabellenname.
df	data.frame mit (ausschließlich) neuen Spalten.

Value

In der Tabelle werden neue leere Spalten angelegt.

Examples

```
## Not run:
testdf1 <- data.frame(spalte1 = rnorm(100),
spalte2 = sample(letters, size = 100, replace = TRUE),
spalte3 = as.POSIXct("2014-01-01 00:00") + 60*0:99)
testdf2 <- data.frame(spalte1 = rnorm(100,2),
                      spalte2 = sample(LETTERS, size = 100, replace = TRUE),
                      spalte3 = as.POSIXct("2014-01-01 00:00") + 60*100:199,
                      spalte4 = sample(1:100),
                      spalte5 = rexp(100))
sqlSave(ch, testdf1, varTypes = c("spalte3" = "datetime"))
neue_spalten_in_tabelle_anlegen(ch, "testdf1", testdf2[,-c(1,2,3)])
sqlSave(ch, testdf2, "testdf1", append = TRUE)
sqlFetch(ch, "testdf1", rownames = FALSE)

## End(Not run)
```

---

outside *Pruefen, ob x ausserhalb des Intervalls [min, max]*

Description

Überprüfen, ob x außerhalb des Intervalls [min, max] bzw. (min, max) liegt.

Usage

```
outside(x, min, max, incl_equal = TRUE)
```

Arguments

x	Zu überprüfender Wert (Skalar oder Vektor).
min	Untere Grenze des Intervalls.
max	Obere Grenze des Intervalls.
incl_equal	Logical, Sollen die Intervallgrenzen eingeschlossen werden.

Details

Überprüft, ob x außerhalb des Intervalls [min, max] (incl_equal = TRUE) bzw (min, max) (falls incl_equal = FALSE).

Value

Wahrheitswert (TRUE/FALSE)

Examples

```
outside(3, 1,2)
outside(2.5, 2,3)
```

`read_wetter` *Funktion zum Einlesen der Wetterdaten.*

Description

Die Funktion `read_wetter` liest die Wetterdaten des Meteorologischen Instituts ein. Falls sich deren Ordner und Benamung und Struktur nicht ändert, muss nur das Argument monat (z.b. "2015_01") angegeben werden

Usage

```
read_wetter(monat, name = "TheresienWetter", ordner,
  header = "Tag Monat Jahr Stunde Minute Aussentemperatur Feuchttemperatur Globalstrahlung Diffuss
  ...)
```

Arguments

`header`	1 string; beinhaltet die Spaltennamen (mit " " getrennt, um sie einfach kopieren zu können).
`...`	Weitere Argumente, die an read.table übergeben werden.
`ordner+name+monat`	ergeben zusammen den Dateinamen

Value

data.frame mit wetterdaten des entsprechenden monats.

`rel_feuchte` *relative Feuchte nach Anleitung des Meteorologischen Instituts berechnen*

Description

Berechnet die relative Feuchte nach Anleitung des Meteorologischen Instituts als rel. Feuchte = Dampfdruck e / Sättigungsdampfdruck E

Usage

```
rel_feuchte(theta_L, theta_strich, p = 1006.6)
```

Arguments

theta_L	Lufttemperatur in °C.
theta_strich	Feuchttemperatur.
p	Druck in hPa (1006.6hPa = 755 Torr). Üblicherweise unabhängig von Temperatur und Feuchte gesetzt.

Value

relative Feuchte.

Examples

```
rel_feuchte(theta_L = -0.1, theta_strich = -0.1)
```

---

rel_feuchte_df *Relative Feuchte aus data.frame berechnen*

Description

Berechnet relative Feuchte aus data.frame nach Anleitung des Meteorologischen Instituts (zur Vereinfachung der Eingabe).

Usage

```
rel_feuchte_df(daten, namen = c("Aussentemperatur", "Feuchttemperatur"), ...)
```

Arguments

daten	data.frame mit Luft- und Feuchttemperatur.
namen	Spaltennamen von Luft- und Feuchttemperatur.

Value

Relative Feuchte.

---

saettigungsdampfdruck *Saettigungsdampfdruck*

Description

Berechnen des Sättigungsdampfdruckes

Usage

```
saettigungsdampfdruck(theta)
```

Arguments

theta	Lufttemperatur für E (Sättigungsdampfdruck bei Lufttemperatur) bzw. Feuchttemperatur für E' (Sättigungsdampfdruck bei Feuchttemperatur)
Sättigungsdampfdruck	bei Lufttemperatur E oder Sättigungsdampfdruck bei Feuchttemperatur E'

Value

Sättigungsdampfdruck

---

sqlGetResults2 *siehe RODBC::sqlGetResults*

Description

siehe RODBC::sqlGetResults

Usage

```
sqlGetResults2(channel, as.is = FALSE, errors = FALSE, max = 0,
  buffsize = 1000, nullstring = NA_character_, na.strings = "NA",
  believeNRows = TRUE, dec = getOption("dec"),
  stringsAsFactors = default.stringsAsFactors(),
  format = "%Y-%m-%d %H:%M:%S", tz = "GMT")
```

---

sqlQuery2 *Variante von* RODBC::sqlQuery*, mit der format und tz der TIMEDATE- bzw. POSIXct-Variablen festgelegt werden kann.*

Description

Aus RODBC Version 1.3-10 abgewandelt (in 1.3-11 keine Veränderung dieser Funktionen entdeckt). Identisch mit sqlQuery/sqlGetResults, nur dass letztere Funktion zusätzlich die Argumente format und tz für die Umwandlung von timestamp-Spalten in der Datenbank- Tabelle in POSIXct hat. Defaults: `format = "%Y-%m-%d %H:%M:%S"` (Datum mit vollständiger Uhrzeit), `tz = "GMT"` (koordinierte Weltzeit). Diese beiden Funktionen sollen verhindern, dass bei der Umwandlung die Zeitzonen CET/CEST (mit Sommer-/Winterzeit) verwendet werden und die Uhrzeit abgeschnitten wird.

Usage

```
sqlQuery2(channel, query, errors = TRUE, ..., rows_at_time)
```

Arguments

...	Weitere Argumente, die an sqlGetResults2 weitergegeben werden, u.a. format und tz.
Rest	Siehe RODBC::sqlQuery.

Details

Soll bei einer erfolglosen Abfrage mit einem Fehler abgebrochen werden siehe sqlQuery2err.

Value

Siehe RODBC::sqlQuery

---

sqlQuery2err *Datenbankabfrage wie sqlQuery2, nur mit Abbruch bei Fehler*

Description

Details siehe sqlQuery2. Die Funktion sqlQuery2err ruft die Funktion sqlQuery2 auf, die eine Abgewandelte Version der Funktion sqlQuery {RODBC} ist, mit Defaults und Übergabemöglichkeit des Datumsformates und der zeitzone für Datetime-Variablen, und überprüft deren Rückgabe auf Fehlermeldungen über sql_error und bricht bei einem Fehler in der Abfrage mit einer Fehlermeldung ab.

Usage

```
sqlQuery2err(...)
```

Arguments

...	Argumente die an sqlQuery2 weitergereicht werden.

Value

Abgefrage Daten aus der Datenbank als data.frame, falls die Abfrage erfolgreich war.

Examples

```
## Not run:
library(RODBC)
ch <- odbcConnect("ansi_system")
sqlQuery2err(ch, "select x7070_01_u138hzg13tem0002mp01
             from glt_hist_oet67 limit 1") # Erfolgreiche Abfrage
sqlQuery2err(ch, "select") # Abbruch wegen Fehler

## End(Not run)
```

sqlQueryGMT *sqlQuery RODBC mit nachtraeglich neu gesetzter Zeitzone*

Description

Abfrage über sqlQuery RODBC, bei der alle datetime-Spalten mit bestimmter Zeitzone formatiert werden (in der Regel "GMT", siehe as.POSIXct)

Usage

```
sqlQueryGMT(channel, ..., tz = "GMT")
```

Arguments

channel,	... Siehe sqlQuery.
tz	Gewünschte Zeitzone.

Details

sqlQuery2 wahrscheinlich die bessere Lösung.

Value

data.frame mit Abfrageinhalt und datetime-Spalten als POSIXct mit gewünschter Zeitzone.

Examples

```
## Not run:
sqlQuery(ch, "select datum from oet67 limit 1")[,]
sqlQueryGMT(ch, "select * from oet67 limit 1")[1,1]
sqlQueryGMT(ch, "select tag, datum, monat, jahr from oet67 limit 1")[1,2]
sqlQueryGMT(ch, "select tag, monat, jahr from oet67 limit 1")

## End(Not run)
```

sql_error *Wirft einen Fehler bei einem aus einer Datenbankabfrage resultierenden Fehler*

Description

Eine Datenbankabfrage über `sqlQuery {RODBC}` oder `sqlQuery2`, die zu einem Fehler führt, hat diesen Fehler als character-Vektor zwar als Rückgabenwert anstelle eines data.frames, aber wirft keinen Fehler. Die Funktion `sql_error` überprüft die Abfrage auf einen Fehler und wirft ggf. einen Fehler.

Usage

```
sql_error(x)
```

Arguments

x	Ergebnis der Datenbankabfrage mit sqlQuery oder sqlQuery2.

Value

nichts, wirft einen Fehler, falls x eine Fehlermeldung als character-Vektor ist.

Examples

```
## Not run:
library(RODBC)
ch <- odbcConnect("ansi_system")
result <- sqlQuery2(ch, "select x7070_01_u138hzg13tem0002mp01
                    from glt_hist_oet67 limit 1")
sql_error(result)
result2 <- sqlQuery2(ch, "select")
sql_error(result2)

## End(Not run)
```

---

tage_zusammenfassen *Tagesdateien wieder zusammenfassen*

Description

Fasst Tagesdatensätze wieder zusammenfassen, indem es aus csv-dateien für jeden Tag eine für jeden Monat macht.

Usage

```
tage_zusammenfassen(quellordner, zielordner, namensteil = NULL)
```

Arguments

quellordner	Ordner, in welchem sich die Tagesdateien befinden.
zielordner	Ordner, in welchen die Monatsdateien sollen.
namensteil	Bestandteil des Namens an welchem die zu verwendeten csv-Dateien erkannt werden sollen.

Details

Es werden nur Dateien aus dem Quellordner ausgewählt, die die Dateiendung .csv haben und deren Namen den im Argument namensteil angegebenen String beinhalten. Falls kein Namensteil angegeben wird (default-Wert = NULL), wird eine Warnung ausgegeben und es werden alle Dateien im Quellordner verwendet.

Value

Eine csv-Datei im Zielordner für jeden Kalendermonat, der in den Dateinamen im Quellordner vorkommt. Das Datum im Namen der Zieldateien ist das früheste Datum im Namen der Dateien, die zu dieser Monatsdatei zusammengefügt werden. Die Zeilennamen (Numerierung) werden nicht in die Datei geschrieben, da sie nicht mehr benötigt werden.

Examples

```
## Not run:
tage_zusammenfassen(quellordner = "C:/historische daten glt 2015_02_20/einzelne_tage/umgebaut",
                    zielordner = "C:/historische daten glt 2015_02_20/umgebaut_zusammen",
                     namensteil = "Oet67")

## End(Not run)
```

---

update_glt2 *Aufbereiten und in MySQL-Datenbank-Schreiben von GLT-Daten*

Description

update_glt2 liest GLT-Daten aus einem Ordner ein, bereitet sie auf und speichert sie in die MySQL-Datenbank ab.

Usage

```
update_glt2(con, tabelle, differenz = NULL, name,
  name_allgemein = "EnMoLMU", ordner,
  ferien = "Daten/sonstige_Daten/ferien.csv", einzeln = FALSE)
```

Arguments

con	RODBC-Verbindung.
tabelle	Tabellenname in der Datenbank (wird angelegt, falls nicht vorhanden, aber Daten vorhanden).
differenz	Spalten für die die Differenzen hinzugefügt werden sollten (Zählerstände). Falls NULL (Default), werden hierfür alle mit bestimmten Eigenschaften gewählt (Differenzen immer >= 0, numerisch,...).
name	Namensteil der csv-Dateien im Ordner, der den Subserver bezeichnet.
name_allgemein	Namensteil der csv-Dateien, der die GLT-Dateien kennzeichnet.
ordner	Pfad zu dem Ordner, in welchem sich die Dateien befinden.
ferien	Pfad zu csv-Datei mit Ferienzeiten ("csv2"-Format, mit Spaltennamen in der ersten Zeile, Komma-Dezimalzahlen, Semikolon-getrennte Spalten und Datum im Format "%d.%m.%Y", also z.B. 22.01.2015), wobei die 1. Spalte die ersten Ferien- tage und die 2. Spalte die letzten Ferientage beinhaltet, falls eine Spalte mit semesterferien (logical) angefügt werden soll. Es kann auch ein data.frame übergeben werden.
einzeln	Sollen die Dateien nacheinander in die Datenbank geschrieben werden (langsamer, aber müsste weniger RAM erfordern).

Details

Falls die Tabelle noch nicht besteht, wird nur der erste (älteste) Datensatz in die neu angelegte Tabelle geschrieben, ansonsten alle neuen Datensätze.

Value

Die entsprechenden Daten werden in die Datenbank geschrieben, die Funktion gibt die neue Zeilenanzahl der Tabelle, NULL (falls keine neuen Daten) und Hinweise, ob eine neue Tabelle angelegt wird bzw. ob neue Daten vorhanden sind, zurück

Examples

```
## Not run: require(RODBC)
ch <- odbcConnect("ansi_system",  DBMSencoding = "utf-8") #"ansi_system ist Name der eingerichte
source('Code_Daten/sqlQuery2_vgl_RODBC.R')
sqlQuery2(ch, "select count(*), min(datum), max(datum) from glt_hist_the46")
system.time({
update_glt2(ch, tabelle = "glt_hist_the46", differenz = NULL, name = "The46",
            name_allgemein = "EnMoLMU", ordner = "Daten\\daten_glt_tagesupdates",
            ferien = "Daten/ferien.csv", einzeln = FALSE)
})

## End(Not run)

## Not run: sqlQuery2(ch, "select count(*), min(datum), max(datum) from glt_hist_oet67")
system.time({
update_glt2(ch, tabelle = "glt_hist_oet67", differenz = NULL, name = "Oet67",
            name_allgemein = "EnMoLMU", ordner = "Daten\\daten_glt_tagesupdates",
            ferien = "Daten/ferien.csv", einzeln = FALSE)
})

## End(Not run)

## Not run: sqlQuery2(ch, "select count(*), min(datum), max(datum) from glt_hist_leo03")
system.time({
  update_glt2(ch, tabelle = "glt_hist_leo03", differenz = NULL, name = "Leo03",
              name_allgemein = "EnMoLMU", ordner = "Daten\\daten_glt_tagesupdates",
              ferien = "Daten/ferien.csv", einzeln = FALSE)
})

## End(Not run)
```

`update_glt_schreiben` *Daten in die Tabelle schreiben*

Description

Von update_glt2 verwendete Funktion zum Schreiben in die Tabelle.

Usage

```
update_glt_schreiben(datensatz, ordner = "", differenz, con, tabelle, ferien,
  neue_tabelle)
```

Arguments

`datensatz`	In die Datenbank-Tabelle zu schreibender csv-Dateiname (character) oder data.frame.

ordner	Pfad zu dem Ordner in dem sich die csv-Datei befindet (nur, falls csv- Datei übergeben wird).
neue_tabelle	Existiert die Tabelle bereits in der Datenbank?
weitere	Weitere Argumente siehe update_glt2.

---

update_wetter *Wetter Tabelle anlegen und fuellen*

Description

Legt Tabelle mit Wetterdaten an bzw. schreibt neue Daten hinein, falls die Tabelle schon in der Datenbank existiert.

Usage

```
update_wetter(con, ordner = "Daten/Wetterdaten", name = "TheresienWetter",
  tabelle = "theresienwetter", spaltennamen = c("jahr", "monat"),
  verschiebung = 0, na = 999.9)
```

Arguments

con	RODBC-Verbindung.
ordner	Pfad zu dem Ordner in welchem sich die (neuen) Daten befinden.
name	Fester Namensteil der Dateien, z.B. "TheresienWetter" für Dateinamen wie TheresienWetter2015_01 (die Dateinamen müssen das Datum in dieser Form beinhalten).
tabelle	Tabellenname in der MySQL-Datenbank.
spaltennamen	Namen der relevanten Datumsspalten in der Tabelle (nicht vollständig implementiert)
verschiebung	um wieviele Tage muss das Datum der Beobachtung in der Tabelle mit dem neuesten Datum verschoben werden, so dass sich ein Datumsbestandteil im Dateinamen ergibt, der neuere Daten beinhaltet (z.B. 31 Tage, falls TheresienWetter2015_01 die Daten von Dezember enthält).
na	Welche Werte fehlenden Werten entsprechen und auf NA gesetzt werden sollen (Vektor mit einem oder mehreren Werten).

Details

Die Dateien im angegebenen Ordner werden danach ausgewählt, ob ihr Name name beinhaltet und 7 Zeichen länger (also das Datum, aber keine Dateiendung angehängt ist) ist, und ob das Datum neu genug ist (also bei verschiebung = 31 der Monat im Dateinamen mindestens 1 höher ist, als bei der Zeile mit dem neuesten Datum in der Tabelle tabelle). Es wird empfohlen, die Dateien mit dem gleichen Datum zu benennen, mit dem auch die Daten enden und für das Argument verschiebung den Default (0) zu belassen.

Value

Legt Tabelle mit Wetterdaten an bzw. schreibt neue Daten hinein.

Examples

```
## Not run:
update_wetter(ch)

## End(Not run)
```

`wetter_dateiname` *Name des Wetterdatensatzes erzeugen*

Description

Die Funktion `wetter_dateiname` erzeugt den Namen des Wetterdatensatzes mit Pfad für einen bestimmten Monat.

Usage

```
wetter_dateiname(jahr, monat, verschiebung = 0, name = "TheresienWetter",
  ordner = "Daten/Wetterdaten", trennung = "_")
```

Arguments

`jahr`	jahreszahl (4-Stellig, Zahl).
`monat`	Monat (2-Stellig, zahl).
`verschiebung`	Verschiebung um Tage (Zahl, 31 für 1 Monat, funktioniert aber nicht für unbegrenzt viele Monate hintereinander, da nicht jeder Monat 31 Tage hat).
`name`	Name der Datei ohne Datums-Zusatz (Default: "TheresienWetter").
`ordner`	Ordner, in welchem die Datei ist (Default: "Daten/Wetterdaten").
`trennung`	Trennzeichen im Namen zwischen Jahr und Monat (Default: "_" für z.B. "2015_01").

Value

Dateiname mit Pfad

Examples

```
## Not run:
# neuestes jahr/monat
jahr_mon <- sqlQuery(ch, "select jahr, monat from theresienwetter where
                     datum = (select max(datum) from theresienwetter)")

letzter <- wetter_dateiname(jahr = jahr_mon[,"jahr"], monat = jahr_mon[, "monat"] )
# um 1 monat verschoben, da wetterdaten immer das wetter vor dem monat in
# welchem sie kommen und nach welchem sie benannt werden:
letzter <- wetter_dateiname(jahr = jahr_mon[,"jahr"], monat = jahr_mon[, "monat"],
                            verschiebung = 31 )

## End(Not run)
```

`wetter_erweitern` *Eingelesenen Wetter-Datensatz erweitern*

Description

Funktion, um eingelesenen Wetter-Datensatz um Datum als Posix ("datum") und 15-Minuten Kategorie ("cut15" mit gesamtem Datum und "cut15min" nur Minuten) und Relative Feuchte zu erweitern.

Usage

```
wetter_erweitern(wetter, relative_feuchte = TRUE, ...)
```

Arguments

`wetter`	Wetterdatensatz als data.frame.
`relative_feuchte`	Soll die relative Feuchte angefügt werden? default = TRUE
`...`	Weitere Argumente die an rel_feuchte_df weitergegeben werden (insbesondere spaltennamen als namen, default-Wert: c("Aussentemperatur", "Feuchttemperatur")).

Value

um 3 oder 4 Spalten erweiterter Wetterdatensatz.

Index

A.1.4 enmoGrafiken

Package 'enmoGrafiken'

September 7, 2015

Title Grafiken zur Erstellung von Berichten zum Energiemonitoring.

Version 1.0

Description
Grafiken zur automatischen Erstellung von Berichten zum universitären Energiemonitoring. Bestandteil von EnMoLMU.

Depends R (>= 3.1.2)

License What license is it under?

LazyData true

Encoding UTF-8

Imports enmoDaten,
RODBC,
ggplot2,
grid,
gridExtra,
lubridate,
reshape2,
gtable,

R topics documented:

`bis_letzter_monat` *String eines Intervalls, das vor letztem Monat endet*

Description

Erzeugt einen String eines Zeit-Intervalls, welches einen festgelegten Anfang hat und am Ende des vorletzten Monats endet

Usage

```
bis_letzter_monat(von = "2013-01-01 00:00:00")
```

Arguments

von	Anfang des Intervalls

Details

Siehe auch `letzter_zeitraum`, `letzter_monat`, `letztes_jahr`, `letzte_2jahre` und `letzte_3jahre` Nutzbar z.B. in den Grafikfunktion `energiesignatur_db`, `hydr_abgleich_db`, `jahresdauerlinien_db` und `tagesverlauf_db` für das Argument between etc.

Anwendungsbeispiel mit `energiesignatur_db` siehe `letzter_monat`.

Value

String des Zeitraumes, der an die Grafikfunktionen übergeben werden kann.

Examples

```
bis_letzter_monat()
```

`cop` *Leistungszahl (LZ)*

Description

Berechnung der Leistungszahl. Die Leistungszahl ε (abgekürzt LZ), bekannt auch unter den englischen Bezeichnungen Energy Efficiency Ratio (kurz EER) für mechanische Kälteanlagen bzw. Coefficient of Performance (kurz COP) für mechanische Wärmepumpen ist das Verhältnis von erzeugter Kälte- bzw. Wärmeleistung zur eingesetzten elektrischen Leistung. (http://de.wikipedia.org/wiki/Leistungszahl)

Usage

```
cop(heizleistung, elektrische_leistung)
```

Arguments

`heizleistung`	Heizleistung $\dot{Q}_H$.
`elektrische_leistung`	elektrische Leistung P_{el}.

Details

Bei elektrischen Wärmepumpen (WP) mit Kältemittel gibt die Leistungszahl bzw. COP das Verhältnis der abgegebenen Heizleistung einer Wärmepumpe zur aufgewendeten elektrischen Leistung des Verdichters an. Eine Leistungszahl von z.B. 4,2 bedeutet, dass von der eingesetzten elektrischen Leistung des Kompressors das 4,2-fache an Wärmeleistung bereitgestellt wird. Anders formuliert kann mit dieser Wärmepumpe aus einem Kilowatt elektrischer leistung 4,2 kW Wärmeleistung zur Verfügung gestellt werden. (http://de.wikipedia.org/wiki/Leistungszahl)

Value

Leistungszahl ϵ_{WP}.

cop_adi *Leistungszahl Waermepumpe*

Description

Bei adiabater Betrachtung der Wärmepumpe ist die Heizleistung die summe aus der extern aufgenommenen Wärmeleistung $\dot{Q}_u$ und der elektrischen Leistung P_{el}. (http://de.wikipedia.org/wiki/Leistungszahl)

Usage

```
cop_adi(waermeleistung, elektrische_leistung)
```

Arguments

waermeleistung	extern aufgenommene Wärmeleistung $\dot{Q}_u$.
elektrische_leistung	elektrische Leistung P_{el}.

Value

Leistungszahl ϵ_{WP}.

cop_plot *Grafiken zur Leistungszahl*

Description

Die Funktion berechnet die Leistungszahl (falls die Heizleistung angegeben wird) oder die Leistungszahl Wärmeleistung (falls die Wärmeleistung angegeben wird) und zeigt deren Verlauf als Balkendiagramm und Loesskurve bzw. als Balkendiagramm die Wochentagdurchschnitte.

Usage

```
cop_plot(datum, elektrische_leistung, wochentag, heizleistung = NULL,
  waermeleistung = NULL, durchschnitt = "Woche")
```

Arguments

datum	Datumsvektor zu den Beobachtungen.
elektrische_leistung	Vektor mit der der Elektrischen Leistung.
wochentag	Vektor mit den Wochentagen zu den Beobachtungen.
heizleistung,waermeleistung	Vektor mit der Heizleistung bzw. Wärmeleistung. Es muss genau einer von beiden angegeben werden. Die Berechnung der Leistungszahl wird für beide nach unterschiedlichen Formeln durchgeführt.
durchschnitt	Welche Zeitabschnitte die einzelnen Balken des Balkendiagramms zeigen sollen (z.B. "Woche", "Monat").

Details

Wird die Heizleistung angegeben, wird die Leistungszahl mit der funktion cop nach der Formel heizleistung/elektrische_leistung berechnet; wird die Wärmeleistung angegeben, so wird die Leistungszahl Wärmepumpe mit der Funktion cop_adi nach der Formel 1+waermeleistung/elektrische_leistung berechnet und dargestellt.

An den Parameter durchschnitt können die Zeitraumwörter sek, min, stunde, tag, woche, monat, quartal und jahr übergeben werden, wobei beliebig Großbuchstaben verwendet werden können.

Value

Grafiken, die den Verlauf der Leistungszahl und die Durchschnitte der einzelnen Wochentage darstellen.

Examples

```
# mit Zufallszahlen:
cop_plot(as.Date("2014-03-01")+1:50, heizleistung = rexp(50),
         elektrische_leistung= 1:50/2, durchschnitt = "Woche",
         wochentag = weekdays(as.Date("2014-03-01")+1:50))
cop_plot(as.Date("2014-03-01")+1:1000, waermeleistung = rexp(1000),
         elektrische_leistung= 1:1000/2, durchschnitt = "Monat",
         wochentag = weekdays(as.Date("2014-03-01")+1:1000))
```

---

cop_plot_db *Grafiken zur Leistungszahl mit Datenbankabfrage*

Description

Die Funktion berechnet die Leistungszahl (falls die Heizleistung angegeben wird) oder die Leistungszahl Wärmeleistung (falls die Wärmeleistung angegeben wird) und zeigt deren Verlauf als Balkendiagramm und Loesskurve bzw. als Balkendiagramm die Wochentagdurchschnitte. Die Daten werden über RODBC aus einer MySQL-Datenbank abgefragt.

Usage

```
cop_plot_db(con, tabelle, elektrische_leistung, wochentag = "wochentag",
  heizleistung = NULL, waermeleistung = NULL, between = NULL,
  durchschnitt = "Woche", datum = "datum")
```

Arguments

con	RODBC-Verbindung.
tabelle	Name der Datenbanktabelle mit den angegebenen Spalten.
elektrische_leistung	Name der Spalte mit der der Elektrischen Leistung.
wochentag	Spalte mit den Wochentagen zu den Beobachtungen. Default: "wochentag".
heizleistung,waermeleistung	Name der Spalte mit der Heizleistung bzw. Wärmeleistung. Es muss genau eine von beiden angegeben werden. Die Berechnung der Leistungszahl wird für beide nach unterschiedlichen Formeln durchgeführt.

between	Zeitraum für den die Daten abgefragt und für die Grafiken verwendet werden sollen in der Form "'2013-01-30' and '2014-12-24'".
durchschnitt	Welche Zeitabschnitte die einzelnen Balken des Balkendiagramms zeigen sollen (z.B. "Woche", "Monat").
datum	Name der Datumsspalte in der Datenbanktabelle.

Details

Wird die Heizleistung angegeben, wird die Leistungszahl mit der funktion cop nach der Formel heizleistung/elektrische_leistung berechnet; wird die Wärmeleistung angegeben, so wird die Leistungszahl Wärmepumpe mit der Funktion cop_adi nach der Formel 1+waermeleistung/elektrische_leistung berechnet und dargestellt.

An den Parameter durchschnitt können die Zeitraumwörter sek, min, stunde, tag, woche, monat, quartal und jahr übergeben werden, wobei beliebig Großbuchstaben verwendet werden können.

Diese Funktion übernimmt in erster Linie die Abfrage aus der Datenbank und ruft dann die Funktion cop_plot mit diesen Daten auf.

Es werden die nach Tagen aggregierten Daten aus der Datenbank abgefragt.

Automatische Anpassung der Zeiträume siehe Beispiel zu letzter_monat.

Value

Grafiken, die den Verlauf der Leistungszahl und die Durchschnitte der einzelnen Wochentage darstellen.

Examples

```
## Not run:
require(RODBC)
ch <- odbcConnect("ansi_system")
# Da noch keine sinnvollen Variablen verfuegbar waren, wurden hier
# zur Demonstration irgendwelche verwendet:
cop_plot_db(ch, "glt_hist_oet67", "x7079_01_t013klt01waz0002zw01", "wochentag",
            waermeleistung = "x7070_01_u138hzg03wmz0001zw04",
            durchschnitt = "Monat")
cop_plot_db(ch, "glt_hist_oet67", "x7079_01_t013klt01waz0002zw01", "wochentag",
            heizleistung = "x7079_01_t013klt01waz0002zw01",
            between = "'2013-01-01' and '2014-01-01'")

## End(Not run)
```

---

cop_plot_db_t *Grafiken zur Leistungszahl mit Datenbankabfrage*

Description

Die Funktion berechnet die Leistungszahl (falls die Heizleistung angegeben wird) oder die Leistungszahl Wärmeleistung (falls die Wärmeleistung angegeben wird) und zeigt deren Verlauf als Balkendiagramm und Loesskurve bzw. als Balkendiagramm die Wochentagdurchschnitte. Die Daten werden über RODBC aus einer MySQL-Datenbank abgefragt. Entspricht cop_plot_db, aber beinhaltet zusätzlich eine Ausnahmebehandlung, so dass im Falle eines Fehlers die Berichte dennoch erstellt werden.

Usage

```
cop_plot_db_t(...)
```

Arguments

...	Argumente, die an die Funktion cop_plot_db weitergereicht werden.

---

datenbereinigung *Einfache Datenbereinigung in den Grafikfunktionen*

Description

Um die Grafikfunktionen mit den echten fehlerhaften Daten verwenden zu können ist oft eine Bereinigung notwendig, da sonst nichts zu erkennen ist. Dies kann mit der Funktion datenbereinigung in einer simplen Form durchgeführt werden.

Usage

```
datenbereinigung(df, variable, untere_grenze = NULL, obere_grenze = NULL,
  quantil_box = NULL, faktor_oben = NULL, faktor_unten = NULL)
```

Arguments

df	data.frame, der bereinigt werden soll.
variable	Name der Variable, deren Werte überprüft werden sollen (gegenwärtig nur 1 Variable möglich).
untere_grenze	Grenze unter der alle darunterliegenden Daten entfernt werden sollen.
obere_grenze	Grenze über welcher alle darüberliegenden Daten entfernt werden sollen.
quantil_box	Unteres und oberes Quantil für das ein Interquantilsabstand berechnet werden soll. Die obere/untere Grenze des erlaubten Intervalls wird durch unteres Quantil - Interquantilsabstand*faktor_unten und oberes Quantil + Interquantilsabstand*faktor_oben berechnet. Für quantil_box = c(0.25, 0.75) würde z.B. der Interquartilsabstand berechnet werden.
faktor_unten, faktor_oben	Siehe quantil_box.

Details

Soll ein Argument für eine Grenze nicht gesetzt werden, so muss NULL angegeben werden (Default).

Value

data.frame df ohne die Zeilen mit den Werten, die mindestens eine der Grenzen überschreiten.

Examples

```
daten <- data.frame(a = 1:20, b = c(0, 1, -3, 200, 12, 5, 3, -100, 1, 3,
                                    4, 7, 1,1,1, 2, 3, 5, 1, 0))
datenbereinigung(daten, "b", untere_grenze = 0, quantil_box = c(0.1, 0.9),
                 faktor_oben = 1.5)
# Grenzen näherungsweise (0,5], also Werte mit 0 werden bereits entfernt
datenbereinigung(daten, "b", untere_grenze = 1e-100, obere_grenze = 5)
```

---

dt2eng_zeitraum *Zeitraumangaben umwandeln*

Description

Die Funktion wandelt deutsche Zeitraumangaben (Großschreibung wird ignoriert) in englische um, um sie an Funktion übergeben zu können. Falls es sich bereits um einen englischen Ausdruck (aus der breaks-Liste von `cut.Date`, siehe `?cut.Date`) handelt, wird dieser unverändert zurückgegeben.

Usage

```
dt2eng_zeitraum(zeitraum)
```

Arguments

`zeitraum`	"sek", "min", "stunde", etc.

Details

Es können die Zeitraumwörter sek, min, stunde, tag, woche, monat, quartal und jahr übergeben werden, wobei beliebig Großbuchstaben verwendet werden können.

Value

Englischer Ausdruck für den Zeitraum, wie er an `cut.Date` übergeben werden kann.

Examples

```
dt2eng_zeitraum("tag")
dt2eng_zeitraum("Woche")
```

---

energiesignatur *Energiesignatur*

Description

Die Funktion teilt den Datensatz mit "alten" Daten in zwei Teile auf, welche über oder unter einem bestimmten Grenzwert der Einflussgröße liegen und schätzt ein lineares Modell für die Daten unterhalb der Grenze und eine Konstante für die Daten darüber. Dazu wird ein Prädiktionsintervall (lineares Modell) bzw. Quantile einer Normalverteilung (Konstante) geschätzt und überprüft, ob die neuen Daten innerhalb der Intervalle liegen. Dies wird grafisch dargestellt.

Usage

```
energiesignatur(formel, daten, neue_daten, grenze = 15, level = 0.999,
  col.alt = "black", col.neu = "green", col.ausreisser = "red",
  col.intervall = "blue", col.erwartungswert = "blue", ggplot = TRUE,
  plot = TRUE, print = TRUE, main = NULL,
  weitere_variablen = c("Datum"), xlab = "Au<df>entemperatur",
  ylab = "W<e4>rmemenge", legend = TRUE, modelltyp = c("lm", "rq"),
  faktor = 3)
```

Arguments

formel	Formel für Regression (lineares Modell) und Konstante, z.B. y~x oder waermemenge~aussentemperatur, wobei vor dem "~" die Ziel- und nach dem "~" die Einflussgröße kommt (die Variablen müssen in daten und neue_daten vorhanden sein)
daten	Alte Daten aus denen die Modelle geschätzt werden sollen. Sie müssen die Variablen aus formel sowie "Datum" enthalten, falls der default-Wert des Arguments weitere_variablen nicht verändert wird.
neue_daten	Daten für die Erwartungswert und Prädiktionsintervalle vorhergesagt werden, anhand derer näher zu betrachtende Werte definiert werden.
grenze	Unter welchem Einflussgrößenwert soll das lineare Modell geschätzt werden bzw. über welchem Wert die Konstante.
level	wird an predict.lm übergeben. Bezeichnet die Wahrscheinlichkeit, dass eine neue Beobachtung innerhalb des Prädiktionsintervalles des linearen Modells liegt. Dieser Wert wird auch für das Intervall um den konstanten Mittelwert verwendet (quantile der Normalverteilung)
col.alt	Farbe der Beobachtungen aus daten (default-Wert: "black")
col.neu	Farbe der Beobachtungen aus neue_daten (default-Wert: "green")
col.ausreisser	Farbe der Beobachtungen aus neue_daten, die außerhalb der Prädiktions-Intervalle liegen (default-Wert: "red")
col.intervall	Farbe der Prädiktions-Intervall-Grenzen (default-Wert: "blue")
col.erwartungswert	Farbe der Erwartungswert-Linien (default: "blue")
ggplot	logical (TRUE oder FALSE). Sollen die Grafiken mit ggplot2 oder mit den R-Standardgrafiken erstellt werden. Empfohlen: ggplot.
plot	logical. Soll die Grafik gezeichnet werden. Falls ggplot=TRUE wird auf jeden Fall ein ggplot-Object zurückgegeben, mit dem die Grafik nachträglich gezeichnet werden kann.
print	logical. Sollen die Ausreißer auf der Konsole ausgegeben werden?
main	Titel der Grafik (default = NULL)
weitere_variablen	Die Ausreißer mit den Variablen aus formel und die Spalten weitere_variablen werden auf der Console ausgegeben und von der Funktion zurückgegeben.
xlab,ylab	x-/y-Achsenbeschriftung.
legend	Soll eine Legende eingezeichnet werden (noch unimplementiert).

`modelltyp` Soll ein Lineares Modell (lm) oder ein Quantilregressions-Modell (rq) verwendet werden? Quantilregression kann eine stabilere Schätzung liefern, in bestimmten Situationen aber auch deutlich schlechtere Ergebnisse liefern. Gegenwärtig wird lm mit entsprechender Datenbereinigung und, falls im Einzelfall nötig, ein Ausweichen auf rq empfohlen. Muss die Modellschätzung ohne manuelle Anpassung funktionieren bietet rq vermutlich ein größeres Potential bei entsprechender Weiterentwicklung.

`faktor` Beim modelltyp rq werden die Intervallgrenzen mit der Formel 75%-Quantil + (75%-Quantil - Median)*Faktor etc. berechnet, da zu hohe/niedrige Quantile zu sehr von einzelnen Datenpunkten beeinflusst werden und damit immer schlechter als die Prädiktionsintervalle des Linearen Modells wären. Ein vernünftiger Wert für Faktor ist stark abhängig von der jeweiligen Datensituation.

Details

Mit den Standardeinstellungen werden die Beobachtungen aus dem Datensatz `daten`, mit dem das Modell geschätzt wird als schwarze Punkte und die Beobachtungen aus dem Datensatz `neue_daten` als grüne Punkte gezeichnet. Die grün gezeichneten Beobachtungen werden zur Vorhersage des Erwartungswertes und der Prädiktionsintervalle verwendet, wobei für den linearen Abschnitt ein Prädiktionsintervall mithilfe des geschätzen linearen Modells und für den konstanten Abschnitt der (nicht bedingte) Erwartungswert und zwei Quantile (0.5*(1-level), level+0.5*(1-level)) geschätzt werden. Erwartunswerte und Prädiktionsintervalle werden in blau eingezeichnet. Punkte aus dem Datensatz `neue_daten`, die außerhalb der Prädiktionsintervalle liegen, werden als Ausreißer und somit näher zu betrachtende Werte angesehen und in rot eingezeichnet und getrennt nach linearem und konstantem Abschnitt auf der Console ausgegeben.

Value

Grafik, falls plot = TRUE liste mit 2 oder 3 Elementen (falls ggplot = TRUE): ausreisser_linearer_abschnitt: neue_daten mit den Variablen aus formel und weitere_variablen und den Beobachtungen, die außerhalb des Prädiktionsintervall des Abschnittes mit dem linearen Modell liegen, also Ausreißer sind. ausreisser_konstanter_abschnitt: neue_daten mit den Variablen aus formel und weitere_variablen und den Beobachtungen, die außerhalb des Prädiktionsintervall des konstanten Abschnittes liegen, also Ausreißer sind. ggplot-Objekt: falls ggplot = TRUE, kann mit diesem Rückgabewert die Grafik mit print(<rückgabe>[[3]]) gezeichnet werden. Falls daten zuwenige Beobachtungen hat (< 1), wird stattdessen für alle drei Elemente die Zeichenkette bzw. Grafik mit dem Hinweis darauf zurückgegeben.

Examples

```
## Not run:
library(RODBC)
ch <- odbcConnect("ansi_system",  DBMSencoding = "utf-8")
sqlQuery2(ch, "select min(datum), max(datum) from oet67")
daten_alt <- sqlQuery2(ch,"select min(datum) as datum,
    sum(diff_x7070_01_u138hzg03wmz0001zw01) as waermemenge,
    avg(x7070_01_u138hzg03tem0003mp01) as aussentemperatur from glt_hist_oet67
    where jahr < 2015 group by jahr, monat, tag")
head(daten_alt)
daten_neu <- sqlQuery2(ch,"select min(datum) as datum,
    sum(diff_x7070_01_u138hzg03wmz0001zw01) as waermemenge,
    avg(x7070_01_u138hzg03tem0003mp01) as aussentemperatur from glt_hist_oet67
    where jahr >= 2015 group by jahr, monat, tag")
head(daten_neu)
```

```
energiesignatur(waermemenge~aussentemperatur, daten_alt, daten_neu,
    weitere_variablen = c("datum"), level = .50,
    main = "diff_x7070_01_u138hzg03wmz0001zw01")
energiesignatur(waermemenge~aussentemperatur, daten_alt, daten_neu,
    weitere_variablen = c("datum"), level = .90, ggplot=F)

## End(Not run)
```

`energiesignatur_aufgeteilt` *Energiesignatur aufgeteilt nach Wochenende und Semesterferien*

Description

Die Funktion schätzt eine Energiesignatur unter Verwendung der Funktion `energiesignatur` aufgeteilt nach Wochenende/Werktag und Semester/Ferien, wobei die Daten als data.frame übergeben werden.

Usage

```
energiesignatur_aufgeteilt(formel, daten, neue_daten, ggplot = TRUE,
  weitere_variablen = c("Datum", "Wochenende", "Semesterferien"), ...)
```

Arguments

`formel`	Formel für Regression (lineares Modell) und Konstante, z.B. y~x oder waermemenge~aussentemperatur, wobei vor dem "~" die Ziel- und nach dem "~" die Einflussgröße kommt (die Variablen müssen in daten und neue_daten vorhanden sein)
`daten`	Alte Daten aus denen die Modelle geschätzt werden sollen. Sie müssen die Variablen aus formel sowie "Datum" enthalten, falls der default-Wert des Arguments weitere_variablen nicht verändert wird.
`ggplot`	logical (TRUE oder FALSE). Sollen die Grafiken mit ggplot2 oder mit den R-Standardgrafiken erstellt werden.
`weitere_variablen`	Die Ausreißer mit den Variablen aus formel und die Spalten weitere_variablen werden auf der Console ausgegeben und von der Funktion zurückgegeben.
`grenze`	Unter welchem Einflussgrößenwert soll das lineare Modell geschätzt
`level`	Wird an predict.lm übergeben. Bezeichnet die Wahrscheinlichkeit, dass eine neue Beobachtung innerhalb des Prädiktionsintervalles des linearen Modells liegt. Dieser Wert wird auch für das Intervall um den konstanten Mittelwert verwendet (quantile der Normalverteilung)
`col.alt`	Farbe der Beobachtungen aus daten (default-Wert: "black")
`col.neu`	Farbe der Beobachtungen aus neue_daten (default-Wert: "green")
`col.ausreisser`	Farbe der Beobachtungen aus neue_daten, die außerhalb der Prädiktions-Intervalle liegen (default-Wert: "red")
`col.intervall`	Farbe der Prädiktions-Intervall-Grenzen (default-Wert: "blue")
`col.erwartungswert`	Farbe der Erwartungswert-Linien (default: "blue")

plot	logical. Soll die Grafik gezeichnet werden. Falls ggplot=TRUE wird auf jeden Fall ein ggplot-Object zurückgegeben, mit dem die Grafik nachträglich gezeichnet werden kann.
print	logical. Sollen die Ausreißer auf der Konsole ausgegeben werden?
main	Titel der Grafik (default = NULL)

Details

daten und neue_daten müssen Variablen Wochenende und Semesterferien beinhalten

Value

nur Grafik

`energiesignatur_aufgeteilt_db`

Energiesignatur aufgeteilt nach Wochenende und Werktage mit Daten aus der Datenbank

---

Description

Diese Funktion zeichnet eine Energiesignatur aufgeteilt nach Wochenende und Werktage mit Daten aus der Datenbank und gibt eine liste der Ausreißer aus.

Usage

```
energiesignatur_aufgeteilt_db(..., print = TRUE)
```

Arguments

...	Weitere Argumente, welche an energiesignatur übergeben werden.
print	Sollen die Ausreißer ausgegeben werden (logical).
con	RODBC-Verbindung.
tabelle	Name der Tabelle in der Datenbank, in der sich die Variablen befinden.
waerme	Variablennamen der Variable für die y-Achse (Wärmemenge, idR beginnend mit "diff_")
aussentemp	Variablenname der Variable für die x-Achse (Außentemperatur)
alt	Zeitraum aus dem das Modell geschätzt werden soll als character mit den Datumsangaben zusätzlich in einfachen Anführungsstrichen (z.B. "'2014-01-01' and '2015-01-01'").
neu	Zeitraum der auf Ausreißer überprüft werden soll.
bedingung	Character-Skalar, der weitere Bedingung für die Abfrage der zu verwendenden Daten angibt (sowohl für alt, als auch neu), z.B. bedingung = "and wochenende = 'Werktag'".

Value

Liste mit 2 Elementen, welche die Rückgabewerte (wieder Listen) der einzelnen Abschnittsweisen Modelle sind (invisible).

Examples

```
## Not run:
library(RODBC)
ch <- odbcConnect("ansi_system",  DBMSencoding = "utf-8")
energiesignatur_aufgeteilt_db(ch, "glt_hist_oet67",
  "diff_x7070_01_u138hzg03wmz0001zw01", "x7070_01_u138hzg03tem0003mp01",
  "'2014-01-01' and '2015-01-01'", "'2015-01-02' and '2015-02-30'",
  aussentemp_between="-4 and 20")

## End(Not run)
```

---

energiesignatur_aufgeteilt_db_ferien

Energiesignatur aufgeteilt nach Wochenende und Ferien mit Daten aus der Datenbank

---

Description

Diese Funktion zeichnet eine Energiesignatur aufgeteilt sowohl nach Wochenende und Werktage, als auch Semesterferien und Vorlesungszeit mit Daten aus der MySQL-Datenbank und gibt eine liste der Ausreißer aus.

Usage

```
energiesignatur_aufgeteilt_db_ferien(..., print = TRUE)
```

Arguments

`...`	Weitere Argumente, welche an `energiesignatur` übergeben werden.
`print`	Sollen die Ausreißer ausgegeben werden (logical).
`con`	RODBC-Verbindung.
`tabelle`	Name der Tabelle in der Datenbank, in der sich die Variablen befinden.
`waerme`	Variablennamen der Variable für die y-Achse (Wärmemenge, idR beginnend mit "diff_")
`aussentemp`	Variablenname der Variable für die x-Achse (Außentemperatur)
`alt`	Zeitraum aus dem das Modell geschätzt werden soll als character mit den Datumsangaben zusätzlich in einfachen Anführungsstrichen (z.B. "'2014-01-01' and '2015-01-01'").
`neu`	Zeitraum der auf Ausreißer überprüft werden soll.
`bedingung`	Character-Skalar, der weitere Bedinung für die Abfrage der zu verwendenden Daten angibt (sowohl für alt, als auch neu), z.B. bedingung = "and wochenende = 'Werktag'".

Details

Interpretation siehe details von `energiesignatur`.

Value

Liste mit 2 Elementen, welche die Rückgabewerte (wieder Listen) der einzelnen Abschnittsweisen Modelle sind (invisible).

Examples

```
## Not run:
library(RODBC)
ch <- odbcConnect("ansi_system",  DBMSencoding = "utf-8")
energiesignatur_aufgeteilt_db_ferien(ch, "glt_hist_oet67",
"diff_x7070_01_u138hzg03wmz0001zw01", "x7070_01_u138hzg03tem0003mp01",
"'2014-01-01' and '2015-01-01'", "'2015-01-02' and '2015-02-30'")

## End(Not run)
```

---

energiesignatur_aufgeteilt_db_ferien_t

Energiesignatur aufgeteilt nach Wochenende und Ferien mit Daten aus der Datenbank

---

Description

Diese Funktion zeichnet eine Energiesignatur aufgeteilt sowohl nach Wochenende und Werktage, als auch Semesterferien und Vorlesungszeit mit Daten aus der MySQL-Datenbank und gibt eine liste der Ausreißer aus. Entspricht `energiesignatur_aufgeteilt_db_ferien`, aber beinhaltet zusätzlich eine Ausnahmebehandlung, so dass im Falle eines Fehlers die Berichte dennoch erstellt werden.

Usage

```
energiesignatur_aufgeteilt_db_ferien_t(...)
```

Arguments

`...`	Argumente, die an die Funktion `energiesignatur_aufgeteilt_db_ferien` weitergereicht werden.

---

energiesignatur_aufgeteilt_db_t

Energiesignatur aufgeteilt nach Wochenende und Werktage mit Daten aus der Datenbank

---

Description

Diese Funktion zeichnet eine Energiesignatur aufgeteilt nach Wochenende und Werktage mit Daten aus der Datenbank und gibt eine liste der Ausreißer aus. Entspricht `energiesignatur_aufgeteilt_db`, aber beinhaltet zusätzlich eine Ausnahmebehandlung, so dass im Falle eines Fehlers die Berichte dennoch erstellt werden.

Usage

```
energiesignatur_aufgeteilt_db_t(...)
```

Arguments

...	Argumente, die an die Funktion energiesignatur_aufgeteilt_db weitergereicht werden.

---

energiesignatur_db *Zeichnet eine Energiesignatur mit Daten aus einer MySQL-Datenbank.*

Description

Zeichnet eine Energiesignatur, wie energiesignatur, nur werden die Daten aus der MySQL-Datenbank geholt.

Usage

```
energiesignatur_db(con, tabelle, waerme, aussentemp, alt, neu, bedingung = "",
  bereinigung = list(untere_grenze = 0, quantil_box = c(0.1, 0.9), faktor_oben
  = 5), aussentemp_between = "-40 and 45", ...)
```

Arguments

con	RODBC-Verbindung.
tabelle	Name der Tabelle in der Datenbank, in der sich die Variablen befinden.
waerme	Variablennamen der Variable für die y-Achse (Wärmemenge, idR beginnend mit "diff_")
aussentemp	Variablenname der Variable für die x-Achse (Außentemperatur)
alt	Zeitraum aus dem das Modell geschätzt werden soll als character mit den Datumsangaben zusätzlich in einfachen Anführungsstrichen (z.B. "'2014-01-01' and '2015-01-01'").
neu	Zeitraum der auf Ausreißer überprüft werden soll.
bedingung	Character-Skalar, der weitere Bedinung für die Abfrage der zu verwendenden Daten angibt (sowohl für alt, als auch neu), z.B. bedingung = "and wochenende = 'Werktag'".
bereinigung	Liste mit Argumenten, die an die Funktion datenbereinigung weitergegeben werden, um Werte, die wahrscheinlich falsch sind, zu löschen. Falls keine Datenbereinigung gewünscht ist, muss bereinigung = NULL angegeben werden. Der Default-Wert ist nicht NULL, es wird also per Default eine Bereinungung durchgeführt. Die Bereinigung wird nur für die Variable waerme des Datensatzes alt (mit dem das Modell geschätzt wird) durchgeführt, da Ausreißer in den neuen Daten ja gerade das sind, was man durch diese Grafik finden will.
aussentemp_between	Intervall, in dem die (unaggregierten) Werte der Außentemperatur als korrekt eingestuft werden. Alle Werte, die außerhalb dieses Intervalls liegen, werden durch NULL (fehlend in MySQL) ersetzt und gehen somit nicht in die Berechnung der Tagesdurchschnitte ein. Das Default-Intervall liegt bei "-40 and 45", was einem etwas größeren Bereich als der bis jetzt in Deutschland gemessenen Extremtemperaturen entspricht.

... Weitere Argumente, welche an energiesignatur übergeben werden.

Details

Interpretation siehe details von energiesignatur.

Automatische Anpassung der Zeiträume siehe Beispiel zu letzter_monat.

Value

Grafik und Ausreißer. Letztere werden auf der Konsole ausgegeben.

Examples

```
## Not run:
library(RODBC)
ch <- odbcConnect("ansi_system",  DBMSencoding = "utf-8")
energiesignatur_db(ch, "glt_hist_oet67", "diff_x7070_01_u138hzg03wmz0001zw01",
   "x7070_01_u138hzg03tem0003mp01", "'2011-01-01' and '2015-01-01'",
   "'2015-01-02' and '2016-01-01'")
# mit Quantilregression zumindest oben besser (unten total daneben)
energiesignatur_db(ch, "glt_hist_oet67", "diff_x7070_01_u138hzg03wmz0001zw01",
   "x7070_01_u138hzg03tem0003mp01", "'2011-01-01' and '2015-01-01'",
   "'2015-01-02' and '2016-01-01'", modelltyp="rq", faktor = 2.5)
# Beobachtungen, die genau 0 sind ausschlieÃŸen und Obergrenze hÃ¶her
energiesignatur_db(ch, "glt_hist_oet67", "diff_x7070_01_u138hzg03wmz0001zw01",
   "x7070_01_u138hzg03tem0003mp01", "'2011-01-01' and '2015-01-01'",
   "'2015-01-02' and '2016-01-01'",
   bereinigung = list(untere_grenze = 0+1e-100,
                      quantil_box = c(0.1, 0.9),
                      faktor_oben = 10)) # beobachtungen, die genau 0 sind ausschlieÃŸen
energiesignatur_db(ch, "glt_hist_oet67", "diff_x7070_01_u138hzg03wmz0001zw01",
   "x7070_01_u138hzg03tem0003mp01", "'2013-05-01' and '2015-01-01'",
   "'2015-01-02' and '2016-01-01'", level = .7)
energiesignatur_db(ch, "oet67", "diff_x7070_01_u138hzg03wmz0001zw01",
   "x7070_01_u138hzg03tem0003mp01", "'2011-01-01' and '2015-01-01'",
   "'2015-01-02' and '2016-01-01'",
                         bedingung = "and wochenende = 'Wochenende'")
energiesignatur_db(ch, "glt_hist_oet67", "diff_x7070_01_u138hzg03wmz0001zw01",
   "x7070_01_u138hzg03tem0003mp01", "'2013-05-20' and '2013-11-30'",
   "'2013-12-01' and '2013-12-31'",
                         bedingung = "and wochenende = 'Wochenende'",
                         aussentemp_between = "0 and 20")
energiesignatur_db(ch, "glt_hist_oet67", "diff_x7070_01_u138hzg03wmz0001zw01",
   "x7070_01_u138hzg03tem0003mp01", "'2013-05-20' and '2014-12-31'",
   "'2015-01-01' and '2015-02-19'",
                         bedingung = "and wochenende = 'Wochenende'
                         and x7070_01_u138hzg03tem0003mp01 between -20 and 17")

## End(Not run)
```

energiesignatur_db_t *Zeichnet eine Energiesignatur mit Daten aus einer MySQL-Datenbank.*

Description

Zeichnet eine Energiesignatur, wie energiesignatur, nur werden die Daten aus der MySQL-Datenbank geholt. Entspricht energiesignatur_db, aber beinhaltet zusätzlich eine Ausnahmebehandlung, so dass im Falle eines Fehlers die Berichte dennoch erstellt werden.

Usage

```
energiesignatur_db_t(...)
```

Arguments

...	Argumente, die an die Funktion energiesignatur_db weitergereicht werden.

hydr_abgleich *Hydraulischer Abgleich*

Description

Grafische Auswertungen zum Hydraulischen Abgleich

Usage

```
hydr_abgleich(datum, t_rl_einzel, t_aussen, t_vl = NULL, t_rl = NULL,
  einzeln = TRUE, breaks = names(labels), labels = NULL)
```

Arguments

datum	Vektor mit dem Datum der Beobachtungen.
t_rl_einzel	Eine Reihe von Rücklauftemperaturen als matrix oder data.frame.
t_aussen	Vektor mit der Außentemperatur.
t_vl	Vektor mit einer einzelnen Vorlauftemperatur.
t_rl	Vektor mit einer einzelnen Rücklauftemperatur.
einzeln	Sollen die einzelne Rücklauftemperatur und Vorlauftemperatur vs. Außentemperatur auch geplottet werden?
breaks,labels	labels kann alternative Bezeichnungen für die Legende für die Variablen in breaks (Spaltennamen von t_rl_einzel) enthalten. Wird labels nicht angegeben, werden die ursprünglichen Variablennamen verwendet, wird nur breaks nicht angegeben, müssen die einzelnen Einträge in labels mit den Spaltennamen von t_rl_einzel benannt sein. breaks und labels wird an scale_colour_discrete übergeben.

Value

Grafik.

Examples

```
## Not run:
require(RODBC)
ch <- odbcConnect("ansi_system")
# tageswerte
daten <- sqlQuery2(ch, "select min(datum) as datum,
                avg(x7070_01_u138hzg03tem0003mp01) as aussentemp,
                avg(x7070_01_u138hzg03tem0001mp01) as vorlauf,
                avg(x7070_01_u138hzg03tem0002mp01) as ruecklauf
                from glt_hist_oet67 group by jahr, monat, tag")
head(daten)
with(daten, hydr_abgleich(datum=datum, t_rl_einzel = daten[,2:4],
                  t_vl = vorlauf, t_rl = ruecklauf, t_aussen = aussentemp))
with(daten, hydr_abgleich(datum=datum, t_rl_einzel = daten[,2:4],
                  t_vl = vorlauf, t_rl = ruecklauf, t_aussen = aussentemp,
                  einzeln = FALSE))
with(daten, hydr_abgleich(datum=datum, t_rl_einzel = daten[,2:4],
                  t_rl = ruecklauf, t_aussen = aussentemp))
with(daten, hydr_abgleich(datum=datum, t_rl_einzel = daten[,2:4],
                  t_vl = vorlauf, t_aussen = aussentemp,
                  einzeln = TRUE))
# minutenwerte
daten2 <- sqlQuery2(ch, "select datum,
                x7070_01_u138hzg03tem0003mp01 as aussentemp,
                x7070_01_u138hzg03tem0001mp01 as vorlauf,
                x7070_01_u138hzg03tem0002mp01 as ruecklauf
                from glt_hist_oet67 where jahr = 2015 and monat = 1 and
                 tag between 3 and 10")
with(daten2, hydr_abgleich(datum=datum, t_rl_einzel = daten2[,2:4],
                   t_vl = vorlauf, t_rl = ruecklauf, t_aussen = aussentemp))
# Beispiel mit Zufallszahlen
hydr_abgleich(datum = as.Date("2014-03-02")+0:19,
       t_rl_einzel = as.data.frame(matrix(rnorm(100, 30, 20), ncol=5)),
       t_vl = rnorm(20, 40, 10),
       t_rl = rnorm(20, 30, 15), t_aussen = rnorm(20, 15, 20))
close(ch)

## End(Not run)
```

---

hydr_abgleich_db *Grafiken zu Hydraulischem Abgleich mit Daten aus einer Datenbank*

Description

Die Funktion erstellt Grafiken zum Hydraulischen Abgleich und verwendet dafür Daten aus einer MySQL-Datenbanktabelle.

Usage

```
hydr_abgleich_db(con, tabelle, variablen, aussentemp, takt = "tag",
  datum = "datum", t_vl = NULL, t_rl = NULL, einzeln = FALSE,
  between = NULL, kurzbezeichnungen = NULL,
  aussentemp_between = "-40 and 45")
```

Arguments

con	RODBC-Verbindung.
tabelle	Name der Datenbanktabelle.
variablen	Character-Vektor mit den Spaltennamen für die Temperaturmesswerte ohne Außentemperatur und Datum.
aussentemp	Name der Spalte mit der Außentemperatur.
takt	Zeitintervalle, die Verwendet werden sollen. Möglich sind Minutentakt ("min"), 15-Minutentakt ("15 min"), Tage ("tag"), und Monate ("mon"). Es wird dabei immer das älteste Datum und der Durchschnitt der Temperaturen verwendet.
datum	Name der Datumsspalte in der Datenbanktabelle.
t_vl	Variablenname einer einzelnen Vorlauftemperatur, die gegen die Außentemperatur geplottet wird. Das Argument einzeln muss dafür TRUE sein (sonst wird diese Grafik nicht angezeigt).
t_rl	Variablenname einer einzelnen Rücklauftemperatur, die gegen die Außentemperatur geplottet wird. Das Argument einzeln muss dafür TRUE sein (sonst wird diese Grafik nicht angezeigt).
einzeln	Soll die rechte Hälfte des Bildes für die einzelne Vorlauf-/ Rücklauftemperatur reserviert werden.
between	Welcher Zeitraum soll dargestellt werden in der Form "'2015-01-01 00:00:00' and '2015-04-04 23:59:00'".
kurzbezeichnungen	data.frame oder Pfade und Name zu einer csv-Datei, die IP.Kommentar für die einzelnen Variablennamen enthält. Falls die Spalte IP.Kommentar nicht gefunden wurde oder darin kein Eintrag für eine bestimmte Variable vorhanden ist, wird die Spaltenbezeichnung in der Tabelle abgekürzt (siehe spalten_suchen und spaltenuebersicht.
aussentemp_between	Intervall, in dem die (unaggregierten) Werte der Außentemperatur als korrekt eingestuft werden. Alle Werte, die außerhalb dieses Intervalls liegen, werden durch NULL (fehlend in MySQL) ersetzt und gehen somit nicht in die Berechnung der Tagesdurchschnitte etc. ein. Das Default-Intervall liegt bei "-40 and 45", was einem etwas größeren Bereich als der bis jetzt in Deutschland gemessenen Extremtemperaturen entspricht.

Details

Automatische Anpassung der Zeiträume siehe Beispiel zu letzter_monat.

Value

Die Grafik wird erstellt.

Examples

```
## Not run:
require(RODBC)
ch <- odbcConnect("ansi_system")
hydr_abgleich_db(ch, "glt_hist_oet67",
              c("x7070_01_u138hzg13tem0002mp01",
                "x7070_01_u138hzg13tem0001vi03",
```

```
                 "x7070_01_u138hzg13tem0001vi02"),
                 "x7070_01_u138hzg03tem0003mp01",takt = "15 min",
                between = "'2015-01-01' and '2015-03-01 23:59'",
                kurzbezeichnungen = "spaltenuebersicht_oet67.csv",
                aussentemp_between = "-5 and 10")
hydr_abgleich_db(ch, "glt_hist_oet67",
              c("x7070_01_u138hzg13tem0002mp01",
                "x7070_01_u138hzg13tem0001vi03",
                "x7070_01_u138hzg13tem0001vi02"),
                "x7070_01_u138hzg03tem0003mp01",
                t_vl = "x7070_01_u138hzg13tem0001vi03",
                takt = "tag", einzeln = TRUE,
                between = "'2015-01-01' and '2015-03-01 23:59'",
                kurzbezeichnungen = "spaltenuebersicht_oet67.csv")
# andere Variablen und Tages-Durchschnitte
 hydr_abgleich_db(ch, "glt_hist_oet67",
                   c("x7070_01_u138hzg13tem0002mp01",
                   "x7070_01_u138hzg10tem0001mp01",
                   "x7070_01_u138hzg11tem0002mp01"),
                   "x7070_01_u138hzg03tem0003mp01",takt = "tag",
                   between = "'2015-01-01' and '2015-03-01 23:59'",
                kurzbezeichnungen = "spaltenuebersicht_oet67.csv")
close(ch)

## End(Not run)
```

`hydr_abgleich_db_t` *Grafiken zu Hydraulischem Abgleich mit Daten aus einer Datenbank*

Description

Die Funktion erstellt Grafiken zum Hydraulischen Abgleich und verwendet dafür Daten aus einer MySQL-Datenbanktabelle. Entspricht `hydr_abgleich_db`, aber beinhaltet zusätzlich eine Ausnahmebehandlung, so dass im Falle eines Fehlers die Berichte dennoch erstellt werden.

Usage

```
hydr_abgleich_db_t(...)
```

Arguments

`...`	Argumente, die an die Funktion `hydr_abgleich_db` weitergereicht werden.

`hydr_abgleich_variablen`
Vereinfachung der Variableneingabe bei hydr_abgleich_db

Description

Funktion zur Vereinfachung der Variablenübergabe an `hydr_abgleich_db` indem untereinanderkopierte csv-Spaltennamen in die Zwischenablage gespeichert werden können und diese dann als Vektor-Code, welcher in den Funktionsaufruf von `hydr_abgleich_db` kopiert werden kann ausgegeben wird.

Usage

```
hydr_abgleich_variablen(x = NULL)
```

Arguments

x	Entweder Vektor mit Variablennamen, aus denen, nachdem sie in das Datenbanktabellen-Format umgewandelt worden sind, der Vektorcode erstellt wird, oder NULL (Default), dann werden die Variablennamen aus der Zwischenablage genommen.

Value

Vektor-Code zum Kopieren wird in der Console ausgegeben.

Examples

```
## Not run:
# Diese 3 Zeilen in die Zwischenablage kopieren:
7070-01_U138HZG13TEM0002MP01
7070-01_U138HZG13TEM0001VI03
7070-01_U138HZG13TEM0001VI02
hydr_abgleich_variablen()
# mit Ãœbergeben der Variablennamen:
hydr_abgleich_variablen(c("7070-01_U138HZG13TEM0002MP01",
"7070-01_U138HZG13TEM0001VI03", "7070-01_U138HZG13TEM0001VI02"))

## End(Not run)
```

---

jahresdauerlinie *Jahresdauerlinie*

Description

Die Funktion zeichnet die Jahresdauerlinie einer Zeitreihe, wobei die Daten im Stunden- oder Tagestakt sein dürfen. Ist die Zeitreihe länger oder kürzer als ein Jahr, so wird eine Warnung ausgegeben, aber die Jahresdauerlinie dennoch gezeichnet.

Usage

```
jahresdauerlinie(werte, intervalle = c("Tage", "Stunden", "Tg", "h"))
```

Arguments

werte	Vektor mit den Werten, für die die Jahresdauerlinie gezeichnet werden soll.
intervalle	Stunden- oder Tagestakt. Wird nur für die Beschriftung der x-Achse und zur Überprüfung der Anzahl der übergebenen Beobachtungen verwendet. Mögliche Werte sind "Tage", "Stunden", "Tg" oder "h", wobei "Tage" der Default ist.

Details

Dies ist eine sehr einfach gehaltene Variante. Für weitere Argumente und eine in Jahresabschnitte aufgeteilte längere Zeitreihe siehe jahresdauerlinien.

Value

Es wird nur die Grafik erstellt und angezeigt.

Examples

```
## Not run:
require(RODBC)
ch <- odbcConnect("ansi_system")
Aus csv-Datei-Spaltennamen Datenbank-Spaltennamen machen fÃ¼r Argument werte:
mache_db_namen2("7070-01_U138HZG03TEM0003MP01")
# jahresdauerlinien fÃ¼r den Stundentakt fÃ¼r ein Jahr
daten_werte <- sqlQuery2(ch, "select min(datum) as datum,
                             max(x7070_01_u138hzg03tem0003mp01) as werte
                             from glt_hist_oet67
                             where jahr = 2014
                             group by jahr, monat, tag, stunde")
jahresdauerlinie(daten_werte$werte, "h")
daten_werte2 <- sqlQuery2(ch, "select min(datum) as datum,
                              max(x7079_01_t013klt01waz0002zw01) as werte
                              from glt_hist_oet67 group by jahr, monat, tag")
jahresdauerlinie(daten_werte2$werte, "Tg") # warnung, da mehr als 1 jahr

## End(Not run)
```

`jahresdauerlinien` *Jahresdauerlinie aufgeteilt in Jahresabschnitte*

Description

Die Funktion zeichnet die Jahresdauerlinie einer Zeitreihe in ein Jahr dauernde Abschnitte aufgeteilt, wobei die Daten im Stunden- oder Tagestakt sein dürfen.

Usage

```
jahresdauerlinien(werte, datum, intervalle = c("Zeiteinheiten", "Tage",
  "Stunden", "Tg", "h"), titel = NULL, ylab = "W<e4>rmeleistung",
  einheit = "kW", agg_fkt = NULL)
```

Arguments

`werte`	Vektor mit den Werten, für die die Jahresdauerlinie gezeichnet werden soll.
`datum`	Vektor mit dem Datum zu den Werten (POSIXct oder Date).
`intervalle`	Stunden- oder Tagestakt. Wird nur für die Beschriftung der x-Achse verwendet. Mögliche Werte sind "Zeiteinheiten", "Tage", "Stunden", "Tg" oder "h", wobei "Zeiteinheiten" der Default ist.
`titel`	Untertitel für die Grafik. Falls keiner angegeben wird, wird der Name des Vektors, der an werte übergeben wurde verwendet.
`ylab,einheit`	Beschriftung für die y-Achse (Default: "Wärmeleistung" und "kW", ergibt "Wärmeleistung [kW]").
`agg_fkt`	Mit welcher Funktion die Daten aggregiert wurden (wird nur für die Titelbeschriftung verwendet).

Value

Es wird nur die Grafik erstellt und angezeigt.

Examples

```
## Not run:
require(RODBC)
ch <- odbcConnect("ansi_system")
# jahresdauerlinien fÃ¼r den Stundentakt fÃ¼r ein Jahr
daten_werte <- sqlQuery2(ch, "select min(datum) as datum,
                             avg(x7079_01_t013klt01waz0002zw01) as werte
                             from glt_hist_oet67
                             where jahr = 2014
                             group by jahr, monat, tag")
jahresdauerlinien(daten_werte$werte, daten_werte$datum, "Tage",
                  ylab = "Leistung", einheit = "MW")
daten_werte2 <- sqlQuery2(ch, "select min(datum) as datum,
                              avg(x7079_01_t013klt01waz0002zw01) as werte
                              from glt_hist_oet67 group by jahr, monat, tag, stunde")
jahresdauerlinien(daten_werte2$werte, daten_werte2$datum,
                  "Stunden", titel = "x7079_01_t013klt01waz0002zw01")

## End(Not run)
```

---

jahresdauerlinien_db *Jahresdauerlinie aufgeteilt in Jahresabschnitte (Datenbankvariante)*

Description

Die Funktion zeichnet die Jahresdauerlinie einer Zeitreihe in ein Jahr dauernde Abschnitte aufgeteilt, wobei die Daten aggregiert zu Stunden oder Tagen aus der Datenbank geholt werden.

Usage

```
jahresdauerlinien_db(con, tabelle, werte, datum = "datum",
  intervalle = c("Tage", "Stunden", "Tg", "h"), between = NULL,
  werte_between = NULL, titel = NULL, agg_fkt = "avg",
  ylab = "W<e4>rmeleistung", einheit = "kW")
```

Arguments

con	RODBC-Verbindung.
tabelle	Name der Datenbanktabelle.
werte	Spaltenname für die Werte, für die die Jahresdauerlinie gezeichnet werden soll.
datum	Name der Datumsspalte.
intervalle	Stunden- oder Tagestakt. Wird für die Beschriftung der x-Achse und die Aggregation der Daten verwendet. Mögliche Werte sind "Tage", "Stunden", "Tg" oder "h", wobei "Tage" der Default ist.
between	Grenzen des Zeitabschnitts, der verwendet werden soll in der Form "'2011-11-11' and '2012-12-31'".

werte_between	Intervall, in dem die (unaggregierten) Werte von werte als korrekt eingestuft werden. Alle Werte, die außerhalb dieses Intervalls liegen, werden durch NULL (fehlend in MySQL) ersetzt und gehen somit nicht in die Berechnung der Tages- oder Stundendurchschnitte etc. ein.
titel	Untertitel für die Grafik. Fall keiner angegeben wird, wird der Name des Arguments werte verwendet.
agg_fkt	Soll bei der Aggregation der Daten zu Stunden- bzw. Tages-werten "max", "min" oder "avg" verwendet werden für die Minute mit dem höchsten Wert, die mit dem geringsten und dem Durchschnittswert aller Minuten im Intervall.
ylab	y-Achsenbeschriftung (Default: "Wärmeleistung"). Falls der character-Wert mit ".csv" bzw. ".CSV" endet, wird er als Pfad und Name zu einer csv-Datei aufgefasst, in welcher nach dem in Datenbanktabellen-Format umgewandelten Namen von wert gesucht wird und der entsprechende Wert aus der Spalte IP.Kommentar abgefragt wird, ansonsten wird der Wert so übernommen.
einheit	Einheit für die y-Achsenbeschriftung entweder eines aus "kW", "MW", "W" und "°C" (wird dann so übernommen) oder Pfad und Name einer csv-Datei mit den Spalten tabelle und Einheit für den Namen der Spalte werte in der Datenbanktabelle und der Einheit für die y-Achse (siehe spalten_suchen und spaltenuebersicht). Default-Wert ist "kW".

Details

Automatische Anpassung der Zeiträume siehe Beispiel zu letzter_monat.

Value

Es wird nur die Grafik erstellt und angezeigt.

Examples

```
## Not run:
require(RODBC)
ch <- odbcConnect("ansi_system")
jahresdauerlinien_db(ch, "glt_hist_oet67", "x7079_01_t013klt01waz0002zw01",
                     intervalle = "Tage",
                     between = "'2011-01-01' and '2016-01-01'",
                     ylab = "KÃ¤lteleistung", einheit = "spaltenuebersicht_oet67.csv",
                     werte_between = "0 and 250")
jahresdauerlinien_db(ch, "glt_hist_oet67", "x7079_01_t013klt01waz0002zw01",
                     intervalle = "Stunden",
                     between = "'2011-01-01' and '2016-01-01'",
                     ylab = "spaltenuebersicht_oet67.csv")

## End(Not run)
```

---

jahresdauerlinien_db_t

Jahresdauerlinie aufgeteilt in Jahresabschnitte (Datenbankvariante)

Description

Die Funktion zeichnet die Jahresdauerlinie einer Zeitreihe in ein Jahr dauernde Abschnitte aufgeteilt, wobei die Daten aggregiert zu Stunden oder Tagen aus der Datenbank geholt werden. Entspricht jahresdauerlinien_db, aber beinhaltet zusätzlich eine Ausnahmebehandlung, so dass im Falle eines Fehlers die Berichte dennoch erstellt werden.

Usage

```
jahresdauerlinien_db_t(...)
```

Arguments

...	Argumente, die an die Funktion jahresdauerlinien_db weitergereicht werden.

---

kaelteerzeugung *Grafiken zur Kaelteerzeugung*

Description

Die Funktion erstellt Grafiken zur Kälteerzeugung. Diese beinhalten den Verlauf von Außentemperatur, Stromverbrauch und Ventilstellung.

Usage

```
kaelteerzeugung(datum, aussentemperatur, diff_strom, ventil, einheit = "kWh",
  label_strom = "Differenz Strom", variante = 8)
```

Arguments

datum	Datumsvektor.
aussentemperatur	Vektor mit den Messungen der Außentemperatur.
diff_strom	Vektor mit dem Stromverbrauch.
ventil	Vektor mit der Ventilstellung ({0,1} oder [0,1]).
einheit	Die Einheit von diff_strom (Differenz Strom). Default: "kWh".
label_strom	Beschriftung für diff_strom (Default: "Differenz Strom").
variante	Obsolet, nur noch wegen Rückwärtskompatibilität vorhanden und nur noch variante 8 vorhanden. (Früher: Welche Grafikvariante soll gezeichnet werden (1 bis 8, wird wahrscheinlich in naher Zukunft auf 1 Variante reduziert).)

Details

Fehlende Werte in ventil können zu Werten außerhalb von Ein/Aus in der Grafik führen.

Value

Die Grafik wird gezeichnet.

Examples

```
## Not run:
require(RODBC)
ch <- odbcConnect("ansi_system")
# minutenwerte
# Datenbank-Spaltennamen herausfinden:
mache_db_namen2("1110-01_0032HZG00TEM0001MA01")
mache_db_namen2("1120+04--529KLT01VEL0003S101")
mache_db_namen2("1120-01_021ANHV01ELZ0001ZW30")

daten <- sqlQuery2(ch, "select datum,
                        x1110_01_0032hzg00tem0001ma01 as aussentemperatur,
                        x1120_04__529klt01vel0003s101 as ventil,
                        diff_x1120_01_021anhv01elz0001zw30 as wirkarbeit
                        from glt_hist_the46 where jahr = 2014 and monat = 3")
daten$ventil <- umschalt2zustand_na(umschalt = daten$ventil)
with(daten, kaelteerzeugung(datum = datum, aussentemperatur = aussentemperatur,
                            diff_strom = wirkarbeit, ventil = ventil))
#Beispiel mit Zufallszahlen:
kaelteerzeugung(datum = as.Date("01-02-2015")+0:9, aussentemperatur = rnorm(10),
      diff_strom = rnorm(10, 10,2), ventil = c(0,0,0,1,1,1,0,0,0,1))

## End(Not run)
```

---

kaelteerzeugung_db *Grafiken zur Kaelteerzeugung (Datenbankversion)*

Description

Die Funktion erstellt Grafiken zur Kälteerzeugung. Diese beinhalten den Verlauf von Außentemperatur, Stromverbrauch und Ventilstellung. Die Daten dafür werden aus einer MySQL-Datenbanktabelle mittels RODBC gelesen.

Usage

```
kaelteerzeugung_db(con, tabelle, aussentemp, diff_strom, ventil,
  between = NULL, takt = "tag", datum = "datum", einheit = "kWh",
  aussentemp_between = "-40 and 45", ...)
```

Arguments

con	RODBC-Verbindung.
tabelle	Name der Datenbanktabelle.
aussentemp	Name der Spalte mit den Messungen der Außentemperatur.
diff_strom	Name der Spalte mit dem Stromverbrauch.
ventil	Name der Spalte mit der Ventilstellung.
between	Zeitraum der dargestellt werden soll (z.B. "'2013-12-24' and '2014-04-01'")
takt	Sollen die Daten im Tages, 15-Minuten oder Monatstakt dargestellt werden ("tag", "15 min", "mon"). Minutendaten sind nicht vollständig implementiert.
datum	Name der Datumsspalte.

`einheit` Die Einheit für die Variable diff_strom (Differenz Strom). Entweder die Einheit direkt oder Pfad und Name einer csv-Datei mit einer Tabelle, welche die Einheit beinhaltet (siehe `spalten_suchen` und `spaltenuebersicht`).

`aussentemp_between` Intervall, in dem die (unaggregierten) Werte der Außentemperatur als korrekt eingestuft werden. Alle Werte, die außerhalb dieses Intervalls liegen, werden durch NULL (fehlend in MySQL) ersetzt und gehen somit nicht in die Berechnung der Tagesdurchschnitte etc. ein. Das Default-Intervall liegt bei "-40 and 45", was einem etwas größeren Bereich als der bis jetzt in Deutschland gemessenen Extremtemperaturen entspricht.

`variante` Welche Grafikvariante soll gezeichnet werden (1 bis 8, wird wahrscheinlich in naher Zukunft auf 1 Variante reduziert).

`label_strom` Beschriftung für diff_strom (siehe `kaelteerzeugung`).

Details

Die Spaltennamen von aussentemperatur und ventil können wie in den ursprünglichen csv-Daten, in den eingelesenen data.frames oder wie in der Datenbanktabelle angegegben werden (diff_strom existiert in der csv-Datei noch nicht). Fehlende Werte in `ventil` können zu Werten außerhalb von Ein/Aus in der Grafik führen.

Automatische Anpassung der Zeiträume siehe Beispiel zu `letzter_monat`.

Value

Die Grafik wird gezeichnet.

Examples

```
## Not run:
require(RODBC)
ch <- odbcConnect("ansi_system")
kaelteerzeugung_db(ch, "glt_hist_the46",
                   "1110-01_0032HZG00TEM0001MA01",
                   "diff_x1120_01_021anhv01elz0001zw30",
                   "1120+04--529KLT01VEL0003S101",
                   between = "'2014-01-01' and '2014-08-08'", takt = "tag",
                   einheit = "spaltenuebersicht_the46.csv",
                   label_strom = "Diff. Strom",
                   aussentemp_between = "0 and 20")
kaelteerzeugung_db(ch, "glt_hist_the46",
                   "x1110_01_0032hzg00tem0001ma01",
                   "diff_x1120_01_021anhv01elz0001zw30",
                   "x1120_04__529klt01vel0003s101",
                   between = "'2014-01-01' and '2014-08-08'", takt = "tag")
kaelteerzeugung_db(ch, "glt_hist_the46",
                   "x1110_01_0032hzg00tem0001ma01",
                   "diff_x1120_01_021anhv01elz0001zw30",
                   "x1120_04__529klt01vel0003s101",
                   takt = "mon")
# geht nicht:
kaelteerzeugung_db(ch, "glt_hist_the46",
                   "x1110_01_0032hzg00tem0001ma01",
                   "diff_x1120_01_021anhv01elz0001zw30",
                   "x1120_04__529klt01vel0003s101",
```

```
                    between = "'2014-01-01' and '2014-01-05'", takt = "min")

## End(Not run)
```

`kaelteerzeugung_db_t` *Grafiken zur Kaelteerzeugung (Datenbankversion)*

Description

Die Funktion erstellt Grafiken zur Kälteerzeugung. Diese beinhalten den Verlauf von Außentemperatur, Stromverbrauch und Ventilstellung. Die Daten dafür werden aus einer MySQL-Datenbanktabelle mittels RODBC gelesen. Entspricht `kaelteerzeugung_db`, aber beinhaltet zusätzlich eine Ausnahmebehandlung, so dass im Falle eines Fehlers die Berichte dennoch erstellt werden.

Usage

```
kaelteerzeugung_db_t(...)
```

Arguments

`...`	Argumente, die an die Funktion `kaelteerzeugung_db` weitergereicht werden.

`letzter_monat` *String des letzten Monats.*

Description

Erzeugt den String eines Zeitintervalls von einem Monat Dauer, welches zum Ende des letzten Monates (um 23:59:59 Uhr) endet.

Usage

```
letzter_monat()
```

Details

Siehe auch `letzter_zeitraum`, `letztes_jahr`, `letzte_2jahre` und `letzte_3jahre` Nutzbar z.B. in den Grafikfunktion `energiesignatur_db`, `hydr_abgleich_db`, `jahresdauerlinien_db` und `tagesverlauf_db` für das Argument between etc.

Value

String des Zeitraumes, der an die Grafikfunktionen übergeben werden kann.

Examples

```
letzter_monat()
## Not run:
library(RODBC)
ch <- odbcConnect("ansi_system",  DBMSencoding = "utf-8")
# Beobachtungen, die genau 0 sind ausschlieÃŸen und Obergrenze hÃ¶her
energiesignatur_db(ch, "glt_hist_oet67", "diff_x7070_01_u138hzg03wmz0001zw01",
   "x7070_01_u138hzg03tem0003mp01", bis_letzter_monat(),
   letzter_monat(),
   bereinigung = list(untere_grenze = 0+1e-100,
                      quantil_box = c(0.1, 0.9),
                      faktor_oben = 10)) # beobachtungen, die genau 0 sind ausschlieÃŸen

## End(Not run)
```

---

letzter_zeitraum *String eines Zeitraumes*

Description

Erzeugt den String eines Zeitintervalls einer bestimmten Dauer, welches zum Ende des letzten Monates (um 23:59:59 Uhr) endet.

Usage

```
letzter_zeitraum(zeitraum)
```

Arguments

zeitraum	Zeitabstand des Enddatums zum Anfangsdatum (z.B. "-1 month" für letzten Monat, siehe seq.POSIXt)

Details

Die Funktion ist eine allgemeine Variante der Funktionen letzter_monat, letztes_jahr, letzte_2jahre und letzte_3jahre und kann z.B. für die Grafikfunktion energiesignatur_db, hydr_abgleich_db, jahresdauerlinien_db und tagesverlauf_db für das Argument between etc. verwendet werden

Anwendungsbeispiel mit energiesignatur_db siehe letzter_monat.

Value

String des Zeitraumes, der an die Grafikfunktionen übergeben werden kann.

Examples

```
letzter_zeitraum("-1 year")
```

letztes_jahr *String des letzten Jahres*

Description

Erzeugt den String eines Zeitintervalls von einem Jahr Dauer, welches zum Ende des letzten Monates (um 23:59:59 Uhr) endet.

Usage

```
letztes_jahr()
```

Details

Siehe auch letzter_monat,letzter_zeitraum, letzte_2jahre und letzte_3jahre Nutzbar z.B. in den Grafikfunktion energiesignatur_db, hydr_abgleich_db, jahresdauerlinien_db und tagesverlauf_db für das Argument between etc.

Anwendungsbeispiel mit energiesignatur_db siehe letzter_monat.

Value

String des Zeitraumes, der an die Grafikfunktionen übergeben werden kann.

Examples

```
letztes_jahr()
```

letzte_2jahre *String der letzten 2 Jahre.*

Description

Erzeugt den String eines Zeitintervalls von zwei Jahren Dauer, welches zum Ende des letzten Monates (um 23:59:59 Uhr) endet.

Usage

```
letzte_2jahre()
```

Details

Siehe auch letzter_monat, letztes_jahr, letzter_zeitraum und letzte_3jahre Nutzbar z.B. in den Grafikfunktion energiesignatur_db, hydr_abgleich_db, jahresdauerlinien_db und tagesverlauf_db für das Argument between etc.

Anwendungsbeispiel mit energiesignatur_db siehe letzter_monat.

Value

String des Zeitraumes, der an die Grafikfunktionen übergeben werden kann.

Examples

```
letzte_2jahre()
```

`letzte_3jahre` *String der letzten 3 Jahre.*

Description

Erzeugt den String eines Zeitintervalls von drei Jahren Dauer, welches zum Ende des letzten Monates (um 23:59:59 Uhr) endet.

Usage

```
letzte_3jahre()
```

Details

Siehe auch `letzter_monat`, `letztes_jahr`, `letzte_2jahre` und `letzter_zeitraum` Nutzbar z.B. in den Grafikfunktion `energiesignatur_db`, `hydr_abgleich_db`, `jahresdauerlinien_db` und `tagesverlauf_db` für das Argument between etc.

Anwendungsbeispiel mit `energiesignatur_db` siehe `letzter_monat`.

Value

String des Zeitraumes, der an die Grafikfunktionen übergeben werden kann.

Examples

```
letzte_3jahre()
```

`merge_csv` *Join (merge) zweier csv-Dateien*

Description

Fügt zwei csv-Dateien anhand einer Variable zusammen (vgl. merge) und speichert sie als csv-Datei ab oder gibt sie als data.frame zurück.

Usage

```
merge_csv(x, y, neu = NULL, format = c("csv2", "csv"),
  check.names = FALSE, ...)
```

Arguments

x	Name und Pfad des ersten Datensatzes.
y	Name und Pfad des zweiten Datensatzes.
neu	Name und Pfad des Ergebnis-Datensatzes oder NULL (dann wird ein data.frame zurückgegeben).
format	Format der einzulesenden und zu schreibenden csv-Dateien (siehe read.csv)
check.names	Logical. Sollen die Spaltennamen darauf hin überprüft werden, ob sie für einen data.frame gültig sind und ggf. mit make.names angepasst werden.
...	Weitere Argumente, die an merge übergeben werden (insbesondere by)

Details

Für data.frames ungültige Spaltennamen werden beim Einlesen verändert, falls check.names = TRUE ist (siehe make.names bzw. read.table). Argumente, die an merge übergeben werden, müssen dann bereits die neuen Spaltennamen verwenden.

Value

Der zusammengefügte Datensatz wird entweder als csv-Datei geschrieben, falls neu ein character ist, oder als data.frame invisible zurückgegeben und die ersten 6 Zeilen auf der Console ausgegeben, falls neu NULL ist.

Examples

```
## Not run:
merge_csv(x = "spaltenuebersicht_test_alt.csv",
          y = "Daten\\historische_daten_glt\\sonstige tabellen\\IP_Master.csv",
          by.x = "csv", by.y = "IP-Name", all = TRUE, check.names = FALSE)
merge_csv(x = "spaltenuebersicht_test_alt.csv",
          y = "Daten\\historische_daten_glt\\sonstige tabellen\\IP_Master.csv",
          neu = "spaltenuebersicht_alt_ip_master.csv",
          by.x = "csv", by.y = make.names("IP-Name"), all = TRUE,
          check.names = TRUE)
merge_csv(x = "spaltenuebersicht_test.csv",
          y = "Daten\\historische_daten_glt\\sonstige tabellen\\IP_Master.csv",
          neu = "spaltenuebersicht_alt_ip_master2.csv",
          by.x = "csv", by.y = "IP-Name", all.x = TRUE)

## End(Not run)
```

---

spaltenuebersicht *Subserver-Spaltenliste erzeugen*

Description

Erzeugt eine Tabelle mit den Spaltennamen in der ursprünglichen csv-Datei, dem eingelesenen data.frame und der entsprechenden MySQL-Tabelle für einen Subserver.

Usage

```
spaltenuebersicht(csv, con, tabelle, als_csv = NULL,
  uebernehmen = c("Kurzbezeichnung", "Einheit"), aus = NULL,
  gebaeude = NULL)
```

Arguments

csv	Eine csv-Datei eines Subservers (Tagesupdates).
con	Eine RODBC-Verbindung.
tabelle	Name der Tabelle in der MySQL-Datenbank.
als_csv	Pfad und Name der csv-Datei, als die die Übersicht gespeichert werden soll oder NULL, falls eine data.frame zurückgegeben werden soll.
uebernehmen	Welche Spalten aus der alten Datei aus übernommen werden sollen.
aus	Aus welcher csv-Datei die Spalten uebernehmen kopiert werden sollen.
gebaeude	Aus welcher csv-Datei (Famos-Export) sollen die Gebäudekurzbezeichnungen (Kurzbezeichnung BW) eingefügt werden.

Details

Für die Suche nach der Entsprechnung zu den Spaltennamen eines anderen Formates spalten_suchen.Um weitere Spalten, die in einer alten Datei manuell eingetragen worden sind zu erhalten, können die Argumente aus und uebernehmen verwendet werden. Die Gebäudekurzbezeichnung kann aus einer csv-Datei der Famos-Exporte mit gebaude extrahiert werden. Die Datei aus muss eine Spalte mit dem Namen csv und den Spaltennamen der csv-GLT-Exporten beinhalten. Für eine allgemeinere Variante des Joins (merge) zweier csv-Dateien siehe merge_csv.

Value

Die Gegenüberstellung der Spaltennamen und Zusatzinformationen eines Subservers in der csv-Datei, einem daraus erzeugten data.frame und einer MySQL-Tabelle als data.frame oder csv-Datei.

Examples

```
## Not run:
library(RODBC)
ch <- odbcConnect("ansi_system")
# als data.frame zurÃ¼ckgeben:
datei_mit_spalten <- spaltenuebersicht(csv =
                  "Daten/daten_glt_tagesupdates/150308The46EnMoLMU.csv",
                  con = ch,
                  tabelle = "glt_hist_the46")
# als csv-Datei speichern:
spaltenuebersicht(csv = "Daten/daten_glt_tagesupdates/150308The46EnMoLMU.csv",
                  con = ch,
                  tabelle = "glt_hist_the46",
                  als_csv = "spaltenuebersicht_test.csv")
# mit EintrÃ¤gen aus alter Datei:
datei_mit_spalten <- spaltenuebersicht(csv =
                  "Daten/daten_glt_tagesupdates/150308The46EnMoLMU.csv",
                  con = ch,
                  tabelle = "glt_hist_the46",
                  aus = "spaltenuebersicht_test_alt.csv")
# mit GebÃ¤udekurzbezeichnungen
```

```
datei_mit_spalten <- spaltenuebersicht(csv =
                  "Daten/daten_glt_tagesupdates/150308The46EnMoLMU.csv",
                  con = ch,
                  tabelle = "glt_hist_the46",
                  aus = "spaltenuebersicht_test_alt.csv",
                  gebaeude = "Daten/Famos_daten/20141216 GebÃ¤udeliste.csv",
                  als_csv = "spaltenuebersicht_test.csv")

## End(Not run)
```

---

spalten_suchen *Entsprechung zu Suchwort in einer anderen Spalte finden*

Description

Sucht Wort was in den Spalten spalten und gibt das entsprechende Wort aus der Spalte aus zurück.

Usage

```
spalten_suchen(was, aus, spalten, daten, na = c("nichts", "was", "csv13",
  "?"), diff_entfernen = FALSE)
```

Arguments

was	Welches Wort gesucht werden soll (character-Vektor mit einem oder mehreren Wörtern).
aus	Aus welcher Spalte die Entsprechung kommen soll.
spalten	In welchen Spalten gesucht werden soll (character-Vektor).
daten	data.frame oder Pfad und Name der csv-Datei in der gesucht werden soll.
na	Was zurückgegeben werden soll, wenn es zum Suchwort was keine Entsprechung gibt oder das gesuchte Wort nicht gefunden wird bzw. die Spalte aus nicht vorhanden ist ("was": das Suchwort wird zurückgegeben; "nichts" (Default): NA bzw. character(0) etc. wird zurückgegeben; "csv13": die Entsprechnung aus der Spalte csv ab Zeichen 13 wird zurückgegeben; "?": Ein "?" wird zurückgegeben). Wird auch ausgeführt, falls die spalten spalten nicht gefunden wurden. Für csv13 wird was verwendet, falls was nicht in der Tabelle gefunden wurde.
diff_entfernen	Soll bei einer diff_-Variable nach der Variable ohne diff_ gesucht werden, falls diese nicht gefunden wird bzw. die Entsprechung NA ist (logical, Default ist FALSE)?

Details

Für das Erzeugen von Dateien mit den Spaltennamen der GLT-Daten im csv-, data.frame- und Datenbanktabellen-Format siehe spaltenuebersicht.

Value

Entsprechendes Wort aus der Spalte aus.

Examples

```
## Not run:
# aus data.frame
spalten_suchen("1110-01_0032HZG03TEM0001VI03", "tabelle", "csv", datei_mit_spalten)
# aus csv-Datei
spalten_suchen(c("1110+03_0303HZG17D__0001BM01","X1110.01_0032HZG15UPW0001S101"),
               "tabelle", c("csv", "data.frame"),
               "spaltenuebersicht_test.csv")
# mit na = "was" und na in entsprechender Zelle
spalten_suchen("diff_x1110_01_0032hzg01wmz0001zw01", "csv","tabelle",
               "spaltenuebersicht_test.csv", na = "was")
# suche nach nicht existentem Wort
spalten_suchen("bla", "csv","tabelle",
               "spaltenuebersicht_test.csv")
spalten_suchen("bla", "csv","tabelle",
               "spaltenuebersicht_test.csv", na = "was")
# auch ohne diff_ suchen
spalten_suchen("diff_x1110_01_0032hzg01wmz0001zw01", "csv","tabelle",
               "spaltenuebersicht_test.csv", diff_entfernen = TRUE)
# aus-Spalte nicht vorhanden und csv13
spalten_suchen("diff_x1110_01_0032hzg01wmz0001zw01", "gibts_nicht","tabelle",
               "spaltenuebersicht_test.csv", diff_entfernen = TRUE, na="csv13")

## End(Not run)
```

`tagesverlauf` *Tagesverlauf (Rasterdiagramm)*

Description

zeichnet den Tagesverlauf eines Wärmemengenzählers etc. als Rasterdiagramm mit den Tagesmittelwerten der Außentemperatur und ggf. der Sonneneinstrahlung.

Usage

```
tagesverlauf(datum, werte, aussentemp, strahlung = NULL,
  legende = "W<e4>rmemenge", einheit = NULL)
```

Arguments

`datum`	Zeitpunkte der Beobachtungen in werte, aussentemp und strahlung als POSIXct (tz="GMT").
`werte`	Zeitreihe mit dem Wärmemengen-Verbrauch (Vektor).
`aussentemp`	Zeitreihe mit der Außentemperatur (Vektor).
`strahlung`	Zeitreihe mit der Sonneneinstrahlung (Vektor).
`legende,einheit`	Überschrift und Einheit für die Legende des Rasterdiagramms.

Value

Nichts, es wird eine Grafik gezeichnet.

Examples

```
## Not run:
daten4 <- sqlQuery2(ch, "select max(datum) as datum,
                           sum(diff_x7070_01_u138hzg03wmz0001zw01) as waermemenge,
                           avg(x7070_01_u138hzg03tem0003mp01) as aussentemperatur
                         from glt_hist_oet67 group by jahr, monat, tag, stunde, cut15min")
with(daten4,tagesverlauf(werte = waermemenge, aussentemp= aussentemperatur,
                          strahlung = aussentemperatur,datum=datum))
mache_db_namen2("1120-01_021ANHV01ELZ0001ZW30")
mache_db_namen2("1110-01_0032HZG00TEM0001MA01")
daten5 <- sqlQuery2(ch, "select datum as datum,
                           diff_x1120_01_021anhv01elz0001zw30
                           as waermemenge, x1110_01_0032hzg00tem0001ma01
                           as aussentemperatur from glt_hist_the46
                           where jahr = 2014")
with(daten5,tagesverlauf(werte = waermemenge, aussentemp = aussentemperatur,
                          strahlung = aussentemperatur,datum=datum))
quantile(daten5$waermemenge, c(0.9,0.999,1), na.rm=TRUE)
with(daten5[daten5$waermemenge<28 & daten5$waermemenge>=0,],
         tagesverlauf(werte = waermemenge, aussentemp= aussentemperatur,
         strahlung = NULL,datum=datum))

## End(Not run)
```

---

tagesverlauf_db *Tagesverlauf (Rasterdiagramm) mit Datenbank*

Description

zeichnet den Tagesverlauf eines Wärmemenge-Zählers als Rasterdiagramm mit den Tagesmittelwerten der Außentemperatur und ggf. der Sonneneinstrahlung. Die Daten dafür werden aus einer MySQL-Datenbank abgefragt.

Usage

```
tagesverlauf_db(con, tabelle, datum, werte, aussentemp,
  abschnitte = c("Minuten", "15 Minuten"), tabelle2 = NULL,
  strahlung = NULL, between = NULL, legende = "W<e4>rmemenge",
  einheit = NULL, bereinigung = list(untere_grenze = 0, quantil_box = c(0.1,
  0.9), faktor_oben = 5), aussentemp_between = "-40 and 45",
  agg_fkt = "sum")
```

Arguments

con	RODBC-Verbindung zu einer MySQL-Datenbank.
tabelle	Tabelle in der Datenbank, in welcher die Variablen sind.
datum	Name der Datumsvariable in der Tabelle.
werte	Name der Wärmemenge-Variable (bei 15-Minuten-Abschnitten wird jeweils die Summe der einzelnen Minuten gebildet). Für Temperatur-Variablen siehe agg_fkt.
aussentemp	Name der Außentemperatur-Variable.
abschnitte	Soll die Wärmemenge im Minuten- oder 15-Minuten-Takt geplottet werden.

`tabelle2` ggf weitere Tabelle in der einige der Variablen sind (Abfrage über tabelle left join tabelle 2, es werden also alle Werte aus tabelle und nur die Messzeitpunkte aus tabelle2, die auch in tabelle vorhanden sind, abgefragt).

`strahlung` Name der Variable für die Sonneneinstrahlung.

`between` Zeitraum für den das Diagramm gezeichnet werden soll in der Form between = "'2015-01-01' and '2015-12-30'" oder "'2015-01-01 00:00:00' and '2015-12-30 00:00:00'"

`legende,einheit`
Legendenbeschriftung des Rasterdiagramms (Default: "Wärmeleistung" und NULL). Falls der character-Wert mit ".csv" bzw. ".CSV" endet, wird er als Pfad und Name zu einer csv-Datei aufgefasst, in welcher nach dem in Datenbanktabellen-Format umgewandelten Namen von wert gesucht wird und der entsprechende Wert aus der Spalte IP.Kommentar bzw. Einheit abgefragt wird (falls dort nichts gefunden wird, wird eine Abkürzung des Spaltennamens bzw. "?" verwendet), ansonsten wird der Wert so übernommen.

`bereinigung` Liste mit Argumenten, die an die Funktion `datenbereinigung` weitergegeben werden, um Werte, die wahrscheinlich falsch sind, zu löschen. Falls keine Datenbereinigung gewünscht ist, muss `bereinigung = NULL` angegeben werden. Der Default-Wert ist nicht NULL, es wird also per Default eine Bereinungung durchgeführt.

`aussentemp_between`
Intervall, in dem die (unaggregierten) Werte der Außentemperatur als korrekt eingestuft werden. Alle Werte, die außerhalb dieses Intervalls liegen, werden durch NULL (fehlend in MySQL) ersetzt und gehen somit nicht in die Berechnung der Tagesdurchschnitte ein. Das Default-Intervall liegt bei "-40 and 45", was einem etwas größeren Bereich als der bis jetzt in Deutschland gemessenen Extremtemperaturen entspricht.

`agg_fkt` SQL-Funktion mit der die werte zu 15-Minuten-Abschnitten aggregiert werden, wobei "sum" der Summe und "avg" dem Durchschnitt entspricht. Defaultwert ist "sum" für Verbrauchsdaten. "avg" ist für Temperaturdaten geeignet. Weitere Möglichkeiten sind z.B. "min" für Minimum oder "max" für Maximum.

Details

Automatische Anpassung der Zeiträume siehe Beispiel zu `letzter_monat`.

Examples

```
## Not run:
# Daten abfragen:
library(RODBC)
ch <- odbcConnect("ansi_system",  DBMSencoding = "utf-8")
# Tagesverlauf zeichnen:
tagesverlauf_db(ch, tabelle = "glt_hist_oet67", datum = "datum",
                werte = "diff_x7070_01_u138hzg03wmz0001zw01",
                aussentemp = "x7070_01_u138hzg03tem0003mp01", abschnitte = "Minuten",
                tabelle2 = "theresienwetter", strahlung = "globalstrahlung",
                between = "'2015-01-19 00:00:00'  and '2015-01-25 23:59:00'")
 # Minutentakt dauert sehr lange
tagesverlauf_db(ch, tabelle = "glt_hist_oet67", datum = "datum", werte =
               "diff_x7070_01_u138hzg03wmz0001zw01",
                aussentemp = "x7070_01_u138hzg03tem0003mp01",
                abschnitte = "15 Minuten",
```

```
                legende = "spaltenuebersicht_oet67.csv",
                einheit = "spaltenuebersicht_oet67.csv")
tagesverlauf_db(ch, tabelle = "glt_hist_oet67", datum = "datum",
               werte = "diff_x7070_01_u138hzg03wmz0001zw01",
               aussentemp = "x7070_01_u138hzg03tem0003mp01",
               abschnitte = "15 Minuten",
               tabelle2 = "theresienwetter", strahlung = "globalstrahlung",
               between = "'2015-01-01 00:00:00'  and '2015-12-30 00:00:00'",
               einheit = "[MWh]")
tagesverlauf_db(ch, tabelle = "the46", datum = "datum",
                werte = "diff_x1120_01_021anhv01elz0001zw30",
                aussentemp = "x1110_01_0032hzg00tem0001ma01",
                tabelle2 = "theresienwetter", strahlung = "globalstrahlung",
                abschnitte = "15 Minuten")
tagesverlauf_db(ch, tabelle = "glt_hist_the46", datum = "datum",
                werte = "diff_x1120_01_021anhv01elz0001zw30",
                aussentemp = "x1110_01_0032hzg00tem0001ma01",
                tabelle2 = "theresienwetter", strahlung = "globalstrahlung",
                abschnitte = "15 Minuten",
                legende = "W\u00e4rme-\nmenge",
                aussentemp_between = "0 and 20")
tagesverlauf_db(ch, tabelle = "glt_hist_the46", datum = "datum",
                werte = "diff_x1120_01_021anhv01elz0001zw30",
                aussentemp = "x1110_01_0032hzg00tem0001ma01",
                tabelle2 = "theresienwetter", strahlung = "globalstrahlung",
                abschnitte = "Minuten",
                between = "'2015-01-19 00:00:00'  and '2015-01-21 00:00:00'")
                # sollte eigentlich 1 minute des 26. beinhalten
                # (bei 10 Min gehts, einfach zu dÃ¼nn, um sichtbar zu sein?)

## End(Not run)
```

`tagesverlauf_db_t` *Tagesverlauf (Rasterdiagramm) mit Datenbank und Ausnahmebehandlung*

Description

zeichnet den Tagesverlauf eines Wärmemenge-Zählers als Rasterdiagramm mit den Tagesmittelwerten der Außentemperatur und ggf. der Sonneneinstrahlung. Die Daten dafür werden aus einer MySQL-Datenbank abgefragt. Entspricht `tagesverlauf_db`, aber beinhaltet zusätzlich eine Ausnahmebehandlung, so dass im Falle eines Fehlers die Berichte dennoch erstellt werden.

Usage

```
tagesverlauf_db_t(...)
```

Arguments

`...`	Argumente, die an die Funktion `tagesverlauf_db` weitergereicht werden.

umschalt2zustand *Ventildaten anpassen*

Description

Macht aus einem Ventildatenvektor, der bei jedem Ein-/Ausschaltvorgang einen bestimmten wert, und bei unverändertem zustand einen anderen wert hat, einen vektor, der mit Beginn aus beginn,umschalten beginnt und dann den wert für den aktuellen zustand aus 0,1 beibehält.

Usage

```
umschalt2zustand(umschalt, beginn = 0, umschalten = 1, bleibend = 0)
```

Arguments

umschalt	Vektor mit Ein-/Ausschaltvorgängen.
beginn	Womit soll der neue vektor beginnen, bis ein Umschaltwert kommt.
umschalten	Welcher Wert im Umschalt-vektor steht für Zustandsänderung.
bleibend	Welcher Wert im Umschalt-Vektor steht für gleichbleibenden Zustand.

Details

Die Funktion ist nicht für Vektoren geeignet, die fehlende Werte enthalten, wie die GLT-Ventildaten (siehe umschalt2zustand_na).

Value

Zustandsvektor mit den Werten 0 oder 1 für die ensprechenden Zustände.

Examples

```
umschaltvektor <- c(0,0,0, 1, 0, 0, 1, 0,0)
umschalt2zustand(umschaltvektor)
```

umschalt2zustand_na *Ventildaten anpassen GLT*

Description

Diese Funktion passt die Ventildaten für Grafiken an, indem sie einen Vektor erzeugt, in welchem jeder Eintrag den aktuell gültigen Wert repräsentiert. Sie ist geeignet, falls die Einträge immer NA sind außer, wenn es zu einer Änderung der Ventilstellung kommt, in diesem Fall ist der Eintrag 0 für Aus und 1 für Ein, was, soweit aus den Daten ersichtlich den Ventil-Spalten aus den GLT-Daten entspricht.

Usage

```
umschalt2zustand_na(umschalt)
```

Arguments

umschalt	Vektor mit den original Ventildaten.

Details

Die Funktion kann auch verwendet werden, falls die Werte nicht aus {0,1} sind.

Value

Vektor mit den angepassten Ventildaten.

Examples

```
originalvektor <- c(NA, NA, 1, NA, NA, 0, NA)
umschalt2zustand_na(originalvektor)
```

---

ventil_anteil *Anteil der Zeit, die das Ventil auf Ein ist*

Description

Die Funktion berechnet den Anteil der Zeit, in der ein Ventil geöffnet ist.

Usage

```
ventil_anteil(ventil, ein = 1, na.rm = TRUE)
```

Arguments

ventil	Vektor dessen Einträge den Zustand eines Ventils zu einem bestimmten Zeitpunkt repräsentieren.
ein	Welcher Wert repräsentiert ein geöffnetes Ventil (bzw. den Zustand für den man den Anteil an der Gesamtzeit berechnet will).
na.rm	Sollen fehlende Werte in ventil in die Berechnung der Gesamtdauer eingehen.

Value

Anteil der Zeit, zu denen das ventil auf 1 ist.

Examples

```
ventil_anteil(ventil = c(0,0,0,1,1,1,0,0,0,1))
```

Index

A.1.5 enmoBerichte

Package 'enmoBerichte'

September 7, 2015

Title Energiemonitoring-Berichte erstellen

Version 1.0

Description Funktionen zum Erstellen von Energiemonitoring-Berichten mit EnMoLMU.

Depends R (>= 3.1.2)

License What license is it under?

LazyData true

Encoding UTF-8

Imports knitr,
markdown,
tools,
methods

R topics documented:

`knit_all` *Alle Berichte erstellen*

Description

Alle Berichte auf einmal erstellen. Es können einzelne Dateien (mit Pfad) oder ganze Ordner mit .Rmd-Dateien übergeben werden, die alle automatisch zu html-Dateien kompiliert werden (mit `knit`). Für weitere Formate sieht `knit_all_format`

Usage

```
knit_all(dateien = NULL, ordner = NULL, output_ordner = NULL,
  encoding = "UTF-8", ...)
```

Arguments

dateien	Von welchen .Rmd-Dateien sollen berichte erstellt werden.
ordner	In welchen ordnern soll von allen enthaltenen dateien ein bericht erstellt werden.
output_ordner	In welche Ordner sollen die Ergebnisse gespeichert werden. Character-Vektor mit 1 Pfad (ohne Dateinamen) je dateien-Element bzw. je ordner- Element.
encoding	Kodierung der übergebenen Dateien, default: UTF-8.
...	Weitere Argumente, die an knit2html übergeben werden.

Details

Falls dateien angegeben wird und output_ordner nicht, werden die neuen Dateien in die selben Ordner wie Dateien gespeichert. Die Namen der Berichte sind identisch mit denen in dateien, nur mit "bericht_" am Anfang. Falls ordner angegeben wird, aber keine output_ordner, kommen die Berichte in neue Ordner, die im selben Verzeichnis, wie ordner liegen, den selben Namen haben, nur dass sie mit "bericht_" beginnen. Falls ordner angegeben werden, aber keine output_ordner, so müssen die ordner ohne "/" oder "\" am Schluss angegeben werden (sonsts landen die neuen Ordner in den Ordnern aus ordner).

Value

Bericht-Dateien in output-Ordner (bzw. Ordner der dateien/"berichte_"+ordner, siehe Details)

Examples

```
## Not run:
knit_all(dateien = "Code_Berichte/knitr_test_neue_pfade.Rmd")# geht
knit_all(ordner = "Code_Berichte/ganzer_ordner_zum_knittern")#geht
weitere tests nÃ¶tig (ordner, output_ordner, mehrere dateien auf einmal)
knit_all(dateien = c("Code_Berichte/ganzer_ordner_zum_knittern/bericht1.Rmd",
"Code_Berichte/ganzer_ordner_zum_knittern/bericht2.Rmd"))

## End(Not run)
```

---

knit_all_format *Mehrere Berichte mit einem Aufruf erstellen.*

Description

Wie knit_all, nur für unterschiedliche Ausgabeformate (nicht nur html, sondern auch pdf, docx, odt). Es können einzelne Dateien (mit Pfad) oder ganze Ordner mit .Rmd-Dateien übergeben werden, die alle automatisch zu html-, pdf-, docx- oder odt-Dateien kompiliert werden.

Usage

```
knit_all_format(dateien = NULL, ordner = NULL, output_ordner = NULL,
  format, encoding = "UTF-8", ...)
```

Arguments

dateien	Von welchen .Rmd-Dateien sollen berichte erstellt werden.
ordner	In welchen ordnern soll von allen enthaltenen dateien ein bericht erstellt werden.
output_ordner	In welche Ordner sollen die Ergebnisse gespeichert werden. Character-Vektor mit 1 Pfad (ohne Dateinamen) je dateien-Element bzw. je ordner- Element.
format	In welches Format soll die Datei umgewandelt werden: "html", "pdf", "docx", "odt", Default: "html" (wird an knit_all übergeben).
encoding	Kodierung der übergebenen Dateien, default: UTF-8.
...	Weitere Argumente, die an knit_format übergeben werden.

Details

falls ein ganzer Ordner angegeben wird, dürfen darin keine Ordner mit .Rmd-Endung enthalten sein (und keine Dateien, mit dieser Endung, die nicht Rmd-Dateien sind). Die Dateinamen der Vorlagen dürfen (außer für html) keine Leerzeichen enthalten.

Examples

```
## Not run:
knit_all_format(dateien = "Code_Berichte/knitr_test_neue_pfade.Rmd", format = "docx")# geht!
knit_all_format(ordner = "Code_Berichte/ganzer_ordner_zum_knittern", format = "pdf")#geht!
knit_all_format(ordner = "Code_Berichte/ganzer_ordner_zum_knittern", format = "html")#geht!
knit_all_format(dateien = c("Code_Berichte/ganzer_ordner_zum_knittern/bericht1.Rmd",
"Code_Berichte/ganzer_ordner_zum_knittern/bericht2.Rmd"), format = "odt") # geht!

## End(Not run)
```

---

knit_format *Rmd in html, pdf, docx oder odt umwandeln*

Description

Einheitlicher Befehl für die jeweils funktionierenden Umwandlungs-Vorgehensweisen von .Rmd-Dateien zu html, pdf, docx, odt in einem Schritt.

Usage

```
knit_format(datei, output = NULL, format = c("html", "pdf", "docx", "odt"),
  encoding = "UTF-8", md_loeschen = TRUE, ...)
```

Arguments

datei	Rmd-Datei die kompiliert werden soll (inkl. Pfad, falls nicht im Arbeitsverzeichnis).
output	Zielpfad und -Datei.
format	In welches Format soll die Datei umgewandelt werden: "html", "pdf", "docx", "odt", Default: "html".
encoding	Kodierung. Dürfte immer "UTF-8" sein, da dies für rmarkdown benötigt wird(?).

md_loeschen	Soll die md-Datei, die als Zwischenstufe erstellt wird wieder gelöscht werden? Default = TRUE.
...	Weitere Argumente, die an knit2html bzw. knit (alles außer html) weitergegeben werden.

Details

Die Dateinamen der Vorlagen dürfen (außer für html) keine Leerzeichen enthalten.

Value

Nichts oder logical, falls md-Datei gelöscht werden sollte

---

knit_pdf *Pdfs aus Rmarkdown erstellen*

Description

Unkompliziertes Erstellen von pdfs aus Rmarkdown Dokumenten. Es muss dafür Pandoc auf dem PC installiert sein.

Usage

```
knit_pdf(input, output = NULL, encoding = "UTF-8",
  zwischenschritte_loeschen = TRUE, ...)
```

Arguments

input	Pfad und Name der Rmarkdown-Datei.
output	Pfad und Name der Pdf-Datei.
encoding	Zeichenkodierung der Rmarkdown-Datei.
zwischenschritte_loeschen	Sollen die auf der Festplatte gespeicherten Zwischenergebnis-Dateien wieder gelöscht werden.
...	Weitere Argumente, die an knit übergeben werden.

Value

Die Dateien werden erstellt.

Examples

```
## Not run:
knit_pdf("Code_Berichte/knitr_test_neue_pfade.Rmd")
# Zwischenschritte behalten, z.B. um sie spÃ¤ter noch in andere Formate umzuwandeln
knit_pdf("Code_Berichte/knitr_test_neue_pfade.Rmd",
         zwischenschritte_loeschen = FALSE)

## End(Not run)
```

neue_endung *Dateiendung eines Dateinamens austauschen*

Description

Tauscht die Endung eines als String übergebenen Dateinamens aus.

Usage

```
neue_endung(datei, neu, alt = NULL)
```

Arguments

datei	Dateiname als string.
neu	Neue Dateiendung ohne "." davor.
alt	Alte Dateiendung ohne "." zur Kontrolle ob Endung richtig erkannt wurde bzw. ein Dateiname mit der richtigen Endung übergeben wurde.

Details

Die Dateiendung wird daran erkannt, dass sie nach dem letzten Punkt im Dateinamen steht

Value

Dateiname mit neuer Endung.

Examples

```
neue_endung("lustiger_name.md", "Rmd")
neue_endung("jetzt.ein.anderer_name.docx", "mp3", "docx")
## Not run:
Fehler:
neue_endung("bla.mp3", "xls", ".mp3")
neue_endung("bla.mp3", ".xls", "mp3")

## End(Not run)
```

pfad_ohne_datei *Pfad in Datei und Pfad aufteilen*

Description

Extrahiert Pfad aus Pfad mit Datei, Pfad ohne letzten Ordner bzw. letzten Ordner/Datei ohne Pfad

Usage

```
pfad_ohne_datei(pfad, datei = FALSE)
```

Arguments

pfad	Character-Vektor aus Pfaden bzw. Pfaden mit Dateien
datei	Logical (TRUE oder FALSE). Soll die Datei oder der letzte Ordner zurückgegeben werden oder (TRUE) oder der Pfad dahin (FALSE). Default-Wert: FALSE.

Details

Sollte der Pfad mit einem Ordner enden, dürfen am Schluss keine "/" oder "\" mehr sein.

Value

Character-Vektor mit Pfaden oder Dateien/Ordnern

Examples

```
pfad_ohne_datei(c("ordner1/ordner2/datei.R", "ordner1\\ordner2\\datei.docx",
                  "datei.csv"))
pfad_ohne_datei(c("ordner1/ordner2/datei.R", "ordner1\\ordner2\\datei.docx",
                  "ordner1/order2","datei.csv"), datei=TRUE)
pfad_ohne_datei(c("ordner1/ordner2/datei.R", "ordner1\\ordner2\\datei.docx",
                  "ordner1/order2","datei.csv"), datei=FALSE)
```

---

sweave_pdf *Pdf-Bericht aus einer Rnw-Datei (Sweave)*

Description

Die Funktion erzeugt einen Bericht im Pdf-Format aus einer Rnw-Datei, falls R Markdown bzw. Pandoc nicht verwendet werden soll oder kann. Rnw ist die Dateiendung für Sweave {utils}.

Usage

```
sweave_pdf(datei, output = NULL, clean = TRUE)
```

Arguments

datei	Input-Datei (.Rnw) mit Pfad.
output	Pdf-Datei mit Pfad. Legt Name und Ordner der output-Datei fest. Der Zielordner muss vorhanden sein.
clean	Sollen einige der Zwischendateien wieder gelöscht werden.

Value

pdf-Datei und Zwischendateien. Falls sich im Zielordner Dateien mit dem Namen der Zieldatei und den Endungen Rnw, tex, etc. befinden, werden diese überschrieben.

Examples

```
## Not run:
# in neuen ordner
sweave_pdf(datei = "Code_Berichte/sweave_test/sweave_test.Rnw",
           output = "Code_Berichte/sweave_test/output/sweave_test_output.pdf")
# in gleichen ordner neuer name
sweave_pdf(datei = "Code_Berichte/sweave_test/sweave_test.Rnw",
           output = "Code_Berichte/sweave_test/sweave_test_output.pdf")
# ohne output
sweave_pdf(datei = "Code_Berichte/sweave_test/sweave_test.Rnw",
           output = NULL)

## End(Not run)
```

Index

A.1.6 enmoFamos

Package 'enmoFamos'

September 7, 2015

Title Funktionen zur Auswertung der Famos-Daten.

Version 1.0

Description Funktionen zur Auswertung der Famos-Daten. Teil der EnMoLMU-Software.

Depends enmoDaten,
mgcv,
enmoGrafiken,
reshape2,
ggplot2,
dplyr

License What license is it under?

LazyData true

Encoding UTF-8

Suggests xlsx

R topics documented:

---

benchmark *Vergleich des Verbrauchs mehrerer Gebaeudeteile oder deren Summen aus GLT und Famos*

Description

Vergleich des Verbrauchs mehrerer Gebäudeteile oder deren Summen aus GLT und Famos mit Gradtagzahl-Bereinigung für unterschiedliche Monate oder aus ganzen Monaten bestehende Zeiträume.

Usage

```
benchmark(con = NULL, tabellen = NULL, datumsspalte = "datum",
  agg_fkt = "sum", datei = NULL, spalten = NULL, methode = c("anteile",
  "gam"), einheiten_angleichen = TRUE, ..., zeitraum, einheit = "kWh",
  gradtagzahlbereinigung = TRUE, addieren = NULL, rueckgabewerte = FALSE)
```

Arguments

tabellen	Zähler aus einer GLT-Datenbank als Liste mit tabelle, spalten und einheit in der die Werte der jeweiligen Spalte gespeichert sind. Sind keine Einheiten angegeben, so werden die original-Werte verwendet. Siehe glt_benchmark.
datei	xlsx-Datei der Famos-Zählerstände.
spalten	Zähler, die aus der Famos-Datei verwendet werden sollen.
methode	Methode, mit der die gewünschten Zeiträume aus den Famos- Zählerständen geschätzt werden sollen (bis jetzt nur anteile fertig implementiert).
einheiten_angleichen	Sollen die Famos-Zähler auf eine gemeinsame Einheit gebracht werden (dazu werden die Einheitsangaben aus der Spalte Kurzbezeichnung der Datei datei verwendet).
zeitraum	Zeiträume, die miteinander verglichen werden sollen (Monate oder aus ganzen Monaten bestehende Zeiträume).
einheit	Einheit, in welche die Verbräuche umgerechnet werden sollen.
addieren	Liste mit Character-Vektoren der Zähler (Geäudeteilename, z.B. 7079), die zu einem übergeordneten Zähler verschmolzen werden sollen. Falls die einzelnen Vektoren Namen haben, werden die neuen Zähler entsprechend benannt, ansonsten wird der Name des ersten Zählers verwendet.
rueckgabewerte	Soll eine Liste mit den berechneten Daten (fertige_daten, addiert, werte) zurückgegeben werden? werte sind die abgefragten, addiert die addierten (und ggf. gradtagzahlbereinigten) Daten und fertige_daten der Datensatz im long-Format aus dem die Grafik erstellt wird.
ch	RODBC-Verbindung. Siehe odbcConnect aus dem CRAN-Paket RODBC.

Details

Bei den Zeiträumen in zeitraum bedeutet z.B. "2015" das komplette Jahr 2015, "2015/01" der komplette Januar 2014 und "2014/1-2014/03" der Zeitraum vom 1. Januar bis 31. März 2014. Wird der Monat angegeben, so kann dies einstellig (z.B. "2013/3") oder zweistellig (z.B. "2013/03") geschehen. Um eine Gradtagzahlbereinigung durchführen zu können, sind nur vollständige Monate möglich.

Es können sowohl Zähler aus Famos und GLT nebeneinander betrachtet werden, als auch nur Zähler aus einer der beiden Datenquellen, allerdings muss bei ausschließlicher Verwendung von Famos-Daten beachtet werden, dass, falls eine Gradtagzahlbereinigung durchgeführt werden soll, die RODBC-Verbindung trotzdem benötigt wird, um die LUFTTEMPERATUR abzufragen.

Examples

```
## Not run:
library(EnMoLMU)
library(RODBC)
ch<- odbcConnect("ansi_system")
benchmark(ch, tabellen = list(
            list("glt_hist_oet67", c("diff_x7070_01_u138hzg03wmz0001zw01",
                                     "diff_x7079_01_t013klt01waz0001zw01"),
                                   einheiten = c("MWh","kWh")),
            list("glt_hist_leo03", c("diff_x0407_01_061bhzg13wmz0001zw01",
                                     "diff_x0631_01_1004wwb01wmz0001zw02"),
                                   einheiten = c("m3", "kWh"))),

        datei="Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.xlsx",
            spalten = c("ZPL0000937","ZPL000487E","ZPL0005022"),

            methode = "anteile",

          zeitraum = list(c("2015/01"),
                          c("2014/1-2014/3"),
                          c("2013")),
          einheit = "MWh", addieren = list("addierte z\u00e4hler"=c("0631","0660")))

# oder mit Famosdaten als csv-Datei, da Einlesen von xlsx nicht sehr robust:
benchmark(ch, tabellen = list(
            list("glt_hist_oet67", c("diff_x7070_01_u138hzg03wmz0001zw01",
                                     "diff_x7079_01_t013klt01waz0001zw01"),
                                   einheiten = c("MWh","kWh")),
            list("glt_hist_leo03", c("diff_x0407_01_061bhzg13wmz0001zw01",
                                     "diff_x0631_01_1004wwb01wmz0001zw02"),
                                   einheiten = c("m3", "kWh"))),

        datei="Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.csv",
            spalten = c("ZPL0000937","ZPL000487E","ZPL0005022"),

            methode = "anteile",

          zeitraum = list(c("2015/01"),
                          c("2014/1-2014/3"),
                          c("2013")),
          einheit = "MWh", addieren = list("addierte z\u00e4hler"=c("0631","0660")))
# Nur Famos-Daten:
benchmark(datei="Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.csv",
```

```
            spalten = c("ZPL0000937","ZPL000487E","ZPL0005022"),

            methode = "anteile",

          zeitraum = list(c("2015/01"),
                          c("2014/1-2014/3"),
                          c("2013")),
          einheit = "MWh", addieren = list("addierte z\u00e4hler"=c("0631","0660")),
          gradtagzahlbereinigung = FALSE)
# Nur Famos-Daten aber Gradtagzahlbereinigung:
benchmark(con = ch, datei="Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.csv",
            spalten = c("ZPL0000937","ZPL000487E","ZPL0005022"),

            methode = "anteile",

          zeitraum = list(c("2015/01"),
                          c("2014/1-2014/3"),
                          c("2013")),
          einheit = "MWh", addieren = list("addierte z\u00e4hler"=c("0631","0660")),
          gradtagzahlbereinigung = TRUE)
 # Nur glt-Daten:
benchmark(ch, tabellen = list(
             list("glt_hist_oet67", c("diff_x7070_01_u138hzg03wmz0001zw01",
                                      "diff_x7079_01_t013klt01waz0001zw01"),
                                    einheiten = c("MWh","kWh")),
             list("glt_hist_leo03", c("diff_x0407_01_061bhzg13wmz0001zw01",
                                      "diff_x0631_01_1004wwb01wmz0001zw02"),
                                    einheiten = c("m3", "kWh"))),

            methode = "anteile",

          zeitraum = list(c("2015/01"),
                          c("2014/1-2014/3"),
                          c("2013")),
          einheit = "MWh", addieren = list("addierte z\u00e4hler"=c("0631","0660")))
# mit Zeitraum, f\u00fcr den keine Wetterdaten vorhanden sind:
benchmark(ch, tabellen = list(
             list("glt_hist_oet67", c("diff_x7070_01_u138hzg03wmz0001zw01",
                                      "diff_x7079_01_t013klt01waz0001zw01"),
                                    einheiten = c("MWh","kWh")),
             list("glt_hist_leo03", c("diff_x0407_01_061bhzg13wmz0001zw01",
                                      "diff_x0631_01_1004wwb01wmz0001zw02"),
                                    einheiten = c("m3", "kWh"))),

        datei="Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.csv",
            spalten = c("ZPL0000937","ZPL000487E","ZPL0005022"),

            methode = "anteile",

          zeitraum = list("2014/01", "2014/2", "2014/4",
                          "2014/5","2014/6","2014/7"),
          einheit = "MWh", addieren = list("addierte z\u00e4hler"=c("0631","0660")))
# ohne Gradtagzahlbereinigung, sodass diese Wetterdaten nicht ben\u00f6tigt werden:
benchmark(ch, tabellen = list(
             list("glt_hist_oet67", c("diff_x7070_01_u138hzg03wmz0001zw01",
                                      "diff_x7079_01_t013klt01waz0001zw01"),
                                    einheiten = c("MWh","kWh")),
```

```
             list("glt_hist_leo03", c("diff_x0407_01_061bhzg13wmz0001zw01",
                                      "diff_x0631_01_1004wwb01wmz0001zw02"),
                                    einheiten = c("m3", "kWh"))),

         datei="Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.csv",
             spalten = c("ZPL0000937","ZPL000487E","ZPL0005022"),

             methode = "anteile",

           zeitraum = list("2014/01", "2014/2", "2014/4",
                           "2014/5","2014/6","2014/7"),
           einheit = "MWh", addieren = list("addierte z\u00e4hler"=c("0631","0660")),
           gradtagzahlbereinigung = FALSE)
close(ch)

## End(Not run)
```

---

benchmark_csv *Aufruf von* benchmark *mit Argumenten, die in einer csv-Datei gespeichert sind*

Description

Die Funktion ruft die Funktion benchmark auf und liest die dazugehörigen Argumente aus einer csv-Datei ein. DIESE FUNKTION IST NOCH NICHT FERTIG UND KORREKT!

Usage

```
benchmark_csv(con, datei, ...)
```

Arguments

con	RODBC-Verbindung.
datei	csv-Datei mit den Argumenten siehe Details.
...	weitere Argumente, die an benchmark übergeben werden. Argumente, die bereits in der csv-Datei stehen, dürfen hier nicht nochmal angegeben werden.

Details

Die csv-Datei datei muss eine Spalte mit den Argumentnamen, eine mit den Argumenten (bei einer Liste mit Unterlisten, das jeweils erste Unterlistenelement), eine mit dem zweiten Unterlistenelement und eine mit dem Gruppennamen des jeweiligen Zählers, falls eine Gruppierung durchgeführt werden soll. Die einzelnen Elemente zu den einzelnen Argumenten stehen untereinander, z.B.

gradtagzahlbereinigung;TRUE;;;
zeitraum;2014/01;;;
;2014/2;;;
;2014/4;;;
;2014/5;;;
datei;Daten/Famos_daten/20150128 Zählerstand Stammgelände Komplett.csv;;;
spalten;ZPL0000937;;;addiertes 1
;ZPL000487E;;;addiertes 1
;ZPL0005022;;;addiertes 3

tabellen;glt_hist_oet67;diff_x7070_01_u138hzg03wmz0001zw01;MWh;addiertes 2
;;diff_x7079_01_t013klt01waz0001zw01;kWh;addiertes 2
;glt_hist_leo03;diff_x0407_01_061bhzg13wmz0001zw01;m3;addiertes 2
;;diff_x0631_01_1004wwb01wmz0001zw02;kWh;addiertes 3

Examples

```
library(RODBC)
ch <- odbcConnect("ansi_system")
benchmark_csv(con = ch, datei = "Daten/Famos_daten/benchmark_thomas.csv")
```

`dgradtagzahl_bereinigung` *Gradtagzahlbereinigung von data.frame mit Verbrauchswerten in bestimmter Form*

Description

Gradtagzahlbereinigung eines data.frames mit den verschiedenen Zeiträumen als Spalten (und den Raumcodes als Zeilen). Die einzelnen Werte repräsentieren den Verbrauch eines Monats, Jahres oder eines aus vollständigen (zusammenhängenden) Monaten bestehenden Zeitraums. Die Funktion ist in erster Linie für die Verwendung durch `benchmark` und `zaehlervergleich` gedacht. Um einen einzelnen Werte zu gradtagzahlbereinigen, siehe `gradtagzahl_bereinigung`.

Usage

```
dgradtagzahl_bereinigung(wert, zeitraum, hist = "wetter_hist", con,
  tabelle = "theresienwetter", raumcodes = TRUE)
```

Arguments

`wert`	Verbrauchswerte (data.frame).
`zeitraum`	liste der Zeiträume des Verbrauchs in der Form %Y-%m%-d, z.B. c("2013-01-01", "2013-01-31") für den kompletten Januar (in der verwendeten Wetterliste existiert eine Spalte, die nur das Datum in Tagen (ohne Stunden) enthält). Jedes Listenelement mit Anfangs- und Endzeitpunkt repräsentiert eine Spalte im data.frame mit den Verbrauchswerten.
`hist`	Pfad zu den historischen (stündlichen) Wetterdaten (Rds-Datei).
`con`	RODBC-Verbindung zur Datenbank mit den aktuellen Wetterdaten.
`tabelle`	Tabelle mit den aktuellen Wetterdaten.
`raumcodes`	Entspricht die erste Spalte im data.frame wert den Raumcodes (also beinhaltet keine Verbrauchswerte)?

Value

Gradtagzahlbereinigter Wert

Examples

```
addiert <- structure(list(raumcodes = structure(c("1020", "7070", "7079",
  "addierte z\u00e4hler"), .Names = c("ZPL0000937", "", "", "ZPL0005022")),
  X2014.12.31.bis.2015.01.31 = c(0, 127.6, 689.31, 0),
  X2013.12.31.bis.2014.03.31 = c(86494.2954863832,
  131.5, 777.09, 0.355302123586856), X2012.12.31.bis.2013.12.31 =
  c(269253.268643628, 2111.29999999999, 9865.007, 4.65660394623347)),
  .Names = c("raumcodes", "X2014.12.31.bis.2015.01.31",
  "X2013.12.31.bis.2014.03.31", "X2012.12.31.bis.2013.12.31"),
row.names = c(NA, -4L), class = "data.frame")
gradtagzahl_bereinigung(addiert, list(c("2013-01-01", "2013-01-31"), c("","")),
                        con = ch, tabelle = "theresienwetter")
```

---

einheiten_angleichen *Angleichen unterschiedlicher Einheiten*

Description

Mit der Funktion können Werte mit unterschiedlichen Einheiten alle in eine bestimmte Einheit gebracht werden.

Usage

```
einheiten_angleichen(x, einheiten, einheit = c("kWh", "m<U+00B3>", "m3", "MWh"))
```

Arguments

x	Vektor der Verbräuche.
einheiten	Vektor der Einheiten zu den Verbräuchen (muss selbe Länge haben oder 1 Einheit, die für alle Verbräuche aus x gilt). Folgende Einheiten können von der Funktion verwendet werden: "m^3", "m3", "m³" für Kubikmeter, "kWh" für Kilowattstunden und "MWh" für Megawattstunden.
einheit	Einheit, in die alle Werte gebracht werden sollen (eines aus "kWh","m³","m3","MWh").

Value

Vektor x auf die Einheit einheit gebracht.

Examples

```
einheiten_angleichen(c(1,1000,1.42), c("MWh","kWh","m3"), "kWh")
einheitNeu <- einheiten_angleichen(c(1,1000,1.42), c("MWh","kWh","m3"), "m3")
einheitNeu
attr(einheitNeu, "einheit")
einheiten_angleichen(c(1,1000,1.42), "kWh", "MWh")
```

famos_anteile_gam *Schaetzung der Famos-Zeitraeume durch Anteile der gemessenen an den gewuenschten Zeitraeumen oder Vorbereitung fuer die Schaetzung durch ein Additives Modell.*

Description

Mit dieser Funktion können die gewünschten Zeiträume aus den Famos-Daten geschätzt werden, indem die vorhandenen Zeiträume durch die Anzahl der Tage geteilt und mit der Anzahl der Überschneidungs-Tage mit dem gewünschten Zeitraum multipliziert werden, oder indem ein Additives Modell geschätzt wird. Die Funktion wird i.d.R. nur von famos_schaetzen und nicht direkt aufgerufen und bereitet die Daten für die Schätzung durch famos_gam vor, falls methode = "gam".

Usage

```
famos_anteile_gam(daten, zeitraum, methode = c("anteile", "gam"), ...)
```

Arguments

daten	Daten als data.frame nach Vorverarbeitung durch famos_schaetzen.
zeitraum	Liste mit jeweils Anfangs- und Endzeitpunkt der zu schätzenden (gewünschten) Zeiträume.
methode	Schätzmethode, die verwendet werden soll.

Details

Es können nur Zeiträume richtig geschätzt werden, die vollständig von Ablesezeiträumen überdeckt werden.

Value

Schätzungen für die gewünschten Zeiträume und Zähler.

Examples

```
## Not run:
# wird von famos_schaetzen aufgerufen
daten <- famos_lesen(datei="Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.xlsx",
               zcode = c("ZPL0000937","ZPL000487E","ZPL0005022"))
famos_schaetzen(daten, list(c("2013-01-01","2013-01-31"),
                            c("2014-01-01","2014-01-31")),
                methode = "anteile")
famos_schaetzen(daten, list(c("2013-01-01","2013-01-31"),
                            c("2014-01-01","2014-01-31")),
                methode = "anteile")

## End(Not run)
```

famos_auffuellen *csv-Datei zur Argumentuebergabe an benchmark aufbereiten*

Description

Füllt leere Zellen mit der darüberliegenden auf, wenn die anderen Spalten nicht auch leer ("") sind.

Usage

```
famos_auffuellen(spalte, data)
```

Arguments

spalte	Welche Spalte soll aufgefüllt werden (Index als Integer).
data	Datensatz mit der Spalte.

Value

Spalte mit den Ergänzungen.

Examples

```
data <- read.csv2("Daten/Famos_daten/benchmark_thomas.csv",
                  header=FALSE, as.is = TRUE)
famos_auffuellen(spalte=1, data = data)
```

famos_gam *Schaetzung der Famos-Zeitraeume durch ein Additives Modell.*

Description

Mit dieser Funktion können die gewünschten Zeiträume aus den Famos-Daten geschätzt werden, indem ein Additives Regressionsmodell mit den Einflussgrößen jährliche Periodizität (glatt), Außentemperatur (glatt), Wochenende (binär) und Semesterferien (binär und Interaktion mit Wochenende) auf den aus den gemessenen Zeiträumen erzeugten täglichen Durchschnittsverbräuchen berechnet wird. Die Schätzung wird nur für die Verteilung des Verbrauchs auf die einzelnen Tage innerhalb der sich mit dem gewünschten Zeitraum überschneidenden gemessenen Zeiträume verwendet, indem der Quotient aus gemessenen überschneidenden Zeiträumen und geschätzten überschneidenden Zeiträumen mit dem geschätzten gewünschten Zeitraum multipliziert wird. Als Gesamtverbrauch für diesen Zeitraum wird der gemessene Wert verwendet, da der aktuelle Verbrauch ermittelt werden soll. Die Funktion wird i.d.R. nur von famos_schaetzen über famos_anteile_gam und nicht direkt aufgerufen.

Usage

```
famos_gam(zeitraum, tages_daten, ferien, aussentemp)
```

Arguments

zeitraum	Liste mit jeweils Anfangs- und Endzeitpunkt der zu schätzenden (gewünschten) Zeiträume.
tages_daten	Daten als data.frame nach Vorverarbeitung durch famos_anteile_gam.

Details

Es können nur Zeiträume richtig geschätzt werden, die vollständig von Ablesezeiträumen überdeckt werden.

Value

Schätzungen für die gewünschten Zeiträume und Zähler.

Examples

```
## Not run:
# RODBC-Verbindung f\u00fcr Au\u00dfentemperatur aus Datenbank-Tabelle
library(RODBC)
ch <- odbcConnect("ansi_system")
# Famos-Daten einlesen
daten <- famos_lesen(datei="Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.xlsx",
               zcode = c("ZPL0000937","ZPL000487E","ZPL0005022"))
# wird von famos_schaetzen aufgerufen
famos_schaetzen(daten, list(c("2013-03-01","2013-03-31"),
                            c("2014-01-01","2014-01-31")),
                methode = "gam",
                ferien = "Daten/ferien.csv",
                aussentemp = list(con = ch, tabelle = "glt_hist_oet67",
                aussentemp = "x7070_01_u138hzg03tem0003mp01"))

## End(Not run)
```

---

famos_lesen *Einlesen der Famos-Daten*

Description

Einlesen der Famos-Daten im xlsx- oder csv-Format, Auswählen der ZCodes und Angleichen der Einheiten (Umrechnen in kWh).

Usage

```
famos_lesen(datei, zcode = NULL, in_kwh = FALSE)
```

Arguments

datei	Pfad und Name der xlsx- oder csv-Datei.
zcode	Welche ZCodes ausgewählt werden sollen.
in_kwh	Sollen die Werte (von MWh und m3) in kWh umgerechnet werden? Falls TRUE werden auch die Einheiten in der Spalte Kurzbezeichnung durch kWh ersetzt.

Value

data.frame der Famos-Zählerstand-Datei im ursprünglichen Format (long). Die Daten werden nach ZCode und Ablesetag sortiert.

Examples

```
## Not run:
library(xlsx) # scheint trotz "xlsx::read.xlsx2..." notwendig zu sein.
head(famos_lesen(datei="Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.xlsx",
                 zcode = c("ZPL0000937","ZPL000487E","ZPL0005022")))

## End(Not run)
```

---

famos_schaetzen *Schaetzen der Verbraeuche fuer die gewuenchten Zeitraeume*

Description

Schätzen der Verbräuche für die gewünschten Zeitraume mit den eingelesenen Daten und einem Additiven Modell oder der Summe der Anteile aus den verfügbaren Zeiträumen.

Usage

```
famos_schaetzen(daten, zeitraum, methode = c("gam", "anteile"),
  letzter_stand = NULL, einheiten_angleichen = TRUE, einheit = "kWh", ...)
```

Arguments

daten	Eingelesene Famos-Zählerstand-Daten.
zeitraum	gewünchte Zeiträume, die geschätzt werden sollen. Für anteile entspricht c("2014-12-31", "2015-01-31") dem kompletten Januar 2015.
methode	Schätzmethode, die verwendet werden soll ("anteile" oder "gam").
letzter_stand	Falls die letzte Ablesung für den jeweiligen ZCode vor diesem Zeitpunkt war, werden die entsprechenden ZCodes entfernt (als Date oder Character der Form "%Y-%m-%d", z.B. "2015-04-21").
einheiten_angleichen	Sollen die Einheiten der Zähler in eine gemeinsame Einheit umgerechnet werden.
einheit	Welche Einheit soll das sein ("kWh","MWh", "m3").
...	Weitere Argumente, die an famos_anteile_gam weitergegeben werden.

Value

Schätzungen für die gewünschten Zeiträume.

Examples

```
## Not run:
daten <- famos_lesen(datei="Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.xlsx",
               zcode = c("ZPL0000937","ZPL000487E","ZPL0005022"))
famos_schaetzen(daten, list(c("2014-01-01","2014-01-31"),
                            c("2015-01-01","2015-01-31")),
                methode = "anteile")

## End(Not run)
```

`ferien_kurve` *Grafik einer Zeitreihe mit den Ferien durch die Hintergrundfarbe markiert und aufgeteilt nach Wochenende/Werktage*

Description

Grafik einer Zeitreihe mit den Ferien durch die Hintergrundfarbe markiert und aufgeteilt nach Wochenende/Werktage

Usage

```
ferien_kurve(daten, x, y, ferien)
```

Arguments

`daten`	Datensatz.
`x`	Name der Datumsspalte.
`y`	Name der Spalte mit den Messwerten.

Value

Grafik

Examples

```
# vorher noch einlesen und aufbereiten der Daten!
oet_w <- read.csv2("Daten/Famos_daten/interpolation/Oet67_WÃ¤rmemenge.csv",
                   stringsAsFactors = FALSE)
oet_w$datum <- as.POSIXct(oet_w$Datum.Uhrzeit,
                          format = "%d.%m.%Y %H:%M:%S", tz = "GMT")
oet_w$diff_w <- c(NA,diff(oet_w[,2]))
oet_w2 <- oet_w[-c(1:83),] #sinnvollen Bereich auswÃ¤hlen
ferien_df <- read.csv2("Daten/ferien.csv", stringsAsFactors = FALSE)
oet_w3 <- ds_hinzufuegen(x = oet_w2, Wochenende = TRUE,
                         semesterferien = ferien_df,
                         ferienformat = "%d.%m.%Y",datumsspalte = "datum")
ferien_kurve(oet_w3, "datum", "diff_w", ferien_df2)
```

formel_oder_zaehler *Rekursives erzeugen einer Gesamtformel fuer einen Zaehler aus den Unterebenen*

Description

Rekursives erzeugen einer Gesamtformel fuer einen Zaehler aus den Unterebenen

Usage

```
formel_oder_zaehler(zaehler, operator, datei)
```

Arguments

zaehler	Zählername, welcher in datei vorhanden ist.
operator	Rechenzeichen ("+" oder "-") vor dem Zähler zaehler.
datei	Hierarchie-Datei der Zähler (mit Ebene, Zähler, Quelle und Formel).

Details

Ein Zähler (egal welcher Ebene) darf nur einmal in der Spalte Zähler in datei vorkommen!

Value

Gesamtformel aus ausschließlich nicht-virtuellen Zählern.

Examples

```
datei <- read.csv2("zaehlerebenen_benchmark4.csv", as.is = TRUE)
formel_oder_zaehler("zähler a", "+", datei)
formel_oder_zaehler("zähler d", "+", datei)
formel_oder_zaehler("zähler e", "-", datei)
```

gebaeudeueberblick *Gebaeudeuebersicht aus Famos-Datei erstellen*

Description

Mit Hilfe dieser Funktion kann eine csv-Datei erstellt werden, die eine Übersicht über die darin enthaltenen Gebäude mit den Variablen "Straße", "BauteilHaus", "PLZ", "Ort", "BW.NGF" und "Kurzbezeichnung.BW" enthält. Dazu wird die aus Famos exportierte Datei verwendet.

Usage

```
gebaeudeueberblick(liste, ueberblick = NULL)
```

Arguments

liste	Pfad und Name der csv-Datei mit den Ursprungsdaten (aus Famos exportierte Gebäudeliste) als character.
ueberblick	Pfad und Name der csv-Datei, die erstellt werden soll als character (falls NULL wird ein data.frame zurückgegeben und keine csv-Datei erstellt).

Value

csv-Datei oder data.frame mit einer Zeile je Gebäudenummer (Bauteil).

Examples

```
## Not run:
# data.frame:
head(gebaeudeueberblick("Daten/Famos_daten/20141216 GebÃ¤udeliste.csv"))
# csv-Datei
gebaeudeueberblick("Daten/Famos_daten/20141216 GebÃ¤udeliste.csv",
                   "Daten/GebÃ¤udeÃ¼berblick.csv")

## End(Not run)
```

---

glt_benchmark *Abfrage und Aggregation der Benchmark-Daten aus mehreren Datenbanktabellen*

Description

Mit der Funktion können die Daten zu bestimmten Zeiträumen aus mehreren Tabellen abgefragt und aggregiert werden.

Usage

```
glt_benchmark(con, tabellen, ...)
```

Arguments

con	RODBC-Verbindung.
tabellen	Liste mit listen aus Argumenten, die für jede Tabelle unterschiedlich sind (z.B. tabelle, spalten, ...) siehe glt_benchmark_tabelle.
...	Argumente, die für alle Tabellen gelten (z.B. zeitraum) siehe glt_benchmark_tabelle.

Value

data.frame mit den Ergebnissen der Aggregation (Spalten als Spalten und Zeiträume als Zeilen) für alle Tabellen.

Examples

```
## Not run:
library(RODBC)
ch<- odbcConnect("ansi_system")
glt_benchmark(ch, tabellen = list(
            list("glt_hist_oet67", c("diff_x7070_01_u138hzg03wmz0001zw01",
                                     "diff_x7079_01_t013klt01waz0001zw01"),
                                   einheiten = c("MWh","kWh")),
            list("glt_hist_leo03", c("diff_x0407_01_061bhzg13wmz0001zw01",
                                     "diff_x0631_01_1004wwb01wmz0001zw02"),
                                   einheiten = c("m3", "kWh"))),
              zeitraum = list(c("2015-01-01","2015-01-31"),
                              c("2014-01-01","2014-01-31")),
            einheit = "MWh")
close(ch)

## End(Not run)
```

`glt_benchmark_tabelle` *Abfrage und Aggregation der Benchmark-Daten aus einer Datenbanktabelle*

Description

Mit der Funktion können die Daten zu bestimmten Zeiträumen aus einer Tabelle abgefragt und aggregiert werden.

Usage

```
glt_benchmark_tabelle(con, tabelle, spalten, zeitraum, datumsspalte = "datum",
  agg_fkt = "sum", einheiten = NULL, einheit = "kWh")
```

Arguments

`tabelle`	Name der Datenbanktabelle, die die Zähler aus spalten enthält.
`spalten`	Character-Vektor mit den Spalten der Vergleichswerte (müssen nicht im Datenbank-Format sein, siehe `mache_db_namen`.
`zeitraum`	Liste mit den Zeiträumen (jeweils 1 Character-Vektor mit 2 Einträgen), die verglichen werden sollen. Der komplette Januar 2015 kann durch c("2015-01-01","2015-01-31 23:59") angegeben werden.
`datumsspalte`	Name der Spalte in der Tabelle, die das Datum enthält.
`agg_fkt`	Aggregationsfunktion die für die Zeiträume jeweils ausgeführt werden soll (z.B. "avg" für Durchschnitt, "sum" für die Summe). Im Normalfall immer der Default-Wert "sum".
`einheiten`	Vektor der Einheiten zu den Spalten in spalten (muss selbe Länge haben oder 1 Einheit, die für alle Verbräuche aus x gilt). Die Einheiten werden für alle Zeiträume verwendet. Folgende Einheiten können von der Funktion verwendet werden: "m^3", "m3", "m³" für Kubikmeter, "kWh" für Kilowattstunden und "MWh" für Megawattstunden. Falls ein Character-Vektor der Länge 1 übergeben wird, der mit ".csv" oder ".CSV" endet, wird der Wert als Pfad und Name einer Tabelle interpretiert, in welcher die Einheit nachgeschlagen wird (siehe `spalten_suchen`).

einheit	Einheit, in die alle Werte gebracht werden sollen (eines aus "kWh","m³","m3","MWh"). Default: "kWh".
ch	RODBC-Verbindung

Value

data.frame mit den Ergebnissen der Aggregation (Spalten als Spalten und Zeiträume als Zeilen)

Examples

```
## Not run:
library(RODBC)
ch<- odbcConnect("ansi_system")
glt_benchmark_tabelle(ch, "glt_hist_oet67", c("diff_x7070_01_u138hzg03wmz0001zw01",
                                    "diff_x7079_01_t013klt01waz0001zw01"),
              zeitraum = list(c("2015-01-01","2015-01-31 23:59"),
                              c("2014-01-01","2014-01-31 23:59")))
glt_benchmark_tabelle(ch, "glt_hist_oet67", c("diff_x7070_01_u138hzg03wmz0001zw01",
                                    "diff_x7079_01_t013klt01waz0001zw01"),
              zeitraum = list(c("2015-01-01","2015-01-31"),
                              c("2014-01-01","2014-01-31")),
              einheiten = c("MWh","kWh"), einheit = "kWh")
glt_benchmark_tabelle(ch, "glt_hist_oet67", c("diff_x7070_01_u138hzg03wmz0001zw01",
                                    "diff_x7079_01_t013klt01waz0001zw01"),
              zeitraum = list(c("2015-01-01","2015-01-31"),
                              c("2014-01-01","2014-01-31")),
              einheiten = c("m3","kWh"), einheit = "MWh")
glt_benchmark_tabelle(ch, "glt_hist_oet67", c("diff_x7070_01_u138hzg03wmz0001zw01",
                                    "diff_x7079_01_t013klt01waz0001zw01"),
              zeitraum = list(c("2015-01-01","2015-01-31"),
                              c("2014-01-01","2014-01-31")),
              einheiten = "spaltenuebersicht_oet67.csv", einheit = "MWh")
close(ch)

## End(Not run)
```

---

gradtagzahl_bereinigung

Gradtagzahlbereinigung eines Verbrauchswertes

Description

Gradtagzahlbereinigung eines Skalars, der den Verbrauch eines Monats, Jahres oder eines aus vollständigen (zusammenhängenden) Monaten bestehenden Zeitraums repräsentiert.

Usage

```
gradtagzahl_bereinigung(wert, zeitraum, hist = "wetter_hist", con,
  tabelle = "theresienwetter")
```

Arguments

wert	Verbrauchswert (Skalar).
zeitraum	Zeitraum des Verbrauchs in der Form %Y-%m-%d, z.B. c("2013-01-01", "2013-01-31") für den kompletten Januar (in der verwendeten Wetterliste existiert eine Spalte, die nur das Datum in Tagen (ohne Stunden) enthält).
hist	Pfad zu den historischen (stündlichen) Wetterdaten (Rds-Datei).
con	RODBC-Verbindung zur Datenbank mit den aktuellen Wetterdaten.
tabelle	Tabelle mit den aktuellen Wetterdatenl.

Details

benchmark verwendet d_gradtagzahl_bereinigung.

Value

Gradtagzahlbereinigter Wert

Examples

```
library(RODBC)
ch <- odbcConnect("ansi_system")
library(enmoDaten)
gradtagzahl_bereinigung(100, c("2013-01-01", "2013-01-31"),
                        hist="wetter_hist",
                        con = ch, tabelle = "theresienwetter")
close(ch)
```

---

mmerge *Join (merge) einer Liste von data.frames*

Description

Führt Joins für eine Liste von data.frames aus indem es jeweils zwei data.frames an merge übergibt.

Usage

```
mmerge(dfs, by = list(Reduce(intersect, lapply(dfs, names))), all = TRUE)
```

Arguments

dfs	Liste mit data.frames.
by	Liste aus Character-Vektoren mit Spaltennamen für die einzelnen data.frames, die für die Joins verwendet werden sollen. Ein einzelner Spaltennamen wird recycled. Der Default ist die Schnittmenge der Spaltennamen aller data.frames.
all	Logical-Vektor. Für welche data.frames auch die Werte ohne Entsprechnung im nächsten data.frame übernommen werden sollen (Gilt nur für jeweil einen merge-Aufruf). Der Default ist TRUE und ein einzlner Wert wird recycled.

Details

Alternativ siehe Reduce in Paket base.

Value

data.frame.

`operatoren_umdrehen` *Vertauscht "-"- und "+"-Zeichen in einer Formel (character).*

Description

Wird von `formel_oder_zaehler` verwendet

Usage

```
operatoren_umdrehen(formel)
```

Arguments

`formel`	Formel als character-Skalar

Value

formel mit umgedrehten "-"- und "+"-Zeichen.

Examples

```
operatoren_umdrehen("a+b-c+d-e")
```

`tag_des_jahres` *Berechnen der wievielte Tag des Jahres ein bestimmtes Datum ist*

Description

Berechnen der wievielte Tag des Jahres ein bestimmtes Datum ist

Usage

```
tag_des_jahres(datum)
```

Arguments

`datum`	Datum als character-Vektor, Date oder POSIX.ct (GMT).

Value

Integer, der den soundso vielten Tag eines Jahres bezeichnet

Examples

```
tag_des_jahres(c("2015-01-09", "2016-02-01"))
tag_des_jahres(as.Date(c("2015-01-09", "2016-02-01")))
tag_des_jahres(as.POSIXct(c("2015-01-09 13:00", "2016-02-01"), tz="GMT"))
# bei Posix immer GMT, damits keine faxen macht!
```

`theresienwetter_historisch_einlesen` *Einlesen und speichern der historischen stuendlichen Wetterdaten.*

Description

Die Funktion liest die historischen stündlichen Wetterdaten des Meteorologischen Instituts für den Zeitraum 1.1.2013 bis 30.6.2014 ein, bringt ihn ins wide-Format, bereitet die Kodierung auf (9999 -> NA, 1/10 Grad Celsius -> Grad Celsius, etc.) und speichert ihn als RDS-Datei bzw. gibt ihn (invisible) als data.frame zurück. Da es sich hier um einen einmaligen Datensatz handelt, kann nach der ersten Umwandlung theoretisch immer einfach die fertige RDS-Datei eingelesen werden. Die Funktion ist ziemlich stark an die konkrete Datei angepasst (z.B. Einlesen von genau 2730 Zeilen) und somit ohne Abwandlungen kaum für andere Dateien verwendbar.

Usage

```
theresienwetter_historisch_einlesen(datei, speichern = NULL, plot = TRUE)
```

Arguments

`datei`	Pfad und Name der Wetterdatei.
`speichern`	Pfad und Name der RDS-Datei, die erstellt werden soll. Falls NULL wird nur der data.frame (invisible) zurückgegeben.
`plot`	Sollen die Ergebnis-Spalten geplottet werden.

Details

Originaldaten: Tag Monat Jahr Kennzahl 24*Stundenwerte Kennzahl Bedeutung 4 RELATIVE FEUCHTE IN 2 M HOEHE IN 1/10 % 7 LUFTTEMPERATUR IN 2 M HOEHE IN 1/10 GRAD CELSIUS 52 NIEDERSCHLAG IN 30M HOEHE IN 1/100 MM 58 DIFFUSSTRAHLUNG IN 30M HOEHE IN 1/10 W/M^2 62 GLOBALSTRAHLUNG IN 30M HOEHEIN 1/10 W/M^2 Fehlende Werte: 9999

Value

data.frame der aufbereiteten Wetterdaten.

Examples

```
## Not run:
# wetter als RDS-Datei erstellen
theresienwetter_historisch_einlesen(datei = paste0("C:\\Users\\Rottner",
                                          "\\Documents\\EnMoLMU\\Daten\\",
                                          "Wetterdaten\\Theresienwetter ",
                                          "historisch\\stadt_auszug",
                                    speichern = paste0("C:\\Users\\Rottner\\",
                                          "Documents\\EnMoLMU\\Daten\\",
                                          "Wetterdaten\\Theresienwetter",
                                          " historisch\\",
                                          "theresienwetter_historisch_2013_2014.Rds"))
fehlende Monate aus anderer Datei einlesen und als data.frame zur\u00fcckgeben
wetter_hist2 <- theresienwetter_historisch_einlesen(datei =
```

```
                                    paste0("Daten\\Wetterdaten\\",
                                    "stadt_auszug2014_07u08"))
# RDS-Datei einlesen
wetter_hist <- readRDS(paste0("C:\\Users\\Rottner\\Documents\\EnMoLMU\\",
                         "Daten\\Wetterdaten\\Theresienwetter historisch\\",
                         "theresienwetter_historisch_2013_2014.Rds"))

# Zeilen mit fehlenden Werten (LUfttemperatur) entfernen, zusammenf\u00fcgen und
#  vollstÃ¤ndige Datei speichern:
wetter_histb <- wetter_hist[!is.na(wetter_hist$LUFTTEMPERATUR),]
wetter_zusammen <- rbind(wetter_histb, wetter_hist2)
plot(wetter_zusammen$Datum, wetter_zusammen$LUFTTEMPERATUR, type="l")
saveRDS(object = wetter_zusammen,
        file = paste0("Daten\\Wetterdaten\\Theresienwetter historisch\\",
                      "theresienwetter_historisch_2013_2014.Rds"))

## Historisches Wetter einlesen und mit aktuellen aus der Datenbank verkn\u00fcpfen:
#RDS-Datei einlesen:
wetter_hist <- readRDS(paste0("C:\\Users\\Rottner\\Documents\\EnMoLMU\\",
                      "Daten\\Wetterdaten\\Theresienwetter historisch\\",
                      "theresienwetter_historisch_2013_2014.Rds"))
# Theresienwetter aus aktueller Datenbank-Tabelle einlesen:
library(RODBC)
ch <- odbcConnect("ansi_system")
library(enmoDaten)
wetter_neu <- sqlQuery2(ch, "select min(datum) as Datum2, max(stunde) as Stunde,
                            tag as Tag, monat as Monat, jahr as Jahr,
                            avg(diffusstrahlung) as DIFFUSSTRAHLUNG,
                            avg(globalstrahlung) as GLOBALSTRAHLUNG,
                            avg(aussentemperatur) as LUFTTEMPERATUR,
                            sum(niederschlag) as NIEDERSCHLAG,
                            avg(rel_feuchte) as RELATIVE_FEUCHTE
                            from theresienwetter
                            where datum > '2014-06-30'
                            group by jahr, monat, tag, stunde")

# beide Tabellen zusammenf\u00fcgen:
wetter_neu$Datum <- as.Date(wetter_neu$Datum2)
wetter_gesamt <- rbind(wetter_hist, wetter_neu)
head(wetter_gesamt, 20)
plot(wetter_gesamt$Datum, wetter_gesamt$LUFTTEMPERATUR, type="l")

## End(Not run)
```

`theresienwetter_mit_hist` *Kombiniertes Einlesen der historischen und aktuellen Wetterdaten*

Description

Diese Funktion liest die (als RDS-Datei gespeicherten) historischen Wetterdaten des Meteorologischen Instituts bis 6/2014 und die aktuellen aus der Datenbanktabelle ein und kombiniert sie zu einem Datensatz als data.frame.

Usage

```
theresienwetter_mit_hist(hist = "wetter_hist", con,
  tabelle = "theresienwetter", grenzen_aussentemp = c(-40, 50))
```

Arguments

hist	Pfad und Dateiname der RDS-Datei mit den historischen Wetterdaten bis 30.6.2014 oder "wetter_hist" (Default; verwendet die im Paket mitgelieferten historischen Wetterdaten).
con	RODBC-Verbindung
tabelle	Datenbanktabelle mit den aktuellen Wetterdaten.
grenzen_aussentemp	Grenzen, innerhalb derer die Lufttemperatur als richtig gemessen angenommen werden soll (sonst werden die Einträge auf NA gesetzt). Default-Grenzen: c(-40,50).

Details

Daten für 7/2014 bis 9/2014 fehlen.

Value

data.frame mit den gesamten Wetterdaten.

Examples

```
library(RODBC)
ch <- odbcConnect("ansi_system")
library(enmoDaten)
# historische Daten aus RDS-Datei:
gesamt_wetter <- theresienwetter_mit_hist(hist =
                        paste0("Daten\\Wetterdaten\\",
                               "Theresienwetter historisch\\",
                               "theresienwetter_historisch_2013_2014.Rds"),
                        con=ch)
# historische Daten aus enmoFamos-Paket:
gesamt_wetter <- theresienwetter_mit_hist(con=ch)
with(gesamt_wetter, plot(LUFTTEMPERATUR~Datum2, type="l"))
```

---

wetter_hist *historische Wetterdaten des Meteorologischen Instituts Theresienstr 2013 und 2014*

Description

Historische Wetterdaten des Meteorologischen Instituts Theresienstr 2013 und 2014 (1.1.2013 bis 1.9.2014) für die Gradtagzahlbereinigung in zaehlervergleich. Stündliche Werte.

Usage

```
wetter_hist
```

Format

Ein data.frame mit 14582 Zeilen und 11 Variablen:

Datum Datum als Date (nur ganze Tage)

Stunde,Tag,Monat,Jahr Datumsbestandteile als einzelne Integerwerte

DIFFUSSTRAHLUNG DIFFUSSTRAHLUNG

GLOBALSTRAHLUNG GLOBALSTRAHLUNG

LUFTTEMPERATUR LUFTTEMPERATUR

NIEDERSCHLAG NIEDERSCHLAG

RELATIVE_FEUCHTE RELATIVE_FEUCHTE

Datum2 Datum als POSIXct (inkl. Uhrzeit)

Source

Meteorologisches Institut der LMU.

`zaehlerliste` *Aufsplitten der Formel fuer einen Zaehler nach Unterzaehler und Operatoren.*

Description

Aufsplitten der Formel fuer einen Zaehler nach Unterzaehler und Operatoren.

Usage

```
zaehlerliste(formel)
```

Examples

```
zaehlerliste("zaehler1-zaehler2+zaehler3-zaehler4-zaehler5")
```

`zaehlervergleich` *Vergleich des Verbrauchs mehrerer Zaehler aus GLT und Famos mit Gradtagzahl-Bereinigung*

Description

Vergleich des Verbrauchs mehrerer Zähler aus GLT und Famos mit Gradtagzahl-Bereinigung für unterschiedliche Monate oder aus ganzen Monaten bestehende Zeiträume, wobei die Zählerhierarchie, aus der die interessierenden virtuellen Zähler berechnet werden, in einer csv-Datei angegeben wird.

Usage

```
zaehlervergleich(zaehler, zaehlerbeschreibung = NULL, hierarchie,
  einheit = "kWh", zeitraum, gradtagzahlbereinigung = TRUE,
  wetter_hist = "wetter_hist", wetter_tabelle = "theresienwetter",
  wetter_con = con, rueckgabewerte = FALSE, con = NULL,
  spaltenuebersichten = NULL, datumsspalte = "datum", agg_fkt = "sum",
  datei = NULL, methode = c("anteile", "gam"),
  einheiten_angleichen = TRUE, ...)
```

Arguments

zaehler	Welche Zähler (auch virtuell) verglichen werden sollen.
zaehlerbeschreibung	Beschreibung, die zusätzlich zu den Zählernamen in zaehler zur x-Achsenbeschriftung verwendet wird. Character-Vektor, dessen Elemente in der selben Reihenfolge wie Zähler angegeben werde müssen. Falls zaehlerbeschreibung angegeben wird muss es die selbe Anzahl an Elementen haben wie zaehler.
hierarchie	csv-Datei mit der Zähler-Hierarchie (für die virtuellen Zähler) und ob sie sich in Famos oder GLT befinden. Die in der Datei enthaltenen Formeln dürfen nur die Rechenzeichen + und - und keine Klammern enthalten. Zwischen den Rechenzeichen und den Zählernamen darf kein Leerzeichen sein.
einheit	Einheit, in die alle Verbrauchswerte gebracht werden sollen (eines aus "kWh","m³","m3","MWh"). Default: "kWh".
zeitraum	Zeiträume, die miteinander verglichen werden sollen (Monate oder aus ganzen Monaten bestehende Zeiträume).
gradtagzahlbereinigung	Falls TRUE (Default) wird eine Gradtagzahlbereinigung der Werte durchgeführt.
wetter_hist	Pfad zu den historischen (stündlichen) Wetterdaten als Rds-Datei (siehe dgradtagzahl_bereinigung
wetter_tabelle	Datenbanktabelle mit den aktuellen Wetterdaten (siehe dgradtagzahl_bereinigung).
wetter_con	RODBC-Verbindung zur Datenbank mit den aktuellen Wetterdaten. Falls nicht angegeben, wird die Verbindung con verwendet.
rueckgabewerte	Soll eine Liste mit den berechneten Daten (fertige_daten = fertige_daten (Daten aus denen die Grafik erzeugt wird), werte (Daten nach der Abfrage), formeln im Code (Gesamtformeln aus denen die Zähler errechnet werden), bildobjekt (Objekt der Klasse "gg" und "ggplot")) zurückgegeben werden? werte sind die abgefragten, addiert die addierten (und ggf. gradtagzahlbereinigten) Daten und fertige_daten der Datensatz im long-Format aus dem die Grafik erstellt wird.
con	RODBC-Verbindung. Siehe odbcConnect aus dem CRAN-Paket RODBC.
spaltenuebersichten	csv-Dateien mit den Spaltenübersichten zu den GLT-Tabellen, die für die Abfrage benötigt werden. Mit den Namen der Tabellen benannter character-Vektor.
datumsspalte	Name der Spalte in der Tabelle, die das Datum enthält.
agg_fkt	Aggregationsfunktion die für die Zeiträume jeweils ausgeführt werden soll (z.B. "avg" für Durchschnitt, "sum" für die Summe). Im Normalfall immer der Default-Wert "sum".
datei	xlsx-Datei der Famos-Zählerstände.

`methode` Methode, mit der die gewünschten Zeiträume aus den Famos- Zählerständen geschätzt werden sollen (bis jetzt nur anteile fertig implementiert).

`einheiten_angleichen` Sollen die Famos-Zähler auf eine gemeinsame Einheit gebracht werden (dazu werden die Einheitsangaben aus der Spalte Kurzbezeichnung der Datei datei verwendet).

`...` Weitere Argumente, die an `famos_schaetzen` weitergegen werden.

Details

Bei den Zeiträumen in zeitraum bedeutet z.B. "2015" das komplette Jahr 2015, "2015/01" der komplette Januar 2014 und "2014/1-2014/03" der Zeitraum vom 1. Januar bis 31. März 2014. Wird der Monat angegeben, so kann dies einstellig (z.B. "2013/3") oder zweistellig (z.B. "2013/03") geschehen. Um eine Gradtagzahlbereinigung durchführen zu können, sind nur vollständige Monate möglich.

Es können sowohl Zähler aus Famos und GLT nebeneinander betrachtet werden, als auch nur Zähler aus einer der beiden Datenquellen, allerdings muss bei ausschließlicher Verwendung von Famos-Daten beachtet werden, dass, falls eine Gradtagzahlbereinigung durchgeführt werden soll, die RODBC-Verbindung trotzdem benötigt wird, um die LUFTTEMPERATUR abzufragen.

Examples

```
## Not run:
hierarchie <- read.csv2("zaehlerebenen_benchmark2.csv", as.is = TRUE)
# Fehlermeldung, da con nicht angegeben:
zaehlervergleich(c("z\u00e4hler a", "x7079_01_t013klt01waz0001zw01", "z\u00e4hler c"),
                 hierarchie = hierarchie,
                 spaltenuebersichten = c(glt_hist_leo03 = "spaltenuebersicht_leo03.csv",
                                         glt_hist_oet67 = "spaltenuebersicht_oet67.csv",
                                         glt_hist_the46 = "spaltenuebersicht_the46.csv"),
                 einheit = "MWh")

library(RODBC)
ch <- odbcConnect("ansi_system")
hierarchie2 <- read.csv2("zaehlerebenen_benchmark4.csv", as.is = TRUE)
# Alle z\u00e4hler mÃ¼ssen in der hierarchiedatei genau 1x vorhanden sein.
# Fehlt einer der angegebenen Z\u00e4hler, kann die Funktion nicht ausgefÃ¼hrt werden:
zaehlervergleich(c("z\u00e4hler a", "z\u00e4hler b", "z\u00e4hler c", "z\u00e4hler d"),
                 hierarchie = hierarchie2,
                 con = ch, zeitraum = "2014/10",
                 spaltenuebersichten = c(glt_hist_leo03 = "spaltenuebersicht_leo03.csv",
                                         glt_hist_oet67 = "spaltenuebersicht_oet67.csv",
                                         glt_hist_the46 = "spaltenuebersicht_the46.csv"),
                 einheit = "MWh",
              datei = "Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.csv")

# nur 1 Zeitraum:
zaehlervergleich(c("z\u00e4hler a", "z\u00e4hler b", "z\u00e4hler d"),
                 hierarchie = hierarchie2,
                 con = ch, zeitraum = "2014/10",
                 spaltenuebersichten = c(glt_hist_leo03 = "spaltenuebersicht_leo03.csv",
                                         glt_hist_oet67 = "spaltenuebersicht_oet67.csv",
                                         glt_hist_the46 = "spaltenuebersicht_the46.csv"),
                 einheit = "MWh",
              datei = "Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.csv")
```

```
# mehrere Zeitr\u00e4ume:
zaehlervergleich(c("z\u00e4hler a", "z\u00e4hler b", "z\u00e4hler d"), hierarchie = hierarchie2,
                 con = ch, zeitraum = c("2014/9","2014/10"),
                 spaltenuebersichten = c(glt_hist_leo03 = "spaltenuebersicht_leo03.csv",
                                         glt_hist_oet67 = "spaltenuebersicht_oet67.csv",
                                         glt_hist_the46 = "spaltenuebersicht_the46.csv"),
                 einheit = "MWh",
               datei = "Daten/Famos_daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.csv",
                 rueckgabewerte = TRUE)
# ohne Gradtagzahlbereinigung und mit Z\u00e4hlerbeschreibung:
zaehlervergleich(c("z\u00e4hler d", "z\u00e4hler a", "z\u00e4hler b"),
                 zaehlerbeschreibung=c("blad", "blaa", "blab"),
                 hierarchie = hierarchie2,
                 con = ch, zeitraum = c("2014/9","2014/10"),
                 spaltenuebersichten = c(glt_hist_leo03 = "spaltenuebersicht_leo03.csv",
                                         glt_hist_oet67 = "spaltenuebersicht_oet67.csv",
                                         glt_hist_the46 = "spaltenuebersicht_the46.csv"),
                 einheit = "MWh",
                datei = "Daten/20150128 Z\u00e4hlerstand Stammgel\u00e4nde Komplett.csv",
                 rueckgabewerte = TRUE, gradtagzahlbereinigung = FALSE)

## End(Not run)
```

`zeitraum_format` *Uebergabewerte fuer zeitraum an benchmark in das richtige Format fuer die GLT bzw. Famos bringen*

Description

Uebergabewerte fuer zeitraum an benchmark in das richtige Format fuer die GLT bzw. Famos bringen

Usage

```
zeitraum_format(zeitraum, zweck = c("glt", "famos"))
```

Arguments

`zeitraum`	Zeitraum als Jahr (z.B. "2015"), Monat (z.B. "2015/1") oder mehrere ganze Monate (z.B. "2014/1-2015/3").
`zweck`	Umwandlung für die Abfrage mittels glt_benchmark ("glt") oder famos_schaetzen ("famos").

Value

Zeitraum im gewünchten Format.

Examples

```
zeitraum_format("2015/01", "glt")
zeitraum_format("2015/1", "famos")
zeitraum_format("2015", "glt")
zeitraum_format("2015", "famos")
zeitraum_format("2014/2-2015/3", "glt")
zeitraum_format("2014/02-2015/03", "famos")
```

Index

A.2 Weitere Codes

A.2.1 Daten nachträglich in einer Datenbanktabelle hinzufügen

```
#mit 2 Spalten:

glt_hist(con = ch, tabelle = "oet_nachtragen_tests", namensteil = "Oet67",
         quellordner = "D:/EnMoLMU/Daten/Glt_updates fuer tests",
         ferien = "D:/EnMoLMU/Daten/sonstige_Daten/ferien.csv")

# welche Spalten in der Tabelle
sqlColumns(ch, "oet_nachtragen_tests")$COLUMN_NAME

# Spalte x7070_01_u138hzg03tem0002mp01 entfernen:
sqlQuery(ch, "ALTER TABLE oet_nachtragen_tests DROP x7070_01_u138hzg03tem0002mp01")
sqlQuery(ch, "ALTER TABLE oet_nachtragen_tests DROP x7070_01_u138hzg03wmz0001zw02")

# beide Spalte jetzt nicht mehr in der Tabelle
sqlColumns(ch, "oet_nachtragen_tests")$COLUMN_NAME

# GLT-Datei für zweiten Tag in den Ordner quellordner verschoben.
update_glt2(con = ch, tabelle = "oet_nachtragen_tests", name = "Oet67",
            ordner = "D:/EnMoLMU/Daten/Glt_updates fuer tests",
            ferien = "D:/EnMoLMU/Daten/sonstige_Daten/ferien.csv")

# Spalte x7070_01_u138hzg03tem0002mp01 undx7070_01_u138hzg03wmz0001zw02
#  wieder vorhanden:
sqlColumns(ch, "oet_nachtragen_tests")$COLUMN_NAME

sqlQuery(ch, "select x7070_01_u138hzg03tem0002mp01,x7070_01_u138hzg03wmz0001zw02
         from oet_nachtragen_tests limit 10")
# Werte des ersten Tags fehlend
sqlQuery(ch, "select x7070_01_u138hzg03tem0002mp01,x7070_01_u138hzg03wmz0001zw02
         from oet_nachtragen_tests WHERE datum >= '2015-02-17' limit 10")
```

```
# Werte des zweiten Tags vorhanden

daten <- read.csv2("D:/EnMoLMU/Daten/Glt_updates fuer tests/150216Oet67EnMoLMU.csv")
# Datum muss von factor in POSIXct umgewandelt werden:
daten <- cbind(daten, datum2 = as.POSIXct(as.character(daten$Datum),
                                          format = "%d.%m.%Y %H:%M:%S", tz="GMT"))
# sollen weitere Spalten, wie Differenzen, hinzugefügt werden, so müssen diese
# an dieser Stelle an daten angehängt werden
names(daten)
db_spalten_hinzufuegen(ch, "oet_nachtragen_tests", df = daten,
                       spalten_tabelle = NULL, # automat. Umwandlung des df-Namens
                       spalten_df = c("X7070.01_U138HZG03TEM0002MP01",
                                      "X7070.01_U138HZG03WMZ0001ZW02"),
                       datum_df = "datum2") # umgewandeltes Datum aus daten verwenden

sqlQuery(ch, "select x7070_01_u138hzg03tem0002mp01,x7070_01_u138hzg03wmz0001zw02
         from oet_nachtragen_tests limit 10")
# jetzt auch wieder werte des ersten Tages vorhanden

#Test-Tabelle wieder löschen:
sqlTables(ch)
sqlDrop(ch, "oet_nachtragen_tests")

# debug(db_spalten_hinzufuegen)
# names(save_argumente[[2]])[-1] <- mache_db_namen(names(save_argumente[[2]])[-1])
# nach
# save_argumente <- list(con, df[, c(datum_df, spalten_df)],
#                        tablename = df_name, rownames = FALSE)
```

A.2.2 MySQL Joins

```
mysql> select * from t1;
+------+------+
```

```
| nr   | wert |
+------+------+
|    1 |   11 |
|    2 |   12 |
|    3 |   13 |
+------+------+
3 rows in set (0.00 sec)

mysql> select * from t2;
+------+-------+
| nr   | wert2 |
+------+-------+
|    2 |    22 |
|    3 |    23 |
|    4 |    24 |
+------+-------+
3 rows in set (0.00 sec)

mysql> select * from t1 join t2 on(t1.nr = t2.nr);
+------+------+------+-------+
| nr   | wert | nr   | wert2 |
+------+------+------+-------+
|    2 |   12 |    2 |    22 |
|    3 |   13 |    3 |    23 |
+------+------+------+-------+
2 rows in set (0.00 sec)

mysql> select * from t1 join t2 using(nr);
+------+------+-------+
| nr   | wert | wert2 |
+------+------+-------+
|    2 |   12 |    22 |
|    3 |   13 |    23 |
+------+------+-------+
2 rows in set (0.00 sec)
```

```
mysql> select * from t1 left join t2 using(nr);
+------+------+-------+
| nr   | wert | wert2 |
+------+------+-------+
|    2 |   12 |    22 |
|    3 |   13 |    23 |
|    1 |   11 |  NULL |
+------+------+-------+
3 rows in set (0.00 sec)

mysql> select * from t1 right join t2 using(nr);
+------+-------+------+
| nr   | wert2 | wert |
+------+-------+------+
|    2 |    22 |   12 |
|    3 |    23 |   13 |
|    4 |    24 | NULL |
+------+-------+------+
3 rows in set (0.00 sec)
```

A.3 Skripte

A.3.1 R-Skripte

Anlegen der GLT-Subserver-Tabellen

```
library(RODBC)
library(EnMoLMU)

ch <- odbcConnect("ansi_benutzer", DBMSencoding = "utf-8")

glt_hist_vorbereiten(datei = "D:\\EnMoLMU\\Daten\\GLT_hist\\alles_h_Oet67EnMoLMU.csv",
```

```
                    tabelle = "oet67", con = ch, zwischenschritte_loeschen = TRUE,
                    ferien = "D:\\EnMoLMU\\Daten\\sonstige_Daten\\ferien.csv")

# Consolenausgabe:
# Fortschrittsanzeige aufteilen des Datensatzes in einzelne Tage
# Auflistung aller Dateien, die zum Umbauen gefunden wurden
# Ausgabe zur groben Überprüfung der Daten in der Tabelle:
# insbesondere auf richtiges Format von Datum und Uhrzeit und Vollständigkeit
#count(*) min(datum)          max(datum)
#1  1051200 2013-02-20 2015-02-19 23:59:00
#(1 Beobachtung für jede Minute im Zeitraum und Beginn 0:00, Ende 23:59)
# gesamt und für die einzelnen Monate
# 0     15     30     45
# 262800 262800 262800 262800
# Häufigkeitstabelle der cut15min-Werte (sollten alle gleich häufig sein)

glt_hist_vorbereiten(datei =
                    c("D:\\EnMoLMU\\Daten\\GLT_hist\\2013_02_19_h_The46EnMoLMU.csv",
                      "D:\\EnMoLMU\\Daten\\GLT_hist\\2014_02_19_h_The46EnMoLMU.csv"),
                    tabelle = "the46", con = ch, zwischenschritte_loeschen = TRUE,
                    ferien = "D:\\EnMoLMU\\Daten\\sonstige_Daten\\ferien.csv")
# historische Daten für Subserver The46 musste auf 2 Dateien aufgeteilt
#  exportiert werden.

glt_hist_vorbereiten(datei = "D:\\EnMoLMU\\Daten\\GLT_hist\\alles_h_Leo03EnMoLMU.csv",
                    tabelle = "leo03", con = ch, zwischenschritte_loeschen = TRUE,
                    ferien = "D:\\EnMoLMU\\Daten\\sonstige_Daten\\ferien.csv")
```

Aktualisieren der GLT-Subserver-Tabellen

```
library(RODBC)
library(EnMoLMU)

ch <- odbcConnect("ansi_benutzer", DBMSencoding = "utf-8")
```

```
# Aktualisierung der Tabellen für die bereits angelegten
#  Subserver Oet67, The46, Leo03

update_glt2(ch, tabelle = "oet67", name = "Oet67", name_allgemein = "EnMoLMU",
            ordner = "D:\\EnMoLMU\\Daten\\GLT_updates",
            ferien = "D:\\EnMoLMU\\Daten\\sonstige_Daten\\ferien.csv",
            einzeln = FALSE
)

update_glt2(ch, tabelle = "the46", name = "The46", name_allgemein = "EnMoLMU",
            ordner = "D:\\EnMoLMU\\Daten\\GLT_updates",
            ferien = "D:\\EnMoLMU\\Daten\\sonstige_Daten\\ferien.csv",
            einzeln = FALSE
)

update_glt2(ch, tabelle = "leo03", name = "Leo03", name_allgemein = "EnMoLMU",
            ordner = "D:\\EnMoLMU\\Daten\\GLT_updates",
            ferien = "D:\\EnMoLMU\\Daten\\sonstige_Daten\\ferien.csv",
            einzeln = FALSE
)
```

Anlegen bzw. Aktualisieren der Wetterdaten-Tabelle

```
library(RODBC)
library(EnMoLMU)

ch <- odbcConnect("ansi_benutzer", DBMSencoding = "utf-8")

# Tabelle theresienwetter anlegen bzw. aktualisieren:
update_wetter(ch, ordner = "D:\\EnMoLMU\\Daten\\Wetterdaten")
```

Erstellen der Berichte

```
library(EnMoLMU)

pfad <- "D:\\EnMoLMU\\"


# Alle Vorlagen als PDF-Bericht erstellen
knit_all_format(ordner = paste(pfad, "Berichte\\Berichtvorlagen",sep=""),
                output_ordner = paste(pfad, "Berichte\\Berichte_fertig", sep=""),
                format = "pdf")
```

A.3.2 Batch-Skripte

Aufruf des R-Skriptes zum Erstellen der Berichte

```
rem Dieses Skript ruft das R-Skript auf, das aus den Vorlagen die Berichte erstellt.
rem Der Pfad zu "Berichte erstellen - buero.R"
rem (und evtl. auch dessen Name, falls das R-Skript anders heißt)
rem  muss ggf. angepaßt werden!

R CMD BATCH "D:\EnMoLMU\Skripte\R\Berichte erstellen - server -nur pdf.R"
pause
rem pause nur damit das Fenster nicht sofort wieder geschlossen
rem wird und man die Ausgabe sieht.
```

A.4 Berichtsvorlage R Markdown

```
---
title: "Beispielbericht"
author: "Thomas"
date: ’‘r Sys.time()‘’
output: pdf_document
```

```
---

# Beispielbericht

Dies ist ein Beispielbericht vom `r Sys.time()` mit einer Reihe von Grafiken.

```{r, echo=FALSE, error=FALSE, warning=FALSE, include=FALSE}
# Laden der benötigten Pakete
library(EnMoLMU)
library(RODBC)

# ODBC-Verbindung zu MySQL öffnen:
ch <- odbcConnect("ansi_benutzer",  DBMSencoding = "utf-8")
```

\pagebreak

Energiesignatur aufgeteilt nach Werktag/Wochenende und Vorlesungszeit/
Semesterferien (energiesignatur_aufgeteilt_db_ferien)

Energiesignatur des Zählers diff_x7070_01_u138hzg03wmz0001zw01 und der
Außentemperatur x7070_01_u138hzg03tem0003mp01
für die Zeiträume vom 1.1.2014 bis 1.4.2015
(Modelle schätzen) und 2.4.2015 bis 30.4.2015
(überprüfen).

```{r Energiesignatur aufgeteilt ferien, echo=FALSE}
 energiesignatur_aufgeteilt_db_ferien(ch, "oet67",
"diff_x7070_01_u138hzg03wmz0001zw01", "x7070_01_u138hzg03tem0003mp01",
"'2014-01-01' and '2015-04-01'", "'2015-04-02' and '2015-04-30'")

```

\pagebreak
```

```
## Tagesverlauf
### Oettingenstraße 67

```{r Tagesverlauf Oet67, echo=FALSE}
 tagesverlauf_db(ch, tabelle = "oet67", datum = "datum",
                werte = "diff_x7070_01_u138hzg03wmz0001zw01",
                aussentemp = "x7070_01_u138hzg03tem0003mp01",
                abschnitte = "15 Minuten",
                tabelle2 = "theresienwetter", strahlung = "globalstrahlung",
                between = "'2013-01-01 00:00:00'  and '2015-12-30 00:00:00'")

```

\pagebreak

Theresienstraße 46

```{r Tagesverlauf The46, echo=FALSE}
 tagesverlauf_db(ch, tabelle = "the46", datum = "datum",
                 werte = "diff_x1120_01_021anhv01elz0001zw30",
                 aussentemp = "x1110_01_0032hzg00tem0001ma01",
                 tabelle2 = "theresienwetter", strahlung = "globalstrahlung",
                 abschnitte = "15 Minuten",
                 between="'2015-01-01 00:00:00' and '2015-05-01 00:00:00'")

```

\pagebreak

Temperaturvergleich (H_Heizkreise Rücklauf)

```{r Temperaturvergleich, echo=FALSE}
 hydr_abgleich_db_t(ch, "oet67",
               c("x7070_01_u138hzg03tem0001mp01",
                 "x7070_01_u138hzg03tem0002mp01",
```
```

```
                  "x7070_01_u138hzg10tem0002mp01",
                  "x7070_01_u138hzg11tem0002mp01",
                  "x7070_01_u138hzg13tem0002mp01"),
                  "x7070_01_u138hzg03tem0003mp01",takt = "15 min",
                 between = "'2015-03-01' and '2015-05-31'",
                kurzbezeichnungen =
                 "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_oet67.csv")
```

```

\pagebreak

## Jahresdauerlinie (Kälteleistung)

Aggregatsfunktion: avg (Stundendurchschnittswerte)

Jahresdauerlinie im Stundentakt über die letzten zwei Jahre vor
 der Berichtserstellung.

```{r Jahresdauerlinie, echo=FALSE}
 jahresdauerlinien_db_t(ch, "oet67", "x7079_01_t013klt01waz0002zw01",
                      intervalle = "Stunden",
                      between = letzte_2jahre(),
                      ylab =
                       "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_oet67.csv",
                      einheit =
                       "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_oet67.csv")
```

```

\pagebreak
## Jahresdauerlinie (Aussentemperatur)

```
Aggregatsfunktion: avg


```{r Jahresdauerlinie 2, echo=FALSE}
 jahresdauerlinien_db_t(ch, "oet67", "x7070_01_u138hzg03tem0003mp01",
                       intervalle = "Stunden",
                       between = letzte_2jahre(),
                       agg_fkt = "avg",
                       werte_between = "-20 and 50",
                       ylab =
                        "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_oet67.csv",
                       einheit =
                        "D:/EnMoLMU/Daten/sonstige_Daten/spaltenuebersicht_oet67.csv")

```


\pagebreak
Jahresdauerlinie (Aussentemperatur Wetterdaten)

Aggregatsfunktion: avg


```{r Jahresdauerlinie 3, echo=FALSE}
 jahresdauerlinien_db_t(ch, "theresienwetter", "aussentemperatur",
                       intervalle = "Stunden",
                       between = letzte_3jahre(),
                       agg_fkt = "avg",
                       ylab = "Außentemperatur")

```
```

A.5 Shiny

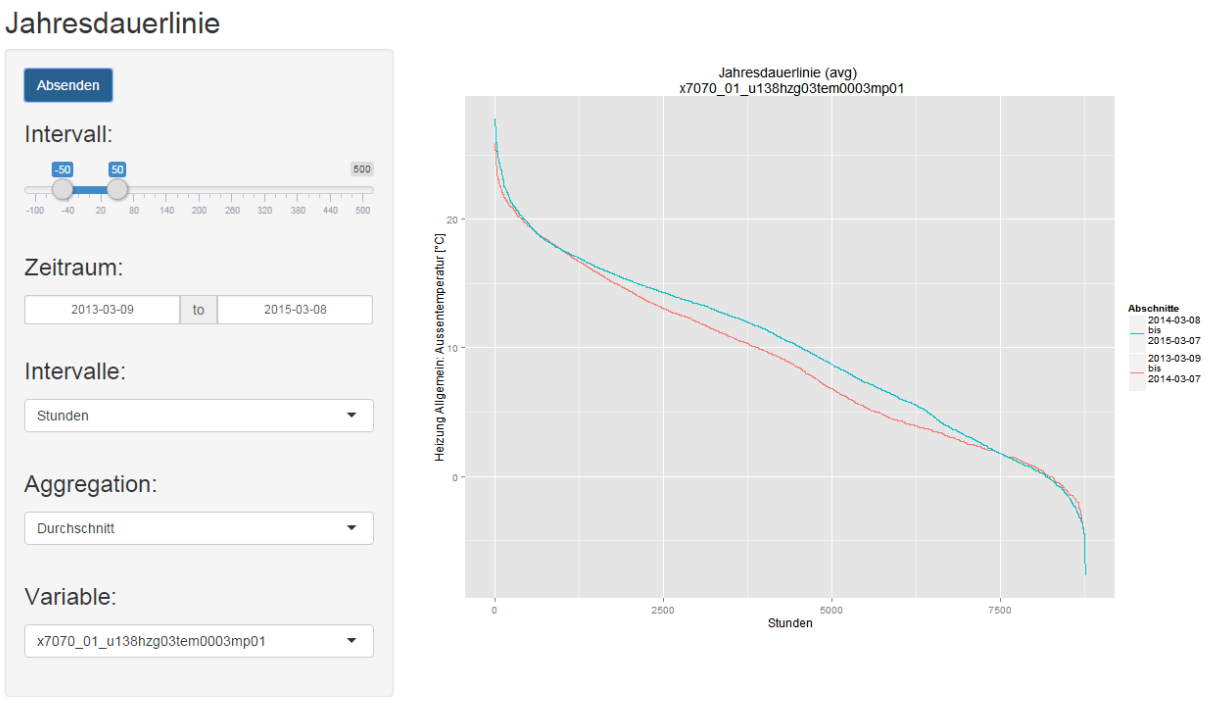


Abbildung A.1: Interaktive Grafik mit *shiny*: Jahresdauerlinie

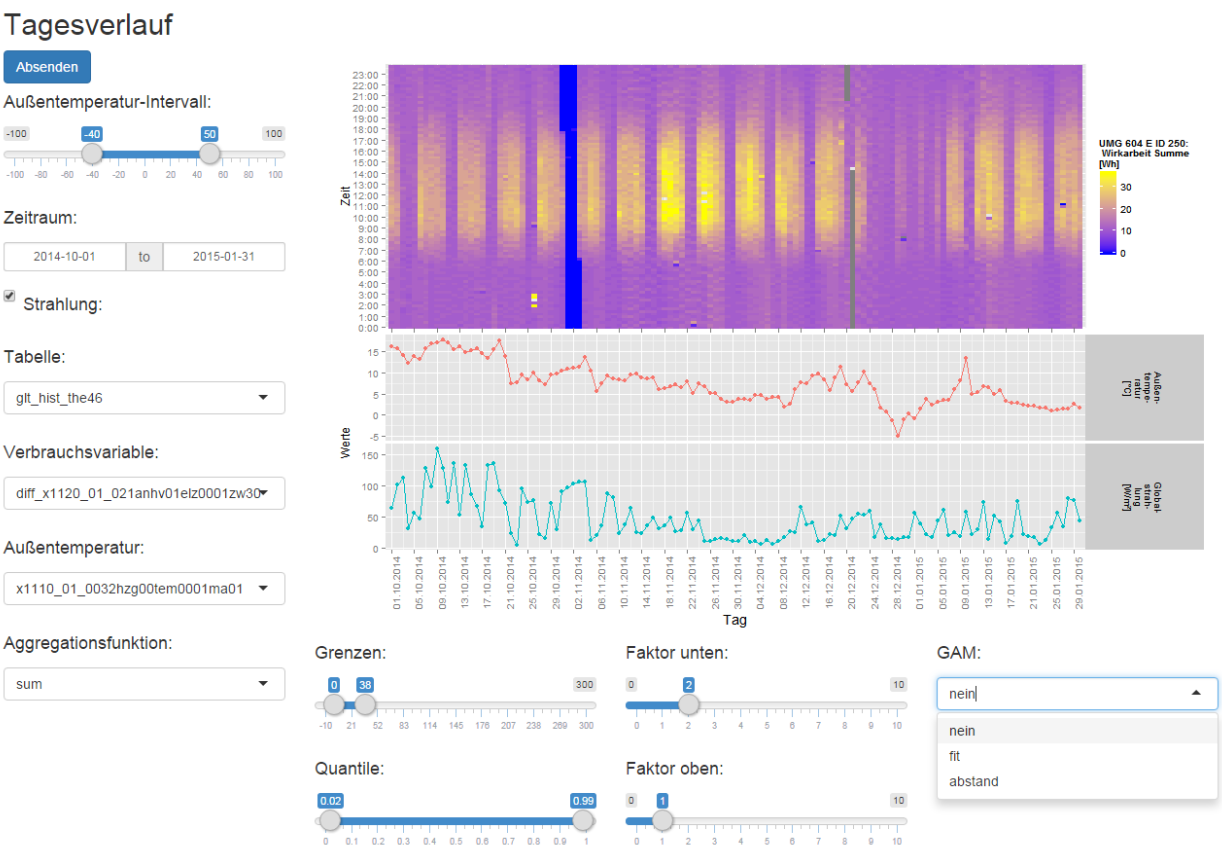


Abbildung A.2: Interaktive Grafik mit *shiny*: Tagesverlauf

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum Unterschrift

Literaturverzeichnis

[1] Jörg Baldenhofer, Gerlinde Baur, Günter Cromm et al, *Betriebskosten und Verbräuche - Kennwerte von Hochbauten*. Herausgegeben im Auftrag des Finanzministeriums Baden-Württemberg, Stuttgart, Juni 2004.

[2] Hadley Wickham, `http://r-pkgs.had.co.nz/intro.html`, aufgerufen am 03.05.2015

[3] `http://dev.mysql.com/doc/`, aufgerufen am 18.03.2015.

[4] `http://dev.mysql.com/doc/refman/5.6/en/table-size-limit.html`, aufgerufen am 18.03.2015.

[5] `http://db-engines.com/de/ranking`, aufgerufen am 18.03.2015.

[6] `http://de.wikipedia.org/wiki/MySQL#cite_note-8`, aufgerufen am 18.03.2015.

[7] `http://dev.mysql.com/doc/refman/5.6/en/column-count-limit.html`, aufgerufen am 18.03.2015.

[8] `http://dev.mysql.com/doc/refman/5.6/en/storage-requirements.html`, aufgerufen am 18.03.2015.

[9] `https://dev.mysql.com/doc/refman/5.6/en/join.html`, aufgerufen am 21.03.2015.

[10] `http://de.wikipedia.org/wiki/Leistungszahl`, aufgerufen am 03.06.2015.

[11] `http://de.wikipedia.org/wiki/Jahresdauerlinie`, aufgerufen am 03.06.2015.

[12] Joachim Liers, *Gastbeitrag: Verwendung von Energiesignaturen zur Betriebsüberwachung von Gebäuden*. HIS:Mitteilungsblatt Arbeits-, Gesundheits- und Umweltschutz, Nr. 2 April 2014.

[13] Christian Neumann, Dirk Jacob, Sebastian Burhenne, Anthony Florita, Eberhard Burger, Fritz Schmidt, *Modellbasierte Methoden für die Fehlererkennung und Optimierung im Gebäudebetrieb - Endbericht*. Frauenhofer ISE, Freiburg, Juli 2011,

[14] `http://www.yourhelpcenter.com/2008/12/mysql-prozesse-auflisten-stoppen`, aufgerufen am 10.06.2015

[15] `http://de.wikipedia.org/wiki/Koordinierte_Weltzeit`, aufgerufen am 13.04.2015

[16] Ludwig Fahrmeir, Thomas Kneib, Stephan Lang, *Regression - Modelle, Methoden und Anwendungen* Springer-Verlag Berlin Heidelberg 2007

[17] Roland Bopp, *Verbrauchsüberwachung Wärme und Kälte*; Vortrag auf dem HISPraxisseminar 'Energiecontrolling und Energieeffizienz in Hochschulen' 19. bis 21.06.2006 an der TU Clausthal

[18] `http://web.mit.edu/~r/r_v3.1.1/lib/R/library/mgcv/html/smooth.terms.html`, aufgerufen am 13.09.2015

[19] `https://de.wikipedia.org/wiki/Quantil`, aufgerufen am 27.09.2015

[20] Christian Heumann, Volker Schmid, *Schätzen und Testen II - Skript zur Vorlesung SS 2014*, `http://www.statistik.lmu.de/institut/lehrstuhl/wisoz/lehre/schaetzen_testen2_ss15/Vorlesung/Skript/Kapitel_6.pdf`, aufgerufen am 27.09.2015

[21] Roger Koenker, Gilbert Bassett, Jr., *Regression Quantiles*, Econometrica, Vol. 46, No. 1. (Jan., 1978), pp. 33-50

[22] Xuming He, Huixida Judy Wang, *A Short Course on Quantile Regression*, `http://www.wise.xmu.edu.cn/UploadFiles/SS2011/Uploadfiles/2013714144421348.pdf`, aufgerufen am 28.09.2015

[23] Ludwig Fahrmeir, Rita Künstler, Iris Pigeot, Gerhard Tutz, *Statistik - Der Weg zur Datenanalyse*, Springer-Verlag Berlin Heidelberg New York 2007

[24] `https://de.wikipedia.org/wiki/Wahrscheinlichkeitsdichtefunktion`, aufgerufen am 28.09.2015

[25] Ronaldo Dias, `http://www.ime.unicamp.br/~dias/loess.pdf`, aufgerufen am 04.10.2015

[26] `https://en.wikipedia.org/wiki/Local_regression`, aufgerufen am 04.10.2015

[27] R Core Team (2014), *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, `http://www.R-project.org/`.