Ludwig-Maximilians-Universität München

Fakultat für Mathematik, Informatik und Statistik

# Forecasting of COVID-19 bed occupancy in German intensive care units:

## A comparison between supervised learning and classical time series forecasting methods

Master's Thesis

Author:

**Mila Yamakova**

Supervisor:

**Prof. Dr. Helmut Küchenhoff**

**Maximilian Weigert**

Department of Statistics

Ludwig-Maximilians-Universität München

## Abstract

Throughout the Coronavirus pandemic, the amount of available COVID-19 related data is increasing steadily and with it the potential to extract valuable information. Data-related insights enable decision-makers in healthcare and politics to make informed and data-driven decisions. Alongside COVID-19 incidence rates and hospitalization rates, the intensive care capacity (ICU) occupancy by COVID-19 patients is an essential key metric for the magnitude of the pandemic outbreak. This thesis focuses on producing reliable forecasts of ICU bed occupancy by COVID-19 patients on German district level for a time forecasting horizon of up to two weeks.

Current literature has been dominated by research on the effective prediction performance of various machine learning models. In the last two decades, machine learning (ML) models have been increasingly adopted to solve forecasting tasks and have established themselves as serious contenders to classical statistical models. However, there is a small amount of work comparing their predictive performance relative to traditional statistical methods. Within this thesis, we conduct a benchmark experiment, which investigates two ensemble methods based on regression trees, random forest and gradient boosting, and compares them to one of the most widely used statistical forecasting models - autoregressive integrated moving average (ARIMA). Alongside data on COVID-19 ICU occupancy, additional COVID-19-related data is included in the machine learning approaches, to enhance prediction performance.

The results from the benchmark experiment show that all three models perform comparably well on average across all districts and prediction horizons. With increasing prediction horizon, random forest and gradient boosting moderately outperform ARIMA. These results suggest that random forest and gradient boosting provide a powerful alternative to ARIMA but are accompanied by higher computational expenses. Therefore, it is strongly recommended to use classical statistical methods as a baseline to more elaborate ML methods in order to justify their usage. The benchmark experiment results are accompanied by variable importance measures and interactive visualizations of the produced forecasts.

**Keywords**: COVID-19, Coronavirus, ICU bed occupancy, time series forecasting, multi-step forecasting, ARIMA, regression trees, random forest, gradient tree boosting, `skranger`, `XGBoost`, walk-forward cross-validation, benchmark experiment, benchmark experiment, variable importance, uncertainty quantification

# Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| ACF | autocorrelation function |
| AR | autoregressive |
| ARIMA | autoregressive integrated moving average |
| BO | bayesian optimization |
| CART | classifications and regression tree |
| CCDF | complementary cumulative distribution function |
| CET | central european time |
| COVID-19 | coronavirus disease 2019 |
| CV | cross-validation |
| DIVI | Deutschen Interdisziplinären Vereinigung für Intensiv- und Notfallmedizin |
| e.g. | exempli gratia |
| GB | gradient boosting |
| GS | grid search |
| i.e. | id est |
| i.i.d. | independent and identically distributed |
| ICU | intensive care unit |
| MA | moving average |
| MAE | mean absolute error |
| ML | machine learning |
| PACF | partial autocorrelation function |
| RF | random forest |
| RKI | Robert Koch Institute |
| RMSE | root mean squared error |
| RS | random search |
| SARS-CoV-2 | severe acute respiratory syndrome coronavirus type 2 |
| WHO | World Health Organization |
| WN | white noise |

# 1. Introduction

On March 11, 2020, the World Health Organization (WHO) characterized the global spread of the novel coronavirus (SARS-CoV-2) as a pandemic. The COVID-19 pandemic is constantly bringing about enormous challenges for the society, the economy, and the health system worldwide. In the middle of a health crisis, it is crucial to adequately control the pandemic in the short-term as well as long-term and to strengthen the response capacity to the health needs of the population. To enable timely decision-making, we need well established and reliable key indicators for evaluating the severity of the coronavirus situation.

Throughout the Coronavirus pandemic, the available COVID-19 related data is increasing steadily, with it the potential to extract valuable information. Data-related insights enable decision-makers in healthcare and politics to make informed and data-driven decisions. The 7-day reporting incidence (number of new infections per 100.000 inhabitants) is an important measurement for the spread of the pandemic and has been the central parameter for assessing the pandemic situation in Germany until October 2021. With the change in the dynamics of the pandemic, such as progressed vaccination and the advanced testing strategy, the number of reported cases has become an insufficient criterion for evaluating the severity of the pandemic situation. Recently, the reporting incidence has been accompanied by measurements of the utilization of the healthcare system as complementary indicators. The occupancy of intensive care units and the new hospital admissions of COVID-19 patients have become important key metrics for describing the pandemic process in many German states. These metrics represent the epidemiological dynamics by focusing on severe courses of the disease and have the advantage of accurately reflecting the magnitude of the pandemic outbreak. Moreover, available data on the level of occupancy of intensive care units (ICUs) by COVID-19 patients is updated daily and is accurate and complete. Meanwhile, data on reported new infections and COVID-19 hospital admissions, in general, is associated with reporting delays or a considerable number of unreported cases, which makes real-time analysis of the current and expected situation more difficult. For these reasons, we set the focus of this thesis on the level of occupancy of ICUs by COVID-19 patients.

The aim of this thesis is to investigate the potential of various machine learning methods for producing reliable forecasts on German intensive care units occupancy by COVID-19 patients and to compare them with a classical statistical time series model as a benchmark. Within the framework of this work, forecasting is based on a time horizon for up to two weeks.

In the last two decades, machine learning models have been increasingly adopted to solve forecasting tasks and have established themselves as serious contenders to classical statistical models (see Ahmed et al., 2010; Zhang et al., 1998). Machine learning models are non-parametric methods, which explicitly evaluate the spectrum or the covariance of the stochastic process without making strong assumptions on the structure of the process. These methods have shown to be very effective but can also be computationally expensive (Boulesteix and Schmid, 2014). Meanwhile, parametric models are often preferred, due to their transparency and moderate computational complexity. The parametric methods assume that the basic stochastic stationary process has a certain structural formation which may be described by utilizing a small number of parameters (Gautam and Singh, 2020).

Current literature has been dominated by research on the effective prediction performance of various machine learning models. However, there is a small amount of work comparing their predictive performance relative to traditional statistical methods (Cerqueira et al., 2019). Within the framework of this work, we conduct a benchmark experiment, which investigates the prediction performance of different machine learning methods for ICU bed occupancy forecasting, compared to a classical statistical forecasting approach. The performed experiment focuses on ensemble methods based on regression trees (ensemble tree methods), more precisely on the methods random forest (RF) and gradient boosting (GB). These models have been widely used in practice and numerous studies have demonstrated their abilities to produce robust prediction models (see Caruana and Niculescu-Mizil, 2006; Chen and Guestrin, 2016; Rokach, 2016). Additionally, these models exhibit many useful properties – the ability to automatically find the structure of the model, successful incorporation of non-linear patterns and interactions in the data, and ability to deal with high-dimensional data (Boulesteix and Schmid, 2014). As a benchmark, we incorporate one of the most widely used statistical forecasting models - autoregressive integrated moving average (ARIMA). The ARIMA model is backed up by a solid theoretical foundation and is particularly effective in short-term forecasting (O'Donovan, 1983). Alongside producing point forecasts, we focus on quantifying the uncertainty of the predicted values by constructing prediction intervals for different prediction horizons. To reflect the uneven spread of the disease across the country, ICU occupancy forecasting is conducted on a regional level, i.e., for each German district.

While ARIMA is a purely autoregressive model, which explicitly relies on past observations of the series, machine learning models have shown to be successful in incorporating various exogenous variables into the modeling procedure. To exploit this advantage and better reflect the dynamic development of the pandemic, we incorporate additional publicly available COVID-19 related data into the ML models. Alongside data on occupied ICU beds, data on the reported number of new cases and deaths in different age groups, as well as data on vaccination rates is included. Our aim is to investigate the contribution of additional key metrics of the pandemic for producing more reliable estimates. For that, different variable importance techniques are explored. In this context, a further desirable property of the investigated tree-based ensemble methods is that they provide embedded variable importance measurement.

Chapter 2 presents the necessary methodological background knowledge of this thesis. This chapter introduces basic concepts of time series analysis and forecasting. In addition, the autoregressive integrated moving average approach to time series forecasting is introduced, including details on multi-step forecasting and the construction of prediction intervals. Furthermore, this chapter introduces relevant machine learning methods, focusing on regression trees, random forest, and gradient boosting. Finally, important concepts of model interpretability are discussed, investigating methods for uncertainty quantification, as well as quantifying the impact of different features, also known as the concept of variable importance.

Chapter 3 focuses on the data used in the project. After a description of the raw data sources and structures, the conducted data pre-processing and manipulation such as feature engineering, missing data imputation, and variable recording are introduced. Additionally, the final choice of features and the creation of a supervised learning dataset is discussed. Finally, we take a look into the structure of the data and the developments of key COVID-19 pandemic measures. We motivate the importance of considering different key metrics and their change over time to better capture dynamic variations in the environment.

Chapter 4 presents the conducted benchmark experiment. This section discusses major aspects of the benchmark implementation strategy - the experimental setup, the choice of model implementations, the evaluation strategies and metrics, and hyperparameter tuning. We discuss time series specific evaluation and tuning strategies, which respect the chronological order of the data and support capturing dynamic changes. In addition, main hyperparameters and their impact on prediction performance are examined and different hyperparameter tuning strategies are discussed.

Chapter 5 comprises the results of the conducted benchmark experiment. After an overview of the results from the hyperparameter tuning procedure, prediction performances of the selected models are compared. We take a look into the average results, as well as into the geographical distribution of prediction accuracy across all German districts. To provide an overview of the prediction curves, we have developed an interactive visualization tool, which retrospectively compares predicted and observed time series for each district and forecast horizon. Additionally, the insights obtained by the variable importance measures are summarized.

The results are accompanied by a discussion on important findings, limitations and future work propositions in Chapter 6. Finally, Chapter 7 summarizes the subject matter and emphasizes the main findings.

# 2. Methodological Background

This section discusses the necessary methodological background knowledge of this thesis. First, we provide an introduction to basic concepts of time series analysis and forecasting. In addition, the autoregressive integrated moving average (ARIMA) approach to time series forecasting is presented, including details on multi-step forecasting and the construction of prediction intervals. Furthermore, we examine the relevant machine learning methods for this thesis, focusing on regression trees, random forest, and gradient boosting. Finally, we focus on model interpretability by introducing methods for quantifying uncertainty, as well as methods for quantifying the impact of different features, also known as the concept of variable importance.

## 2.1 Basic Concepts of Time Series Analysis

### 2.1.1 Time Series Data

A time series is a set of observations measured sequentially through time. These measurements may be made continuously through time (continuous time series) or be taken at a discrete set of time points (discrete time series). From a statistical point of view, time series are regarded as the recording of the stochastic process which varies over time (Shumway and Stoffer, 2006).

A time series $\{y_t : t \in T\}$ is referred to as a stochastic process which in turn is composed of random variables observed over time. For the scope of this thesis $t \in \mathbb{Z}$ is discrete and varies over a subset $\mathbb{Z}$ of the integers. We assume that the time series values we observe $y_1, y_2, ..., y_n$ are the realisations of random variables $Y_1, Y_2, ..., Y_n$ of the stochastic process $\{y_t : t \in T\}$. Throughout this thesis and depending on the context, we use the term time series for both references: the process or the particular realization.

### 2.1.2 Stationarity

One of the most useful concepts in time series modeling is to assume some form of distributional invariance over time or stationarity. When we observe a time series, the fluctuations appear random, but often exhibit the same type of stochastic behaviour from one time period to the next. Stationary stochastic processes are probability models for time series with time-invariant behaviour.

A time series is said to be *strictly stationary* if the probabilistic behaviour of every collection of values remains unchanged by shifts in time (Shumway and Stoffer, 2006). Mathematically, stationarity is defined as the requirement that for every $m$ and $n$, the joint distribution of $y_1, ..., y_n$ is the same as the joint distribution of $y_{1+m}, ..., y_{n+m}$. The version of stationarity is too strong for most applications and it is often difficult to assess strict stationarity from a single dataset (Shumway and Stoffer, 2006). Moreover, many important questions relating to a stochastic process can be adequately answered by imposing fewer conditions on the process.

A *weakly stationary* time series $\{y_t : t \in T\}$ is a process meeting two conditions. First, the mean value function

$$\mu_t = \mathbb{E}_t$$

is constant and does not depend on time $t$. Second, the autocovariance function

$$\gamma_{t,t+s} = Cov(y_t, y_{t+s}) = \mathbb{E}[(y_t - \mu_t)(y_{t+s} - \mu_{t+s})]$$

of two time point $t$ and $t+s$ depends only through their difference $s = |t+s-t|$. This means that for all $s$ the time series $\{y_t\}$ moves in a similar way as the 'shifted' time series $\{y_{t+s}\}$. Throughout this thesis, we will use the term stationary to mean weakly stationary.

The simplest example of a stationary process is white noise (WN). White noise is a time series of uncorrelated random variables with constant $\mu_t = 0$ and constant variance $\sigma^2$. We denote this process as

$$w_t \sim WN(0, \sigma^2).$$

A particularly useful white noise series is Gaussian white noise, wherein the $w_t$ are independent normal random variables, with mean 0 and variance $\sigma^2$,

$$w_t \overset{\text{i.i.d.}}{\sim} WN(0, \sigma^2).$$

### 2.1.3 Time Series Forecasting

The objective of time series analysis is to identify the nature of the phenomenon represented by the sequence of observations, understand patterns and predict further development of the time series variable. The latest is often referred to as the process of forecasting. The fundamental idea of time series forecasting is to use past observations to predict the future. The model that describes best the data will later be used to predict the future based on records. Time series forecasting can be based on endogenous variables, which are the past values of the series, or exogenous variables, which are external factors that can be correlated to the value of the series. Within the scope of this thesis, we take both types into consideration: models using only endogenous variables (autoregressive integrated moving average models) and models that incorporate further exogenous variables (machine learning approaches).

Suppose we have an observed time series $y_1, y_2, ..., y_n$ and wish to forecast future values such as $y_{n+h}$. The integer $h$ is called the lead time or the forecasting horizon and the forecast of $y_{n+h}$ made at time $n$ for $h$ steps ahead will be denoted by $\hat{y}_n(h)$.

**One-step and Multi-step time series forecasting**

Time series forecasting is typically discussed, where only a one-step prediction is required. Predicting a sequence of values in a time series is called multi-step time series forecasting. Formally, a multi-step ahead time series forecasting task consists of predicting the next $h$ values $y_{n+1}, ..., y_{n+h}$ of a historical time series $y_1, ..., y_n$, with $h > 1$. Strategies for predicting time series multi-step ahead have been extensively discussed in Ben Taieb et al. (2012). For the scope of this thesis, we are interested in two general strategies: the recursive and the direct strategy.

The *recursive strategy* (also called iterated or multi-stage) is the most intuitive forecasting strategy. In this strategy, a single model $f$ is trained to perform a one-step-ahead forecast

$$\hat{y}_t(1) = f(y_1, ..., y_t)$$

When forecasting $h$ steps ahead, we first forecast the first step by applying the model. Subsequently, we use the forecasted value as part of the input variables for forecasting the next step. For prediction $t + h$ values we obtain

$$\begin{aligned}
\hat{y}_t(1) &= f(y_1, ..., y_{t1}) \\
&\vdots \\
\hat{y}_t(h-1) &= f(y_1, ..., y_t, \hat{y}_t(1), ..., \hat{y}_t(h-2)) \\
\hat{y}_t(h) &= f(y_1, ..., y_t, \hat{y}_t(1), ..., \hat{y}_t(h-1))
\end{aligned}$$

The iterative approach is used generally by autoregressive models, where each prediction is based on previous records. The weakness of this method is that it propagates the error committed in earlier forecasts to the future which might render the quality of long-term forecasts unreliable (Ing, 2003).

The *direct strategy* (also called independent) consists of forecasting each horizon independently from the others. In other terms, $h$ models are learned (one for each horizon) from the time series.

$$\hat{y}_t(h) = f(y_1, ..., y_t)$$

This implies that the direct strategy does not use any approximated values to compute the forecasts, being then immune to the accumulation of errors. This approach is applied to the machine learning forecasting in the conducted benchmark experiment. Since we use a supervised learning approach, where we shift the target to $h$ steps to create a supervised learning set (see Chapter 2.3.1), the above equation should be rewritten into

$$\hat{y}_t(h) = f(y_1, ..., y_{t-h})$$

## 2.2 Statistical Techniques for Time Series Modeling

A variety of time series forecasting models have been evolved in the literature. One of the most well-known statistical methods for time series forecasting is the ARIMA model, also called the Box-Jenkins model, originally proposed by Box and Jenkins (1970).

### 2.2.1 ARIMA

ARIMA stands for 'autoregressive integrated moving average'. As the acronym indicates, $ARIMA(p, d, q)$ captures the key elements of the model: Auto Regression (AR): A model, which incorporates past values in forecasting future values. Integrated Term (I): Method to make the time series stationary by measuring the differences between consecutive observations. Moving Average (MA): A model, which incorporates past forecast errors in forecasting future values.

#### Auto Regressive AR(p)

The autoregressive model simply follows the idea that any value of a time series can be modeled as a weighted average of past observations plus a white noise 'error', which is also called the 'noise' or 'disturbance'. An autoregressive model of order $p$, abbreviated $AR(p)$, is of the form

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t$$
$$y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \epsilon_t$$

where $y_t$ is stationary series, $\phi_1, ..., \phi_p$ are constants and $\epsilon_t$ an error term (or residual). As usual in time series modeling, a basic assumption is that the random error $\epsilon_t$ is Gaussian white noise. The parameter $p$ of $AR(p)$ determines the number of lags that will be used in predicting the value of $y_t$.

#### Moving Average MA(q)

A moving average process can be expressed as a weighted average of the past values of the white noise process $\{\epsilon_t\}$. The moving average model of order $q$, or $MA(q)$ model, is of the form

$$y_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q}$$

$$y_t = \sum_{j=1}^{q} \epsilon_j \theta_{t-j}$$

where $y_t$ is stationary series, $\theta_1, ..., \theta_q$ are constants and $\epsilon_t$ is assumed to be a Gaussian white noise.

**Differencing I(d)**

Stationary time series with complex autocorrelation behaviour often are more accurately modeled by mixed autoregressive and moving average (ARMA) processes than by either a pure AR or pure MA process. An $ARMA(p, q)$ model combines both $AR$ and $MA$ terms and is defined by the equation

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q} + \epsilon_t$$
$$y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \epsilon_j \theta_{t-j} + \epsilon_t$$

where $\phi \neq 0, \epsilon \neq 0$ and $\sigma^2 > 0$. The equations show how $y_t$ depends on lagged values of itself and lagged values of the white noise process.

A weakness of the ARMA model is its assumption that the underlying time series is stationary. Since many real-life cases time series do not meet the assumption of stationarity, ARMA processes can not be applied directly. One possible way of handling non-stationary series is to apply the differencing operation to stationarize the series. This operation is handled by the ARIMA model which generalizes the ARMA model for non-stationary time series. The I (integrated) indicates the process of making a time series stationary with differencing. The differencing operator is defined as $\triangle = 1 - B$, where $B$ is the backward operator, so that

$$\triangle y_t = y_t - By_t = y_t - y_{t-1}$$

Differencing may be repeated as required and $\triangle^d$ is called the $d$th-order differencing operator. A process $\{y_t\}$ is said to be an $ARIMA(p, d, q)$ process if

$$\triangle^d y_t = (1 - B)^d y_t$$

is $ARMA(p, q)$. In general, the model can be represented as follows

$$\phi(B)(1 - B)^d y_t = \theta(B)\epsilon_t$$

**Multi-horizon forecating**

Given the necessary input values, ARIMA models make point forecasts one time step into the future. To obtain multi-horizon forecasts an iterative (recursive) strategy is applied, where the model is trained to predict one step ahead and the predicted value is used as an input to predict the value for the next step. Assuming an $ARMA(p,q)$ model, the $h$-step-ahead prediction is obtained as follows:

Initially, a one-step-ahead prediction is conducted. For the one-step-ahead prediction $t$ is replaced by $t+1$.

$$y_t(1) = \phi_1 y_t + ... + \phi_p y_{t-p+1} + \theta_1 \epsilon_t + ... + \theta_q \epsilon_{t-q+1} + \epsilon_{t+1}.$$

Assuming we have observations up to time $t$, all values on the right side of the equation are known except $\epsilon_{t+1}$, which we replace by 0, and $\epsilon_t, ..., \epsilon_{t-q+1}$, which we replace with the last observed residuals $e_t, ..., e_{t-q+1}$

$$\hat{y}_t(1) = \phi_1 y_t + ... + \phi_p y_{t-p+1} + \theta_1 e_t + ... + \theta_q e_{t-q+1} + 0.$$

Subsequently, we take the estimate $\hat{y}_{t+1}$ for $y_{t+1}$ and plug it in to obtain the second-step-ahead prediction

$$\hat{y}_t(2) = \phi_1 \hat{y}_{t+1} + ... + \phi_p y_{t-p+2} + \theta_2 e_t + ... + \theta_q e_{t-q+2} + 0 + 0$$

This process is then repeated, until $\hat{y}_t(h)$ is reached (Hyndman and Athanasopoulos, 2018).

### 2.2.2 Uncertainty Quantification

Usually, a point estimate $\hat{y}$ returns a value that does not match the true future outcome $y$ of a time series. A prediction interval is a quantification of the uncertainty on a prediction. It provides probabilistic upper and lower bounds such that $y$ lies within that range with a desired probability $(1-\alpha)$.

A $h$-step-ahead $(1-\alpha)$-level prediction interval for the ARMA(p,q) process can be constructed as follows

$$[\hat{y}_{t+h} - q_{\alpha/2}\hat{\sigma}(h); \hat{y}_{t+h} + q_{1-\alpha/2}\hat{\sigma}(h)]$$

where $q_\alpha$ is the $\alpha$-level quantile of the error disturbance and

$$\hat{\sigma}^2(h) = \sigma^2 \sum_{j=0}^{h-1} \hat{\theta}_j^2$$

is the estimate of $\sigma^2(h)$, the variance of the errors $\{\epsilon_t\}$. $\hat{\theta}_j$, $j = 0, .., h-1$ are the estimated coefficients of the moving-average representation of the ARMA(p,q) process (Brockwell and Davis, 2002).

With the assumption of normally distributed error, the prediction interval is given by

$$\hat{y}_{t+h} \pm z_{\alpha/2} \sqrt{\sigma^2 \sum_{j=0}^{h-1} \hat{\theta}_j^2}$$

where $z_\alpha$ is the $\alpha$-level quantile of the standard normal distribution.

The introduced prediction interval for the ARIMA-based models only takes into account the variability from the errors, thus it tends to be too narrow. Alongside the random error term, there are several sources of uncertainty, such as the uncertainty in the model building and selection process, which is not adequately considered (Hyndman and Athanasopoulos, 2018). Furthermore, the prediction intervals for ARIMA models are asymptotically valid only under the assumption that the residuals are uncorrelated and normally distributed.

## 2.3 Machine Learning Techniques for Time Series Modeling

Machine learning techniques have gained lots of attention in recent years with their applications in many disciplines. The success of non-parametric methods lies in the ability to learn by trial-error method and by improving model accuracy over iterations. Furthermore, machine learning models exhibit various useful properties, which can effectively improve time series forecasting performance - ability to automatically identify the structure and pattern of the data, successful incorporation of non-linear patterns and interactions, and ability to deal with high-dimensional data (Boulesteix and Schmid, 2014).

The presented methodological background on decision trees, random forest and gradient boosting is general, i.e., the notation is not adapted to a time series forecasting task. Moreover, the theoretical overview of these models is closely based on a project report of a statistical consulting project conducted at the LMU Munich ('Supervised learning for day-ahead wind power forecasting', 2021).

### 2.3.1 Supervised Learning Setting

Supervised learning consists in modeling the relationship between a set of input variables and output variables, which are considered somewhat dependent on the inputs. In formal notation, the supervised dataset is given in the following form

$$\mathcal{D} = ((\mathbf{x}^{(1)}, y^{(1)}), ..., (\mathbf{x}^{(n)}, y^{(n)}) \in (\mathcal{X} \times \mathcal{Y})^n$$

with $\mathcal{X}$ the input space, $\mathcal{Y}$ the output space and the tuple $(\mathbf{x}^{(i)}, y^{(i)})$ the $i$-th observation.

Supervised learning can be seen as the task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples.

One-step forecasting can be tackled as a problem of supervised learning. Assuming a purely autoregressive model of order p, we want to forecast future observations by using $p$ previous observations of the time series. Effectively, we construct a set of observations which are based on the past $p$ lags of the time series. Each observation is composed of a feature vector $\mathbf{x}_i \in \mathbb{X} \subset \mathbb{R}^p$, which denotes the previous $p$ values and a target $y_i \in \mathbb{Y} \subset \mathbb{R}$, which represents the value we want to predict. The objective is to construct a model $f : \mathbb{X} \mapsto \mathbb{Y}$, where $f$ denotes the regression function. For a task, which is trained on $n$ data points and a fixed forecast horizon of $h$, the training set is derived by creating the $[p \times n]$ input data matrix and the $[1 \times n]$ output vector

$$Y = \begin{bmatrix} y_t \\ y_{t-1} \\ \vdots \\ y_{t-n} \end{bmatrix} \quad X = \begin{bmatrix} y_{t-h} & y_{t-1-h} & y_{t-2-h} & \cdots & y_{t-p-h} \\ y_{t-1-h} & y_{t-2-h} & y_{t-3-h} & \cdots & y_{t-p-h-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{t-n-h} & y_{t-n-h-1} & y_{t-n-h-2} & \cdots & y_{t-n-h-p} \end{bmatrix}$$

Assuming the presence of $p$ exogenous features in the model, the feature matrix and output vector for training on $n$ data points for predicting the $h$-th prediction horizon can be represented as follows

$$Y = \begin{bmatrix} y_t \\ y_{t-1} \\ \vdots \\ y_{t-n} \end{bmatrix} \quad X = \begin{bmatrix} x_{1_{t-h}} & x_{2_{t-h}} & x_{3_{t-h}} & \cdots & x_{p_{t-h}} \\ x_{1_{t-1-h}} & x_{2_{t-1-h}} & x_{3_{t-1-h}} & \cdots & x_{p_{t-1-h}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{1_{t-n-h}} & x_{2_{t-n-h}} & x_{3_{t-n-h}} & \cdots & x_{p_{t-n-h}} \end{bmatrix}$$

The conducted supervised learning modeling in this thesis is based on a dataset, which consists of both - endogenous and exogenous features. Therefore, the constructed feature matrix in this thesis can be represented as a combination of both presented feature matrices above.

### 2.3.2 Regression Trees

Decision trees are a non-parametric method for predicting the response $y$ from the predictor variables $x_1, ..., x_p$ and build accurate, flexible, and easily interpretable models. One of the most popular tree-building algorithms is the methodology of the classifications and regression trees (CART), proposed by Breiman et al. (1984). In the case of a continuous response, one speaks of a regression tree. On the contrary, a classification tree aims to model data with a categorical response. The basic idea of decision trees is to recursively partition the covariate space to form subsets that are more homogeneous concerning the response variable. To choose the binary partition in a way that the arising child nodes are "purer" (i.e., more homogeneous) than the parent nodes, a measure of node impurity is minimized.

The algorithm starts at the root node, which contains all observations $(\mathbf{x}^{(i)}, y^{(i)})$ with $i = 1, ..., n$ and performs a search through all potential binary splits (through all variables $x$ and variable values $s$), then selects the best one according to a splitting criterion. A split with respect to variable $x_j$ and split point $s$ divides the root node $N$ into two disjoint subsets (left and right daughter nodes):

$$N_L = \{(\mathbf{x}, y) \in N : x_j \leq s\} \quad \text{and} \quad N_R = \{(\mathbf{x}, y) \in N : x_j > s\}.$$

The most common splitting criteria are based on impurity reduction in the daughter nodes. This corresponds to minimizing response differences in the daughter nodes. A natural measure of node impurity in regression trees is mean squared error. Formally the goal is to minimize the expected value of the loss function, the so-called empirical risk $\mathcal{R}(N)$.

We estimate the mean squared error by

$$\mathcal{R}(N) = \frac{1}{|N|} \sum_{(\mathbf{x},y) \in N} (y - c)^2$$

with the mean response as a constant prediction

$$c = \overline{y}_N = \frac{1}{|N|} \sum_{(\mathbf{x},y) \in N} y.$$

In order to evaluate how good a split is, we compute the empirical risks in both child nodes and sum them up

$$\mathcal{R}(N, j, s) = \frac{|N_L|}{|N|} \mathcal{R}(N_L) + \frac{|N_R|}{|N|} \mathcal{R}(N_R)$$

Finding the best way to split $N$ into $N_L$, $N_R$ formally means solving

$$\arg\min_{j,t} \; \mathcal{R}(N, j, s).$$

We want to find variable $x_j^*$ and split point $t^*$, such that $\mathcal{R}(N, j^*, s^*) \leq \mathcal{R}(N, j, s)$ for all $x_j$ and $c$. The resulting subsets are referred to as daughter nodes. These are further split into daughter nodes throughout the entire tree construction. The process is repeated recursively until a stopping criterion is reached. This method eventually creates disjoint subsets (terminal nodes) in the predictor feature space $\mathcal{X}$. Suppose we partition the data into G terminal nodes $N_g$ with $g = 1, ..., G$. We model the response $y$ with a constant $c_g$ in region $N_g$:

$$f(x) = \sum_{g=1}^{T} c_g I(x \in N_g).$$

The estimate of $c_g$ is usually given by the average of $y^{(i)}$ in region $N_g$. Hence, predictions for new observations can be uniquely determined by tracking the observations down the tree until it arrives at a terminal node. The prediction for a new observation $i$ falling into the terminal node $N_g$ is given by

$$\hat{y}^{(i)} = \frac{1}{|N_g|} \sum_{i \in N_g} y^{(i)}.$$

The CART algorithm usually produces large and complex trees, which often lead to over-fitting and large prediction errors. A natural solution is to stop growing the tree when a pre-defined stopping criterion is met - e.g., a given threshold for the impurity measure that is not exceeded by any split, total purity for terminal nodes, or a minimum number of observations in a node for splitting it further. Alternatively, a tree can be grown very large and subsequently "pruned" back to circumvent the issue of over-fitting. Breiman et al. (1984) proposed the cost-complexity criterion, which takes both the cost (prediction error) and complexity (size) of the tree into consideration and gives an overall score for further selection.

Alternatively, an ensemble tree method can be used to avoid the problem of selecting a single tree of the appropriate size. Ensemble methods aim at improving the predictive performance of a given statistical learning or model fitting technique (Bühlmann, 2004). The general principle of ensemble methods is to construct a linear combination of some model fitting method, instead of using a single fit of the method. Ensemble tree methods can essentially improve the accuracy of the prediction since single tree models can suffer in terms of stability to small changes in the learning data. There are two main ensemble techniques. The bagging procedure improves accuracy by reducing the variance of the prediction. On the other hand, boosting methods are primarily reducing the model bias of the base procedure.

The following chapter introduces the main properties of both ensemble tree techniques: bagging (random forests) and boosting.

### 2.3.3 Random Forest

Bagging, a portmanteau for "bootstrap aggregating", is an ensemble method for improving the unstable estimation of rather weak prediction methods. Breiman (1996) motivates bagging as a variance reduction technique for a given base procedure, such as decision trees. In bagging, several trees are fit to bootstrapped or sub-sampled data. Predictions from every tree are averaged to produce the final prediction of the ensemble.

The random forest (RF) method proposed by Breiman (2001) is an extension of bagging, which enforces further diversity between trees. In bagged trees, bootstrap samples are drawn randomly from the learning sample and an individual tree is grown on each sample. This already implies that bagged trees differ in their structure. Random forest trees are grown fully and differ from CART as they are grown non-deterministically using a two-stage randomization procedure. In addition to the randomization, which is introduced by growing the tree using a bootstrap sample of the original data, the second level of randomization is added - at each node of each tree, only a random subset of variables is selected as candidates for splitting. The purpose of both randomization steps is to create decorrelated trees, which, when aggregated, reduce the variance of the ensemble and improve accuracy.

The random forest algorithm is described in Algorithm 1:

**Algorithm 1:** Random Forest Algorithm

```
1. Draw $B$ bootstrap samples from the original dataset. Each bootstrap sample
   excludes on average about 37% of the data, called out-of-bag (OOB data).

2. Grow a random-forest tree $T_b$ for each bootstrap sample $b = 1, ..., B$ . At each node
   randomly select $mtry \leq p$ covariates as candidates for splitting (default:
   $mtry = \sqrt{p}$). Find the best variable/split-point. Grow the tree to full size.

3. Output the ensemble of trees $\{T_b\}_1^B$. To make a prediction at a new point, aggregate
   information from B trees, i.e., majority for classification, average for regression.

4. Using OOB data, calculate error rate (not applicable to this thesis).
```

**Ensemble Estimation**
The key estimate, produced by random forest is the average prediction over all $B$ trees. We denote the produced tree ensemble $\{T_b\}_1^B$ with $b = 1, ..., B$. For a new observation $i$ with predictor $\mathbf{x}^{(i)}$ the response estimation is given by

$$\hat{f}^B(\mathbf{x}^{(i)}) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(\mathbf{x}^{(i)})$$

where $\hat{f}^{*b}(\mathbf{x}^{(i)})$ denotes the prediction for observation $i$ for tree $b = 1, ..., B$.

### 2.3.4 Gradient Tree Boosting

In machine learning theory, boosting is considered to be one of the most powerful learning ideas. The main idea of boosting is to take a weak learner, e.g., a decision tree, and sequentially apply it to modified versions of the training data. In each step, the model tries to compensate for the weaknesses of its predecessor. The gradient boosting technique, which was introduced by Friedman (2001) is probably the most widely used boosting technique. Gradient boosting combines two different techniques, *boosting* and the *gradient descent* method. The theoretical overview in this section is closely based on Hastie et al. (2009).

**Gradient descent**
Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The gradient descent algorithm moves towards the local minimum in every iteration. In the literature, gradient descent has been often proposed as a method for minimizing empirical risk $\mathcal{R}_{emp}$. Let $f(x)$ be an arbitrary, differentiable objective function that is to be minimized. The gradient $\nabla f(x)$ can be understood as the direction of the steepest ascent of the function. Correspondingly, $-\nabla f(x)$ points towards the steepest descent of $f(x)$. For a function $f : \mathbb{R}^n \to \mathbb{R}$ we denote

$$\nabla f(x) = \left( \frac{\partial f(x)}{x_1}, ..., \frac{\partial f(x)}{x_n} \right)^T .$$

For a current iteration step, we minimize the function $f(x)$ by updating the current point $x^{[m]}$ by

$$x^{[m+1]} = x^{[m]} - \nu \nabla f(x^{[m]}))$$

for $m = 1, ..., M$. The update results in a lower value of the objective function $f(x^{[m]}) > f(x^{[m+1]})$. The step length $\nu$ controls the step size towards the steepest descent. For a very small $\nu$ the learning process will converge very slowly. On the other side, a bigger $\nu$ might miss the minimum.

**Forward Stagewise Additive Modeling**
Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions. The approach builds strong learners by slightly improving weaker learners and over a series of iterations. The boosting framework is based on the *forward stagewise additive model*, which estimates the prediction function as an additive model in a forward stagewise way. An additive basis function expansion can be expressed as

$$f(x) = \sum_{m=1}^{M} f^{[m]}(x) = \sum_{m=1}^{M} \beta^{[m]} b(x, \theta^{[m]})$$

with weights $\beta^{[m]}$, $m = 1, ..., M$ and base learner parameter $\theta^{[m]}$. The goal is to minimize the empirical risk, given by

$$\mathcal{R}_{emp}(f) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) = \sum_{i=1}^{n} L\left(y^{(i)}, \sum_{m=1}^{M} \beta^{[m]} b\left(\mathbf{x}^{(i)}, \theta^{[m]}\right)\right)$$

with the training data $(y^{(i)}, \mathbf{x}^{(i)}), i = 1, ..., n$ and some loss function $L$.

Minimizing $\mathcal{R}_{emp}(f)$ with respect to $((\beta^{(1)}, \theta^{(1)}), ..., (\beta^{[m]}, \theta^{[m]}))$ requires computationally intensive numerical optimization techniques, since the parameter space is a high-dimensional. Alternatively, the forward stagewise additive modeling approach can be used to sequentially minimize the empirical risk only with respect to the next component

$$\min_{\beta,\theta} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{f}^{[m-1]}\left(\mathbf{x}^{(i)}\right) + \beta b\left(\mathbf{x}^{(i)}, \theta\right)\right)$$

Algorithm 2 describes the process of the forward stagewise additive modeling.

**Algorithm 2:** Forward Stagewise Additive Modeling

```
Initialize $\hat{f}^{(0)} = 0$
for $m = 1 \to M$ do
    $(\hat{\beta}^m, \hat{\theta}^m) = \arg\min_{\beta,\theta} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{f}^{[m-1]}\left(\mathbf{x}^{(i)}\right) + \beta b\left(\mathbf{x}^{(i)}, \theta\right)\right)$
    Update $\hat{f}^{[m]} \leftarrow \hat{f}^{[m-1]}(x) + \hat{\beta}^{[m]} b\left(\mathbf{x}^{(i)}, \hat{\theta}^{[m]}\right)$
end
```

**Gradient Boosting**

Gradient boosting incorporates the methods of gradient descent and forward stagewise additive modeling. Concretely, gradient boosting uses stagewise additive models for which the empirical risk $\mathcal{R}_{emp}$ is minimized via gradient descent. The gradient of the empirical risk $\mathcal{R}_{emp}$ with respect to each component of the parameter vector for a differentiable loss function $L$ is given by

$$\frac{\partial \mathcal{R}_{emp}}{\partial f(\mathbf{x}^{(i)})} = \frac{\partial L(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f(\mathbf{x}^{(i)})}$$

The gradient descent update for each vector component of $f$ is given by

$$f(\mathbf{x}^{(i)}) \leftarrow f(\mathbf{x}^{(i)}) - \beta \frac{\partial L(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f(\mathbf{x}^{(i)})}$$

Consequently, we can determine the direction of the steepest descent for each observation. In combination with the iterative additive procedure of forward stagewise modeling, we are at the spot $f[m-1]$ during minimization. At this point we calculate the direction of the negative gradient and define the pseudo residuals $r^{[m](i)}$

$$r^{[m](i)} = -\left[\frac{\partial L(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f(\mathbf{x}^{(i)})}\right]_{f=f^{[m-1]}}$$

The pseudo-residuals $r^{[m](i)}$ match the usual residuals for squared loss.

For regression problems, the parameter $\theta^{[m]}$ in base learner $b(\mathbf{x}^{(i)}, \theta^{[m]})$ can be estimated by minimizing the sum of squared error

$$\theta^{[m]} = \arg\min_{\theta} \sum_{i=1}^{n} (r^{[m](i)} - b(\mathbf{x}^{(i)}, \theta))^2$$

In general, one boosting iteration is exactly one approximated gradient step in function space, which minimizes the empirical risk as much as possible. The gradient boosting algorithm is described in Algorithm 3.

**Algorithm 3:** Gradient Boosting Algorithm

```
Initialize $\hat{f}^{(0)} = \arg\min_{\theta} \sum_{i=1}^{n} L(y^{(i)}, b(\mathbf{x}^{(i)}, \theta))$
for $m = 1 \to M$ do
    For all i: $r^{[m](i)} = -\left[\frac{\partial L(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f(\mathbf{x}^{(i)})}\right]_{f=\hat{f}^{[m-1]}}$
    Fit a regression base learner to the pseudo-residuals $r^{[m](i)}$:
    $\theta^{[m]} = \arg\min_{\theta} \sum_{i=1}^{n} (r^{[m](i)} - b(\mathbf{x}^{(i)}, \theta))^2$
    Line search: $\hat{\beta}^{m} = \arg\min_{\beta} \sum_{i=1}^{n} L\left(y^{(i)}, f^{[m-1]}(\mathbf{x}^{(i)}) + \beta b(\mathbf{x}^{(i)}, \theta^{[m]})\right)$
    Update $\hat{f}^{[m]} \leftarrow \hat{f}^{[m-1]}(x) + \hat{\beta}^{[m]} b(\mathbf{x}^{(i)}, \hat{\theta}^{[m]})$
end
Output $\hat{f}(x) = \hat{f}^{[m]}(x)$
```

**Gradient Boosting and Trees**
The most commonly used base learner for gradient boosting are decision trees. A tree can be formally expressed as

$$b(x) = \sum_{t=1}^{T} c_t I(x \in N_t)$$

with $N_t$ the disjoint terminal regions and $c_t$ the corresponding means. Accordingly, this specific additive structure can be exploited by boosting

$$f^{[m]} = f^{[m-1]}(x) + \beta^{[m]} b^{[m]}(x) = f^{[m-1]}(x) + \beta^{[m]} \sum_{t=1}^{T[m]} c_t^{[m]} I(x \in N_t^{[m]}).$$

The above expression can be rewritten as

$$f^{[m]} = f^{[m-1]}(x) + \sum_{t=1}^{T[m]} \gamma^{[m]} I(x \in N_t^{[m]})$$

with $\gamma^{[m]} = \beta^{[m]} c_t^{[m]}$ Again, minimizing the loss function provides the optimal coefficients for $\gamma^{[m]}$

$$\gamma^{[m]} = \arg\min_{\theta} \sum_{i=1}^{n} L\left(f^{[m-1]}(x) + \sum_{t=1}^{T[m]} \gamma^{[m]} I(x \in N_t^{[m]})\right)$$

**Regularization and Shrinkage**
Due to its aggressive loss minimization, gradient boosting can easily overfit the model. An overfitted model tends to fit the noise in the training data and fails to generalize to new data. Regularization methods attempt to prevent overfitting by constraining the fitting procedure.

One natural regularization parameter is the number of components $M$. Limiting the number of additive components by stopping the boosting iterations early ("early stopping") regulates the degree to which expected loss on the training data can be minimized (Friedman, 2001). Further, the model complexity can be regulated by limiting the depth of the trees. Another way is to shorten the step length $\beta^{[m]}$ in each iteration by multiplying a shrinkage parameter (learning rate) $\nu \in (0, 1]$:

$$f^{[m]}(x) = f^{[m-1]}(x) + \nu^{[m]} b(x, \theta^{[m]})$$

All regularization parameter control the trade-off between complexity and accuracy of the model and do not operate independently. It has been empirically shown that smaller values of $\nu$ favor better test error, and require correspondingly larger values of $M$ (Friedman, 2001). Generally it is advisable to tune all regularization parameters simultaneously.

Some gradient boosting implementations, e.g., `XGBoost` introduce further advanced regularization ($L1$ and $L2$), which improves model generalization. Furthermore, feature subsampling similar to the random forest is introduced in `XGBoost`. Only a random subset of all features is considered for each split. A more detailed look into the theoretical background of `XGBoost` can be found in the paper of Chen and Guestrin (2016).

### 2.3.5 Uncertainty Quantification

Machine learning techniques are often highly effective in keeping the reducible error at a minimum as they require little to no assumptions to be made on the function for making predictions. However, this often makes statistical inference difficult. Due to their black-box nature, models like random forest and gradient boosting are difficult to interpret and the inherent modeling and input uncertainties are difficult to quantify. However, we are often not only interested in accurate prediction, but in valuable insights into the complex process, e.g., transparency with regards to the quantification of prediction uncertainties. Instead of minimizing the prediction error, the objective in uncertainty quantification is to quantify how large this expression of errors could be for a newly predicted unobserved point. In this context, multiple approaches to uncertainty quantification for machine learning models have been proposed in the literature. One existing method for uncertainty quantification in random forest involves estimating the conditional distribution of the response variable given the predictor vector via quantile regression forests (Meinshausen, 2006). In this method, an empirical cumulative distribution function (CCDF) is calculated by keeping the complete distribution of all observed response values of every node of each tree in the forest. Prediction intervals are therefore derived from the empirical CCDF. Quantile regression can be used to estimate prediction intervals for the gradient boosting algorithm as well. As shown in Kriegler and Berk (2007) the gradient tree boosting algorithm predicts the $Q_\alpha$ quantile, when implemented with an appropriate cost function. Another possible approach with random forest models is to construct prediction intervals using the empirical distribution of out-of-bag prediction errors (Zhang et al., 2020). These prediction intervals can be obtained as a by-product of a single random forest. Mentch and Hooker (2016) have proposed an approach for constructing confidence intervals from a procedure similar to random forests. By training a multitude of trees on strict subsample combinations of the training set and averaging their results, random forest can be seen as a U-statistic which is proven to be asymptotically normal. Furthermore, Mentch and Hooker (2016) developed a consistent estimator for the variance of the relevant limiting normal distribution that naturally leads to a confidence interval for the mean of their estimator.

Another way to think about the length of the prediction interval is in terms of the magnitude of the prediction error. Analogously to the prediction intervals introduced in the ARIMA modeling, regression models can produce prediction intervals analytically. Assuming normally distributed errors, a $(1-\alpha)$-level prediction interval can be acquired by

$$\hat{y}_{t+h} \pm z_{\alpha/2}\hat{\sigma}_t$$

where $z_\alpha$ is the $\alpha$-level quantile of the standard normal distribution and $\hat{\sigma}$ is an estimate of the standard deviation. A 95% prediction interval is then asymptotically given by

$$\hat{y}_{t+h} \pm 1.96\hat{\sigma}_t$$

### 2.3.6 Variable Importance

In the context of machine learning, we are often not only interested in accurate prediction, but in valuable insights into the complex process. Furthermore, gaining a well-grounded understanding of a model is essential for further improvement and tackling shortcomings. Besides quantifying uncertainty (see Chapter 2.3.5), we are often interested in quantifying the impact of different features and gaining information on the association between features and the target. Unfortunately, random forests and gradient boosting are not intrinsically interpretable since their prediction results from averaging or updating several hundreds of decision trees. However, there are established methods that focus on model interpretation and a better understanding of the relationship between features and the outcome variables. This methods are often referred as variable importance or feature importance measures. In this section we introduce the two most widely used variable importance measures for tree-based methods: impurity-based importance and permutation-based importance, as well as assess their advantages and disadvantages.

The *impurity-based importance* measure (also called *information gain-based importance*, or *Gini importance*) quantifies variable importance by measuring the contribution to impurity reduction of each variable. As decribed above, CART splitting rules aim to minimize variance or impurity, in child nodes. As a consequence variables used for splitting at important splits are also considered important. Based on this idea, impurity-based measures calculate the importance of a variable $x_j$ by taking the sum of the impurity decrease of all nodes at which a split on $x_j$ has been conducted. In random forests, this measure is normalized by the number of trees.

Impurity-based measures are relatively intuitive and easy to calculate. A disadvantage of impurity-based measures is that they tend to be biased in favor of variables with high cardinality, since variables with higher cardinality have more split points, which affords them a higher probability of being selected to split variables (Wright et al., 2016a). Moreover, an important issue, which is tightly related the structure of time series data, is the behavior of variable importance measures in the presence of correlated features. Archer and Kimes (2008) observe that the Gini measure is less able to detect the most relevant variables when the correlation increases. Hence, interpretation of variable importance outputs must be cautious, to not overstate the meaningfulness of results.

The *permutation importance* measure, proposed by Breiman (2001) is one of the most common and most advanced variable importance measures. It measures how prediction error for the ensemble is influenced when a variable is randomly "shuffled", meaning that its values are moved to different positions to create noise. By randomly permuting the predictor variable $x_j$, its original association with the response $y$ is affected. If the variable consists of purely random noise, prediction accuracy likely won't be markedly changed. On the contrary, prediction performance is expected to drop for relevant variables. In random forest, the out-of-bag observations have proven useful for computing the permutation variable importance.

Permutation importance has the advantage of dealing well with interactions between variables (Wright et al., 2016b; Scornet, 2020). By shuffling the variable, the association is untethered not only from the response but also the other variables with which it interacts (Scornet, 2020). Its interpretation is also relatively clear: the contribution of a variable to the model is measured by how the model behaves when the association of the response is removed. A main disadvantage of the permutation variable importance is its behavior in the presence of correlated features. Adding a correlated feature to the model can decrease the importance of the associated feature by splitting the importance between both features (Nembrini et al., 2018).

Both methods display different advantages and disadvantages, which should be considered when interpreting the metrics. Both concepts of feature importance can be regarded as complements that enable interpretability from different angles. Examining different importance metrics while bearing their shortcomings and the structure of the data in mind, can help avoid misinterpretations.

# 3. Data Foundation

This section focuses on the necessary data for the conducted benchmark experiment. Within the framework of this thesis, we do not limit modeling only to purely autoregressive approaches. Therefore, alongside data on COVID-19 ICU occupancy, additional key metrics for the progression of the pandemic are considered. After a description of the raw data sources and structures, the conducted data pre-processing and manipulation such as feature engineering, missing data imputation and variable recording are introduced. Additionally, the final choice of features and the creation of a supervised learning dataset is discussed. Finally, we take a look into the structure of the data and the developments of key COVID-19 pandemic measures. We motivate the importance of considering different key metrics and their change over the course of the pandemic in response to new conditions.

## 3.1 Raw Datasets

### COVID-19 bed occupancy in German intensive care units

Data on free and occupied capacities in German ICUs is publicly available and provided by the German Interdisciplinary Association for Intensive Care and Emergency Medicine (DIVI) intensive care register[1]. Moreover, information on beds occupied by COVID-19 patients is included. Data is recorded at around 1,300 intensive care hospitals in Germany on a daily basis and aggregated and provided on district level. The available information for 397 German districts is provided daily until 13:00 CET and exclusively reflects the data according to the status of the considered day. For the scope of this thesis, we are interested in data, which reflects the respective day and does not include any subsequent reporting corrections. This assures data representativeness for future forecasts.

The available data should be treated cautiously. The number of occupied beds is based on the postal code of the intensive care unit. The postal code of the patients is not included in the data. Therefore, a patient may reside in a district other than the district of the intensive care unit he is hospitalized in.

[1] `https://www.intensivregister.de`

### COVID-19 cases data of new infections and deaths

The publicly available dataset, provided by the Robert Koch-Institut, represents the daily updated case numbers of positive COVID-19 infections and deaths reported by public health authorities in Germany according to the requirements of the German Law on Prevention and Control of Infectious Diseases (Infektionsschutzgesetzes). This data represents a daily updated status (00:00 CET) of all previously reported cases of infection in Germany. Similar to the ICU data, we are interested in the values reported on the given day and are not updated by cases transmitted on subsequent days. Therefore, we consider not only the current cases dataset, but all previous datasets, updated on each respective day of interest. The RKI Github-Repository [2], contains an archive with a collection of all previous datasets, since 01.04.2021. For the time frame until 01.04.2021, the datasets are provided by CODAG (COVID-19 Data Analysis Group) of Ludwig-Maximilians-University Munich. The raw data is provided on district level. This data forms the basis for the calculation of COVID-19 new infections and deaths incidence rates. The exact procedure is explained in Chapter 3.2.

### COVID-19 vaccination data

The COVID-19 vaccination data represents a daily update (8:30 CET) of all vaccinations reported to the RKI on district level in Germany. Similar to the COVID-19 cases data, RKI provides a Github-Repository [3], containing an archive with a collection of all previous vaccination datasets, since 21.07.2021. The raw data is provided on federal, as well as on district level. Based on each dataset, a vaccination rate up to the respective day is calculated, resulting in a new vaccination rate time series for each district.

The proportion of vaccinated population residing in a district should be treated cautiously. The allocation of vaccination figures to federal states and countries is based on the postal code of the vaccinating sites. Only this indication of the place of vaccination is included in all data sources. The postal code of the vaccinated person is not included in the data. Therefore, a vaccinated person may reside in a district other than the district of the vaccinating site.

### German districts demographic data

For the calculation of incidence and vaccination rates, population data on district level in total, as well as population data, broken down by age groups, is needed. This data is available in the main database of the Federal Statistical Office [4]. Since age groups in both datasets do not entirely match (affected groups: '0 to 4 years' and '80+' years), some assumptions are met. We assume that the population distribution of the affected groups on district level is the same as the population distribution on national level. More granular population data on national level is available in the same database.

[2]`https://github.com/robert-koch-institut/SARS-CoV-2_Infektionen_in_Deutschland`
[3]`https://github.com/robert-koch-institut/COVID-19-Impfungen_in_Deutschland`
[4]`https://www-genesis.destatis.de/genesis/online`

## 3.2 Data Preprocessing

### Feature engineering

While ARIMA is a purely autoregressive model, which explicitly relies on past observations of the series, machine learning models have shown to be successful in incorporating various exogenous variables into the modeling procedure. Alongside past observations of ICU occupancy by COVID-19 patients, we want to add additional COVID-19 related data to the machine learning models to better reflect the dynamic development of the pandemic situation. This includes data on new infections, deaths and vaccination rates. In this section we explain how key metrics are derived from the raw data and discuss the final choice of features for the analysis.

#### *Lag features*

To address the specifics of time series data, we primarily model the number of COVID-19 patients in intensive care units based on endogenous variables. These are previous periods of the target, also called lag features of the target. Lags are very useful due to the tendency for values within a time series to be correlated with previous observations of itself. Lags can contribute to the identification of patterns and trends within the time series. A critical step is the right determination of the number of lags. Within the framework of this thesis, the number of lags is handled as a hyperparameter. The choice of the optimal length is based on the lowest average out-of-sample loss over a time series cross-validation sample. More on this procedure can be found in Chapter 4.3. We denote the derived $l$ lagged variables as *cases_ covid_ intensive_ t-1,...,cases_ covid_ intensive_ t-l.*

#### *Other ICU-related features*

Alongside information on previous observations of number of COVID-19 patients in intensive care units, we add exogenous variables into the machine learning models, to potentially enhance prediction performance. The number of COVID-19 patients in intensive care units, receiving invasive ventilation treatment, adds additional information on the severity of the infection. Moreover, we include the number of free beds in ICUs. An increase in bed occupancy can only be expected, if respective capacities are available.

#### *Incidence rates*

The 7-day reporting incidence is another important measurement for the spread of the pandemic. This metric measures the number of new cases reported within the last 7 days per 100.000 inhabitants and indicates the speed at which infections spread. We calculate 7-day incidence values based on the RKI data for new infections and deaths. The calculation is based on the reporting date, i.e., the date on which the local health department became aware of the case and recorded it electronically. For today's 7-day incidence, cases with reporting dates in the last 7 days are counted. The current day is not included in the calculation because additional reports may occur on that day and thus a full 7 days would not be included. As already mentioned, we are interested in the values reported until a given day and are not updated by cases transmitted on subsequent days. Therefore, the

7-day incidence rate is calculated based on each dataset in the archive for the respective day. Consequently, the 7-day incidence values derived from the single datasets are merged into a new 7-day incidence time series for each district. To gain more detailed information into the spread of the pandemic, incidence values by age groups are derived. This results in the following new variables: *7_day_incidence_cases*, as well as *7_day_incidence_cases* for the age groups *A05-A14*, *A15-A34*, *A35-A59*, *A60-A79*, *A80+*. In order to consider the time between the occurrence of symptoms, worsening of symptoms, and admission in the ICU, we add lagged observations of the incidence as well. This results in the following new variables: *7_day_incidence_cases_t-1,...,7_day_incidence_cases_t-m*. The number of lags is handled as a hyperparameter (see Chapter 4.3).

Analogously, the *7-day incidence of deaths* is calculated, as this is an additional important metric for the progression of the pandemic. The 7-day death incidence measures the number of deaths reported within the last 7 days per 100.000 inhabitants. Once again, we are interested in the values reported until a given day and are not updated by cases transmitted on subsequent days. However, in comparison to new cases, which are recorded by their reporting date, new deaths appear only in the respective dataset from the respective day, on which the death has been reported. The death reporting day in this case is the timestamp of the respective dataset. In order to obtain today's 7-day death incidence, we aggregate all deaths reported in the past 7 datasets from the archive, regardless the cases' reporting date. This procedure is done for each day of interest, resulting in a new 7-day death incidence time series for each district. Additionally, values by age groups are calculated here as well. This results in the following new variables: *7_day_incidence_deaths*, as well as *7_day_incidence_deaths* for the age groups *A05-A14*, *A15-A34*, *A35-A59*, *A60-A79*, *A80+*.

***Vaccination coverage***

Another important factor of the pandemic is the vaccination coverage. The vaccine coverage shows the percentage of people who are vaccinated against COVID-19 until a certain point in time. To calculate vaccination coverage, the number of fully vaccinated persons up to a point is aggregated and the proportion to the total population in the respective district is calculated. This results in a new vaccination coverage time series for each district. Vaccination coverage rates are calculated for fully vaccinated citizens, as well as for persons with a booster vaccination. Values by age groups are derived as well. This results in the following new variables: *vaccination_rate_full*, *vaccination_rate_booster* as well as *vaccination_rate_full*, *vaccination_rate_booster* for the age groups *A12-A17*, *A18-A59*, *A60+*.

As already mentioned, there is often a discrepancy between the district of the intensive care unit or the vaccination site and the district of residence of a person. While ICU and vaccination data is based on the postal code of the hospital or vaccination site, RKI cases data is based on the persons' residential postal code. This decreases the quality of the data and can possibly lead to lower prediction performance of the models. One easy solution, which partially addresses this issue, is to assume, that people from the country could go to a hospital in the city. To model this effect, we add an additional variable, which signifies whether the district is a country district (Landkreis) or a city district (Stadtkreis). This results in a new variable *is_ city*.

***Final dataset***

Finally, all relevant features are merged into one final dataset. The resulting dataset has district-level granularity and daily frequency. Table 3.1 summarizes all features included in the final merged dataset, which is used for the benchmark experiment.

| **data source** | **features** | |
|---|---|---|
| Infections and deaths data | 7_day_incidence_cases | 7_day_incidence_deaths |
| | 7_day_incidence_cases_A05-A14 | 7_day_incidence_deaths_A05-A14 |
| | 7_day_incidence_cases_A15-A34 | 7_day_incidence_deaths_A15-A34 |
| | 7_day_incidence_cases_A35-A59 | 7_day_incidence_deaths_A35-A59 |
| | 7_day_incidence_cases_A60-A79 | 7_day_incidence_deaths_A60-A79 |
| | 7_day_incidence_cases_A80+ | 7_day_incidence_deaths_A80+ |
| Vaccination data | vaccination_rate_full | vaccination_rate_booster |
| | vaccination_rate_full_A12-A17 | vaccination_rate_booster_A12-A17 |
| | vaccination_rate_full_A18-A59 | vaccination_rate_booster_A18-A59 |
| | vaccination_rate_full_A60+ | vaccination_rate_booster_A60+ |
| Lagged features | cases_covid_intensive_t-1 | 7_day_incidence_cases_t-1 |
| | cases_covid_intensive_t-2 | 7_day_incidence_cases_t-2 |
| | . | . |
| | . | . |
| | cases_covid_intensive_t-l | 7_day_incidence_cases_t-m |
| ICU data features | cases_covid_intensive_ventilated | |
| | beds_free | |
| Additional features | districtid | |
| | is_city | |

Table 3.1: Overview of features of the final dataset.

### Data pre-processing

After creating the final dataset, further basic data pre-processing, such as missing data imputation, and variable recording is required.

While some of the applied machine learning implementations (e.g., `XGBoost`) support internal missing values imputation, other do not (e.g., `skranger`). To guarantee consistency between models, we employ a unified missing data imputation method. The final dataset exhibits little missing data with small gaps of up to a couple of periods. Such missing data periods are imputed via a forward fill strategy, also known as the last value carried forward strategy. This method replaces every missing value with the value of the last observed period. Imputation is not directly applied to the data but integrated into `scikit-learn pipelines`. Pipelines create machine learning flows, by combining singular data and model manipulation steps into the learner. Hence, we incorporate missing data imputation operations into a learner, which is later directly applied for resampling, benchmarking, and tuning.

None of applied machine learning implementations in the benchmark experiment requires data scaling and normalizing. However, some do not internally handle categorical variables. Therefore, the single categorical variable in our dataset *districtid* is converted to numerical, via one-hot-encoding. This data pre-processing step is not directly applied to the data but included in the pipeline as well.

An important assumption of the ARIMA model is that the underlying time series is stationary. One possible way of handling non-stationary series is to apply the differencing operation to stationarize them. The differencing operation is integrated into the ARIMA algorithm and therefore directly applied to the data by the model. The order of the differencing operator is handled as a hyperparameter and optimized over a time series cross-validation sample. More on this procedure can be found in Chapter 4.3.

### Creation of supervised learning dataset

We are interested in forecasting ICU bed occupancy of COVID-19 patients $h$-step into the future. For modelling via ARIMA, only the time series of the target variable is required and no further manipulation is needed. However, supervised learning approaches need labeled data to learn from. One-step forecasting can be tackled as a problem of supervised learning. For that, the target variable is shifted by $h$-steps. Multi-step forecasting is conducted by training $h$ individual models, each trained to predict $1, .., h$ steps into the future. An overview of this procedure can be found in Chapter 2.3.1.

## 3.3 Data Exploration

### Interactive visualization of key COVID-19 metrics

To enable a comprehensive overview of different key metrics for the progression of the COVID-19 pandemic, we have created an interactive visualization tool, which gives insights into the dynamics of the pandemic for each German district. The following examples illustrate the provided charts and are based on district 'SK Hamburg'.

#### ***COVID-19 bed occupancy in German ICUs***

Figures 3.1 and 3.2 illustrate the development of reported COVID-19 cases treated in intensive care units over the course of the pandemic, as well as the proportion of COVID-19 cases to all occupied and free beds in intensive care units over time.

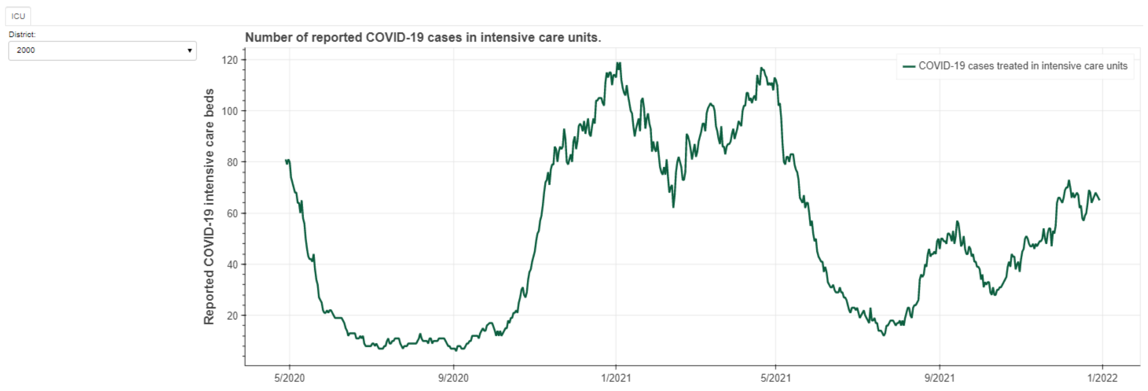


Figure 3.1: Number of reported COVID-19 cases treated in intensive care units at the respective observation day. Example based on district 'SK Hamburg'.

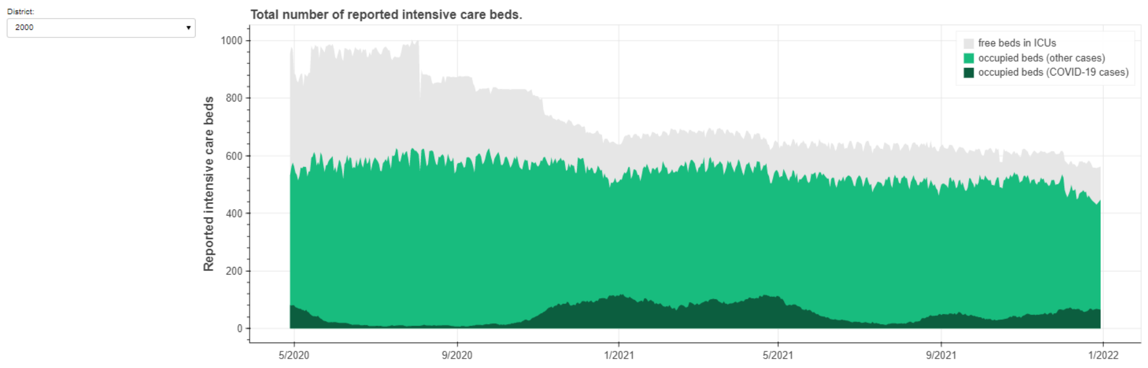


Figure 3.2: Stacked area diagram of number of reported intensive care beds. The 2 main areas (green and grey) each show the number of occupied and free, operable intensive care beds at the respective observation day. The dark green area shows the number of beds, occupied by COVID-19 patients. The capacities are "stacked" on top of each other and illustrate the reported total potential capacity. Example based on district 'SK Hamburg'.

Figure 3.2 shows that the total bed capacities are not constant over time. Since the beginning of the pandemic, bed capacities in many districts have been reduced. One reason for the reduction is the shortage of personnel in hospitals. Furthermore, only operable intensive care beds appear in the statistics, i.e., beds for which rooms, equipment, and personnel are available. Due to the inconsistency of total bed capacities over time, the target variable in our benchmark experiment is the absolute number of occupied ICU beds by COVID-19 patients. The proportion of occupied ICU beds by COVID-19 patients to total ICU bed capacity can be calculated subsequently after prediction.

### *Incidence of reported COVID-19 infections*

Figure 3.3 illustrates the development of weekly reported new infections in the selected district, compared to the average development in Germany. Figure 3.4 represents the development in total and among different age groups. Values are represented per 100.000 inhabitants and are based on data for the year 2021.

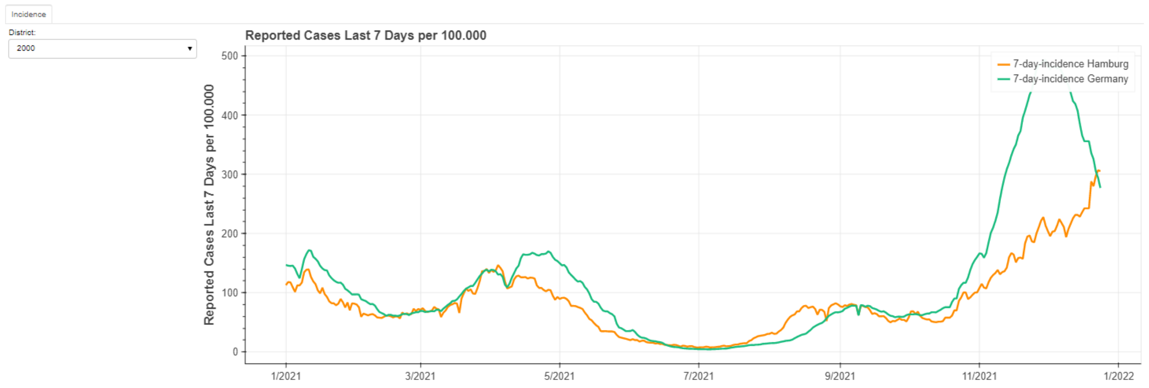


Figure 3.3: Number of reported COVID-19 cases last 7 days per 100.000 inhabitants over the course of year 2021. Example based on district ’SK Hamburg’.

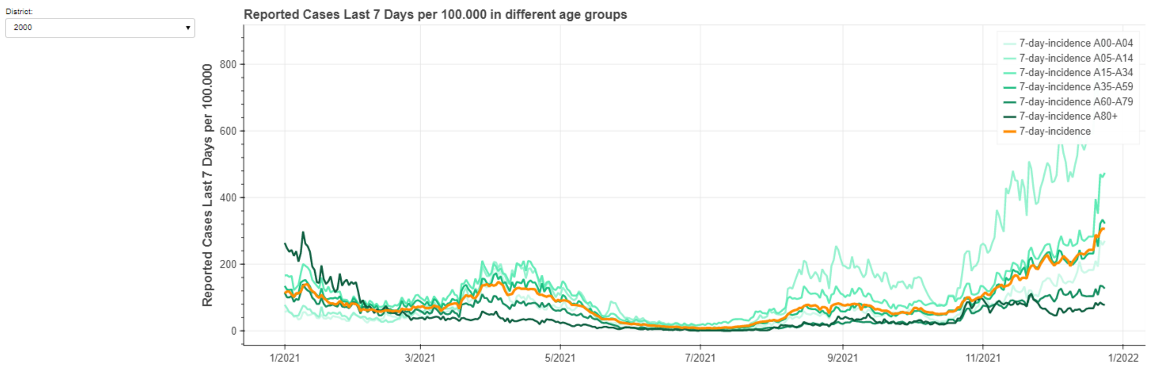


Figure 3.4: Number of reported COVID-19 cases in different age groups last 7 days per 100.000 inhabitants over the course of year 2021. Example based on district ’SK Hamburg’.

Following COVID-19 incidence rates by age over time, there is a visible shift of infection from the older to the younger population. As testing expands, it becomes evident that all age groups are affected by COVID-19 infection and predominantly the most socially active, least likely to be symptomatic younger population. As the illustration shows, a persistently high percentage of current infections exist among the age 5–14 and 15–34 population who may also be at the highest risk of contracting and spreading the virus but not at high risk of hospitalization or mortality. It is crucial to account for such developments and incorporate them in modeling the risk for admission in intensive care units.

### *Incidence of reported COVID-19 deaths*

Figure 3.5 illustrates the development of weekly reported deaths in the selected district, compared to the average development in Germany. Figure 3.6 represents the development in total and among different age groups. Values are represented per 100.000 inhabitants and are based on data for the year 2021.

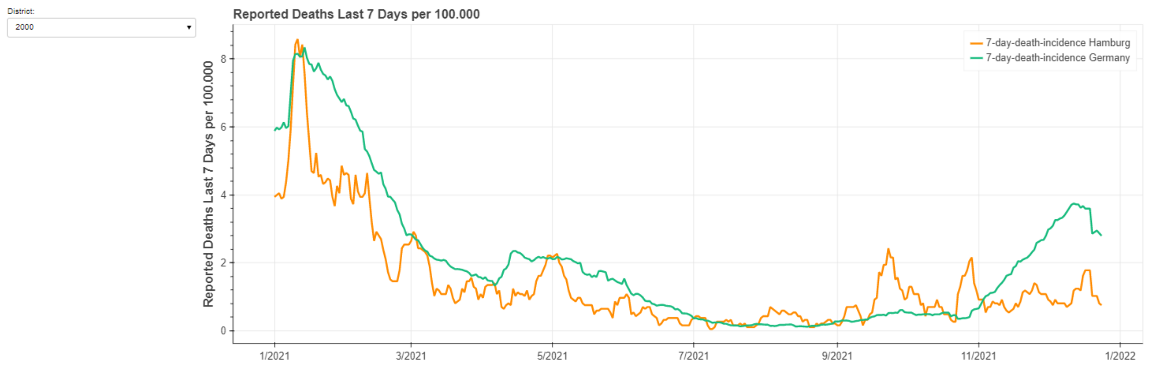


Figure 3.5: Number of reported COVID-19 deaths last 7 days per 100.000 inhabitants over the course of year 2021. Example based on district 'SK Hamburg'.

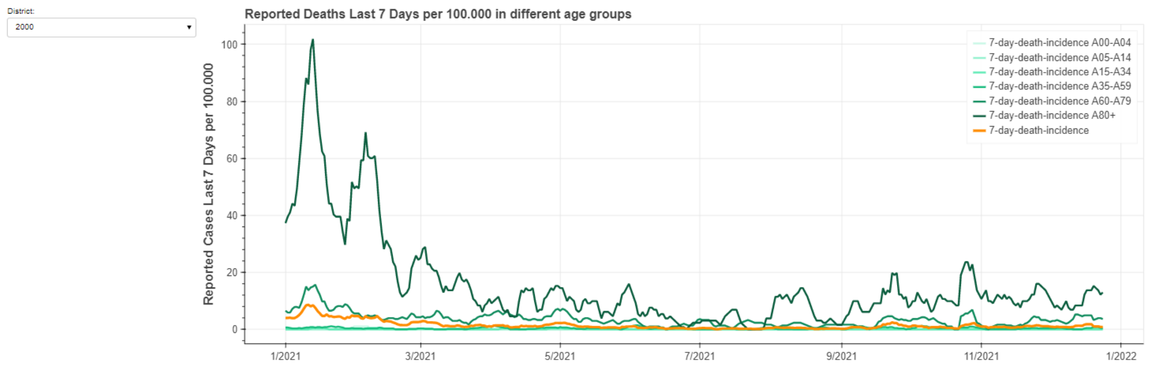


Figure 3.6: Number of reported COVID-19 deaths in different age groups last 7 days per 100.000 inhabitants over the course of year 2021. Example based on district 'SK Hamburg'.

Unsurprisingly, among all age groups, the older population, age 80 or above, are at a greater risk of hospitalization or dying with COVID-19 infection due to age-related decline in immune function and potential pre-existing medical conditions. However, due to the advanced stage of vaccination, there is a significant decrease in mortality, especially among the older population groups.

### ***COVID-19 vaccination coverage***

Figure 3.7 illustrates the vaccination coverage until a certain point in time. The vaccination coverage signifies the proportion of the population, which is vaccinated against COVID-19. The figure depicts both, the percentage of the fully vaccinated population and the percentage of the population, which has been re-vaccinated with a booster vaccination. Figure 3.8 represents the vaccination development among different age groups. Here, only fully vaccinated population is illustrated.

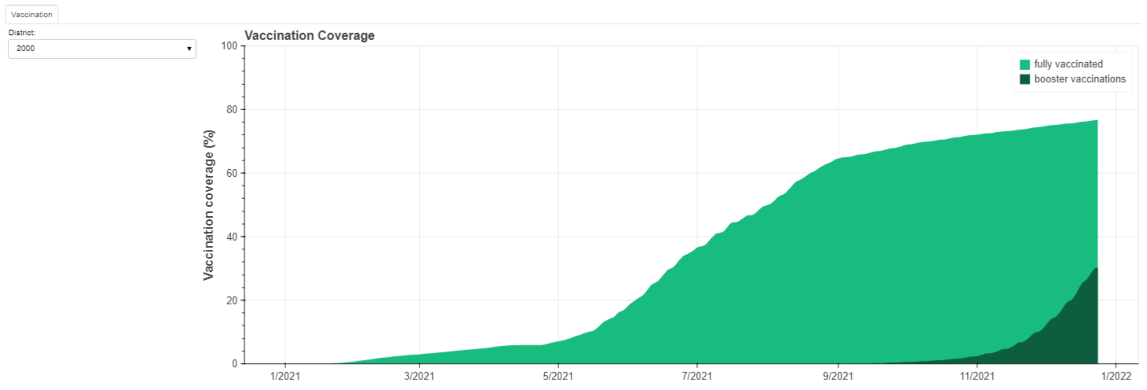


Figure 3.7: Vaccination coverage (%) of fully vaccinated and re-vaccinated population. Example based on district 'SK Hamburg'.

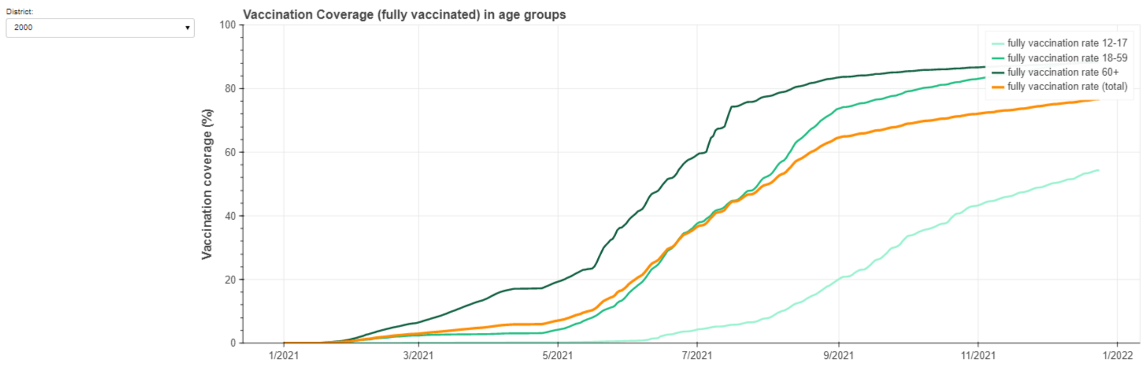


Figure 3.8: Vaccination coverage (%) of fully vaccinated population in different age groups. Example based on district 'SK Hamburg'.

### Changes in the dynamics of the pandemic

All discussed key indicators represent different aspects of the pandemic. They are all related and could contribute to explaining the development of COVID-19 cases in intensive care units. A correlations heatmap, which measures the average correlation between features and the target over the course year 2021 can be found in the Appendix. This representation gives insights into the average magnitude of the relationship between each feature and the admission in intensive care units. However, the respective importance of key metrics constantly changes over time in response to new conditions. It is crucial to choose an appropriate modeling strategy, which accounts for changes in the association between variables. The modeling strategy of the applied benchmark experiment in this thesis is presented and motivated in Chapter 4.

One example of the varying association between key metric is the tendency of decoupling between the reported new COVID-19 infections and the number of COVID-19 patients in intensive care units, which is illustrated in Figure 3.9. The figure depicts the development of the 7-day incidence of infections and the number of ICU beds, occupied by COVID-19 patients over the course of year 2021 for district 'SK Hamburg'. It is visible how in time, both metrics drift further apart. Despite the significantly higher number of cases by the end of the year, there is no notable increase in the number of particularly serious infections requiring treatment in intensive care units. This development can be explained by the advanced stage of vaccination, which significantly decreases the risk for serious infections. Moreover, there is an observable shift of the pandemic to younger age groups, which generally exhibit a lower risk of severe courses of infection.

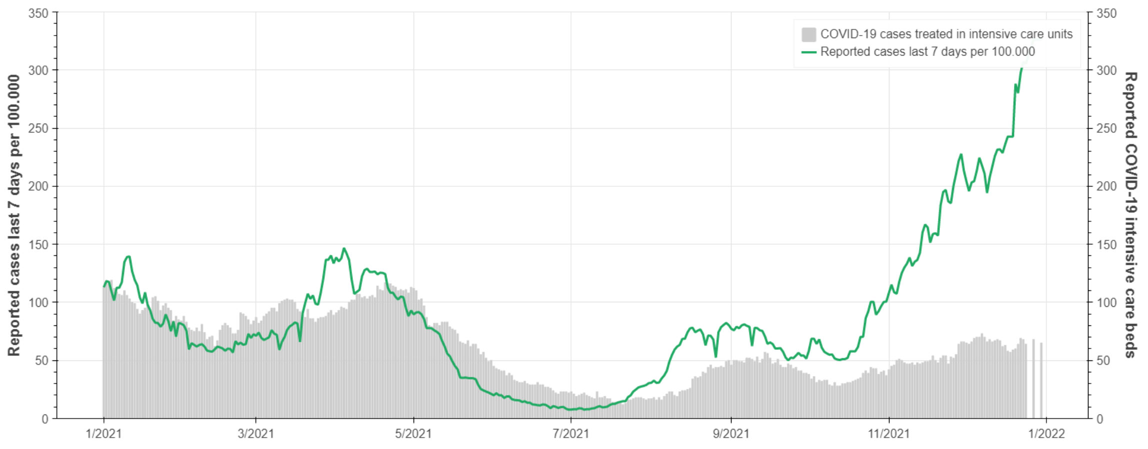


Figure 3.9: Comparison of reported COVID-19 cases last 7 days per 100.000 inhabitants (green) with the number of COVID-19 patients in intensive care units (grey) over the course of year 2021.

# 4. Benchmark Experiment

To evaluate prediction performances of selected machine learning methods - random forest and gradient boosting, and compare them to a classical statistical forecasting model - ARIMA, a benchmark experiment is conducted. This chapter discusses major aspects of the experimental design - the experimental setup, the choice of model implementations, the evaluations strategies and metrics, and hyperparameter tuning. We discuss time series specific evaluation and tuning strategies, which respect the chronological order of the data and support capturing dynamic changes. In addition, main hyperparameters and their impact on prediction performance are examined and different hyperparameter tuning strategies are discussed. The results and findings are presented in Chapter 5.

## 4.1 Benchmark Setup

### 4.1.1 Environment

The benchmark experiment is performed via Python 3.8. The conducted benchmark experiment is based on a unified framework for machine learning, based on the Python package `scikit-learn` (Pedregosa et al., 2011), which makes them easily extendable and reproducible. A model configuration is further addressed as "learner".

For the ARIMA method, we use the time series analysis model class from the `statsmodels` package (Seabold and Perktold, 2010). For the random forest method, the experiment focuses on the implementations `skranger`, which provides compatible Python bindings to the C++ random forest implementation `ranger` (Wright and Ziegler, 2017). For the gradient boosting method, the implementation from the package `XGBoost` (Chen and Guestrin, 2016) is chosen. Hyperparameter procedures for random search are implemented via `scikit-learn`. The required splitting methods for the walk-forward validation and hyperparameter tuning have been implemented by us additionally and integrated as customized evaluation methods into the `scikit-learn` built-in functions. The produced splitting methods have been designed to handle repeated measurements, since splitting in the machine learning methods is conducted simultaneously for all districts.

Visualizations are mostly made using the `plotnine` implementation of a grammar of graphics in Python, which is based on the R package `ggplot2` (Wickham, 2016). Additionally `plotly` (Plotly Technologies Inc., 2015) and `Matplotlib` (Hunter, 2007) are used. The implemented interactive vizualisation tools in this thesis are programmed via the interactive visualization library `Bokeh` (Bokeh Development Team, 2018).

### 4.1.2 General Strategy and Data

#### Benchmark Strategy and Data

The current benchmark experiment focuses on comparing performances of ARIMA, random forest and gradient boosting. Table 4.1 demonstrates the examined learners, including an overview of the considered comparison strategy. A full specification of the applied resampling strategy and performance measures is available in Chapter 4.2.

| method | learner | package | evaluation measure | evaluation method |
|---|---|---|---|---|
| ARIMA | `ARIMA` | `statsmodels` | RMSE | Rolling Window CV |
| RF | `RangerForestRegressor` | `skranger` | RMSE | Rolling Window CV |
| GB | `XGBRegressor` | `XGBoost` | RMSE | Rolling Window CV |

Table 4.1: Summary of algorithm implementations used for the benchmark experiment, incl. evaluation measures and evaluation methods.

In the scope of this benchmark experiment, we compare candidate algorithms, based on a single main performance criterion, e.g., the RMSE metric. However, we want to point out that it is advisable to evaluate model performance, based on multiple criteria, e.g., as a trade-off between predictive power, computational resources and the number of features used (Bischl et al., 2013).

Model comparison is conducted on the complete dataset of one year, from 01.01.2021 to 31.12.2021, to examine generalizability across different stages of the pandemic. For ARIMA, training, prediction, resampling and hyperparameter tuning are conducted individually for each district. Meanwhile, since machine learning models can deal with repeated measurements, procedures are performed on all provided districts simultaneously. This strategy supplies the models with more data and enables the incorporation of potential interactions between districts.

#### Hyperparameter Tnung Strategy and Data

Prior to benchmarking prediction performances of gradient boosting and random forest against ARIMA, hyperparameter tuning for each model is performed. A detailed description of the hyperparameter strategy, as well as relevant hyperparameters and their contribution to the models, is available in Chapter 4.3. Here we give a brief overview of the procedure and the underlying data.

Classical machine learning algorithms are not designed to handle time series data by default. To adapt machine learning models to the specifics of time series data, different adjustments are applied: e.g., addition of lagged variables of the target and training the models on a subset of most recent data, also referred to as rolling window. In this experiment, we handle the number of lags and the length of the rolling window as hyperparameters. We denote this hyperparameters as *time-series-specific hyperparameter*. The build-in hyperparameters of the selected algorithms are then referred to as *algorithm-specific hyperparameter*. Generally, different hyperparameters are often related to each other. Consequently, an appropriate hyperparameter tuning procedure takes possible interaction effects between hyperparameters into account. On that note, it would be advisable to apply a tuning procedure, which is optimizing both hyperparameter sets (time-series-specific and algorithm-specific) simultaneously. However, this process would require additional technical implementation and computational capacity, which goes beyond the scope of this experiment. Therefore, we consider both sets individually. First, tuning for the time-series-specific hyperparameter is performed. Consequently, algorithm-specific hyperparameter tuning is conducted with the best hyperparameter from the first step. Since the hyperparameter of ARIMA already incorporates the specifics of time series data, for ARIMA only algorithm-specific tuning is required.

Hyperparameter tuning can be very computationally and memory expensive. To keep the computational complexity in a reasonable range, tuning is performed on a time frame of approximately 4 months, from 01.08.2021 to 30.11.2021. The selected time frame incorporates data with very low to very high COVID-19 bed occupancy rates and reflects important structural differences of the time series over time. ARIMA hyperparameter tuning is conducted individually for each district. Machine learning hyperparameter tuning is conducted on all districts simultaneously. ARIMA tuning is conducted only for a prediction horizon of 14 days. Due to the recursive multi-horizon prediction strategy of ARIMA (see Chapter 2.2.1), tuning on the longest prediction horizon and applying the same model to all previous horizons is a reasonable strategy. Meanwhile, the constructed machine learning models use a direct strategy for multi-step forecasting, where $h$ independent models are built for each prediction horizon. Therefore, corresponding hyperparameter tuning is performed for each of the investigated prediction horizons individually.

To ensure generalization of the selected hyperparameters across the full dataset for year 2021, we carry out a performance comparison of default against tuned models on two different datasets: on the tuning dataset, from 01.08.2021 to 30.11.2021, as well as on a validation dataset, from 01.01.2021 to 31.07.2021. The main idea of this comparison is to ensure that no overfitting on the tuning dataset has occurred.

## 4.2 Performance Evaluation

### 4.2.1 Resampling Strategy

Cross-validation (CV) is one of the most widely used methods to assess the generalizability of algorithms in regression (Hastie et al., 2009). In $k$-fold cross-validation, the data is randomly divided into $k$ equally-sized folds. Each fold is a subset of the data comprising $n/k$ randomly assigned observations, where $n$ is the number of observations. Iteratively the model is trained on $k-1$ folds and performance is estimated on the left-out fold. This procedure is repeated $k$ times and a summary measure is calculated.

However, there is evidence in the literature that classical $k$-fold cross-validation does not apply to time series forecasting. To reflect real-world forecasting, in which we stand in the present and forecast the future, the chronological order of the data must be preserved. Observed data cannot be randomly split in training and testing samples, due to the temporal relationships in the data (Tashman, 2000). In addition, classical cross-validation assumes independence and identical distribution of the observations. Not meeting this assumption might lead to overly optimistic estimations and consequently, poor generalization ability of predictive models on new observations (Roberts et al., 2017).

The above considerations can be incorporated into cross-validation methods, adapted to time series data. To consider temporal relationships in the data, a rolling window (also called walk-forward validation or rolling out-of-sample validation) strategy can be employed (Hyndman and Athanasopoulos, 2018). In rolling out-of-sample forecasting, one produces a sequence of pseudo out-of-sample forecasts using a fixed number of the most recent data at each point of time. The selected data is called a fixed-sized training window of length $l$, which slides over the entire history of time series and is repeatedly tested against a future observation. The model is trained on the train set, train $= \{t_i, ..., t_{i+l}\}$, validated on the subsequent $h$-th period, test $= \{t_{i+l+h}\}$, guaranteeing that the train set temporally precedes the test set. Stepwise moving through the entire data, forecasting error is recorded at each step and prediction accuracy is computed by averaging over the test sets. This approach is illustrated in Figure 4.1, where the dark green observations form the training sets, and the light green observations form the test sets. The illustrated example is based on a 4-steps ahead prediction.

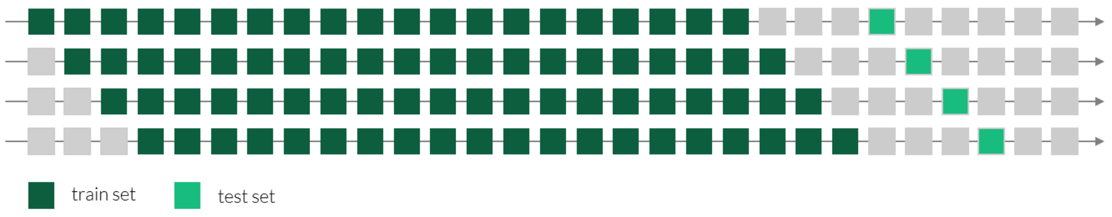


Figure 4.1: Illustration of the time series walk-forward validation. Example, based on prediction horizon of 4 steps. Train sets (dark green), test set (light green).

For hyperparameter tuning, a nested resampling strategy is required to ensure reliable performance estimation. An outer loop is performed to provide a performance estimate for the model. An inner loop is required for evaluating different hyperparameter configurations. This strategy ensures that the test data in each iteration of the outer resampling has not been used to optimize the performance of the model (Bischl et al., 2012).

Nested cross-validation can be adapted to time series data as well. This can be achieved by splitting the data into a train, validation, and test split. The train/validation split is constructed analogously to the rolling window validation process, e.g., train = $\{t_i, ..., t_{i+l}\}$ and val = $\{t_{i+l+h}\}$. The test set is the subsequent $h$-th period, test = $\{t_{i+l+2h}\}$. In an inner loop, different hyperparameter combinations are evaluated, by training on the train splits and evaluation on the validation split. Consequently, the model with the best hyperparameter is trained again on the whole inner loop data and evaluated with the test set. This approach is illustrated in Figure 4.2. The illustrated example is based on a 2-steps ahead prediction.

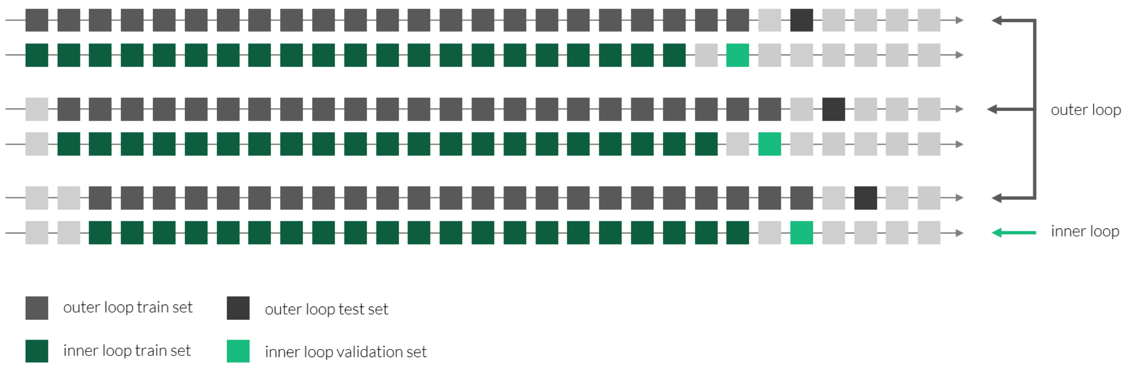


Figure 4.2: Illustration of the nested time series walk-forward validation. Example, based on prediction horizon of 2 steps.

A crucial aspect of the rolling-window evaluation is the choice of the window length or how much of the most recent observations should be used in the estimation. To enable learning of complex structures in the data, enough data must be fed into the machine learning models. However, in time series data, especially in our case, the association between variables changes over time, due to the dynamic nature of the pandemic (see Chapter 3.3). For regression tasks, instability can be handled by forecasting based on a subset of the data. There is enough evidence in the literature, that forecasting, based on only the most recent observations, produces a reasonably good and robust forecasting performance (Inoue et al., 2017). Consequently, the optimal window length must provide a reasonable trade-off between efficiency and accuracy of the model. Within the framework of this thesis, window length is handled as a hyperparameter. The choice of the optimal length is based on the lowest average out-of-sample loss over a cross-validation sample. More on this procedure can be found in Chapter 4.3.

### 4.2.2 Performance Measures

The most commonly used error measures for regression models are the mean absolute error (MAE) and the root mean squared error (RMSE). Appropriate performance measures depend strongly on the intended application. The mean absolute error is obtained by measuring the average of the absolute differences between prediction and actual observation, where all individual differences have equal weight:

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} |y_t - \hat{y}_t| \in [0; \infty)$$

with $\hat{y}_t$ being the forecasted value for period $t$ and $y_t$ being the actual value at the respective period.

An advantage of the MAE measure is that it gives more insight into the average magnitude of the errors over the entire dataset. Additionally, MAE is less sensitive to an outlier in the data. However, the RMSE can be a preferred metric for evaluating COVID-19 bed occupancy errors, since it intrinsically places more weight on larger error terms. Large deviations can have additional pressure on the healthcare systems. Penalizing and reducing big errors as much as possible is essential to facilitate the planning of COVID-19 and other activities in hospitals. Therefore, prediction performance in the conducted benchmark experiment is assessed via the RMSE metric:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2} \in [0; \infty)$$

To produce results independent of district sizes and bed capacities and enable comparison of prediction performance across different districts, the RMSE measure can be normalized by the total ICU bed capacity. The normalized RMSE is then given by:

$$\text{nRMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(\frac{y_t}{c_t} - \frac{\hat{y}_t}{c_t}\right)^2} \in [0; \infty)$$

with $c_t$ the total ICU bed capacity (total number of available beds) at time $t$.

**Prediction intervals**

Model selection in the conducted benchmark experiment is based exclusively on the RMSE metric. However, alongside minimizing the prediction error and producing reliable point forecasts, we are interested in uncertainty estimates associated with the forecasts. Incorporating prediction uncertainty into deterministic forecasts helps to enhance the reliability and credibility of the model outputs.

In Chapter 2.2.2 we introduce how uncertainty quantification, i.e., the calculation of prediction intervals is handled in ARIMA. One of the drawbacks of the machine learning approach is that it does not have any built-in capability to calculate prediction intervals, while most statistical time series forecasting implementations, such as ARIMA, have it. However, there are different methods for the calculation of prediction intervals for machine learning models, which we discuss in Chapter 2.3.5. In the current experiment, our fist attempt for producing prediction intervals for random forest and gradient boosting has been via the quantile regression approach (see Meinshausen, 2006, Kriegler and Berk, 2007). Both algorithm implementations `skranger` and `XGBoost` have optional loss functions for quantile regression. However, in our opinion, the produced results have been rather unreliable. Producing customized loss functions for quantile regression or conducting further research on alternative approaches and existing implementations is out of the scope of this thesis.

Therefore, within this thesis, we employ an analytical approach to producing prediction intervals for random forest and gradient boosting. Assuming normally distributed errors, a 95% prediction interval is acquired by exploiting the RMSE metric, calculated on the last 50 days:

$$\hat{y}_{t+h} \pm 1.96 \times \text{RMSE(t)} \quad \text{with} \quad \text{RMSE(t)} = \sqrt{\frac{1}{50} \sum_{i=t-51}^{t-1} (y_i - \hat{y}_i)^2} \in [0; \infty)$$

Despite the build-in ability of ARIMA for prediction intervals calculation, we apply this analytical approach to ARIMA as well. This ensures consistency and allows for comparability of prediction uncertainty across ARIMA, random forest and gradient boosting.

## 4.3 Hyperparameter Tuning

Forecasting and machine learning algorithms involve a number of hyperparameters that have to be configured before running. In contrast to first-level model parameters, which are estimated during training, these second-level tuning parameters often have to be carefully optimized to achieve maximal performance. In order to select an appropriate hyperparameter configuration, a user can rely on default values of hyperparameters that are specified in implementation software packages (Probst et al., 2019). However, noticeable performance improvements can be achieved with hyperparameter tuning in some cases (Weerts et al., 2020). For that hyperparameter tuning strategies can be used, which try to minimize the expected generalization error of the algorithm over a hyperparameter search space of considered candidate configurations. There is evidence in the literature, that hyperparameter tuning of some algorithms holds higher potential of performance improvements than other. Random forest is well known to be relatively robust against hyperparameter configurations (Probst et al., 2019). Meanwhile, gradient boosting can have high variability in accuracy dependent on their hyperparameter settings (Probst et al., 2019).

### 4.3.1 ARIMA

This section presents the central hyperparameters of the ARIMA algorithm, by shorty discussing their influence on prediction performance. Additionally, the choice of the hyperparameter search space, on which the tuning is executed in this experiment, is motivated. Moreover, the selected hyperparameter tuning strategy is presented and motivated.

**ARIMA hyperparameter and tuning search spaces**

Table 4.2 displays the general ARIMA hyperparameter with description.

| hyperparameter | description |
|---|---|
| p | number of autoregressive terms |
| d | number of nonseasonal differences needed for stationarity |
| q | number of lagged forecast errors in the prediction equation |

Table 4.2: Overview of ARIMA hyperparameter.

$p$ captures the autoregressive nature of ARIMA and reflects the number of lag observations in the model, also known as the lag order. The autoregressive part shows how the variable is impacted by its values on the previous observations. For the current experiment the range between 1 and 15 lags is explored.

$d$ represents the number of times the data has to be differenced to produce a stationary signal, also known as the degree of differencing. Working with differences rather than with the original data means that we deal with changes and rates of changes, not with just values. For the current experiment, the differencing order between 0 and 2 is explored.

$p$ captures the moving average component, explaining how the previous white noise impacts the variable. For the current experiment, the range between 0 and 2 lags is explored.

### ARIMA hyperparameter tuning strategy

A classical way of determining ARIMA hyperparameters is by using the ACF plot and the closely related PACF plot to determine appropriate values for $p$ and $q$, as well as applying some statistical tests, e.g., the Augmented Dickey-Fuller test (Dickey and Fuller, 1979) to check for stationarity and the need for differencing. However, for the scope of this thesis, we need to fit approximately 400 ARIMA models (one for each district), which requires a rather automatic approach to hyperparameter determination. Moreover, applying the same hyperparameter tuning approach to all models of the benchmark experiment creates consistency and better comparability across models. Therefore, hyperparameter tuning is conducted by optimizing the RMSE via the rolling window cross-validation process. Hyperparameter tuning for ARIMA is executed with the discussed parameter and defined search spaces above. Table 4.3 displays an overview of the performed strategy.

| hyperparameter | search space | tuning strategy | evaluation metric | evaluation method |
|---|---|---|---|---|
| ARIMA | | | | |
| p | [1, 15] | | | |
| d | [0, 2] | grid search | RMSE | rolling window CV |
| q | [0, 2] | | | |

Table 4.3: Overview of the hyperparameter tuning strategy for ARIMA: selected hyperparameter, hyperparameter search spaces and evaluation method.

The procedure is performed automatically via grid search. Grid search (GS) can be thought of as an exhaustive search for selecting a model. In this approach, every combination of hyperparameter values is tried which can be very inefficient. Since this process can be extremely costly in both, computing power and time, it is only applied in cases of small parameter spaces (Ghawi and Pfeffer, 2019). One alternative to grid search is random search (RS). Random search (RS) is a naive, model-free, automated strategy for hyperparameter tuning, in which hyperparameter values are drawn randomly from the defined hyperparameter space and performance is compared (Bergstra and Bengio, 2012). This allows to explicitly control the number of parameter combinations that are attempted. The random search procedure is easy to implement and the number of trials is easily extendible. Additionally, trials can be carried out asynchronously. The random search method oversteps some disadvantages of grid search such as high time resources but meanwhile brings a major disadvantage with its inability to converge to the global optimum (Andradóttir, 2015).

As already discussed, the hyperparameter tuning strategy for ARIMA is applied for each Geman district individually and only for a prediction horizon of 14 days (see Chapter 4.1.2).

### 4.3.2 Random Forest and Gradient Boosting

As discussed in Chapter 4.1.2, hyperparameter tuning for random forest and gradient boosting consists of two steps: tuning of time-series-specific hyperparameter and tuning of algorithm-specific hyperparameter. This section presents the central time-series-specific hyperparameter, as well as the algorithm-specific hyperparameter of the random forest and gradient boosting algorithms, by shorty discussing their influence on prediction performance. Additionally, the choice of the hyperparameter search space, on which the tuning is executed in this experiment, is motivated. Moreover, the selected hyperparameter tuning strategy is presented and motivated.

#### Time-series-specific hyperparameter

The discussed hyperparameters in this section are not related to a specific algorithm and are the product of the time-series-specific machine learning strategy implemented in this thesis. These hyperparameters include the length of the test sets, defined by the sliding window length, as well as the number of lags of selected variables. Table 4.4 displays the relevant time-series-specific hyperparameters for this experiment.

| hyperparameter | description | selected default values |
|---|---|---|
| window_length | Number of most recent time periods, used for parameter estimation. | 50 |
| num_lags | Number of past observation of an endogenous or exogenous variable, included in the model. | 5 |

Table 4.4: Overview of time-series-specific hyperparameter.

The parameter *window_length* denotes the number of most recent time periods, used for parameter estimation. This parameter controls an important trade-off between stability and accuracy of the models. To enable the learning of complex structures in the data, enough data must be fed into the machine learning models. On the other hand, structural changes and instability of the series can be incorporated by training on a smaller set, including only the most recent observations. As a default value, we consider the length of 50 days. Within the scope of the hyperparameter tuning procedure, we explore window lengths between 10 and 200 days.

Another critical step for time-series forecasting is the right determination of the number of past observations (lags). This is controlled by the *num_lags* parameter. Lagged variables can be added for the target, as well as for other important exogenous features of the model. The proper selection of the time-lag value becomes important to find the section of data where values of independent variables are highly correlated, which would positively contribute to the forecasting accuracy. Here, we consider lagged variables of the target - number of ICU beds, occupied by COVID-19 patients. In order to consider the time between the occurrence of symptoms, worsening of symptoms, and admission in the ICU, we add lagged observations of the 7-day reporting incidence as well. As default values, we consider 5 lags of the target and 5 lags of the 7-day reporting incidence. Within the scope of the hyperparameter tuning procedure, we explore lag numbers between 1 and 30 for both selected variables.

**Random forest hyperparameter**

Table 4.5 displays the main random forest hyperparameter with description and their common default values as implemented in the `skranger` package, used in this experiment.

| hyperparameter | description | common default values |
|---|---|---|
| n_estimators | Number of trees that constitute forest. | 100, 500, 1000 |
| mtry | Number randomly drawn candidate variables in each split. | $\sqrt{p}$ |
| sample_fraction | Number of observations drawn for each tree. | n |
| replace | Draw observations with (bootstrapping) or without replacement. | TRUE (with) |
| min_node_size | Minimum number of observations in a terminal node. | 5 |
| max_depth | Maximum depth of the tree. | unlimited |
| split_rule | The rule by which each split is considered in a tree. | variance |

Table 4.5: Overview of main hyperparameter of random forest and common default values. $n$: number of observations, $p$: number of features in the dataset.

The parameter *n_ estimator* denotes the number of trees that constitute the forest. This parameter does not necessarily require tuning but should be set sufficiently high to stabilize the error rate. More trees provide more robust and stable error estimates. In general, the higher the number of trees the more reliable is the prediction. However, the impact on computation time increases linearly with the number of trees. After evaluating performance with a different number of trees, the default value of 100 trees shows sufficient results. Therefore, no future tuning of this parameter is conducted.

*mtry* is one of the central RF hyperparameters, defining the number of variables, which are randomly selected as candidate variables at each split. A low value of *mtry* leads to less correlated trees, which improves the stability of the model when aggregating. Moreover, a low value increases the chance of selecting features with moderate effect, which may contribute to prediction performance improvement. On the other hand, a high value of *mtry* reduces the risk of disregarding important features. To ensure a feasible trade-off between stability and accuracy of the model, the search space for *mtry* incorporates from one to up to all available features.

The sample size reflects the number of observations drawn for each tree. Similar to *mtry* the *sample_fraction* parameter maintains a trade-off between stability and accuracy of the model. Smaller sample sizes lead to a lower correlation between trees. On the contrary, decreasing the sample size implicates less information available for each tree. For the current experiment, a fraction range between 0.1 and 1 is explored. Additionally, sampling is available with and without replacement. Both options are considered in parameter tuning.

The parameters *min_ node_ size* and *max_ depth* control the complexity of individual trees. *min_ node_ size* represents the minimum number of observations in a terminal node. Larger values lead to smaller trees. *max_ depth* represents the depth of the tree. Respectively smaller values produce smaller, less complex trees. Meanwhile, computational time decreases approximately exponentially with increasing node size and decreasing tree depth. Node sizes between 1 and 50, and tree depth between 2 and 30 are explored in the tuning process. As suggested by Segal (2004), a higher node size than the default value decreases runtime substantially, often without contributing to a considerable loss of accuracy.

### Gradient Boosting hyperparameter

Table 4.6 displays the main gradient boosting hyperparameter with description and their common default values as implemented in the `XGBoost` package, used in this experiment.

| hyperparameter | description | common default values |
|---|---|---|
| n_estimators | Number of trees in the sequence. | 100, 500, 1000 |
| learning_rate | Boosting learning rate. | 0.3 |
| colsample_by_level | Fraction of randomly drawn candidate variables in each split. | 1 |
| colsample_by_tree | Fraction of randomly drawn candidate variables in each tree. | 1 |
| subsample | Subsample ratio drawn for each iteration | 1 |
| min_child_weight | Minimum number of observations in a terminal node. | 1 |
| max_depth | Maximum depth of the tree. | 6 |
| reg_alpha | $L_1$ regularization term on weights | 0 |
| reg_lampda | $L_2$ regularization term on weights | 1 |

Table 4.6: Overview of main hyperparameter of gradient boosting and common default values.

Similar to the random forest, the parameter *n_ estimator* denotes the number of trees of the ensemble. Unlike random forest, where averaging of many independently grown trees makes it very difficult to overfit the model, each tree in gradient boosting is grown in sequence to fix up the error of the previous tree. Although the gradient boosting algorithm often needs a lot of trees, it can easily lead to overfitting. For this benchmark experiment, we keep the default value of 100 trees to maintain stability.

The shrinkage parameter, *learning_ rate*, is used to prevent overfitting and makes the boosting process more conservative by reducing the impact of each additionally fitted base-learner. It reduces the size of incremental steps and thus penalizes the importance of each consecutive iteration. Generally, it is better to improve a model by taking many small steps than by taking fewer large steps. Smaller values make the model robust to the specific characteristics of each individual tree, which allows better generalization and mitigates overfitting. However, a smaller learning rate increases the risk of not reaching the optimum with a fixed number of trees and comes with a high computational cost. Learning rates between 0.1 and 1 are explored in the tuning process.

A further technique used to prevent overfitting in the gradient boosting model is the *subsample* technique. Similar to the random forest, subsampling techniques introduce some randomness onto the fitting procedure. Additionally to fitting the consecutive tree only to a random part of the training (*subsample*), `XGBoost` introduces additional subsampling on column level - subsampling columns before creating each tree (*colsample_ by_ tree*) and subsampling columns before considering each split (*colsample_ by_ level*). According to user feedback, Chen and Guestrin (2016) state that using column subsampling prevents overfitting even more than the traditional row subsampling. For all 3 randomization parameters, a range between 0.1 and 1 is explored in the tuning process.

Analogously to random forest, the parameters *min_ node_ size* and *max_ depth* control the complexity of individual trees. Smaller depth trees or trees with larger terminal nodes are computationally efficient and help with mitigating overfitting. Higher depth trees help the algorithm to capture unique interactions but increase the risk of overfitting. Hence, both parameters are set to maintain a trade-off between the stability and accuracy of the model. Node sizes between 1 and 50 and tree depth between 2 and 30 are explored in the tuning process.

Finally, two more traditional regularization parameters *alpha* and *lambda* are included. The $L_1$ regularization (*alpha*), also called lasso penalty, combats overfitting by shrinking the parameters towards 0. The $L_2$ regularization (*lambda*), also called ridge penalty, controls the estimated coefficients by pushing them to approximately, but not equal zero. Generally, these regularization parameters limit how extreme the weights of the leaves in a tree can become. In our experiment, we explore the whole possibility range between 0 and 1 for both parameters.

**Random forest and gradient boosting tuning strategy**

Hyperparameter tuning for random forest and gradient boosting is executed in two steps with the discussed parameter and defined search spaces above. First, tuning for the time-series-specific hyperparameter is performed. Consequently, algorithm-specific hyperparameter tuning is conducted with the best hyperparameter from the first step. Table 4.7 displays an overview of the performed strategy.

The tuning procedure is performed automatically via grid search for the time-series-specific hyperparameters and via random search for the algorithm-specific hyperparameter with 100 to 200 evaluations. Hyperparameter tuning is performed for all districts simultaneously and for each prediction horizon individually, which results in 14 optimal hyperparameter combinations (see Chapter 4.1.2).

| hyperparameter | search space | tuning strategy | evaluation metric | evaluation method |
|---|---|---|---|---|
| **Time-series-specific** | | | | |
| window_length | [10,200] (step 10) | grid search | RMSE | rolling window CV |
| num_lags (target) | [1,15] | | | |
| num_lags (7-day incidence) | [1,15] | | | |
| **Algorithm-specific** | | | | |
| **Random Forest** | | | | |
| mtry | [1, p] | random search with 100 evaluations | RMSE | rolling window CV (outer and inner) |
| sample_fraction | [0.1, 1] | | | |
| replacement | [TRUE, FALSE] | | | |
| min_node_size | [1, 50] | | | |
| max_depth | [2, 30] | | | |
| **Gradient Boosting** | | | | |
| learning_rate | [0.1, 1] | random search with 200 evaluations | RMSE | rolling window CV (outer and inner) |
| colsample_by_level | [0.1, 1] | | | |
| colsample_by_tree | [0.1, 1] | | | |
| subsample | [0.1, 1] | | | |
| min_child_weight | [1, 50] | | | |
| max_depth | [2, 30] | | | |
| reg_alpha | [0, 1] | | | |
| reg_lampda | [0, 1] | | | |

Table 4.7: Overview of the hyperparameter tuning strategy for random forest and gradient boosting: selected hyperparameter, hyperparameter search spaces and evaluation method.

# 5. Results

This chapter comprises the results of the conducted benchmark experiment. First, we take a look into the results from the hyperparameter tuning procedure and evaluate how well the selected hyperparameter configurations generalize on previously unseen data. After appropriate hyperparameter setting, prediction performances of random forest and gradient boosting are benchmarked against the ARIMA model. We take a look into the average prediction results, as well as into the geographical distribution of prediction accuracy across all German districts. Moreover, to provide an overview of the prediction curves we have developed an interactive visualization tool, which retrospectively compares predicted and observed time series for each district and forecast horizon. Here we provide an example visualization of a selected district over different prediction horizons. Finally, this chapter summarizes the insights obtained by the variable importance measures of the models.

## 5.1 Hyperparameter Tuning

Prior to benchmarking prediction performances of gradient boosting and random forest against ARIMA, hyperparameter tuning for each model is performed. Hyperparameter tuning is conducted on data for 4 months, from 01.08.2021 to 30.11.2021. To ensure generalization of the selected hyperparameters across the full dataset for the year 2021, we carry out a performance comparison of default against tuned models on two different datasets: on the tuning dataset, from 01.08.2021 to 30.11.2021, as well as on a validation dataset, from 01.01.2021 to 31.07.2021. The main idea of this comparison is to ensure that no overfitting on the tuning dataset has occurred.

The hyperparameters of ARIMA already incorporate the specifics of time series data. Therefore, for ARIMA only algorithm-specific tuning is required. ARIMA hyperparameters are tuned for each district individually. To keep computational complexity in a reasonable range, tuning is performed only for a prediction horizon of 14 days. For random forest and gradient boosting, we perform a 2-step tuning process. Step 1 includes tuning of time-series-specific hyperparameter, e.g., window length, number of lags. Step 2 includes tuning of the algorithm-specific hyperparameter. Here, hyperparameter tuning is performed for each prediction horizon individually and for all districts simultaneously.

### ARIMA tuning

We compare tuned ARIMA models to a simpler ARIMA(1,0,0) as a benchmark, to evaluate the contribution of the hyperparameter tuning procedure. Figure 5.1 and Table 5.1 summarize the performance differences between the investigated models. We denote the compared algorithms as *arima(1,0,0)* for the model prior tuning and *arima tuned* for the model after algorithm-specific hyperparameter tuning. The given performances are obtained by averaging the RMSE metric over all German districts and all prediction horizons.

The comparison results show prediction performance improvement in both datasets after the appropriate configuration of hyperparameters. This result implies that the selected hyperparameter configurations generalize well on previously unseen data. When comparing both datasets, we observe an overall average forecasting error increase in the validation dataset, compared to the tuning dataset. The observed differences are due to the different ICU bed occupancy in the selected time frames. Therefore, models comparison within each dataset should be regarded individually. On average across both datasets, hyperparameter tuning leads to a notable reduction of the RMSE by 0.224. The tuned models for each German district are selected as final models for our benchmark experiment and further denoted as *arima tuned*.

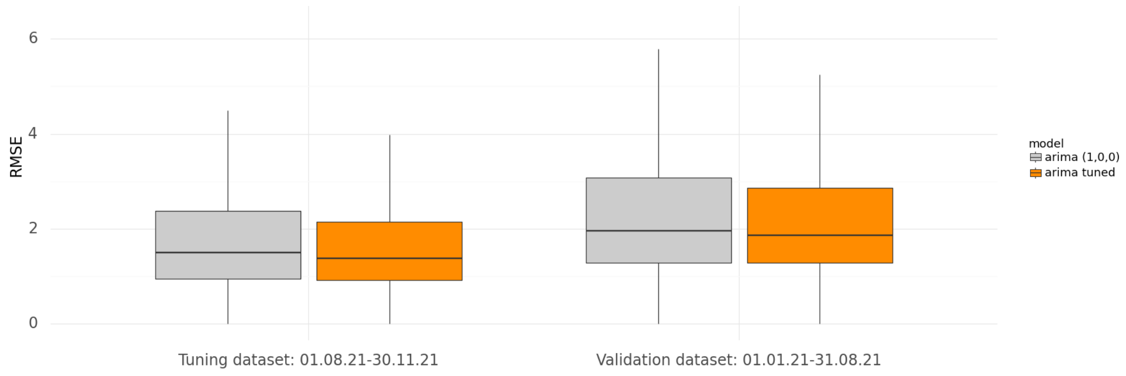


Figure 5.1: Boxplot of performance differences of ARIMA(1,0,0) (grey) against ARIMA with tuned hyperparameter configurations (orange), evaluated on different datasets. Performances are based on the average RMSE metric over all German districts and all prediction horizons.

| data | arima(1,0,0) | arima tuned |
|---|---|---|
| Tuning dataset | 2.023 | **1.810** |
| Validation dataset | 2.765 | **2.530** |

Table 5.1: Benchmark performance results of ARIMA(1,0,0) against ARIMA with tuned hyperparameter configurations, evaluated on different datasets. Performances are based on the average RMSE metric over all German districts and all prediction horizons.

### Random forest tuning

We compare the prediction performance of models with default hyperparameter configurations to the models after each step of the hyperparameter tuning procedure. Figure 5.2 and Table 5.2 summarize the performance differences between the investigated models for random forest. We denote the compared algorithms as *rf baseline* for the model prior tuning, *rf tuned step 1* for the model after time-series-specific hyperparameter tuning and *rf tuned step 2* after algorithm-specific hyperparameter tuning. The given performances are obtained by averaging the RMSE metric over all German districts and all prediction horizons. It is visible that both hyperparameter tuning steps lead to a small reduction of the forecasting error. Improvement is achieved for both datasets: the dataset, on which hyperparameter tuning is performed (left), as well as on the external validation dataset (right). This implies that the selected hyperparameter configurations generalize well on previously unseen data.

On average across both datasets, time-series-specific hyperparameter tuning leads to a reduction of the RMSE by 0.05. This result suggests that the forecasting performance of random forest is relatively insensitive to the window length and number of lagged features. Even smaller is the improvement achieved after the algorithm-specific hyperparameter tuning - RMSE reduction by 0.018. In total across both sets, the RMSE is merely reduced by 0.068 after both tuning steps. The small difference between default and tuned algorithm-specific hyperparameter of random forest is not very surprising as the results by Probst et al. (2019) suggest that random forest hyperparameters are less tunable compared to other machine learning algorithms. The tuned models for each prediction horizon are selected as final models for our benchmark experiment and further denoted as *random forest tuned*.

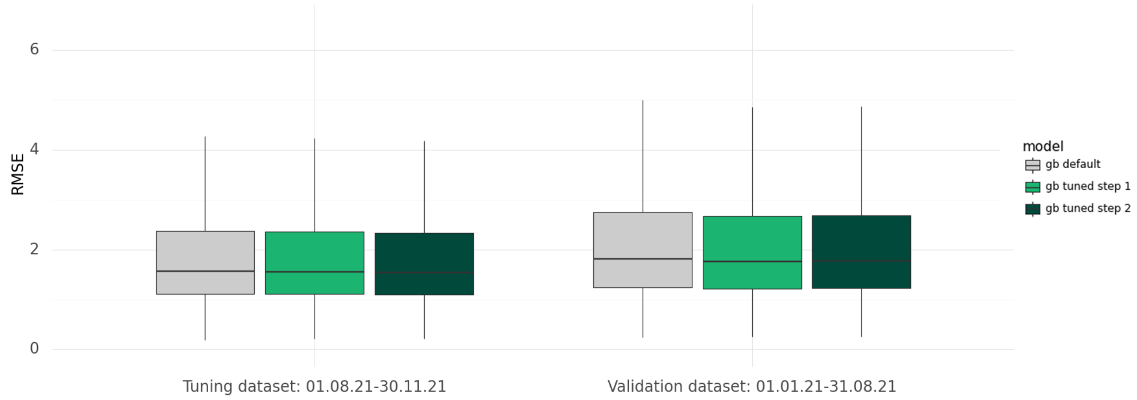


Figure 5.2: Boxplot of performance differences of random forest with default (grey) against tuned hyperparameter configurations, evaluated on different datasets. Tuning step 1 (green) includes time-series-specific hyperparameters. Tuning step 2 (dark green) includes algorithm-specific hyperparameters. Performances are based on the average RMSE metric over all German districts and all prediction horizons.

| data | random forest baseline | random forest tuned step 1 | random forest tuned step 2 |
|---|---|---|---|
| Tuning dataset | 2.116 | 2.051 | **2.041** |
| Validation dataset | 2.520 | 2.484 | **2.459** |

Table 5.2: Benchmark performance results of random forest with default against tuned hyperparameter configurations, evaluated on different datasets. Tuning step 1 includes time-series-specific hyperparameter. Tuning step 2 includes algorithm-specific hyperparameter. Performances are based on the average RMSE metric over all German districts and all prediction horizons.

## Gradient boosting tuning

Figure 5.3 and Table 5.3 summarize the performance differences between the investigated models for gradient boosting. We denote the compared algorithms as *gb baseline* for the model prior tuning, *gb tuned step 1* for the model after time-series-specific hyperparameter tuning and *gb tuned step 2* after algorithm-specific hyperparameter tuning. The given performances are based on the average RMSE metric over all German districts and all prediction horizon.

Similar to random forest, both tuning steps lead only to a small improvement in prediction performance. On average across both datasets, time-series-specific hyperparameter tuning reduces the RMSE by 0.044. In accordance with the results of random forest, it becomes visible that appropriate adjustment of window length and number of lags holds only a limited potential for prediction performance improvement. Further forecasting error reduction by 0.036 is achieved for the tuning dataset, by setting optimal algorithm-specific hyperparameters. However, the selected hyperparameter configuration fails to generalize on previously unseen data. Gradient boosting performs an extensive loss minimization, which can lead to noticeable overfitting. In terms of time series data, it is difficult to find a set of algorithm-specific hyperparameters, which generalize well across the full dataset of one year. The default gradient boosting hyperparameters generalize better on the provided data and are therefore kept in the final models. The best models for each prediction horizon after the tuning procedure are selected as final models and denoted as *gradient boosting tuned*.

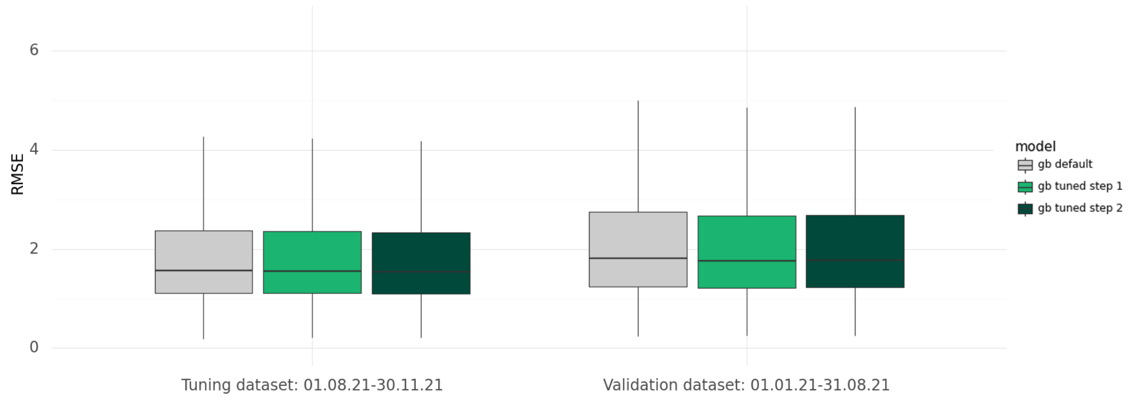


Figure 5.3: Boxplot of performance differences of gradient boosting with default (grey) against tuned hyperparameter configurations, evaluated on different datasets. Tuning step 1 (green) includes time-series-specific hyperparameters. Tuning step 2 (dark green) includes algorithm-specific hyperparameters. Performances are based on the average RMSE metric over all German districts and all prediction horizons.

| data | gradient boosting baseline | gradient boosting tuned step 1 | gradient boosting tuned step 2 |
|---|---|---|---|
| Tuning dataset | 2.209 | 2.166 | **2.130** |
| Validation dataset | 2.459 | **2.414** | 2.430 |

Table 5.3: Benchmark performance results of gradient boosting with default against tuned hyperparameter configurations, evaluated on different datasets. Tuning step 1 includes time-series-specific hyperparameter. Tuning step 2 includes algorithm-specific hyperparameter. Performances are based on the average RMSE metric over all German districts and all prediction horizons.

## 5.2 Benchmark Evaluation

### Prediction performance comparison

After setting optimal hyperparameter configurations for each investigated algorithm, the prediction performance of random forest and gradient boosting is benchmarked against ARIMA. Validation is based on the complete dataset for the year 2021. Training and prediction for gradient boosting and random forest are executed simultaneously for all German districts. For ARIMA, predictions are made for each district individually.

Figure 5.4 provides an overview of the algorithm performances. The illustrated boxplots depict the distribution of the RMSE metric over all German districts and all prediction horizons. Here, the RMSE metrics are not normalized. Therefore, higher RMSE values mostly signify districts with higher ICU bed capacities. Table 5.4 presents the summary learner performances via the mean and the median of the RMSE metric over all German districts and all prediction horizons. On average, all models perform comparably well with an RMSE mean of 2.277 for ARIMA, 2.277 for gradient boosting, and 2.276 for random forest. When comparing the RMSE median, the investigated machine learning models exhibit a slightly smaller prediction error in comparison to ARIMA. However, considering the small differences between the prediction performances of all models, we can conclude that, on average, all models perform comparably well.

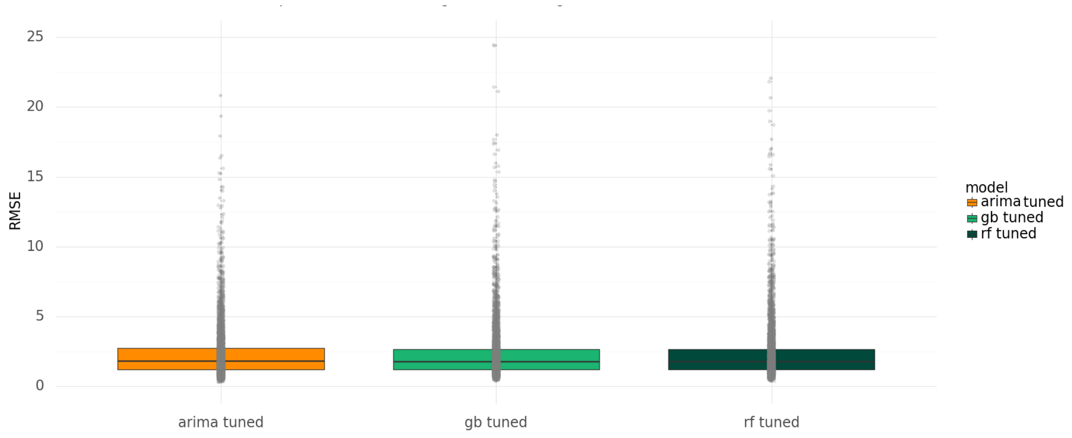


Figure 5.4: Boxplot of performance differences between ARIMA (orange), random forest (green) and gradient boosting (dark green) models. Performances are based on average learner performances, obtained by averaging the RMSE metric over all German districts and all prediction horizon.

| | arima tuned | gradient boosting tuned | random forest tuned |
|---|---|---|---|
| RMSE mean | 2.277 | 2.277 | 2.276 |
| RMSE median | 1.793 | 1.762 | 1.762 |

Table 5.4: Benchmark performance results of tuned ARIMA, random forest and gradient boosting models. Performances are based on summary algorithm performances via the mean and the median of the RMSE metric over all German districts and all prediction horizon.

### Prediction performance comparison with respect to the prediction horizon

A more detailed prediction performance summary for each prediction horizon is illustrated in Figure 5.5 and Table 5.5. Outlier values are excluded in the provided boxplot representation. Unsurprisingly, for all models, prediction performance decreases with a declining forecast time horizon. While 1-day ahead forecasts exhibit an average error of 1.152 RMSE, 14-day ahead prediction shows an average of 3.165 RMSE over all models. It is visible, that ARIMA exhibits the best prediction performance in the shorter term ($h = 1$ to $h = 3$). This result is not surprising, since ARIMA has proved to be an excellent short-term forecasting model for a wide variety of time series (O'Donovan, 1983). While ARIMA exhibits better prediction performance in the short-term prediction horizons, gradient boosting and random forest moderately overpower ARIMA in the more distant lookout periods. When comparing random forest with gradient boosting, it is visible that both models show very similar results in terms of their average RMSE over all German districts. Gradient boosting exhibits a slightly lower average prediction error in the furthest forecast horizon.

The increasing difference in prediction performance between ARIMA and the machine learning models with increasing horizons can be explained by the different multi-step horizon prediction strategies used in both approaches. The constructed machine learning models use a direct strategy for multi-step forecasting, where $h$ independent models are built for each prediction horizon. Since the direct strategy does not use any approximated values to compute the forecasts, it is not prone to any accumulation of errors. Meanwhile, in ARIMA each prediction is based on previous records. Therefore, this method can propagate the error committed in earlier forecasts to the future which might render the quality of long-term forecasts unreliable. Another reason for the lower prediction error of random forest and gradient boosting can be the potential benefiting by the additional information added from the exogenous features. Moreover, the selected machine learning models are known for their ability to uncover interactions between features, which could have a contribution to the exhibited prediction performance.

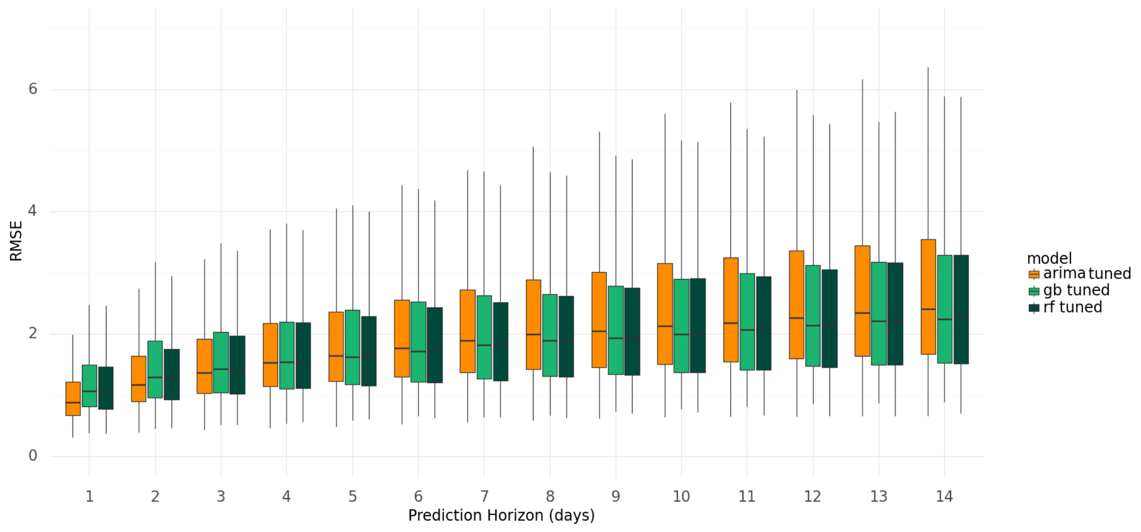


Figure 5.5: Boxplot of performance differences between ARIMA (orange), random forest (green) and gradient boosting (dark green) models for each prediction horizon. Performances are based on average learner performances, obtained by averaging the RMSE metric over all German districts.

| prediction horizon | arima tuned | gradient boosting tuned | random forest tuned |
|---|---|---|---|
| 1 | **1.017** | 1.242 | 1.196 |
| 2 | **1.357** | 1.523 | 1.458 |
| 3 | **1.602** | 1.702 | 1.655 |
| 4 | **1.802** | 1.842 | 1.817 |
| 5 | 1.982 | 1.999 | **1.968** |
| 6 | 2.142 | 2.119 | **2.096** |
| 7 | 2.291 | 2.279 | **2.216** |
| 8 | 2.432 | 2.365 | **2.363** |
| 9 | 2.568 | **2.466** | 2.498 |
| 10 | 2.694 | **2.589** | 2.642 |
| 11 | 2.819 | **2.709** | 2.787 |
| 12 | 2.940 | **2.867** | 2.942 |
| 13 | 3.059 | **3.035** | 3.051 |
| 14 | 3.176 | **3.138** | 3.181 |

Table 5.5: Benchmark performance results of tuned ARIMA, random forest and gradient boosting models for each prediction horizon. Performances are based on average learner performances, obtained by averaging the RMSE metric over all German districts.

**Prediction performance comparison across German districts**

The uneven spread of the disease across the country, as well as the uneven distribution of ICU bed occupancy and capacities, imply the need for accurate forecasts of ICU demand not only at a national but also at a regional level. Therefore, within the framework of the conducted benchmark experiment, we focus on producing ICU bed occupancy forecasts on district level. Producing forecasts on this spatial granularity supports decision-makers in healthcare and politics to make informed and data-driven decisions, such as resource management to allocate resources such as specialized nurses, physicians, or medical devices between districts. Moreover, patient transfer and admission policies can be adjusted in response to the occupancy forecasts.

The presented benchmark results above reflect the average prediction performance over all German districts. However, due to the varying dynamics of the pandemic, as well as the different ICU capacities and demand across different German regions, we expect prediction performance differences between regions and districts in Germany. To obtain a brief overview of prediction performances across German districts, we take a look into the geographical distribution of the RMSE metric over all evaluated districts. To enable comparison between regions, here we use a normalized RMSE metric. The RMSE metric is normalized by the total capacity (free and occupied) of ICU beds for each district. The exact calculation of the nRMSE metric is explained in Chapter 4.2.2.

Figure 5.6 depicts the geographical distribution of the nRMSE metric for ARIMA, random forest, and gradient boosting for a prediction horizon of $h = 7$ days. The presented nRMSE metric is based on the full data for the year 2021. This illustration allows a comparison between districts within one model, as well as a comparison between models. Light green signifies better prediction performance, while darker green implies higher forecasting errors. The grey districts (districts: Rhein-Pfalz-Kreis, Neustadt an der Waldnaab, Coburg, Fürth) are missing, due to unavailable ICU data. Generally, we can see that there are overall differences between districts. As an example, there is a visible smaller average forecasting error in some federal states, such as Mecklenburg-Vorpommern, Nordrhein-Westfalen, and Hessen, while other federal states exhibit multiple districts with rather higher forecasting error, such as Thüringen, Bayern and Baden Württemberg. This observation can be possibly explained by an overall higher relative occupancy in the states with higher error. However, a detailed analysis of the factors of the varying distribution of prediction performance across districts and federal states is not in the scope of this thesis. When comparing the geographical performance distribution between ARIMA, random forest, and gradient boosting, we do not observe a noticeable difference.

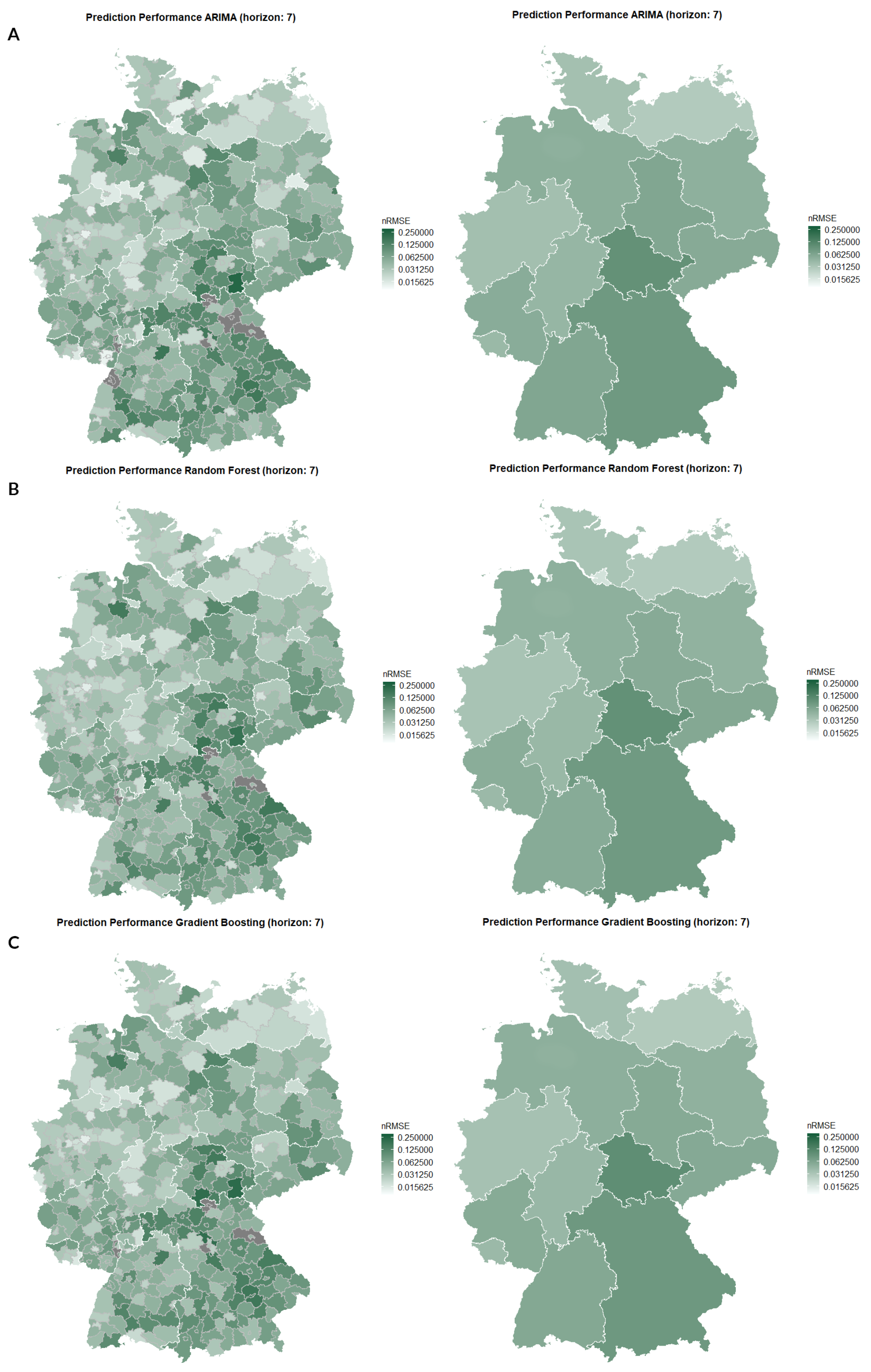

A
Prediction Performance ARIMA (horizon: 7)
Prediction Performance ARIMA (horizon: 7)
B
Prediction Performance Random Forest (horizon: 7)
Prediction Performance Random Forest (horizon: 7)
C
Prediction Performance Gradient Boosting (horizon: 7)
Prediction Performance Gradient Boosting (horizon: 7)
nRMSE
0.250000
0.125000
0.062500
0.031250
0.015625

Figure 5.6: Geographical distribution of prediction performance across German districts (left) and states (right) for A: ARIMA, B: random forest and C: gradient boosting. Example, based on prediction horizon of 7 steps. Performances, based on a normalized RMSE metric.

**Retrospective one-step forecasting incl. prediction intervals**

The benchmark experiment results show that the selected classical statistical approach, as well as the evaluated machine learning models, show comparable results in terms of prediction performance, measured via the RMSE metric. Random forest and gradient boosting moderately dominate ARIMA, especially in the distant lookout periods.

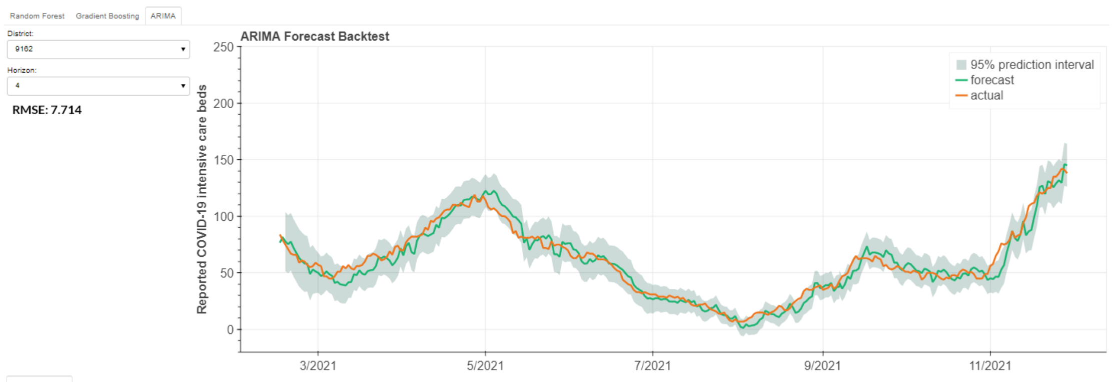
In time series forecasting, we are not only interested in a summary measure of average prediction performance, but also in the course and characteristics of the prediction curves, as well as the uncertainty associated with the prediction. Therefore, we have created an interactive visualization tool, which gives insights into the dynamics of the prediction curves. The interactive visualization tool is created via the interactive data visualization framework for Python `Bokeh` (Bokeh Development Team, 2018) and provides a comparison of model forecasts against the actual occupancy of intensive care capacity beds by COVID-19 patients for each German district and for each prediction horizon. Alongside point forecast, prediction intervals are included in the visualization. More information on the exact calculation of the presented prediction intervals can be found in Chapter 4.2.2. All prediction intervals are calculated based on the same analytical approach, which ensures comparability of prediction uncertainty across models.

The following examples in Figures 5.7, 5.8 and 5.9 illustrate the provided visualizations and are based on district 'SK München' for the lookout periods of $h = 4$, $h = 8$ and $h = 14$. The provided RMSE values are based on the RMSE metric, calculated on the total data for 2021. It is overall visible, how prediction accuracy declines and uncertainty increases with an increasing forecast horizon. In accordance with the results from the benchmark experiment, in district 'SK München' ARIMA exhibits better prediction performance in the short-term forecast periods, while random forest and gradient boosting show better results in further lookout periods. The gradient boosting algorithm exhibits the best performance for prediction horizons $h = 8$ and $h = 14$.

Generally, all models are successful in predicting the correct trend of the time series. However, with increasing forecast horizons, the course of the prediction curves seems to run a couple of steps behind for all models. A classical limitation of AR-models is their inability to identify radical shifts in trend since forecasts are determined only by the past behavior of the variable. The additional information from exogenous variables in the machine learning models and the ability of these models to incorporate the underlying structure and interactions in the data possibly have a contribution to the exhibited prediction performance. These aspects are investigated in section 5.3. Moreover, gradient boosting and random forest do not use any estimated values to forecast future values. In ARIMA, the recursive multi-step forecast strategy leads to even higher error rates after each step. This tendency is demonstrated in Figure 5.10, which illustrates an example of multi-step forecasting of 14 days into the future for each model. Once again, it can be observed how the prediction uncertainty in ARIMA grows bigger within the higher horizons, compared to gradient boosting and random forest.

*4-day forecast comparison*

A

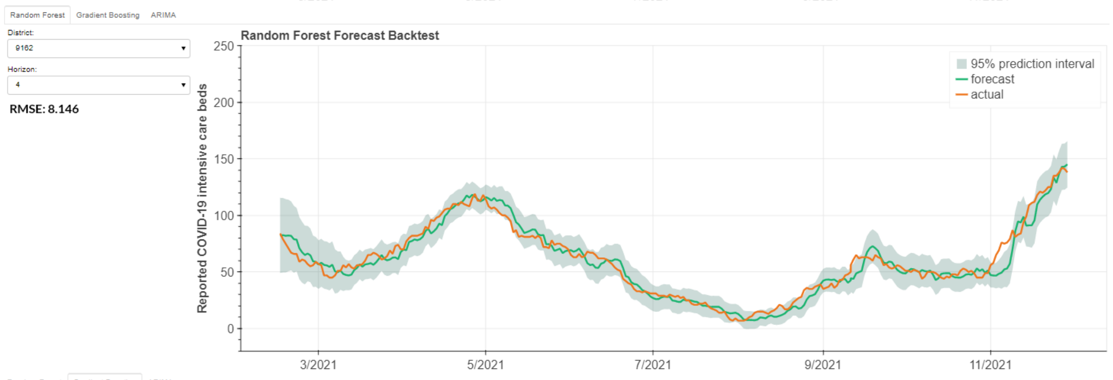


District: 9162

Horizon: 4

RMSE: 7.714

B

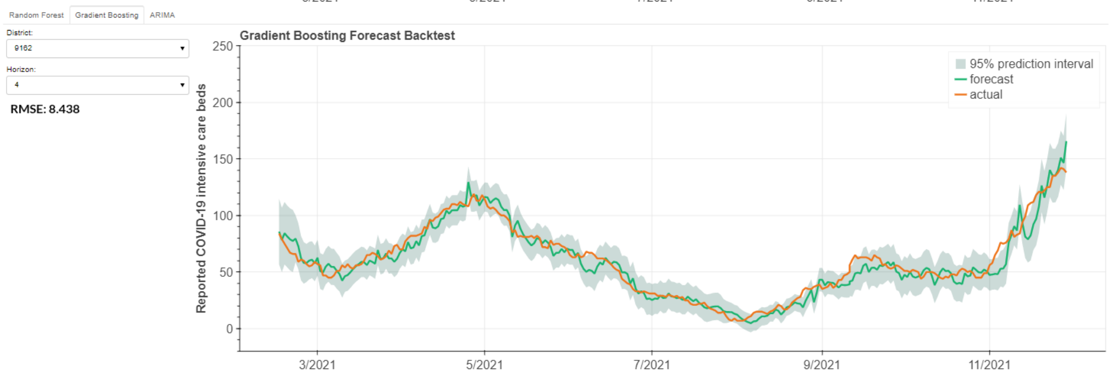


District: 9162

Horizon: 4

RMSE: 8.146

C



District: 9162

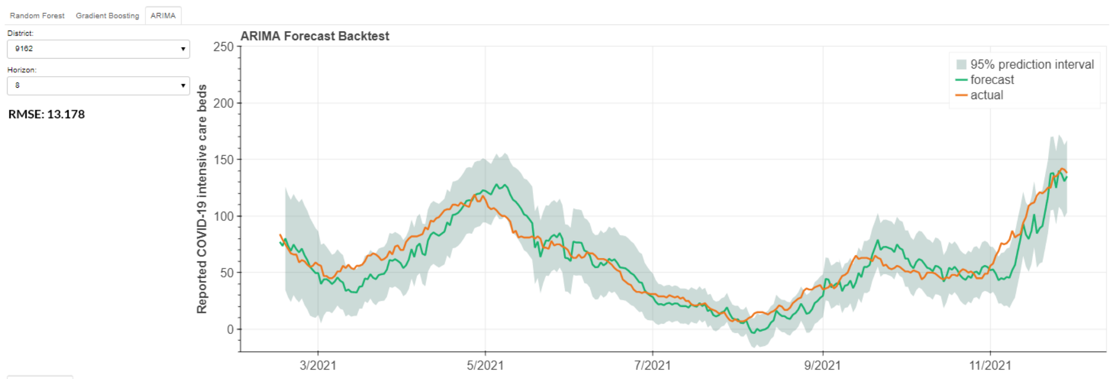
Horizon: 4

RMSE: 8.438

Figure 5.7: Retrospective comparison of observed (orange) and forecasted (green) reported intensive care beds, occupied by COVID-19 patients in 'SK München' over the course of year 2021 incl. 95% prediction interval. Example, based on prediction horizon of 4 days. A: ARIMA, B: random forest, C: gradient boosting.

*8-day forecast comparison*

A

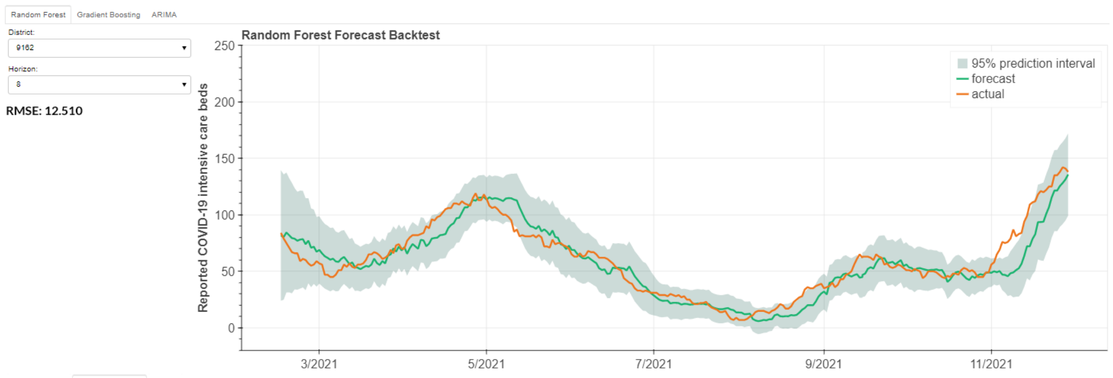


District: 9162

Horizon: 8

**RMSE: 13.178**

B

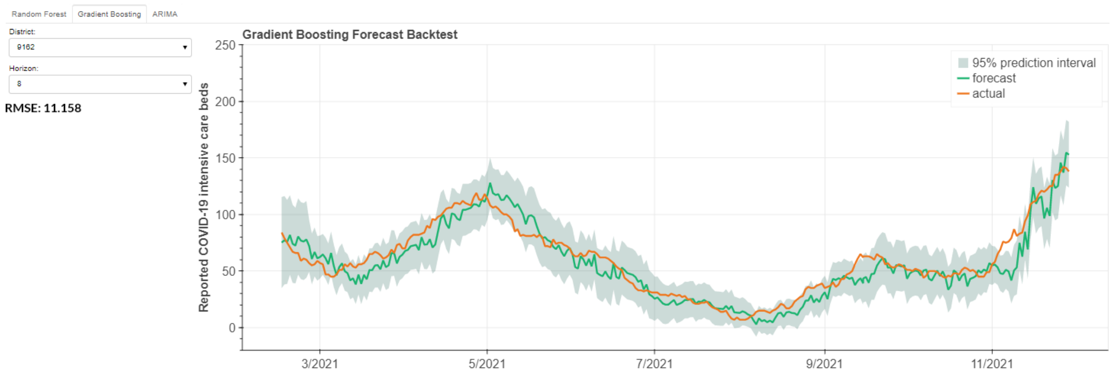


District: 9162

Horizon: 8

**RMSE: 12.510**

C



District: 9162

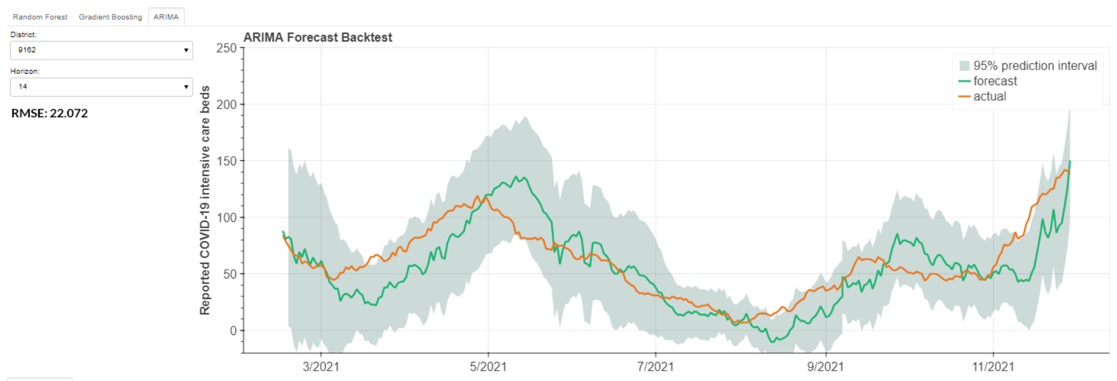
Horizon: 8

**RMSE: 11.158**

Figure 5.8: Retrospective comparison of observed (orange) and forecasted (green) reported intensive care beds, occupied by COVID-19 patients in 'SK München' over the course of year 2021 incl. 95% prediction interval. Example, based on prediction horizon of 8 days. A: ARIMA, B: random forest, C: gradient boosting.

*14-day forecast comparison*

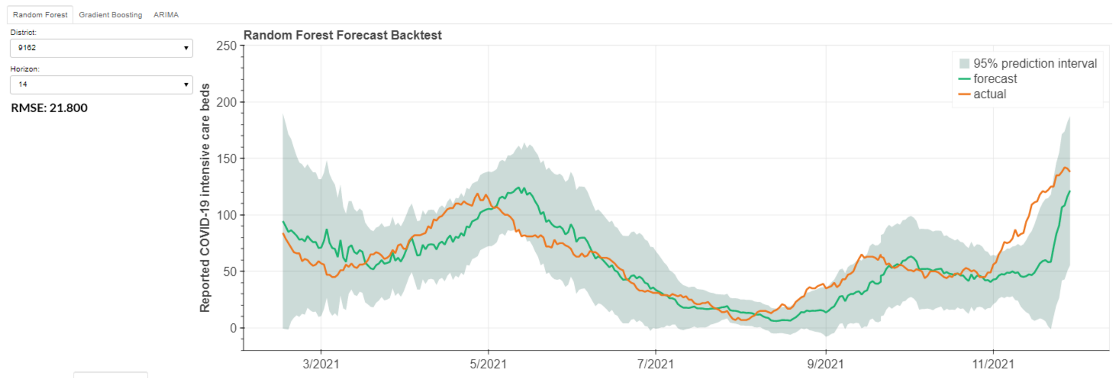
A


District: 9162

Horizon: 14

RMSE: 22.072

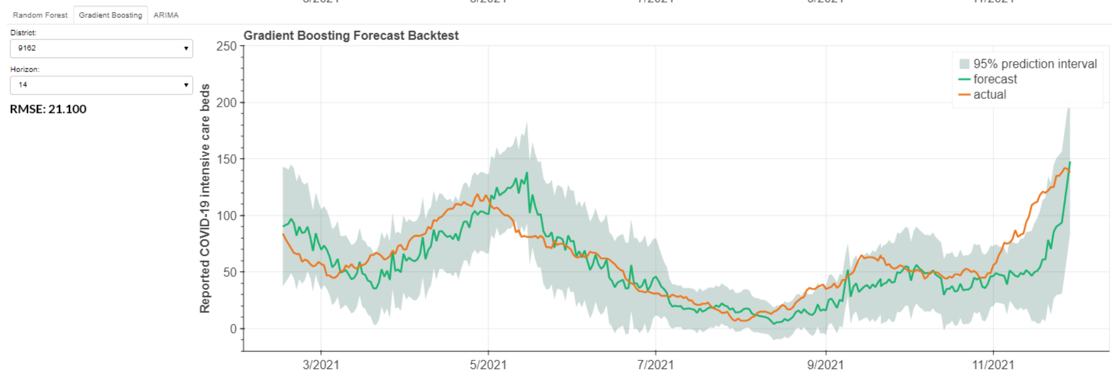
B


District: 9162

Horizon: 14

RMSE: 21.800

C


District: 9162

Horizon: 14

RMSE: 21.100

Figure 5.9: Retrospective comparison of observed (orange) and forecasted (green) reported intensive care beds, occupied by COVID-19 patients in 'SK München' over the course of year 2021 incl. 95% prediction interval. Example, based on prediction horizon of 14 days. A: ARIMA, B: random forest, C: gradient boosting.

## Multi-step forecasting

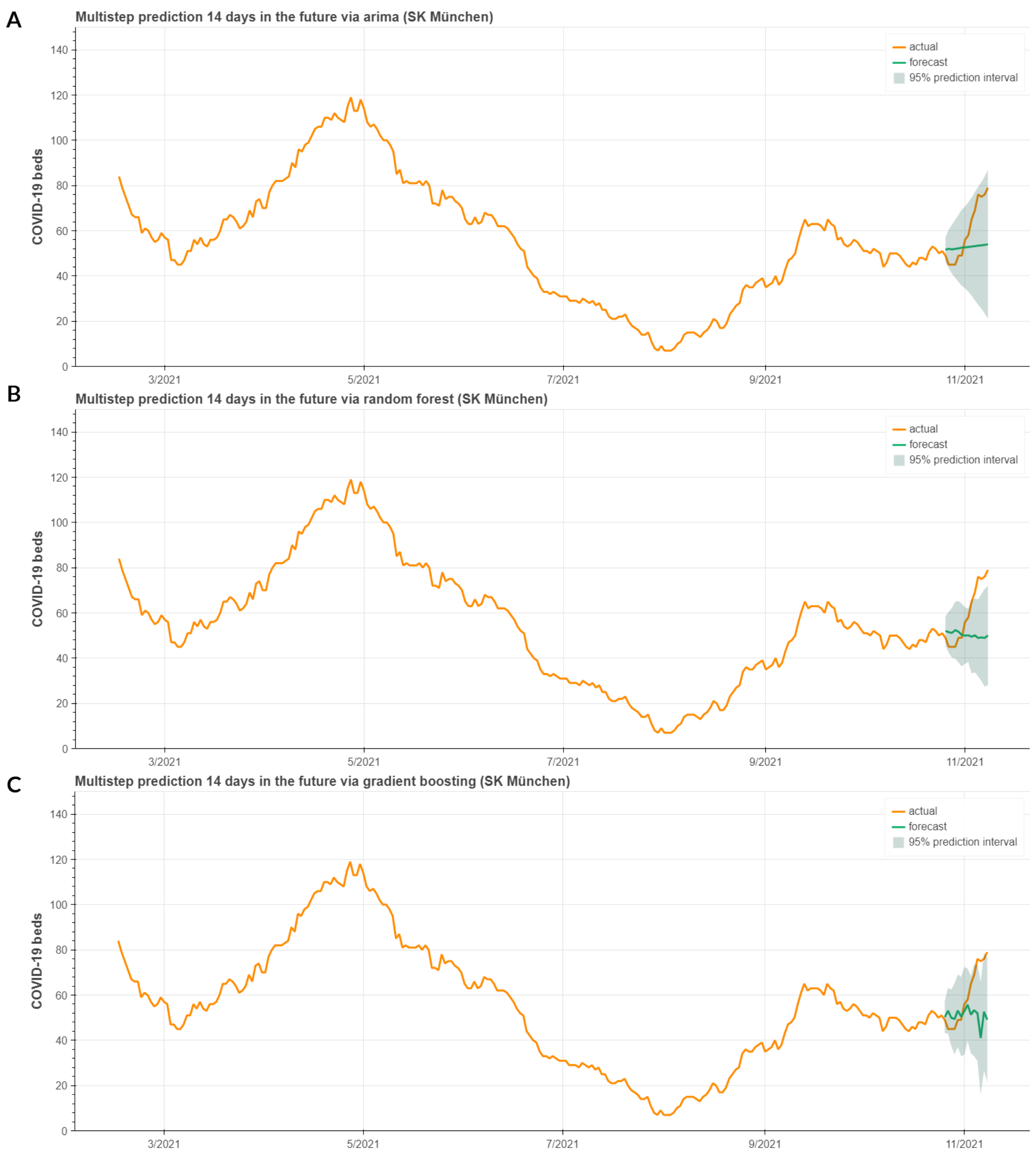


Figure 5.10: Example of multi-step prediction of A: ARIMA, B: random forest and C: gradient boosting for the next 14 days incl. 95% prediction interval. Example based on 'SK München'.

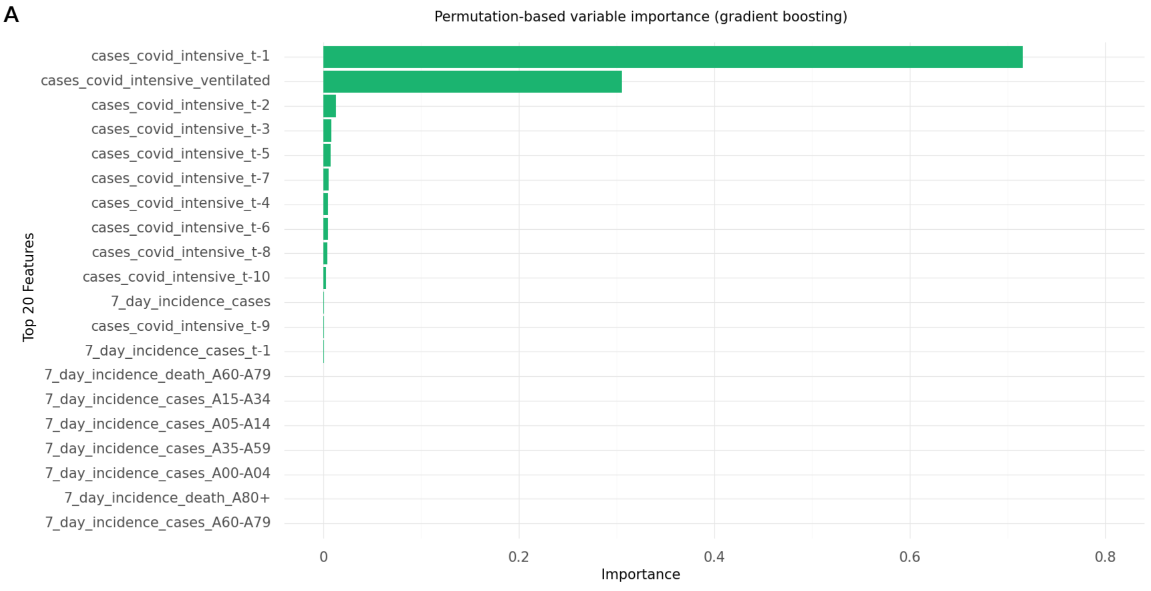
## 5.3 Variable Importance

The primary goal of the benchmark experiment is to measure and improve prediction accuracy for intensive care unit bed occupancy by COVID-19 patients. However, in the context of machine learning, we are often not only interested in accurate prediction, but in valuable insights into the complex process, e.g., understanding the relationship between features and target and quantifying their impact. While ARIMA is a purely autoregressive model, which explicitly relies on past observations of the series, machine learning models have shown to be successful in incorporating various exogenous variables into the modeling procedure. To exploit this advantage of machine learning models and better reflect the dynamic development of the pandemic situation, we have incorporated not only data on occupied ICU beds, but additional publicly available COVID-19 related data, such as data on the reported number of new cases and deaths in different age groups, as well as data on vaccination rates. We aim to investigate the contribution of additional key metrics of the pandemic. For that, different variable importance techniques are explored. Both algorithm implementations `skranger` for random forest and `XGBoost` for gradient boosting provide embedded variable importance metrics.

For both machine learning models, we measure variable importance via both metrics, permutation importance, and impurity-based importance. As already discussed in Chapter 2.3.6, different metrics can not be rated and should be understood as mutual complements that enable interpretability from different angles. Figure 5.11 illustrates the measured feature importance by gradient boosting via both measurement methods. The provided metrics reflect an average variable importance measure across all German districts and forecast horizon. Variable importances are obtained by averaging variable importance metrics over the walk-forward validation iterations and are based on the complete dataset for the year 2021, which results in 365 iterations. Due to the direct strategy for multi-step prediction of the machine learning models, here we deal with 14 individual models, one for each prediction horizon. Therefore, each of the 14 models has an individual optimal number of lagged variables of the target. For consistency across all forecast horizons, we have included 10 lags of the target in the dataset for each prediction horizon.

The provided figures illustrate the top 20 features, based on the corresponding metric. It is visible for both metrics, that the first lag of the target is by far the most important feature in the dataset, followed by the subsequent lagged features of the target. Further included exogenous features, such as incidence rates and vaccination coverage, have very little to no contribution to the models. An exception is the number of COVID-19 patients in intensive care units, receiving invasive ventilation treatment, which is a subset of the target, therefore strongly correlated to it. When interpreting the results, the possible shortcoming of the measurements should be taken into consideration. Since the endogenous features are strongly correlated, interpretation of variable importance outputs must be extremely cautious, to not overstate the meaningfulness of results.

A



B

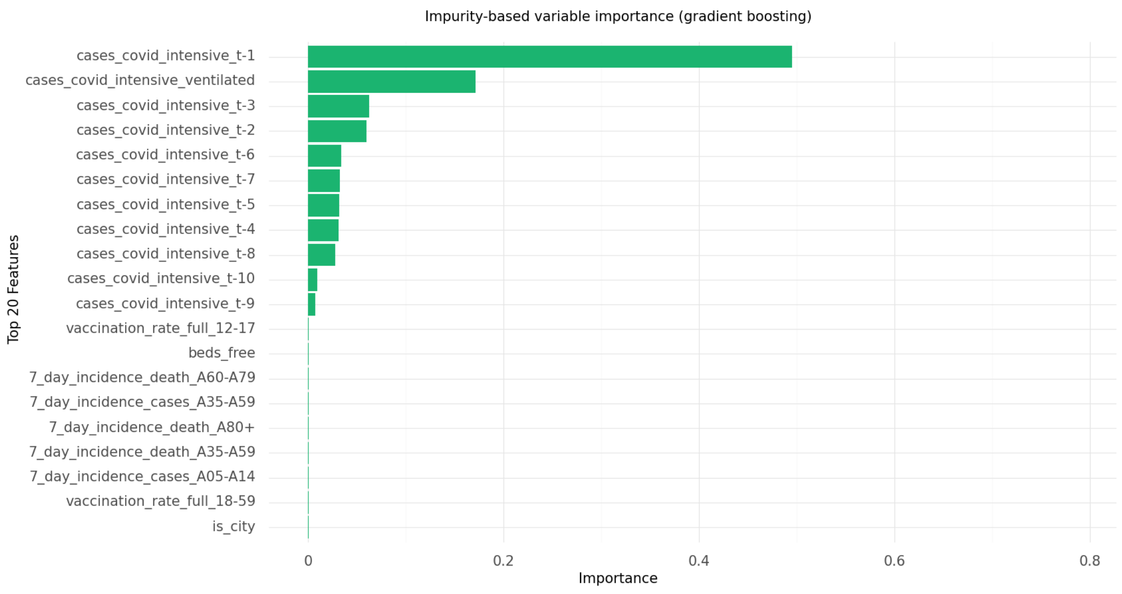


Figure 5.11: Top 20 features, based on the variable importance in the gradient boosting model, calculated via A: the permutation-based measure, B: the impurity-based measure. Average variable importance metrics across all German districts and prediction horizons, obtained by averaging over the walk-forward validation iterations on total data for 2021.

# 6. Discussion

The main focus of this thesis lies on investigating the potential of various machine learning methods for producing reliable forecasts on German intensive care units occupancy by COVID-19 patients and on providing a comparison with a classical statistical time series model as a benchmark. Moreover, to exploit the numerous advantages of machine learning models and to better reflect the complex dynamics of the pandemic situation over time, various exogenous variables are included in the modeling procedure. In this context, alongside prediction, the explanatory aspect of determining the importance of different exogenous variables for prediction accuracy is explored.

The results of the conducted benchmark experiment suggest that random forest and gradient boosting provide a powerful alternative to the classical statistical foresting model ARIMA. However, the overall observed differences in prediction performance between the investigated models are not substantial. Despite the moderately better predicting performance of the ML models, especially in the more distant prediction horizon, there are multiple aspects of these more elaborate methods that need to be considered. Random forest and gradient boosting show reliable prediction performance, but at the expense of longer times for training and hyperparameter tuning, as well as very high computational resources. Meanwhile, ARIMA offers more simplicity and rapidity of the forecasting procedure. These findings strongly encourage the use of classical methods, such as ARIMA as a baseline to more elaborate methods in order to justify their usage. However, in this context, it is worth mentioning, that compared to ARIMA, where models are built for each German district individually, random forest and gradient boosting enable training, tuning and forecasting for all German districts simultaneously, which has contributed to some computational time reduction.

As discussed in the results of the benchmark experiment, ARIMA exhibits the best prediction performance in the short-term. With increasing horizons, gradient boosting and random forest outperform ARIMA. As a possible reason for these results, we have discussed the different multi-step horizon prediction strategies used in both approaches. ARIMA uses a recursive multi-step prediction strategy, in which each prediction is based on previous records. Therefore, this method can propagate the error committed in earlier forecasts to the future which might render the quality of long-term forecasts unreliable. Meanwhile, the constructed machine learning models use a direct strategy for multi-step forecasting, where $h$ independent models are built for each prediction horizon. Since the direct strategy does not use any approximated values to compute the forecasts, it is not prone to any accumulation of errors. Notwithstanding, it has some weaknesses. First, since the $h$ models are learned independently inducing conditional independence of the $h$ forecasts, no statistical dependencies between the predictions of different steps are considered (Ben Taieb et al., 2012). Also, this strategy requires larger computational resources since there are as many models to learn as the size of the horizon.

As another reason for the lower prediction error of random forest and gradient boosting, we have discussed the potential benefit of additional information included from the exogenous features. Random forest and gradient boosting are known to generally work well with high-dimensional problems and can identify strong predictors of a specified outcome without making assumptions about an underlying model (see Breiman, 2001; Bühlmann and Yu, 2003). However, based on the results from the variable importance measures, the first lag of the target has shown to be by far the most important feature in the dataset, followed by the subsequent lagged features of the target. Further incorporated exogenous features have shown very little to no contribution to the models. Nonetheless, since the endogenous features are strongly correlated, the applied variable importance methods can be strongly misleading. In the literature there are some suggestions on how to deal with collinearity with respect to feature importance. One suggestion is to perform permutation feature importance measurement by permuting correlated features together instead of individually (Parr et al., 2018). Other alternative measures are based on the Conditional Feature Importance (Strobl et al., 2018), which focus on the idea of permuting new values of a feature by taking the distribution conditional on the remaining features into consideration. Applying these newer metrics and comparing them to their classical counterparts may provide better insights into the relationship between features and target.

The wide range of exogenous variables added in the models has the potential to enhance prediction performance but could also lead to overfitting. Variable importance measurements can be used for improving prediction performance by identifying and eliminating redundant or noise variables. One commonly used feature selection method is the recursive feature elimination (RFE) method (Guyon et al., 2002). Recursive feature elimination is a backward selection of the predictors. This technique begins by building a model on the entire set of predictors and computing an importance score for each predictor. The least important predictor(s) is then removed, the model is re-built, and importance scores are computed again. Iteratively, the least important feature is removed, until an optimal subset of features is achieved. Alongside overcoming problems, such as overfitting, reducing the dimensionality supports reducing computational times of the models.

ARIMA modeling in this experiment is conducted for each district individually. Meanwhile, training, hyperparameter tuning, and prediction for random forest and gradient boosting are performed for all German districts simultaneously. This procedure can have many benefits, including improved prediction accuracy by adding more data and modeling interactions between districts, increased data efficiency and reduced training times. However, the development and dynamics of the pandemic, as well as the capacities and demand for ICU beds can strongly differ across different German districts. Therefore, considering all districts as a joint task can have a negative impact on prediction performance. Investigating the potential of building individual models for each district, can be an area for improving the forecasting accuracy of the ML models.

We see a further limitation of the conducted experiment related to hyperparameter tuning. We have resorted to standard techniques hyperparameter tuning, such as random search. The random search method oversteps some disadvantages of grid search, such as high time resources, but has a major disadvantage with its inability to converge to the global optimum (Andradóttir, 2015). The randomly selected hyperparameter combinations cannot guarantee a steady and competitive result. In the conducted experiment, only little improvement was achieved by the appropriate hyperparameter setting. Moreover, for gradient boosting, the selected hyperparameter configurations have failed to generalize on previously unseen data. Instead of resorting to standard techniques like grid search, random search or manual tuning, more advanced methods, such as bayesian optimization (BO) offer a useful alternative in which information from previous tuning iterations is considered to quickly find optimal ranges of parameter settings (Snoek et al., 2012). Bayesian optimization seeks to balance the exploration of well-known optimal configurations with the exploitation of high-uncertainty regions in the parameter space. There is a lot of evidence in the literature, that BO is superior to RS for machine learning hyperparameter tuning (Turner et al., 2021). Furthermore, to reduce computational complexity, we have based hyperparameter tuning on data for the time span of approximately four months. In this regard, it is advisable to experiment with the data volume to find an appropriate trade-off between computational efficiency and the accuracy of the model.

Another limitation is the applied method for prediction intervals calculation. Our original aim has been to calculate prediction intervals for random forest and gradient boosting via the quantile regression approach (see Meinshausen, 2006, Kriegler and Berk, 2007). Due to implementation difficulties, we have limited the calculation to an analytical approach, approximating prediction intervals via the RMSE metric. Moreover, to enable comparison between models, we have applied the same approach to ARIMA, regardless it's well-established build-in method. Therefore, further examination of uncertainty quantification for random forest and gradient boosting, including existing technical implementations, is required.

Within the scope of this project, we investigate only two machine learning approaches - random forest and gradient boosting. The conducted benchmark experiment can be extended to the use of further machine learning and deep learning methods. Despite being relatively new, the field of deep learning has attracted a lot of interest in the past few years. Current research of time series forecasting sets a strong emphasis on the successful prediction performance and robustness of artificial neural network models (Gamboa, 2017).

Another possible extension of the benchmark experiment would be to conduct further examination into the performance differences across different districts. By visualizing the geographical distribution of the normalized RMSE metric across districts and federal states, we have seen how strong the metric can differ. A detailed analysis of the factors for these differences can be valuable for improving the forecasting accuracy of the models.

# 7. Conclusion

Containing and mitigating the spread and infection rate of the Coronavirus is essential. Moreover, strengthening the capacity of health systems to respond swiftly and effectively is inevitable. The daily updated recording of ICU beds, occupied by COVID-19 patients, together with further key metrics of the pandemic, such as the hospitalization rate and reporting incidence, enable a regionally and temporally resolved real-time analysis of the current and expected situation. Reliable forecasts of key pandemic indicators enable timely data-driven decision-making, which is crucial for healthcare and politics.

The conducted project pursues the goal of evaluating different machine learning approaches for forecasting occupancy of COVID-19 patients in German ICUs and comparing them to a classical statistical forecasting approach. This investigation is done by conducting a benchmark experiment, which compares prediction performance of random forest and gradient boosting to ARIMA. Within the framework of this thesis, forecasting is based on a time horizon for up to two weeks. Additionally, to reflect the uneven spread of the disease across the country, forecasting is conducted on German district level. Alongside comparison of the forecasting accuracy, we examine the contribution of exogenous variables - further key measures of the pandemic.

Three main findings can be highlighted. On average across all districts and prediction horizons, all three models perform comparably well. Based on this finding, it is strongly recommended to use classical statistical methods as a baseline to more elaborate machine learning methods in order to justify their usage. Secondly, ARIMA exhibits the best prediction performance in the short-term, while random forest and gradient boosting moderately outperform ARIMA in the more distant periods. These results suggest that random forest and gradient boosting provide a powerful alternative to ARIMA for medium-term forecasting. Finally, based on the variable importance measurements, it can be concluded that further exogenous features have no substantial contribution to a prediction performance improvement. The first lag of the target has shown to be by far the most important feature in the dataset, followed by the subsequent lagged features of the target. However, this conclusion is limited, due to autocorrelation between observations of the target.

# A. Appendix

**To Chapter 3:**
**Correlation matrix heatmap**

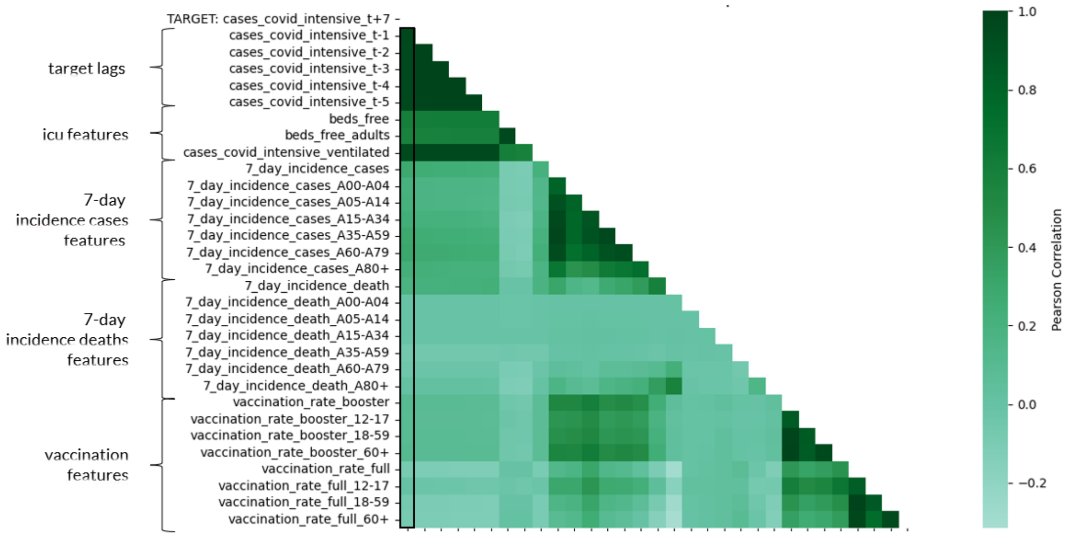


Figure A.1: Correlation coefficient heatmap of final data features, based on data for 2021. The marked area represents the association of each feature with the target - number of COVID-19 cases in intensive care units one week ahead (prediction horizon of 7 days).

Figure A.1 represents the correlation matrix between all variables of the final data set, based on the Pearson correlation coefficient. The example is based on the target variable - number of COVID-19 cases in intensive care units 7 days in the future. Correlation coefficients are calculated for all districts simultaneously on the total data set for year 2021. While +1.0 (dark green) indicates a perfect positive correlation, 1.0 (light green) signifies a perfect inverse correlation and 0 means no correlation. Here we have included only 5 lags of the target for illustration purposes.

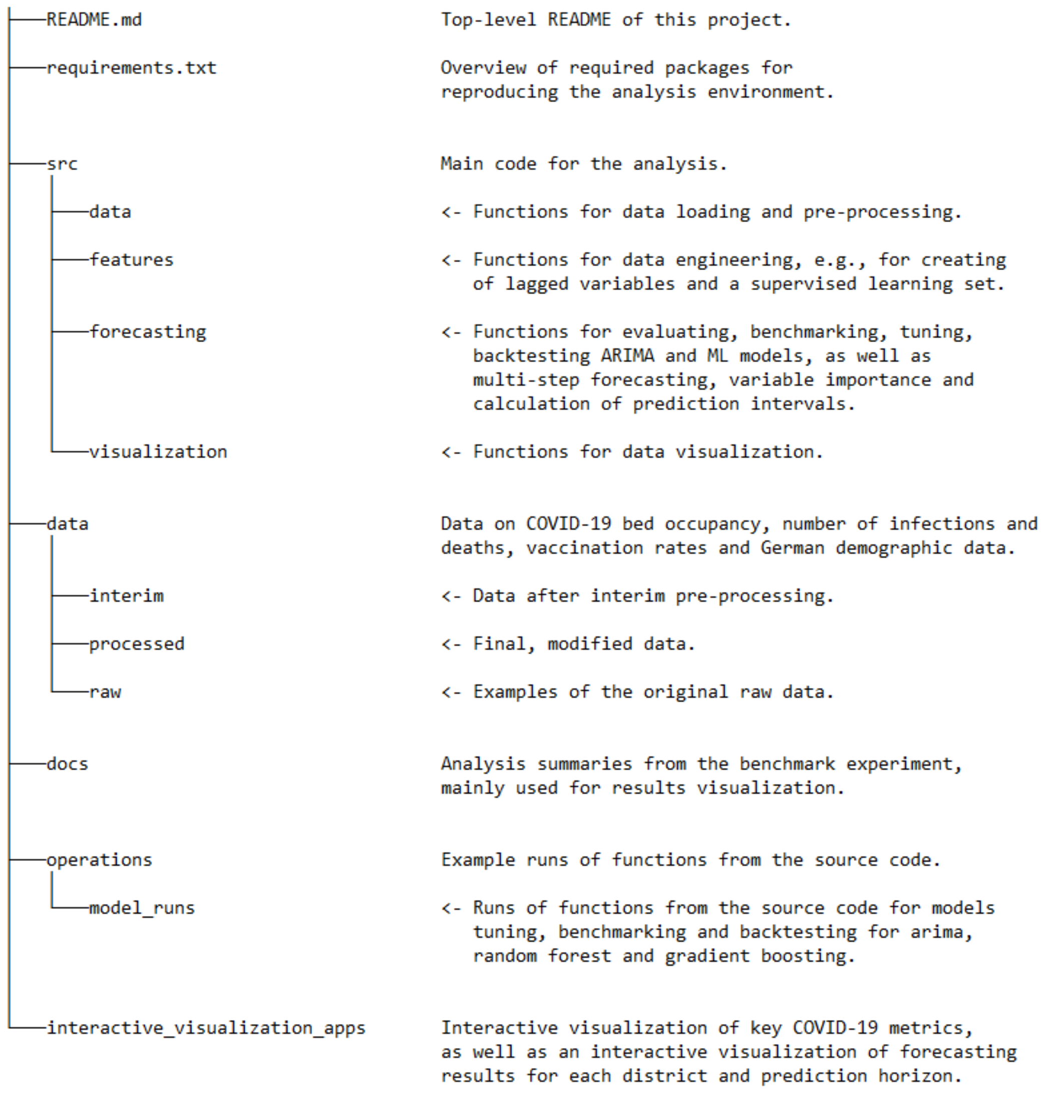
# B. Electronic Appendix

**Contents**

- An electronic form of this thesis.
- Repository containing the `Python`-Code of this project:
  https://gitlab.com/master-thesis-stat/intensive-care-capacity-forecasting

**Python Project Structure**

- `Data` folder: Contains examples of the raw data, as well as pre-processed interim and final data, ready for modeling.
- `SRC` folder: Main code for the analysis: functions for data loading and pre-processing, functions for data engineering, e.g., for creation of lagged variables and a supervised learning set, functions for evaluating, benchmarking, tuning, backtesting ARIMA and ML models, as well as multi-step forecasting, variable importance and calculation of prediction intervals, functions for data visualization.
- `Operations` folder: Concrete example runs of the source code, e.g. executed hyper-parameter tuning, benchmarking and backtesting for each model.
- `Docs` folder: Contains output summaries from the analysis, mainly used for results visualization.
- `Interactive Visualization Apps` folder: Contains the following interactive visualization tool:
  - Interactive visualization of ICU bed capacities and COVID-19 occupancy for each German district.
  - Interactive visualization of reported COVID-19 infections and deaths in total and per age groups for each German district.
  - Interactive visualization of vaccination rates in total and per age group for each German district.
  - Interactive visualization of the forecasting results. Retrospective comparison of predicted and observed values of best arima, random forest, gradient boosting models, incl. prediction intervals, for each district and each prediction horizon.



```
├───README.md                       Top-level README of this project.
│
├───requirements.txt                Overview of required packages for
│                                   reproducing the analysis environment.
│
├───src                             Main code for the analysis.
│   │
│   ├───data                        <- Functions for data loading and pre-processing.
│   │
│   ├───features                    <- Functions for data engineering, e.g., for creating
│   │                                  of lagged variables and a supervised learning set.
│   │
│   ├───forecasting                 <- Functions for evaluating, benchmarking, tuning,
│   │                                  backtesting ARIMA and ML models, as well as
│   │                                  multi-step forecasting, variable importance and
│   │                                  calculation of prediction intervals.
│   │
│   └───visualization               <- Functions for data visualization.
│
├───data                            Data on COVID-19 bed occupancy, number of infections and
│   │                               deaths, vaccination rates and German demographic data.
│   │
│   ├───interim                     <- Data after interim pre-processing.
│   │
│   ├───processed                   <- Final, modified data.
│   │
│   └───raw                         <- Examples of the original raw data.
│
├───docs                            Analysis summaries from the benchmark experiment,
│                                   mainly used for results visualization.
│
├───operations                      Example runs of functions from the source code.
│   │
│   └───model_runs                  <- Runs of functions from the source code for models
│                                      tuning, benchmarking and backtesting for arima,
│                                      random forest and gradient boosting.
│
└───interactive_visualization_apps  Interactive visualization of key COVID-19 metrics,
                                    as well as an interactive visualization of forecasting
                                    results for each district and prediction horizon.
```

Figure B.1: Overview of the Python project directory structure.

# Bibliography

Ahmed, N.and Atiya, A., Gayar, N., and El-Shishiny, H. (2010). An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29(5-6):594–621.

Andradóttir, S. (2015). A review of random search methods. In *Handbook of Simulation Optimization. International Series in Operations Research  Management Science.*, volume 216, pages 277–292. Springer New York, New York, NY.

Archer, K. J. and Kimes, R. V. (2008). Empirical characterization of random forest variable importance measures. *Computational Statistics and Data Analysis*, 52(4):2249–2260.

Ben Taieb, S., Bontempi, G., Atiya, A. F., and Sorjamaa, A. (2012). A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition. *Expert Systems with Applications*, 39(8):7067–7083.

Bergstra, J. and Bengio, J. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305.

Bischl, B., Mersmann, O., Trautmann, H., and Weihs, C. (2012). Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary computation*, 20(2):249–75.

Bischl, B., Schiffner, J., and Weihs, C. (2013). Benchmarking local classification methods. *Computational Statistics*, 28(6):2599–2619.

Bokeh Development Team (2018). *Bokeh: Python library for interactive visualization.*

Boulesteix, A. and Schmid, M. (2014). Machine learning versus statistical modeling. *Biometrical journal*, 56(4):588–593.

Box, G. E. P. and Jenkins, G. M. (1970). *Time-Series Analysis, Forecasting and Control.* Holden-Day, San Francisco.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J., Olschen, R., and C.J., S. (1984). *Classification and regression trees.* Monterey, CA.

Brockwell, P. J. and Davis, R. A. (2002). *Introduction to time series and forecasting.* Springer, New York.

Bühlmann, P. (2004). Bagging, boosting and ensemble methods. Papers 2004,31, Berlin.

Bühlmann, P. and Yu, B. (2003). Boosting with the l2 loss. *Journal of the American Statistical Association*, 98(462):324–339.

Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd international conference on Machine learning - ICML '06*, 148:161–168.

Cerqueira, V., Torgo, L., and Soares, C. (2019). Machine learning vs statistical methods for time series forecasting: Size matters. *arXiv:stat.ML/1909.13316.*

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Dickey, D. A. and Fuller, W. A. (1979). Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American statistical association*, 74(366a):427–431.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics.*, 29(5):1189–1232.

Gamboa, J. (2017). Deep learning for time-series analysis. *CoRR*, abs/1701.01887.

Gautam, A. and Singh, V. (2020). Parametric versus non-parametric time series forecasting methods: A review. *Journal of Engineering Science and Technology Review*, 13(3):165–171.

Ghawi, J. and Pfeffer, J. (2019). Efficient hyperparameter tuning with grid search for text categorization using knn approach with bm25 similarity. *Open Computer Science*, 9(1):160 – 180.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. 46(1-3):389–422.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction.* New York, Springer, 2nd edition.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science and Engineering*, 9(3):90–95.

Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: Principles and Practice.* OTexts, Australia, 2nd edition.

Ing, C. (2003). Multistep prediction in autoregressive processes. *Econometric Theory*, 19(2):254–279.

Inoue, A., Jin, L., and Rossi, B. (2017). Rolling window selection for out-of-sample forecasting with time-varying parameters. *Journal of Econometrics*, 196(1):55–67.

Kriegler, B. and Berk, R. (2007). Boosting the quantile distribution: A cost-sensitive statistical learning procedure. *Department of Statistics, UCLA*.

Meinshausen, N. (2006). Quantile regression forests. *Journal of Machine Learning Research*, 7(35):983–999.

Mentch, L. and Hooker, G. (2016). Quantifying uncertainty in random forests via confidence intervals and hypothesis tests. *J. Mach. Learn. Res.*, 17(1):841–881.

Nembrini, S., König, I. R., and Wright, M. N. (2018). The revival of the gini importance? *Bioinformatics*, 34(21):3711–3718.

O'Donovan, T. M. (1983). *Short term forecasting: An introduction to the Box-Jenkins approach.* Wiley, Chichester.

Parr, T., Turgutlu, K., Csiszar, C., and Howard, J. (2018). Beware default random forest importances. `https://explained.ai/rf-importance/index.html`.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Plotly Technologies Inc. (2015). *Collaborative data science.* Plotly Technologies Inc., Montreal, QC.

Probst, P., Bischl, B., and Boulesteix, A. (2019). Tunabiliy: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(1):1934–1965.

Roberts, D. R., Bahn, V., Ciuti, S., Boyce, M. S., Elith, J., Guillera-Arroita, G., Hauenstein, S., Lahoz-Monfort, J. J., Schröder, B., Thuiller, W., Warton, D. I., Wintle, B. A., Hartig, F., and Dormann, C. F. (2017). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography*, 40(8):913–929.

Rokach, L. (2016). Decision forest: Twenty years of research. *Information Fusion*, 27:111–125.

Scornet, E. (2020). Trees, forests, and impurity-based variable importance. *arXiv:2001.04295*.

Seabold, S. and Perktold, J. (2010). statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*.

Segal, M. (2004). Machine learning benchmarks and random forest regression. *UCSF: Center for Bioinformatics and Molecular Biostatistics*.

Shumway, R. H. and Stoffer, D. S. (2006). *Time series analysis and its applications: With R examples.* Springer, New York.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *arXiv:1206.2944*.

Strobl, C., Boulesteix, A., Kneib, T., Augustin, T., and Zeileis, A. (2018). Conditional variable importance for random forests. *BMC Bioinformatics*, 9(307):1471–2105.

Tashman, L. J. (2000). Out-of-sample tests of forecasting accuracy: an analysis and review. *International Journal of Forecasting*, 16(4):437–450.

Turner, R., David Eriksson, D., McCourt, M., Kiili, J., Laaksonen, E., Xu, Z., and Guyon, I. (2021). Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. *arXiv: 2104.10201*.

Weerts, H., Mueller, A. C., and Vanschoren, J. (2020). Importance of tuning hyperparameters of machine learning algorithms. *arXiv: 2007.07588*.

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.

Wright, M., Dankowski, T., and Ziegler, A. (2016a). Unbiased split variable selection for random survival forests using maximally selected rank statistics. *Statistics in Medicine*, 36(8):1271–1284.

Wright, M. and Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in c++ and r. *Journal of Statistical Software*, 77(1):1–17.

Wright, M. N., Ziegler, A., and König, I. R. (2016b). Do little interactions get lost in dark random forests? *BMC bioinformatics*, 17(1):1–10.

Zhang, G., Eddy Patuwo, B., and Y. Hu, M. (1998). Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting*, 14(1):35–62.

Zhang, H., Zimmerman, J., Nettleton, D., and Nordman, D. J. (2020). Random forest prediction intervals. *The American Statistician*, 74(4):392–406.

# Statement of authorship

I declare that I completed this master thesis on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this nor a similar work has been presented to an examination committee.

Munich, March 1, 2022 . . . . . . . . . . . . . . . . . . . . . . . . . . . .