LUDWIG-MAXIMILIANS-UNIVERSITY MUNICH

FACULTY FOR MATHEMATICS, COMPUTER SCIENCE AND STATISTICS

BACHELOR’S THESIS

COMPARISON BETWEEN TWO PROGRAMMING LANGUAGES R AND PYTHON

Author:

Thuy Ngoc Phuong TA

Supervisor:

Prof. Dr. Christian HEUMANN

22nd July 2020

Abstract

This thesis is a dedicated comparison between two popular programming languages: R and Python. An experiment will be conducted in order to provide the most unbiased substantive information. This experiment is split into quantitative and qualitative analysis. The quantifiable experiment will be performed on two projects, with two different sizes of dataset. The objective of the two projects is to apply multiple traditional data analysis tasks, with respect to the following criteria: processing time, code length, code complexity, design of graphics and used packages, without drawing conclusions. The qualitative analysis will support the findings of the quantitative analysis. The conclusion will provide useful guidance for software beginners to choose the correct tools in field data analysis or data science as well as provide comparative information in data driven areas. A basic knowledge of programming is required to understand this thesis.

Keywords: R, Python, programming language, quantitative analysis, qualitative analysis

Content

List of Figures

List of Tables

Abbreviations

CRAN = Comprehensive R Archive Network

IDE = Integrated Development Environment

1. Introduction

Over the previous decades manufacturing systems, processes, and data are expanding and becoming more complex. Additionally, the demand for automated extraction of valuable knowledge from enormously huge amount of data has been dramatically increased. As a consequence, a large variety of automated analysis and discovery tools have been built [1].

Among these tools, R and Python are becoming the most important programming languages in analytics and data science. According to a recent poll from KDNuggets (KDNuggets annual software poll, 2014) given the fact that R (1st place) and Python (3rd place) were in the top 4 of dominant languages for analytics, data mining and data science [2]. Five years later, a flash survey conducted by Burtch Works, with more than one-thousand responses to assess the preferences for Python, R, and SAS, showed that Python with 41 percent of voters is now the most common programming language, while about 30 percent chose R as their preferred programming tool [3]. Furthermore, according to R Bloggers in 2019, Python and R are the most required data science software skills on Indeed.com with 27,374 jobs and around 13,000 jobs respectively (Figure 1).

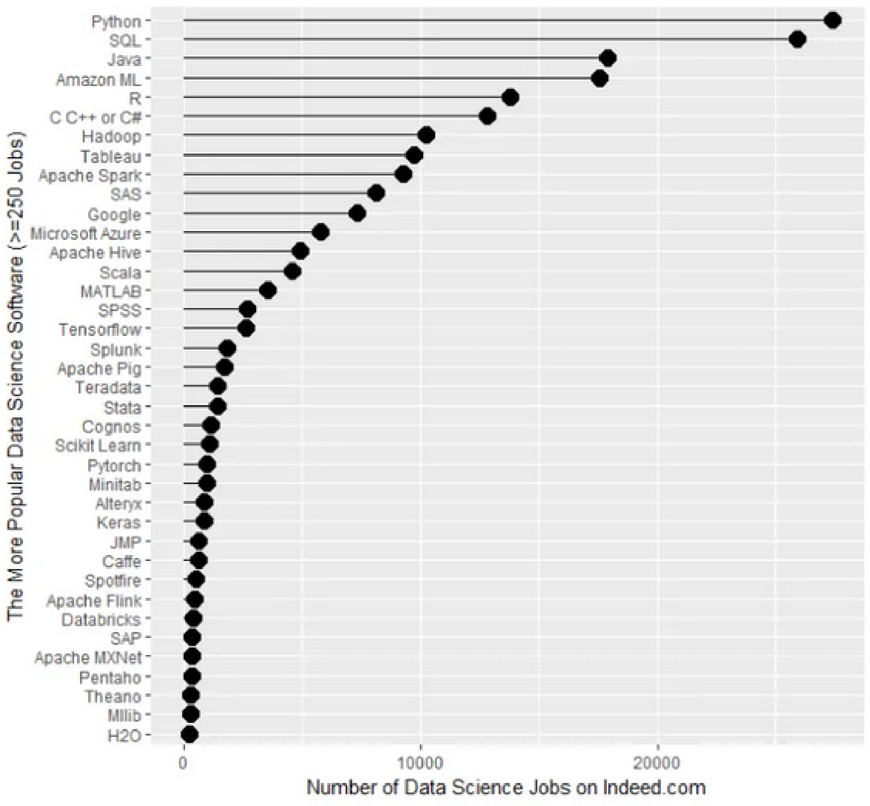


Figure 1. Job Postings for Popular Data Science Software in 2019 [20].

There have been thousands of online articles written comparing programming languages, there are, however, only a few academic research papers on the subject. For example, in 2018, Brittain and his colleagues published their paper about the comparison of R, Python, and SAS performance. Similarly, Ozgur et al. (2017) compared R, Python and Matlab in their research [5,6]. However, there has never been a dedicated research paper comparing only R and Python. Additionally, as described by Brittain et al. (2018), the articles written comparing programming languages ‘often include bias and the qualifying element is not measurable’ [5]. They are normally written from an overly subjective standpoint. Therefore, this thesis aims to look at these programming languages more objectively by walking through their structures. It shall also apply the languages on different projects with different sizes of dataset to compare their processing time, code complexity, design of graphics and the number of used packages. The research work of Brittain et al. (2018) was a major influence on how this thesis was written and the approach taken to compare these two tools. Nevertheless, the work of Brittain et al. (2018) did not write deeply enough about how the qualifiable elements were compared. This thesis will focus in more detail on the comparison between R and Python in order to provide an insight about these programming tools.

The remainder of this thesis is organized as follows. First, Section 2 provides an overview of the software metrics of the code complexity. Following this, Section 3 provides an overview of R and Python. This includes definitions, strengths, weaknesses, IDEs of these tools and how to install them. Next is the methodology used to compare their performances. In this Section, an experiment was conducted and separated into the quantifying and qualifying elements. The object of this experiment is to wrangle data, visualize data, build linear regression models, split the dataset into train and test data, and measure random forest prediction with respect to the following criteria: processing time, lines of codes, code complexity, design of graphics and used packages. The results drawn from this work can be found in Section 5. Finally, Section 6 sums up all the work and findings, as well as future work. The findings of this thesis will offer useful guidance for software beginners to choose their correct tools in field data analysis or data science, as well as provide comparative information in data driven areas.

2. Literature Review

Software quality is one of the most important factors that determine the development of a software driven industry. In software engineering software metrics are the only tools to control this quality. Therefore, there have been many different scales proposed for software metrics, such as Lines of code, McCabe's Cyclomatic complexity metrics or Hallstead complexity metrics. Tashtoush et al. (2014) stated that 'Cyclomatic complexity covers the control flow of the program, whereas Hallstead complexity measures the data flows' [17].

Cyclomatic Complexity[1], introduced by Thomas J. McCabe in 1976, was designed to 'indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a source code' [24]. Equation 1 gives the cyclomatic number v(G):

$$v(G) = e - n + p \tag{1}$$

where e is the number of edges in the control flow graph (CFG), n is the number of nodes in the CFG and p are separate components. Consider this example:

```
If (Condition 1)                   If (Condition 2)

Statement 1                        Statement 3

Else                               Else

Statement 2                        Statement 4
```

Cyclomatic Complexity for this program will be 8-7+2=3.

There is much discussion about McCabe's measurement. Many researchers believe that there is a high correlation between McCabe's CC and Lines of code (LOC) [16, 17]. According to Jay et al. (2019), 'LOC and CC have a stable practically perfect linear relationship that holds across programmers, languages, code paradigms (procedural versus object-oriented), and software processes' [16].

[1] R Documentation, „cyclocomp," [Online]. Available: https://www.rdocumentation.org/packages/cyclocomp/versions/1.1.0/topics/cyclocomp/. [Accessed June 2020].

One year later, Maurice Howard Halstead introduced his complexity measures – known as Halstead complexity metrics. According to Halstead, 'a computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operands' [15]. All Halstead's metrics are functions based on the number of operators and operands. Based on these parameters, the program length (N), program vocabulary (n), volume (V) and program difficulty (D) are computed as follows:

$$N = \textit{the total number of occurrences of the operators } (N_1) + \textit{the total number of occurrences of the operands } (N_2) \tag{2}$$

$$n = \textit{the number of unique operators } (n_1) + \textit{the number of unique operands } (n_2) \tag{3}$$

$$V = N * log_2 n \tag{4}$$

$$D = \frac{n_1}{2} * \frac{N_1}{n_2} \tag{5}$$

Additionally, programming effort (E) and programming time (T) can be computed differently using the following equations:

$$E = D * V \tag{6}$$

$$T = \frac{E}{S} \textit{ (seconds)} \tag{7}$$

Where S is the Stroud number[2], defined as 'the number of mental discriminations performed by the human mind per second' [15]. According to Hamer and Frewin (1982) this value for software scientists is set to 18 [5]. One critic pointed out about Halstead's complexity metrics that there is no strict rule which distinguishes between operators and operands [15]. In response, the following convention is used in this thesis to define operators and operands. Operators[3] are clarified as:

- break case continue default do else for go to if return size of switch while
- function call (Counts as one operator.)

[2] In 1967, psychologist John M. Stroud suggested that the human mind is capable of making a limited number of mental discrimination per second (Stroud Number), in the range of 5 to 20.

[3] R Documentation, "3.1.4 Operators," [Online]. Available: https://cran.r-project.org/doc/manuals/r-release/R-lang.html#Operators. [Accessed July 2020].

- {} () [] (Each pair counts as one operator.)
- >>= <<= += -= *= /= %= &= ^= |= >> << ++ -- -> && || <= >= == != ; , : = . & ! ~ - + * / % < > ^ | ?

And identifiers, numbers, characters ('x'), strings ("...") are defined as operands

For example:

```
If (k < 10)

{

        If (k > 4)

        x = x * k;

}
```

- Distinct operators: if () {} > < = *;

- Distinct operands: k 10 4 x.
- $n_1 = 8$
- $n_2 = 4$
- $N_1 = 10$
- $N_2 = 7$
- $D = \frac{8}{2} * \frac{10}{4} = 10$
- $N = N_1 + N_2 = 10 + 7 = 17$
- $n = n_1 + n_2 = 8 + 4 = 12$
- $V = 17 * \log_2 12 = 60.94$
- $E = 10 * 60.94 = 609.4$

While the Halstead's calculations do not appear complex, there are some issues found in these measures [15]. The first issue is the scale type of each equation. There is ambiguity and uncertainty about the scale types in Halstead's metrics. The second issue is the units of measurement for both the left-hand and the right-hand sides of most of Halstead's equations. In general, there are lots of debates about these software metrics, they are, however, still continuously used by practitioners [5].

3. Overview of R and Python

3.1 R

3.1.1 Definition

R is a free and versatile open source programming language for statistics and data science. It is derived from the statistical language S developed by AT&T. Unlike Python, R is a functional programming language and includes some support for object-oriented programming (OOP). Many R packages are written using R Objects, including the core statistics package, *lattice*, and *ggplot2* [9]. R is available on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS. R is not only a statistics system but also an environment within which statistical techniques are implemented. Thanks to its flexibility and user friendly interface, R has become more popular than its inspiration – S. R was initially written by Robert Gentleman and Ross Ihaka – from the Statistics Department of the University of Auckland, however; the current version of R is the result of contributions from R core Team and many academics, statisticians, engineers and scientists, all of whom are contributing to a vast community of R users [26]. R has now one of the richest ecosystems in which to perform data analysis [5]. There are more than 14,000 packages available in CRAN (open-source repository). They are meticulously validated (with a hybrid automated-peer review process) before they get to CRAN. These packages extend the R language in every field (e.g. business, industry, government, medicine, academia, and so on).

3.1.2 Advantages and Disadvantages of R

The first advantage of R is its *tidyverse* library, written in part by Hadley Wickham. It is a collection of packages (e.g. *readr, databases, dplyr, tidyr, tibble, stringr, ggplot2*, etc.) that makes common data science tasks simple, elegant, reproducible and fast [10]. Next is RMarkdown. This is a documentation system which ties together simple markdown syntax with R code ‘chunks.’ This process is straight-forward, and the documentation can be saved in various formats (e.g. html, pdf, word, etc.) [10]. Lastly, R provides its users with a fully functional web application – *shiny,* by using the *runApp()* command. Shiny is highly accessible when it comes to make the leap from analysis to analytics [10].

When compared to Python, R can be difficult for beginners who have no software engineering experience or statistical background. Indeed, due to a large number of packages, it can be time-consuming to find the required package in R. Additionally, there are also many dependencies between its packages. Finally, a poor code in R will be executed slower than other programming languages [8].

3.1.3 IDEs for R

RStudio[4] is an IDE with a console and syntax-highlighting editor that supports direct code execution. It also has tools for plotting, history, debugging and workspace management. There are also StatET[5] and ESS[6], which provide the R programmer with an IDE in the Eclipse and Emacs settings, respectively [6].

3.1.4 Installing R and Rstudio

R is available for download via CRAN[7], whereas RStudio can be downloaded on its official website. There are two main options to download RStudio: RStudio Desktop and RStudio Server. Depending on the users' purpose and own interest, these two options can be downloaded either with Open Source License or with Commercial License.

3.2 Python

3.2.1 Definition

Python – invented by Guido van Rossum and officially released in 1991 – is an interpreted, object-oriented (OO), high-level programming language with accessible syntax. Like R, Python is also a free and open source computer programming language and can be installed using Windows, Linux/Unix and Mac OS X [27]. While R is designed mostly for statistical purposes, Python is a general-purpose programming tool, meaning anybody can use the language and modify it to suit their specific needs. Python's application domains range from web development, cell phone scripting, and education to desktop GUIs, software development and business application [28]. While the implementation to this programming language is varied, the most common implementation (also known as a default byte-code interpreter of Python) is *CPython*. According to Pedregosa et al. (2011),

[4] RStudio, [Online]. Available: https://rstudio.com/. [Accessed May 2020]

[5] StatET, [Online]. Available: http://www.walware.de/goto/statet/. [Accessed June 2020]

[6] ESS (Emacs Speaks Statistics), [Online]. Available: http://ess.r-project.org/. [Accessed June 2020]

[7] CRAN (The Comprehensive R Archive Network), [Online]. Available: https://cran.r-project.org/. [Accessed June 2020]

‘*CPython* makes it easy to reach the performance of compiled languages with Python-like syntax and high-level operations’ [13]. Beside *CPython*, there are several other variants (e.g. *Pypy, Jython, IronPython, IPython, etc.*) that implement the same language standard, but increase the execution time. For example, by compiling before or during execution [18]. There are two main versions of Python, Python 2.x and Python 3.x.

Like R, Python also has a large and active software community, namely Python Users Group (PUGs)[8] [12]. Together with Python Software Foundation (PSF) they have contributed to the rich set of built-in libraries in Python, which are available in source or binary form. For data analysis, interactive computing and data visualization, there are some essential Python libraries such as *NumPy* (numerical Python) – for mathematical computing, data and model parameters or linear algebra*; pandas* (derived from panel data) – for multidimensional structured datasets; *matplotlib* – for data visualization; *SciPy* – for numerical integration and optimization; *scikit-learn* – for data science; and *statsmodels* – for statistical analysis [11].

3.2.2 Advantages and Disadvantages of Python

Apart from the aforementioned benefits, Python also increases programmers’ efficiency [20, 21]. ‘Python due to its flexibility and simplicity reduces the amount of time taken from conceptualization of an idea to building the application and marketing it, resulting in more demand for Python programmers in Enterprise setup’ [11]. In case of embedded system, Python, due to its small size, may be a best solution for cost reduction. Additionally, Python is an agile code, meaning it is easy to change, add or remove modules in Python [19]. Moreover, the “two-languages” problem can be solved using Python. This means programmers and developers do not need to switch their programming languages when doing research, prototyping or building production systems [12]. Finally, Python is a tool to deploy and implement machine learning at a large-scale, because Python codes are easier to maintain and more robust than R. Replicability and accessibility are also easier with Python compared to R. Python recently began to provide its users with advanced API for Machine Learning or Artificial Intelligence [7].

However, most Python code will run substantially slower than code written in a compiled language like Java or C++, as Python is an interpreted programming language [12]. Furthermore, Python may not be a suitable programming language to execute multithreaded parallel code. Due to the *global*

[8] Python Software Foundation, "Diversity Statement," [Online]. Available: https://www.python.org/community/diversity/. [Accessed May 2020].

interpreter lock (GIL) Python instruction cannot be executed more than one at a time. To overcome this obstacle, Python C extensions can be used, unless they need to regularly interact with Python objects [12].

3.2.3 IDEs for Python

Unlike R, Python has a various list of IDE. PyCharm[9] is one of the most popular IDEs for Python programming language. PyCharm was created by the Czech company Jetbrains, a team responsible for one of the most famous Java IDE, the IntelliJ IDEA. It comes in three options namely, Professional Edition, Community Edition and Educational Edition. The Professional Edition of PyCharm requires a subscription, while the Community Edition and Educational Edition are free. In addition to Python, PyCharm provides support for JavaScript, HTML/CSS, Angular JS, Node.js, and so on. This makes it a good option for web development. It provides a graphical debugger, an integrated unit tester, coding assistance, support for web development with Django, and integration with Anaconda's data science platform [11]. Beside Pycharm, there are also other IDEs for Python:

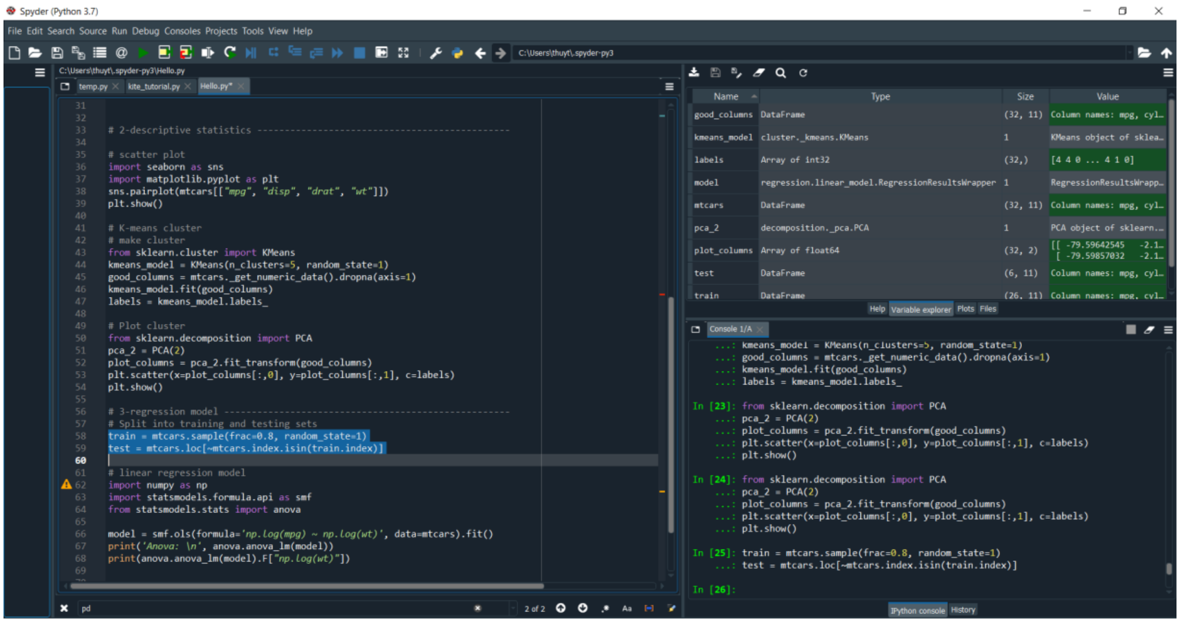
- PyDev[10] (free), an IDE built on the Eclipse platform;
- Python for Visual Studio Code (VSC)[11] (for Windows users);
- Spyder[12] (free), an IDE currently included with Anaconda. It includes editing, interactive testing, debugging, introspection features, and has the interface similar to RStudio (Figure 2);
- Komodo IDE (commercial) [11].

[9] JetBrain, "Pycharm," [Online]. Available : https://www.jetbrains.com/pycharm/. [Accessed June 2020]

[10] Anaconda Documentation, „Eclipse and PyDev," [Online]. Available: https://docs.anaconda.com/anaconda/user-guide/tasks/integration/eclipse-pydev/. [Accessed June 2020]

[11] Anaconda Documentation, „Python for Visual Studio Code," [Online]. Available: https://docs.anaconda.com/anaconda/user-guide/tasks/integration/python-vsc/. [Accessed June 2020]

[12] Anaconda Documentation, „Spyder," [Online]. Available: https://docs.anaconda.com/anaconda/user-guide/tasks/integration/spyder/. [Accessed June 2020]


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\thuyt\.spyder-py3
C:\Users\thuyt\.spyder-py3\Hello.py
temp.py
kite_tutorial.py
Hello.py*
pd
2 of 2
Help
Variable explorer
Plots
Files
Console 1/A
IPython console
History

```python
# 2-descriptive statistics ------------------------------------------

# scatter plot
import seaborn as sns
import matplotlib.pyplot as plt
sns.pairplot(mtcars[["mpg", "disp", "drat", "wt"]])
plt.show()

# K-means cluster
# make cluster
from sklearn.cluster import KMeans
kmeans_model = KMeans(n_clusters=5, random_state=1)
good_columns = mtcars._get_numeric_data().dropna(axis=1)
kmeans_model.fit(good_columns)
labels = kmeans_model.labels_

# Plot cluster
from sklearn.decomposition import PCA
pca_2 = PCA(2)
plot_columns = pca_2.fit_transform(good_columns)
plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=labels)
plt.show()

# 3-regression model ------------------------------------------------
# Split into training and testing sets
train = mtcars.sample(frac=0.8, random_state=1)
test = mtcars.loc[~mtcars.index.isin(train.index)]

# linear regression model
import numpy as np
import statsmodels.formula.api as smf
from statsmodels.stats import anova

model = smf.ols(formula='np.log(mpg) ~ np.log(wt)', data=mtcars).fit()
print('Anova: \n', anova.anova_lm(model))
print(anova.anova_lm(model).F["np.log(wt)"])
```

Name	Type	Size	Value
good_columns	DataFrame	(32, 11)	Column names: mpg, cyl…
kmeans_model	cluster._kmeans.KMeans	1	KMeans object of sklea…
labels	Array of int32	(32,)	[4 4 0 ... 4 1 0]
model	regression.linear_model.RegressionResultsWrapper	1	RegressionResultsWrapp…
mtcars	DataFrame	(32, 11)	Column names: mpg, cyl…
pca_2	decomposition._pca.PCA	1	PCA object of sklearn.…
plot_columns	Array of float64	(32, 2)	[[-79.59642545 -2.1… [-79.59857032 -2.1…
test	DataFrame	(6, 11)	Column names: mpg, cyl…
train	DataFrame	(26, 11)	Column names: mpg, cyl…

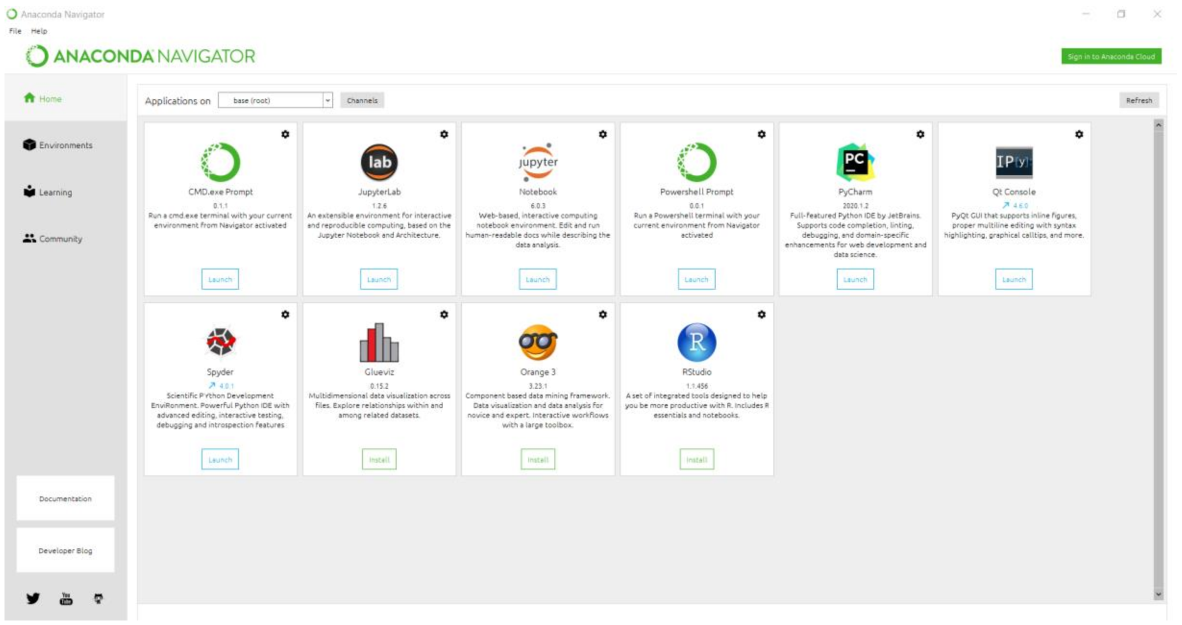
```
   ...: kmeans_model = KMeans(n_clusters=5, random_state=1)
   ...: good_columns = mtcars._get_numeric_data().dropna(axis=1)
   ...: kmeans_model.fit(good_columns)
   ...: labels = kmeans_model.labels_

In [23]: from sklearn.decomposition import PCA
   ...: pca_2 = PCA(2)
   ...: plot_columns = pca_2.fit_transform(good_columns)
   ...: plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=labels)
   ...: plt.show()

In [24]: from sklearn.decomposition import PCA
   ...: pca_2 = PCA(2)
   ...: plot_columns = pca_2.fit_transform(good_columns)
   ...: plt.scatter(x=plot_columns[:,0], y=plot_columns[:,1], c=labels)
   ...: plt.show()

In [25]: train = mtcars.sample(frac=0.8, random_state=1)
   ...: test = mtcars.loc[~mtcars.index.isin(train.index)]

In [26]:
```

Figure 2. Interface from Spyder (version 4.0.1)

3.2.4 Installing Anaconda - Python Distribution

PSF released Python interpreter with standard libraries. However, to work in a scientific or enterprise environment users also need to install other packages such as *Scikit-learn*, *PyTorch*, *TensorFlow* and *SciPy*. This is complicated and time-consuming. In order to simplify package management and deployment Anaconda[13] Distribution was created. It is one of the most popular distributions of Python programming language for data scientists. Anaconda contains '*Conda* and *Anaconda Navigator*, core Python interpreter and hundreds of scientific packages. *Conda* works on local command line interface such as Anaconda Prompt on Windows and terminal on macOS and Linux, [whereas] *Anaconda Navigator* is a desktop graphical user interface that allows its users to launch applications and easily manage Conda packages, environments, and channels without using command-line commands' (Figure 3) [4]. Anaconda Distribution can be downloaded via its official website.

[13] Anaconda Documentation, „Installation," [Online]. Available: https://docs.anaconda.com/anaconda/install/. [Accessed June 2020]



CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated

JupyterLab
1.2.6
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
6.0.3
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated

PyCharm
2020.1.2
Full-featured Python IDE by JetBrains. Supports code completion, linting, debugging, and domain-specific enhancements for web development and data science.

Qt Console
4.6.0
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder
4.0.1
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Glueviz
0.15.2
Multidimensional data visualization across files. Explore relationships within and among related datasets.

Orange 3
3.23.1
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Figure 3. Anaconda Navigator

4. Methodology

In this section, an experiment was conducted to compare the performance of R and Python without diving into the statistical data and analysis results. The experiment was divided into quantifiable and qualifiable attributes to provide the most unbiased substantive information.

4.1 Quantifiable Experiment

In the quantifiable experiment, two projects with two different datasets were identified to measure the quantifying attributes: the processing time and the code complexity of selected tools. The objective of the projects was data wrangling, exploratory data analysis, multiple linear regression and random forest prediction without drawing a conclusion. Each test run of the comparative programs was run with all extraneous applications closed.

Table 1. Machine specifications.

Specifications	Machine
Processor	Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50GHz
HD Size	455 GB
HD Free Space	362 GB
RAM	8 GB
OS	Windows 10

Table 2. Software specifications for both test machines.

Tool	Version	Installation
Python	3.7.6	Local PC
R	4.0.1	Local PC

Project 1 (small built-in data set) – Motor Trend Car Road Tests (mtcars)

mtcars data is a built-in data, extracted from the 1974 Motor Trend US magazine. It is a small data set, which contains information about 32 cars, including their weight, fuel efficiency (in miles-per-gallon), aspects of automobile design and performance for many automobiles [21]. The mtcars data comes with the *dplyr* package in R, which is included in *tidyverse* package. In Python, users can access the mtcars dataset not only through the *datasets* package in *statsmodels* library by using the *get_rdataset* function[14], but also through the *ggplot* library, a port of a popular R plotting library called *ggplot2* (see below).

In R:

```
data(mtcars)
```

In Python:

```
import statsmodels.api as sm

mtcars = sm.datasets.get_rdataset("mtcars", "datasets", cache = True).data

from ggplot import mtcars
```

The initial code was written in R and then replicated into Python with some adjustments for balance. The projects began with the loading of the data. The mtcars data has 11 columns consisting of 11 numeric variables and 32 rows (observations). The data wrangling of mtcars data began with replacing the codes of Engine and Transmission variables with human readable value labels. The variable labels were added, which make the variables easier to understand without renaming them. Next, some codes were written to explore the dataset. Some cross-tabulations were then created to show the relationship between Engine variable and Transmission variable. The other cross-tabulation was performed and graphed to show the numbers of cylinders per Engine by Transmission. Multiple bar graphs, histograms, boxplots and scatter plots were generated to visualize the data. Beside that a K-means cluster plot was developed to group similar data points together and discover underlying patterns. Next a heatmap was created for visualization of the correlation. A multiple linear regression model was built to understand the relationship between variables. The linear regression was firstly performed on 11 variables.

$$mpg = ß_0 + ß_1 cyl + ß_2 disp + \cdots + ß_{10} carb \tag{8}$$

[14] statsmodels, "The Datasets Package," [Online]. Available: https://www.statsmodels.org/devel/datasets/index.html/. [Accessed June 2020]

To fit the model above, the Akaike Information Criteria[15] (AIC) was selected. The formula used is:

$$-2 * logL + k * npar \tag{9}$$

where L is the value of the likelihood, npar represents the number of estimated parameters, and k = 2 for the usual AIC.

After running AIC functions, the statistically significant variables were selected: Weight, ¼ mile time and Transmission in R or Weight, number of cylinders and gross horsepower in Python. This optimal model with three (3) variables was trained using 80% of the data and tested with the remaining 20%. Finally, a random forest prediction was developed to compare the data mining algorithms functions of both selected programming languages.

Project 2 (meta data set) – Crash Report Sampling System (CRSS)

The second dataset was obtained from the Crash Report Sampling System (CRSS), which builds on the retiring long-running National Automotive Sampling System General Estimates System (NASS GES) of the National Highway Traffic Safety Administration (NHTSA)[16]. A single file focusing on person data for the year 2018 was used. Person data contains all information of involved persons, ranging from property-damage-only crashes to those that result in fatalities. The original complete data set is composed of 120,230 observations and 54 variables. The data was pre-processed and cleaned before usage. After the missing data and untrue values were dropped, the final dataset contained 107,993 observations and 54 variables.

Like project 1, the initial code in this second project was written in R and then replicated into Python with similar adjustments. The experiment also started with the loading of the data. Since it was a large dataset, replacing variable names with human readable values or variable labels was not done in the data wrangling. However, finding missing values was added to this step. All other data analysis tasks were replicated from the first project, except creating the boxplot and cluster plot. More barcharts were created to compare the performance of the selected tools. The multiple linear model was built with eight (8) variables from the heatmap. Finally, the data was split into train data and test data in order to measure a random forest prediction.

[15] R Documentation, "stepAIC," [Online]. Available: https://www.rdocumentation.org/packages/MASS/versions/7.3-51.6/topics/stepAIC/. [Accessed June 2020]

[16] Data can be retrieved from https://www.nhtsa.gov/node/97996/221.

4.1.1 Lines of code (LOC)

In order to count the number of lines in both R code and Python code, cloc[17] (Count lines of code), a command in Windows command prompt was applied. The version cloc-1.86.exe was installed to count the physical lines of source code in each task in both R and Python programming languages. Comments and white space were not counted to the final number of lines in the code because they often depend on users' own methods and familiarity. The result was reviewed individually to minimize the mistakes.

4.1.2 Halstead metrics

The convention, introduced in the Literature Review Section, was applied to the code for all programs to calculate the number of operators and operands. The total operators and operands were consolidated to provide a total, while the unique operators and operands were then identified. The counting was done three times on all tasks of each project. The final result is the average of the counting. The program length (N), program vocabulary (n), volume (V), program difficulty (D), programming effort (E), and programming time (T) were then measured based on the number of operands and operators. The formulas (2) to (7) can be found in the Literature Review Section.

4.2 Qualifiable Research

Beside the quantitative aspects, the qualifiable elements of the selected tools were also taken into account in order to provide the most unbiased performance comparison of selected tools possible. According to Brittain et al. (2018), 'a tool dedicated to data and statistical analysis should be readable, writable, able to handle various data types, have different options to manage missing values properly, and provide at least basic mathematical and statistical functions, such as the ability to generate random numbers and probabilistic distributions, as well as high-level visualizations' [5] (Originally Huber [14]) . A comparison was then developed based upon these criteria.

[17] CLOC, [Online]. Available: http://cloc.sourceforge.net/. [Accessed July 2020]

5. Results

This section provides the results of the previous experiments.

5.1 Quantifiable Experiment

The main objective of the quantifiable experiment presented here is to compare the processing time and the code complexity between two selected tools. However, a comparison of the graphic design of each tool will also be given to support the qualitative experiment. This is in order to provide unbiased substantive information. Two projects with two different sizes of data sets were performed. The codes of these projects were run in the aforementioned computer. ‘Wall clock’ was used to measure the processing time a program needs to execute all codes in order to complete a task. The processing time is the overall time of all mentioned data analytics tasks of each project. As explained in subsection 4.1.1, the number of code lines were counted without comment lines and white space. The summary below highlights the key results of the selected tools.

5.1.1 Processing time

The final time performance of each tool, shown in Table 3, was the average of five time measurements with the condition that all other applications were closed. Figure 4 visualizes the average results of each tool in three different scenarios: (1) Both tools had to run the codes to complete all required tasks, (2) both tools ran the codes without the plotting model task, and (3) all the codes were run excluding the plotting model and correlation matrix plot codes.

Table 3. The time performance of each programming language.

Project	Tasks	Tool	IDE	Processing time
1	All tasks	R	Rstudio	**6,02**
		Python	Spyder	6,23
	All tasks (without plotting model)	R	Rstudio	6,01
		Python	Spyder	**5,64**
	All tasks (without plotting mode and creating correlation matrix plot)	R	Rstudio	4,24
		Python	Spyder	**3,44**
2	All tasks	R	Rstudio	**92,15**
		Python	Spyder	288,27
	All tasks (without plotting model)	R	Rstudio	67,19
		Python	Spyder	**24,72**
	All tasks (without plotting mode and creating correlation matrix plot)	R	Rstudio	23,39
		Python	Spyder	**10,36**

As displayed in Figure 4, there is no clear conclusion about which tool performed better. While R's performance was faster than Python's in the first scenario, Python allowed the code to run faster than R in the other two scenarios. It was noticeable that Python started to run slower, when it came to plot the linear regression model. R can easily generate diagnostic plots for the model by using the *plot(lm)* command. Python however, does not have a specific function to call the four plots of the model. Each parameter had to be defined and some wrapper functions needed to be built for the four plots. It might be the reason why Python performed poorer than R when it came to plot the multiple linear regression model. The processing time was tested without creating a correlation matrix, because R appeared to perform slower when running the *ggpairs()* function. While it took less than 10 seconds in Python to create a correlation matrix plot, R needed more than 30 seconds. After considering all three scenarios, Python is a better tool to start with, when users' tasks does not include plotting statistical models, in which case R would be preferable.

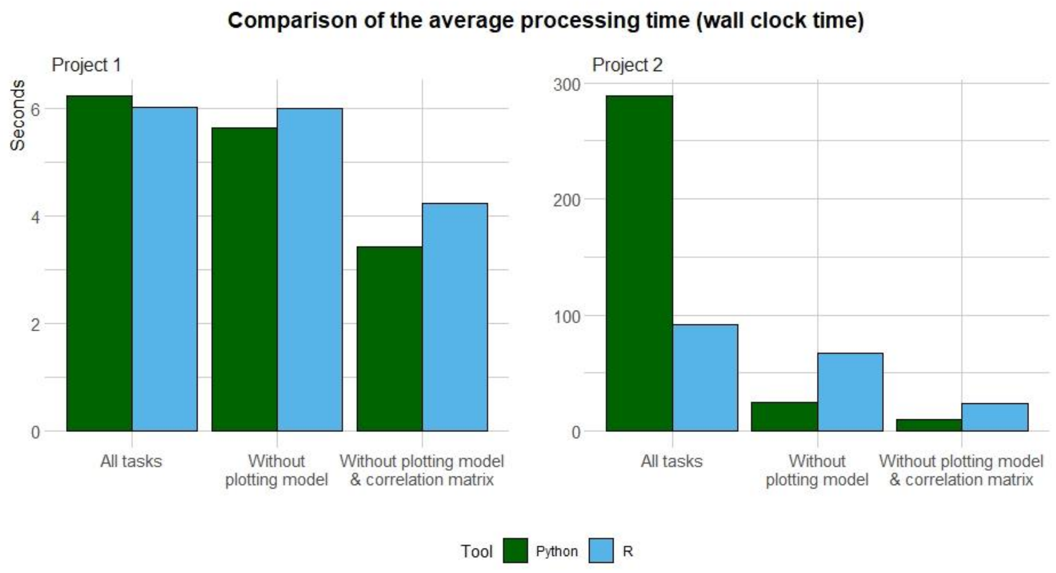


Figure 4. Comparison of the average processing time (wall clock time).

5.1.2 Lines of code

As mentioned previously, there were five main tasks to identify: data wrangling, descriptive analytics, linear regression model, train and test data, and random forest prediction. The number of lines in the code needed to perform each data analysis task in project 1 and project 2 were illustrated in Figure 5 and Figure 6 respectively.

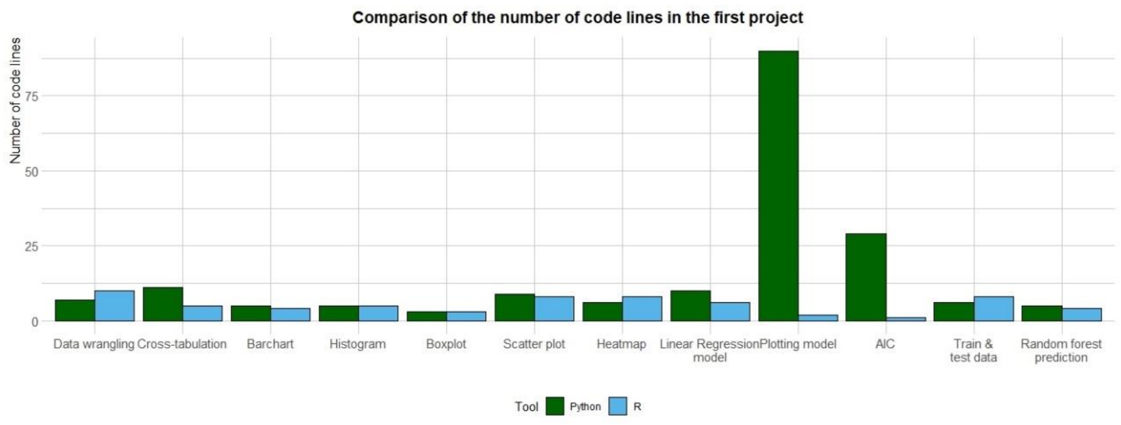


Figure 5. Comparison of the number of code lines in the first project (mtcars data).

In general, there are more lines of code in Python compared to R when looking at Figure 5 and 6. However, the deviation between the tools in most of the activities was small, except in plotting model and AIC tasks. In these activities Python needed a huge amount of codes to complete the tasks due to the lack of a built-in command in its library. It shows that R has more advantages than Python when it comes to data analysis tasks because of its huge number of packages, readily usable tests, and the advantage of using formulas. The slight deviations in the number of code lines between R and Python in Figure 5 were displayed clearer in Figure 6 due to the larger amount of charts in the second project. Even though the number of code lines in Python was generally higher, its syntax is still readable. The syntax of R however, requires its user to have a knowledge in statistics to understand it.

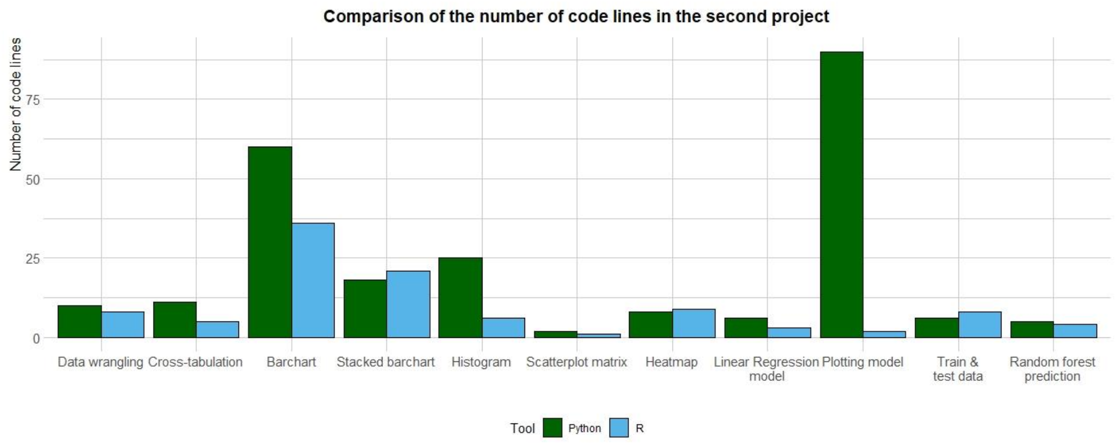


Figure 6. Comparison of the lines of code in the second project (Person data).

The analysis of the code from both projects was measured and compiled in Table 4. Again, no specific tool had better advantages or disadvantages. The results from Table 4 confirmed that the overall code requirements to complete all of the tasks in Python were more extensive than in R, while more packages were installed and used in Python than in R. It should be noted however, that a conclusion cannot be made about the quality of a programming language based solely on the number of lines of code it has [5].

Table 4. Summary of the overall activity in each project.

	Project 1		Project 2	
Measured activity	R	Python	R	Python
The total code lines	73	195	103	241
The total code lines without plotting model and AIC	70	76	101	151
Packages used	7	16	7	10
Packages used without plotting model and AIC	6	11	6	8

Despite the differences in the amount of code and the level of coding in the tools, the experiments were able to be completed and duplicated in all tools including graphics. Due to the similarity of all the tasks in both projects, only the charts and plots built on mtcars dataset were shown below (Figure 7 – Figure 18). In the end, both tools provide the same insight, even though the graphic design of each tool is slightly different.

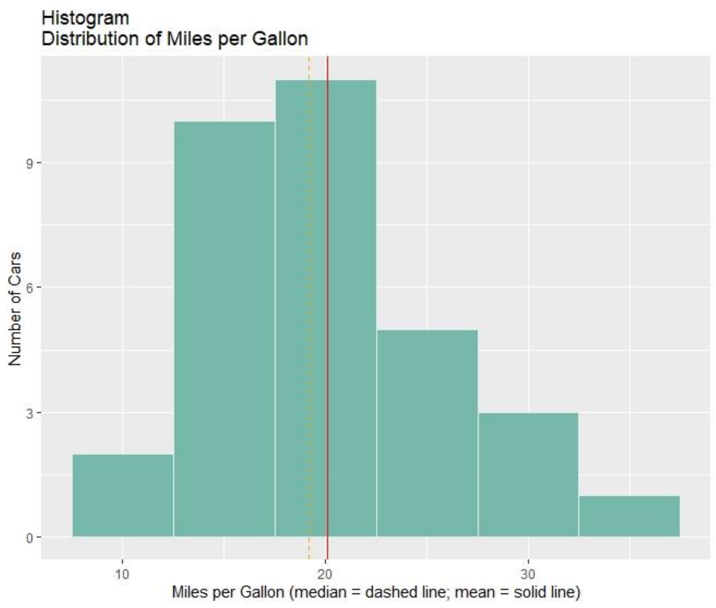


Figure 7. Histogram from R.

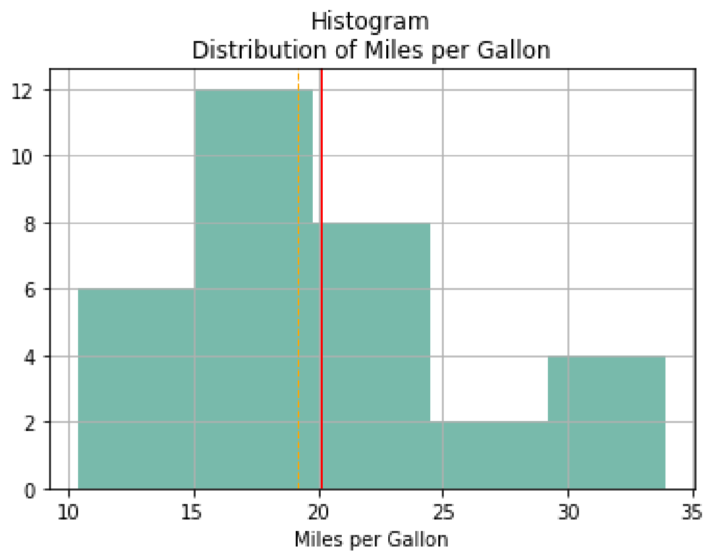


Figure 8. Histogram from Python.

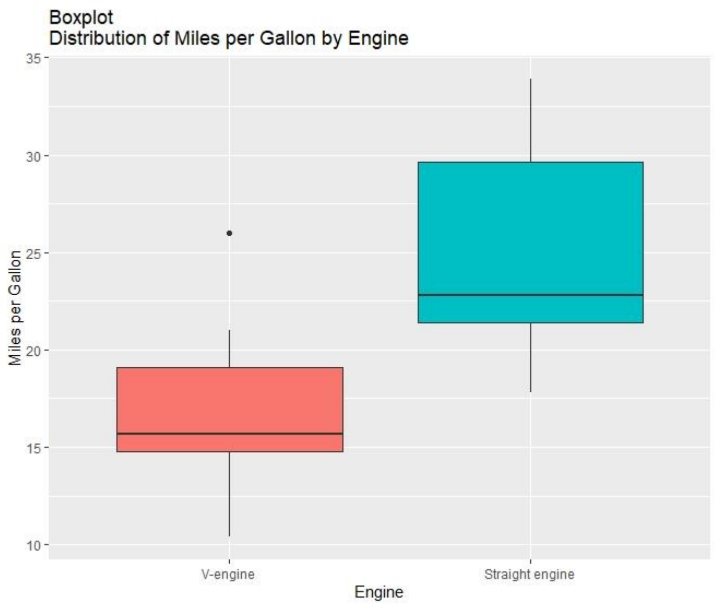


Figure 9. Boxplot from R.

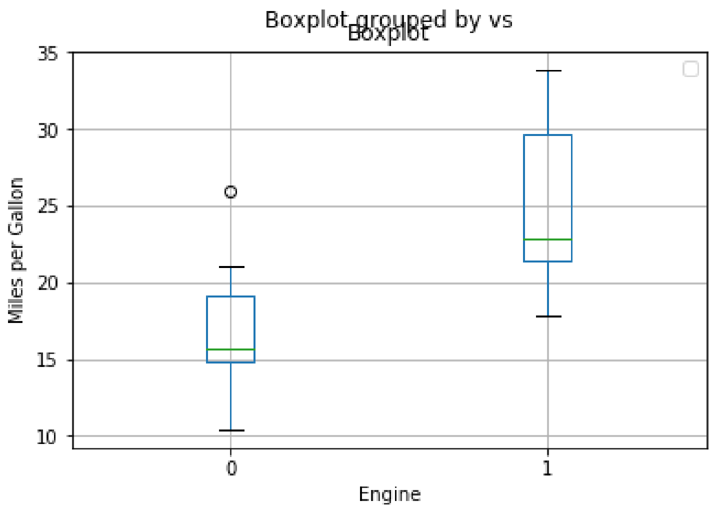


Figure 10. Boxplot from Python.

In R, the function *ggpairs()* was applied to graphically display a correlation matrix, which indicates correlation coefficients among the continuous variables. The *GGally* package (a helper package of *ggplot2*) needs to be installed to support this function. The generated matrix is shown in Figure 11 and 13. With *seaborn* and *matplotlib*, Python offered a similar figure to the original one made in R, which shows the relationship between variables (Figure 12 and 14).

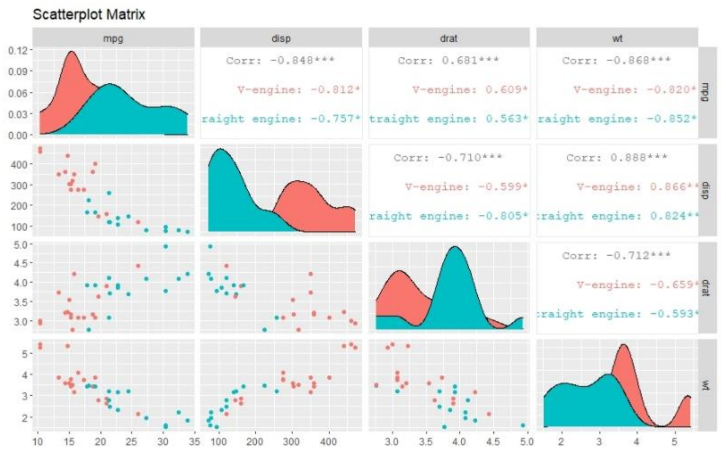


Figure 11. Scatter plot matrix from R.

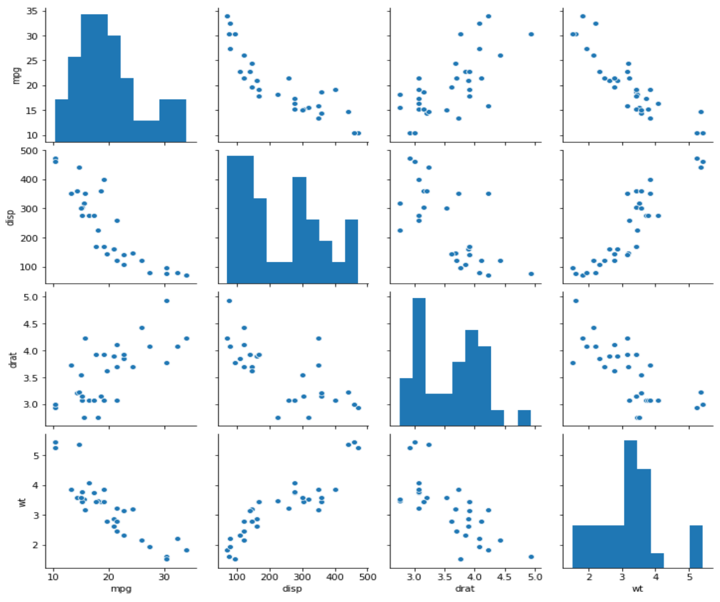
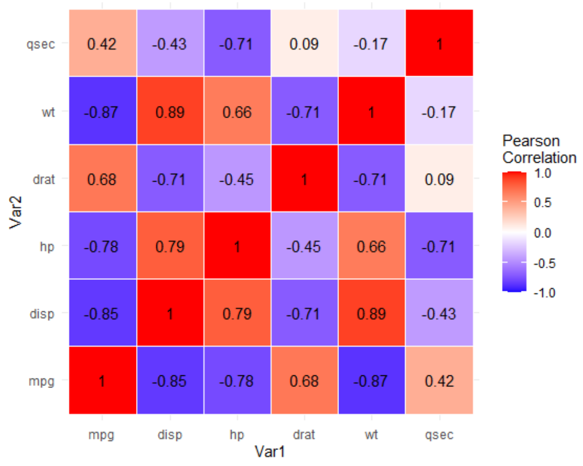


Figure 12. Scatter plot matrix from Python.



Var2	mpg	disp	hp	drat	wt	qsec
qsec	0.42	-0.43	-0.71	0.09	-0.17	1
wt	-0.87	0.89	0.66	-0.71	1	-0.17
drat	0.68	-0.71	-0.45	1	-0.71	0.09
hp	-0.78	0.79	1	-0.45	0.66	-0.71
disp	-0.85	1	0.79	-0.71	0.89	-0.43
mpg	1	-0.85	-0.78	0.68	-0.87	0.42

Figure 13. Heatmap from R.

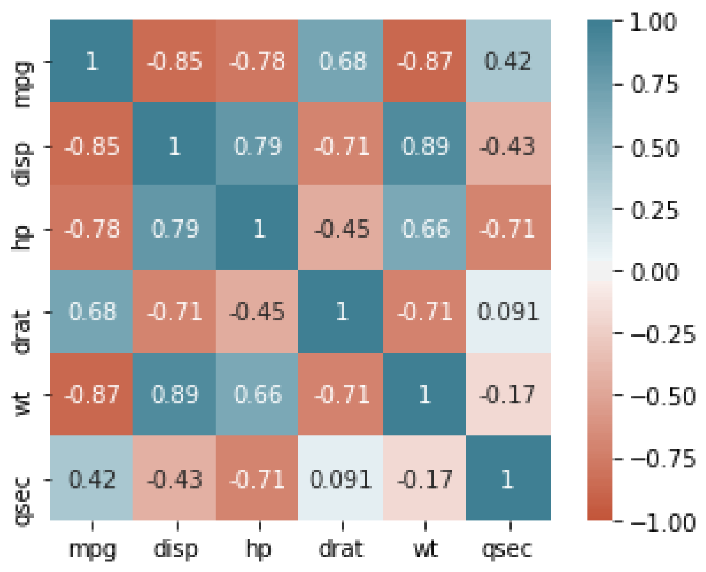


Figure 14. Heatmap from Python.

Again, the similarity between R and Python can be seen through the K-means cluster plots (Figure 15 and 16). Despite the slightly different appearance, the way of creating K-means cluster plot in R is similar to in Python. First, all non-numeric columns and missing values were removed. K-mean was then calculated and all clusters were formed. Finally, a plot which shows many different groups of cluster was created.

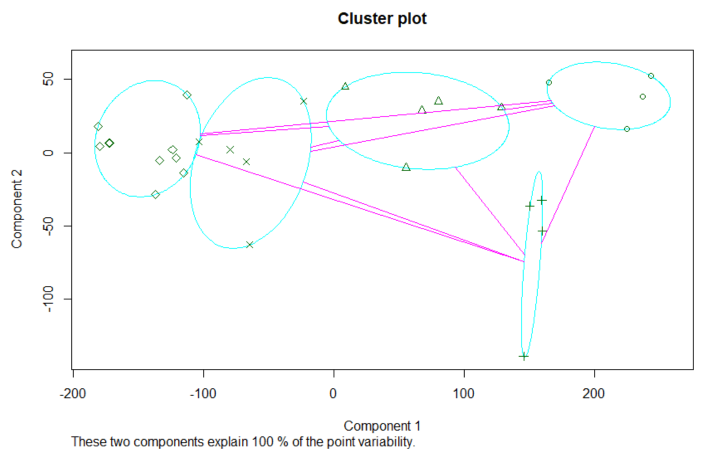


Figure 15. K-means cluster plot from R.

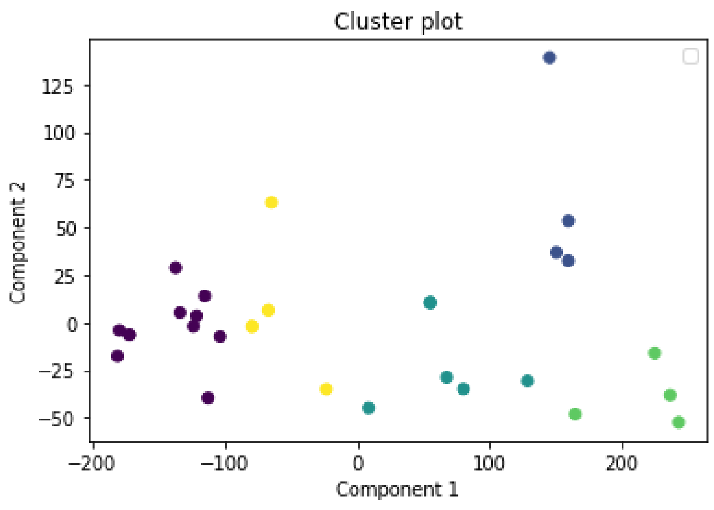
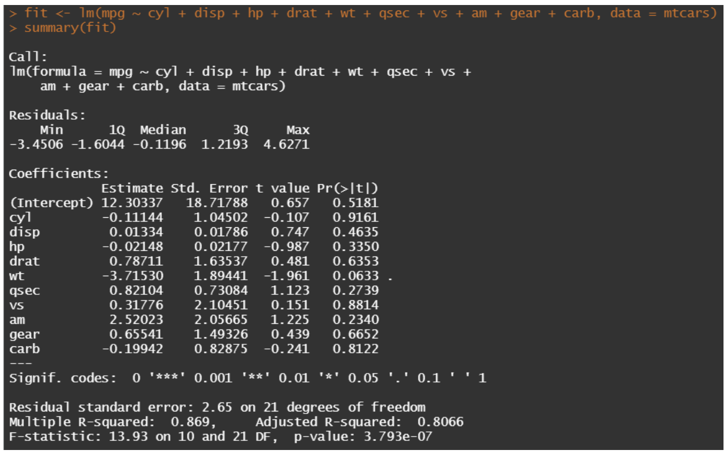


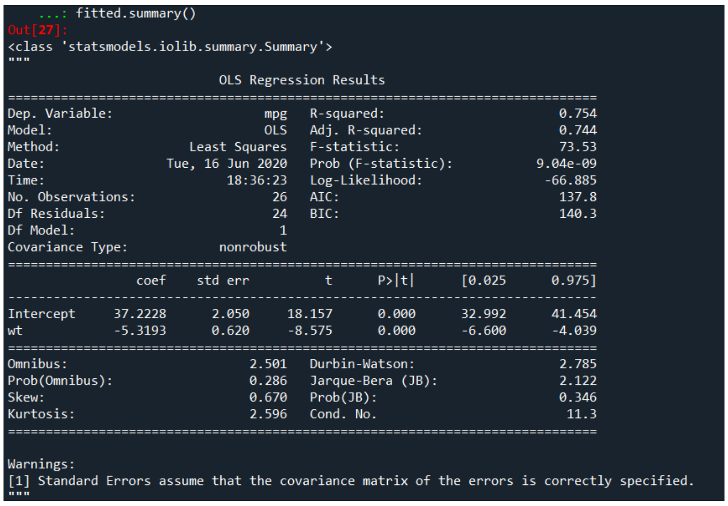
Figure 16. K-means cluster plot from Python.

Lastly, unlike R with only one single line of built-in command *lm()*, Python needed one more step to add an intercept into the model and one more step to fit the model in order to create a multiple linear regression. The results are shown in Figure 17 and 18.



```
> fit <- lm(mpg ~ cyl + disp + hp + drat + wt + qsec + vs + am + gear + carb, data = mtcars)
> summary(fit)

Call:
lm(formula = mpg ~ cyl + disp + hp + drat + wt + qsec + vs +
    am + gear + carb, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-3.4506 -1.6044 -0.1196  1.2193  4.6271

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 12.30337   18.71788   0.657   0.5181
cyl         -0.11144    1.04502  -0.107   0.9161
disp         0.01334    0.01786   0.747   0.4635
hp          -0.02148    0.02177  -0.987   0.3350
drat         0.78711    1.63537   0.481   0.6353
wt          -3.71530    1.89441  -1.961   0.0633 .
qsec         0.82104    0.73084   1.123   0.2739
vs           0.31776    2.10451   0.151   0.8814
am           2.52023    2.05665   1.225   0.2340
gear         0.65541    1.49326   0.439   0.6652
carb        -0.19942    0.82875  -0.241   0.8122
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.65 on 21 degrees of freedom
Multiple R-squared:  0.869,	Adjusted R-squared:  0.8066
F-statistic: 13.93 on 10 and 21 DF,  p-value: 3.793e-07
```

Figure 17. Multiple linear regression model from R.



```
   ...: fitted.summary()
Out[27]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.754
Model:                            OLS   Adj. R-squared:                  0.744
Method:                 Least Squares   F-statistic:                     73.53
Date:                Tue, 16 Jun 2020   Prob (F-statistic):           9.04e-09
Time:                        18:36:23   Log-Likelihood:                -66.885
No. Observations:                  26   AIC:                             137.8
Df Residuals:                      24   BIC:                             140.3
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     37.2228      2.050     18.157      0.000      32.992      41.454
wt            -5.3193      0.620     -8.575      0.000      -6.600      -4.039
==============================================================================
Omnibus:                        2.501   Durbin-Watson:                   2.785
Prob(Omnibus):                  0.286   Jarque-Bera (JB):                2.122
Skew:                           0.670   Prob(JB):                        0.346
Kurtosis:                       2.596   Cond. No.                         11.3
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

Figure 18. Multiple linear regression model from Python.

In conclusion, both R and Python are well-equipped for data visualization. Customizing graphics is easier and more intuitive in R with the help of *ggplot2* than in Python with *matplotlib*. The *seaborn* library helps to overcome this, and offers good standard solutions which get by with relatively few lines of code. *Seaborn* uses a programmatic approach whereby the user can access the classes in *Seaborn* and *Matplotlib* to manipulate the plots. *ggplot2* however, uses a layered approach wherein the user can add aesthetics and formats in any order to create the figure. It was also noticeable that Python plots, when saved as graphics, take up significantly more disk space than R generated graphics.

5.1.3 Halstead metrics

A summary of Halstead metrics results of each project are displayed in Table 5. While Python had higher program length, program vocabulary and volume, it had, compared to R, lower difficulty, effort and time scores.

Table 5. Summary of Halstead metrics result

	Project 1		Project 2	
Parameters	R	Python	R	Python
Program length (N)	583	632	983	1,201
Program vocabulary (n)	139	170	184	218
Volume (V)	4,150	4,683	7,396	9,330
Difficulty (D)	181	106	151	109
Effort (E)	751,270	494,071	1,113,550	1,020,424
Time (T)	41,737	27,448	61,864	56,690

5.2 Qualitative Research

In this section a qualifiable comparison between R and Python is described based on the criteria mentioned in subsection 4.2. According to Huber, ‘a programming language dedicated to data and statistical analysis should be writable and readable by human not by computers only’ [14]. Therefore, this experiment began with writing some basic codes about how to explore a dataset in both languages, in order to provide an objective point of view on how one language is similar to or different from the other (Table 6). Next, in order to explore more deeply the characteristics of each tool, a detailed side by side comparison of each of Huber’s requirements between the tools was made.

Table 6 displays how the codes were written in each language in order to complete the same activity. As shown in Table 6, a data frame in Python needed to be written before the function call (e.g. `df.head()`), whereas in R it is usually inside the function parentheses. In general, R and Python had in most activities similar function calls.

Table 6. Qualitative comparison of basic codes in R and Python.

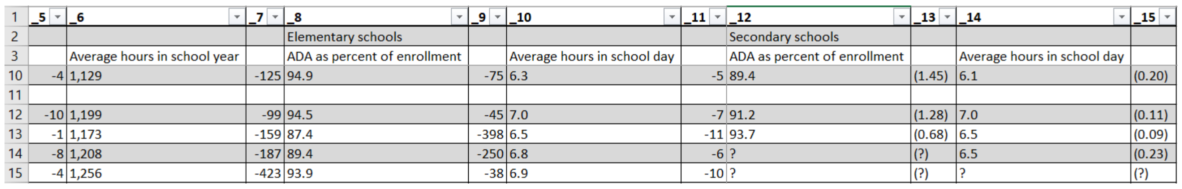
Activity	R	Python
Finding the number of observations and variables in the dataset	`dim(df)`	`df.shape()`
Printing the first five row of the data	`head(df, 5)`	`df.head(5)`

Finding the average of each statistic	`df %>%`
	`  select_if(is.numeric) %>%`
	`  map_dbl(mean, na.rm = TRUE)`
		`df.mean()`

Calculating quantile and variance of a variable	`quantile(x)` or `summary(df$x)`
	`var(x)`
		`df.x.describe()` or
		`df.x.quantile(0.25)`
		`df.x.var()`

Summary of a data set	`summary(df)`	`df.describe()`

Next, an objective comparison of how R and Python handle the missing values in a dataset was made. Missing values can be either an empty cell (no values provided in the dataset) or a special string of characters or numbers (e.g. ? and (?)) as displayed in Figure 19.

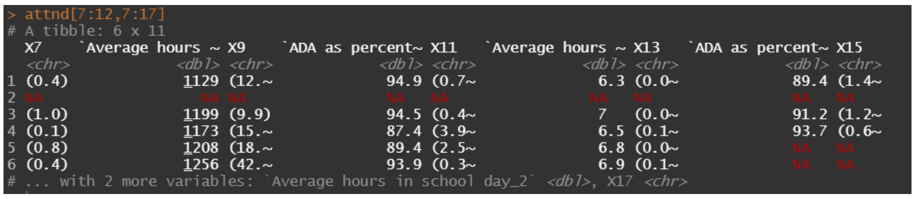


1	_5	_6	_7	_8	_9	_10	_11	_12	_13	_14	_15
2				Elementary schools				Secondary schools			
3		Average hours in school year		ADA as percent of enrollment		Average hours in school day		ADA as percent of enrollment		Average hours in school day	
10	-4	1,129	-125	94.9	-75	6.3	-5	89.4	(1.45)	6.1	(0.20)
11											
12	-10	1,199	-99	94.5	-45	7.0	-7	91.2	(1.28)	7.0	(0.11)
13	-1	1,173	-159	87.4	-398	6.5	-11	93.7	(0.68)	6.5	(0.09)
14	-8	1,208	-187	89.4	-250	6.8	-6	?	(?)	6.5	(0.23)
15	-4	1,256	-423	93.9	-38	6.9	-10	?	(?)	?	(?)

Figure 19. Overview of the original data.

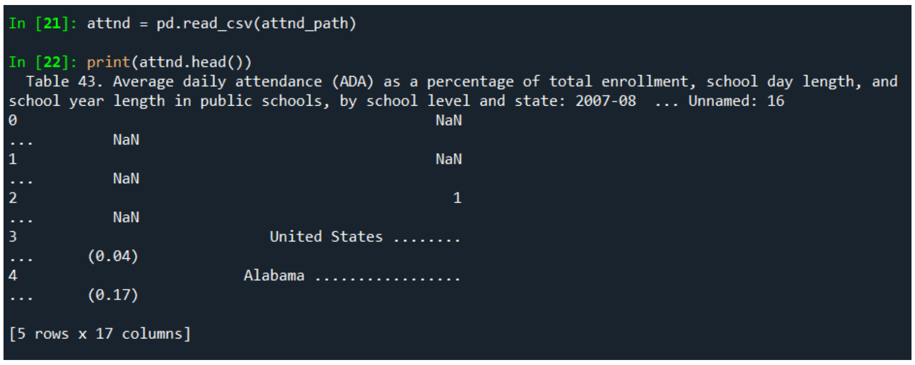
As stated by Banghart (2019), 'most data processing languages have their own special object to indicate a missing data value' [23]. In R "NA" is identified as the missing data object, whereas the missing value object is "NaN" in Python (Figure 20 and 21). However, according to R Documentation "NaN" or "Not a Number" is also used in R for exceptional numeric calculations (see `help(NA)`). Normally these missing values will be changed automatically when using the R or Python's read functions: function `read_csv()` and `pd.read_csv()` respectively. However, Python and R can only detect the standard missing values (e.g. NA, NaN, and empty cell). The non-standard missing values will then be ignored (e.g. na, N/A, "--", etc.). Hence, it is recommended to create a list of all possible missing values then add this list to `na` parameter of the function `read_csv`() or `na_values` parameter of function `pd.read_csv()` in order to avoid some unexpected missing values that are still in the data. The codes in either language can be written as below:

```
In R:
missing_values <- c("", "NA", "?", "(?)")
df <- read_csv(data, na = missing_values, col_types = cols())
In Python:
missing_values = ["n/a", "na", "--", "?", "(?)"]
df = pd.read_csv("data.csv", na_values = missing_values)
```



```
> attnd[7:12,7:17]
# A tibble: 6 x 11
  X7     `Average hours ~ X9     `ADA as percent~ X11    `Average hours ~ X13    `ADA as percent~ X15
  <chr>             <dbl> <chr>             <dbl> <chr>             <dbl> <chr>             <dbl> <chr>
1 (0.4)              1129 (12.~              94.9 (0.7~               6.3 (0.0~              89.4 (1.4~
2 NA                   NA NA                   NA NA                   NA NA                   NA NA
3 (1.0)              1199 (9.9)              94.5 (0.4~               7   (0.0~              91.2 (1.2~
4 (0.1)              1173 (15.~              87.4 (3.9~               6.5 (0.1~              93.7 (0.6~
5 (0.8)              1208 (18.~              89.4 (2.5~               6.8 (0.0~                NA NA
6 (0.4)              1256 (42.~              93.9 (0.3~               6.9 (0.1~                NA NA
# ... with 2 more variables: `Average hours in school day_2` <dbl>, X17 <chr>
```

Figure 20. Missing value object in R.



```
In [21]: attnd = pd.read_csv(attnd_path)

In [22]: print(attnd.head())
  Table 43. Average daily attendance (ADA) as a percentage of total enrollment, school day length, and
school year length in public schools, by school level and state: 2007-08  ... Unnamed: 16
0                                                NaN
...         NaN
1                                                NaN
...         NaN
2                                                  1
...         NaN
3                             United States ........
...       (0.04)
4                          Alabama .................
...       (0.17)

[5 rows x 17 columns]
```

Figure 21. Missing value object in Python.

In addition to that, Table 7 compares the way each tool finds, replaces and drops the missing values in the dataset. In general, despite the different types of the missing value object, both R and Python offer various ways to allow for their detection or transformation. Additionally, they also allow for the replacement of both numeric and character type missing values, as well as the dropping of unknown values in the dataset.

Table 7. Summary of handling missing data in each language.

	R	Python
Detecting missing values	`is.na(df)` or `anyNA`	`df.isnull()` or `df.isnull().values.any()`

Summarizing missing values	`sum(is.na(df))`	`df.isnull().sum()`
Replacing missing values by a median	`df <- df %>% mutate(col_name = replace(col_name, is.na(col_name), median(col_name, na.rm = T)))`	`df[col_name].fillna(df[col_name].median(), inplace=True)`
Dropping missing values	`drop_na(df)`	`df.dropna()`

Moreover, as illustrated in Table 8, both tools provide functions which help its users reproduce their results from random numbers. The difference here is that using function `set.seed()` in R does not require any package, whereas *random* package needs to be installed in order to use function `seed()` in Python.

Table 8. Random generator functions in R and Python.

	R	Python
Function	`set.seed(1)`	`random.seed(1)`
Used package	-	`import random`

In conclusion, R and Python can handle numbers, characters, logical, complex, and arbitrary data types. The capabilities of visualizing data using high-level customizable visualization packages, computing linear algebra functions, or doing probabilistic distributions were illustrated and proved in the quantitative experiment.

6. Conclusion and Future Work

Notably, the following comparison results are all drawn based on the quantitative and qualitative analysis, which is used as the criterion in this study. Similar to the results found by Brittain et al. (2018), the quantitative experiment showed that neither language is completely superior to the other. Python performed better than R when the plotting model task was not included in the projects, whereas R allowed the codes run faster than Python when the statistical model was required. Additionally, the processing time of Python was two times faster than R when creating the correlation matrix plot. They did, however, complete all the data analytics tasks, from exploring datasets and visualizing data to building linear models and measuring random forest predictions. Nevertheless, the amount of code needed to complete the analysis for the experiment different between the tools. Python had in most cases more lines in the code than R. As stated by Brittain et al. (2018), 'although less code is often considered preferable, it may also make the code less readable and more difficult to understand. An example of this would be positional parameters used in functions' [5]. Additionally, the results of the Halstead metrics also showed Python with higher vocabulary, length, and volume for both projects, but its program difficulty, program effort and time score were lower than these of R.

The results of the qualifiable research in this thesis confirmed one more time that no tool stood out based on Huber's criteria. Either language has its strengths and weaknesses in various ways. Ultimately, both languages offer the possibility to visualize data in a clear and appealing manner. Despite the differences, they both satisfied Huber's requirements to be a programming language dedicated to data and statistical analysis. The final results of the two experiments shows that R is an excellent choice if data analytics or visualization is at the core of the project, whereas Python might be a preferable alternative if the user's project needs a flexible, multi-purpose programming language with a large community of developers and one that is extendable with Machine Learning packages.

Some limitations in this thesis should be acknowledged. First, as mentioned in the literature review Section, there are some issues found in the Halstead's metrics which would bias the results of the overall analyses. Second, all calculations were done only by one individual. This may also cause bias in the results due to author's own interest, and programming experience. This could be prevented by each researcher performing the coding in their best language. Third, due to the restraints of these experiments, not all of the characteristics and suitability for various user groups are included in this thesis. Finally the use of 'Wall-Clock' means the times recorded are not machine perfect and are open to slight deviations due to human error.

In conclusion, the decision between R or Python should consider the programming-language preferences and experiences of the user as well as the objectives of users' mission. R is mainly used when the data analysis tasks require standalone computing or analysis on individual servers. For exploratory work, R is easier for beginners. Statistical models can be written with a few lines of code. Python, on the other hand, is generally used to develop and demonstrate web applications or piping the statistical codes into a production database. Since it is an OO programming language, Python is a good tool to implement algorithms for use in production. Furthermore, the choice between the tools also depends on which tool is the most-used of the company or industry. According to DZone, Python is becoming increasingly popular in data science platforms due to its better performance in production. R, on the other hand, is designed to do data science and statistics from the bottom up, which also makes it an essential language for data science [25].

Appendix

Table 9. Processing time measurement

Project	**Tasks**	**Tool**	**IDE**	**1st M**[18]	**2nd M**	**3rd M**	**4th M**	**5th M**
1	all tasks	R	Rstudio	6,20	5,93	5,93	6,03	5,99
1	all tasks	Python	Spyder	6,33	6,23	6,25	6,13	6,20
1	without plotting model	R	Rstudio	6,16	5,89	6,19	5,99	5,80
1	without plotting model	Python	Spyder	5,66	5,80	5,63	5,59	5,53
1	without plotting mode and creating correlation matrix plot	R	Rstudio	4,20	4,22	4,30	4,09	4,38
1	without plotting mode and creating correlation matrix plot	Python	Spyder	3,69	3,50	3,27	3,33	3,40
2	all tasks	R	Rstudio	85,70	86,96	108,60	86,37	93,10
2	all tasks	Python	Spyder	270,65	260,65	386,70	262,16	261,17
2	without plotting model	R	Rstudio	67,00	67,23	67,50	66,78	67,43
2	without plotting model	Python	Spyder	24,63	24,70	24,66	24,70	24,90
2	without plotting mode and creating correlation matrix plot	R	Rstudio	23,64	23,13	23,53	23,47	23,20
2	without plotting mode and creating correlation matrix plot	Python	Spyder	10,56	10,39	10,30	10,40	10,17

[18] Measurement

Table 10. Result of the Halstead’s metrics measurements in the first project.

	R				**Python**			
Tasks	**N1**	**N2**	**n1**	**n2**	**N1**	**N2**	**n1**	**n2**
Data wrangling	25	10	16	4	33	12	13	6
Cross-tabulation	35	12	9	0	45	29	3	17
Barchart	32	13	9	8	21	11	5	8
Histogram	49	35	3	14	37	22	7	14
Boxplot	30	14	2	2	18	12	1	3
Correlation matrix	81	44	6	11	96	53	13	21
Cluster	45	28	13	15	46	26	8	15
Linear regression	23	15	4	5	51	31	4	7
Plot model	8	4	2	1	496	294	32	69
AIC	2	1	1	0	122	74	12	20
Train and test data	35	14	5	7	24	18	4	12
Random forest prediction	23	20	1	5	26	21	1	8
Total	**388**	**210**	**71**	**72**	**1,015**	**603**	**103**	**200**
Total without plot model and AIC	**378**	**205**	**68**	**71**	**397**	**235**	**59**	**111**

Table 11. Result of the Halstead’s metrics measurements in the second project.

	R				**Python**			
Tasks	**N1**	**N2**	**n 1**	**n 2**	**N1**	**N2**	**n1**	**n2**
Data wrangling	42	29	13	16	68	39	15	25
Cross-tabulation	30	13	9	5	47	29	4	20
Bar chart	198	74	7	36	222	111	5	42
Stacked bar chart	132	78	5	12	90	75	3	9
Histogram	54	30	2	6	148	117	5	35
Correlation matrix	109	70	12	35	81	43	6	13
Linear regression	15	9	4	3	34	18	5	5
Plot model	8	4	2	1	496	294	32	69
Train and test data	35	24	6	7	23	16	6	11
Random forest prediction	23	18	1	5	22	18	1	8
Total	**646**	**349**	**61**	**126**	**1231**	**760**	**82**	**237**
Total without plot model	**638**	**345**	**59**	**125**	**735**	**466**	**50**	**168**

References

[1] Barlas, P., Lanning, I. & Heavey, C., (2015), "A survey of open source data science tools", International Journal of Intelligent Computing and Cybernetics, Vol. 8 Iss 3 pp. 232 – 261.

[2] Piatetsky, G., “Four main languages for Analytics, Data Mining, Data Science,” 18 August 2014. [Online]. Available: https://www.kdnuggets.com/2014/08/four-main-languagesanalytics-data-mining-data-science.html. [Accessed 24 May 2020].

[3] Burtch Works, “2019 SAS, R, or Python Survey Update: Which Tool do Data Scientists & Analytics Pros Prefer?,” 21 August 2019. [Online]. Available: https://www.burtchworks.com/2017/06/19/2017-sas-r-python-flash-survey-results/. [Accessed 24 May 2020].

[4] Anaconda Documentation, „Installation,“ [Online]. Available: https://docs.anaconda.com/anaconda/user-guide/getting-started/. [Accessed 21 June 2020].

[5] Brittain, J., Cendon, M., Nizzi, J. & Pleis, J. (2018). Data Scientist’s Analysis Toolbox: Comparison of Python, R, and SAS Performance. *SMU Data Science Review:* Vol. 1 : No. 2 , Article 7. URL: https://scholar.smu.edu/datasciencereview/vol1/iss2/7/.

[6] Ozgur, C., Colliau, T., Rogers, G., Hughes, Z. & Myer-Tyson, B. (2017). MatLab vs. Python vs. R. *Journal of Data Science 15,* pp. 355-372.

[7] Guru99, “R Vs Python: What’s the Difference?,” [Online]. Available: https://www.guru99.com/r-vs-python.html. [Accessed 24 May 2020]

[8] Dataquest, “Python vs R: Head to Head Data Analysis,” 13 June 2019. [Online]. Available: https://www.dataquest.io/blog/python-vs-r/. [Accessed 24 May 2020]

[9] Adler, J. (2012). R in a Nutshell, 2nd Edition, *O'Reilly Media, Inc.,* Chapter 10.

[10] towards data science, “5 Lines of Code to Convince You to Learn R,” February 2019. [Online]. Available: https://towardsdatascience.com/5-lines-of-code-to-convince-you-to-learn-r-81efb2e3c836/. [Accessed 21 June 2020].

[11] Gowrishankar, S. & Veena, A. (2018). Introduction to Python Programming, CRC Press LLC. ProQuest Ebook Central, pp. 12 – 48.

[12] McKinney, W. (2013). Python for Data Analysis, *O'Reilly Media Inc.*, pp. 3 – 11.

[13] Kdnuggets, “What is the Best Python IDE for Data Science?,” November 2018. [Online]. Available: https://www.kdnuggets.com/2018/11/best-python-ide-data-science.html. [Accessed 03 June 2020]

[14] Huber, P. T. (2000). Languages for Statistics and Data Analysis, *Journal of Computational and Graphical Statistics,* vol. 9, no. 3, pp. 600-620.

[15] Abran A. (2010). Software metrics and software metrology. *IEEE Computer Society Publications, IEEE Computer Society / Wiley Partnership,* pp. 145 – 159.

[16] Jay, G., Hale, J. E., Smith, R. K., Hale, D., Kraft, N. A. & Ward, C. (2009). Cyclomatic Complexity and Lines of Code: Empirical Evidence of a Stable Linear Relationship. *J. Software Engineering & Applications,* pp. 137 – 143. URL: http://www.SciRP.org/journal/jsea/.

[17] Tashtoush, Y., Al-Maolegi, M. & Arkok, B. (2014). The Correlation among Software Complexity Metrics with Case Study. *International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970),* Vol. 4 No. 2 Iss. 15, pp. 414 – 419.

[18] Knabner, P., Reuter, B. & Schulz, R. (2019), Mit Mathe richtig anfangen. Anhang A, pp. 419.

[19] Misra, S. & Cafer, F. (2011). Estimating complexity of programs in python language. *Technical Gazette 18, 1, ISSN: 1330-3651, UDC/UDK 004.412:004.43 Python*, pp. 23 – 32.

[20] R Bloggers, “Data Science Job Report 2019,” May 2019. [Online]. Available: https://www.r-bloggers.com/. [Accessed 21 June 2020].

[21] R Documentation, “mtcars,”. [Online]. Available: https://www.rdocumentation.org/packages/datasets/versions/3.6.2/topics/mtcars/. [Accessed 21 June 2020].

[22] Sebesta, R. W. (2016). Concepts of Programming Languages, 11th Edition, *Pearson Education Limited*, pp. 30 – 41.

[23] Banghart, M. (2019). Data Wrangling Essentials. *Supporting Statistical Analysis for Research, SSCC Social science computing cooperative*. Chapter 2: Dataframe. URL: https://ssc.wisc.edu/sscc/pubs/DWE/book/preface.html/.

[24] R Documentation, „cyclocomp,” [Online]. Available: https://www.rdocumentation.org/packages/cyclocomp/versions/1.1.0/topics/cyclocomp/. [Accessed 21 June 2020].

[25] DZone, “Python vs. R: Which Should You Choose For Your Next ML Project?,” [Online]. Available: https://dzone.com/articles/python-or-r-which-should-you-choose-for-your-next/. [Accessed 19 July 2020].

[26] The R Project, [Online]. Available: https://www.r-project.org/. [Accessed 21 May 2020].

[27] Python Software Foundation, [Online]. Available: https://www.python.org/. [Accessed 21 May 2020].

[28] Python Software Foundation, "Applications for Python," [Online]. Available: https://www.python.org/about/apps/. [Accessed 21 May 2020].

R Packages

R Development Core Team (2010). R: A Language and Environment for Statistical Computing. R version 4.0.1. Available at: http://www.R-project.org/.

[tidyverse] Wickham, H. (2019). tidyverse: Easily Install and Load the 'Tidyverse'. R package version 1.3.0. Available at: http://tidyverse.tidyverse.org/.

[GGally] Schloerke, B., Cook, D., Larmarange, J., Briatte, F., Marbach, M., Thoen, E., Elberg, A., Toomet, O., Crowley, J., Hofmann, H. & Wickham, H. (2020). GGally: Extension to 'ggplot2'. R package version 2.0.0. Available at: https://CRAN.R-project.org/package=GGally/.

[expss] Demin G., Jeworutzki S. (2020). expss: Tables, Labels and Some Useful Functions from Spreadsheets and 'SPSS' Statistics. R package version 0.10.5. Available at: https://cran.r-project.org/web/packages/expss/index.html/.

[cluster] Maechler M., Rousseeuw P., Struyf A., Hubert M., Hornik K., Studer M. & Roudier P. (2019). cluster: ``Finding Groups in Data'': Cluster Analysis Extended Rousseeuw etal. R package version 2.1.0. Available at: https://svn.r-project.org/R-packages/trunk/cluster/.

[ggthemes] Arnold, J., B., Daroczi, G., Werth, B., Weitzner, B., Kunst, J., Auguie, B., Rudis, B., Wickham, H., Talbot, J. & London, J. (2019). ggthemes: Extra Themes, Scales and Geoms for 'ggplot2'. R package version 4.2.0. Available at: http://github.com/jrnold/ggthemes/.

[RColorBrewer] Neuwirth, E. (2014). RColorBrewer: ColorBrewer Palettes. R package version 1.1-2. Available at: https://CRAN.R-project.org/package=RColorBrewer/.

[reshape2] Wickham, H. (2020). reshape2: Flexibly Reshape Data: A Reboot of the Reshape Package. R package version 1.4.4. Available at: https://github.com/hadley/reshape/.

[MASS] Ripley, B., Venables, B., Bates, D. M., Hornik, K., Gebhardt, A. & Firth, D. (2020). MASS: Support Functions and Datasets for Venables and Ripley's MASS. R package version 7.3-51.6. Available at: http://www.stats.ox.ac.uk/pub/MASS4/.

[randomForest] Breiman, L., Cutler, A., Liaw, A. & Wiener, M. (2018). randomForest: Breiman and Cutler's Random Forests for Classification and Regression. R package version 4.6-14. Available at: https://www.stat.berkeley.edu/~breiman/RandomForests/.

[hrbrthemes] Rudis, B., et al. (2020). hrbrthemes: Additional Themes, Theme Components and Utilities for 'ggplot2'. R package version 0.8.0. Available at: http://github.com/hrbrmstr/hrbrthemes/.

Python Packages

Python Software Foundation. Python Language Reference, version 3.7.6. URL: http://www.python.org/.

[matplotlib] Hunter, J., et al. (2007). matplotlib: Python plotting. Available at: http://matplotlib.sourceforge.net/.

[Numpy] Oliphant, T. (2006). A guide to NumPy, *USA: Trelgol Publishing*. Available at: http://numpy.scipy.org/.

[SciPy] Jones, E., et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods, in press*. Available at: http://scipy.org/.

[SciL] Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, 12, 2825-2830. Available at: http://scikit-learn.sourceforge.net/.

[pandas] McKinney, W. (2010). pandas: a python data analysis library, *AQR Capital Management*. Available at: http://pandas.sourceforge.net/.

[Cython] Ewing, G., et al. (2011). The Cython compiler. Available at: http://cython.org/.

[IPython] Perez, F., et al. (2007). IPython: an interactive computing environment. Available at: http://ipython.scipy.org/.

[Statmodels] Perktold, J., et al. (2010). Statsmodels: Econometric and statistical modeling with python. *Proceedings of the 9th Python in Science Conference*. Available at: https://www.statsmodels.org/stable/index.html/.

[plotnine] Kibirige, H. et al. (2018). has2k1/plotnine: v0.5.0, *Zenodo*. Available at: https://zenodo.org/record/1464204#.XxAkmedCRPY/.

[seaborn] Waskom, M. et al. (2017). mwaskom/seaborn: v0.8.1, *Zenodo*. Available at: https://doi.org/10.5281/zenodo.883859/.

[itertools] Van Rossum, G. (2020). The Python Library Reference, release 3.8.2, *Python Software Foundation*. Available at: http://www.python.org/.

Declaration

I hereby declare that I have authored this thesis independently, that I have not used other than the declared sources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources. I furthermore declare that this thesis has not been submitted to any other board of examiners yet.

München, 22/07/2020

Date and location	Signature