CENG 484 Data Mining

Data Mining with Python and R

Asst. Prof. Serap Şahin

Res. Asst. Altuğ Yiğit

March 2020

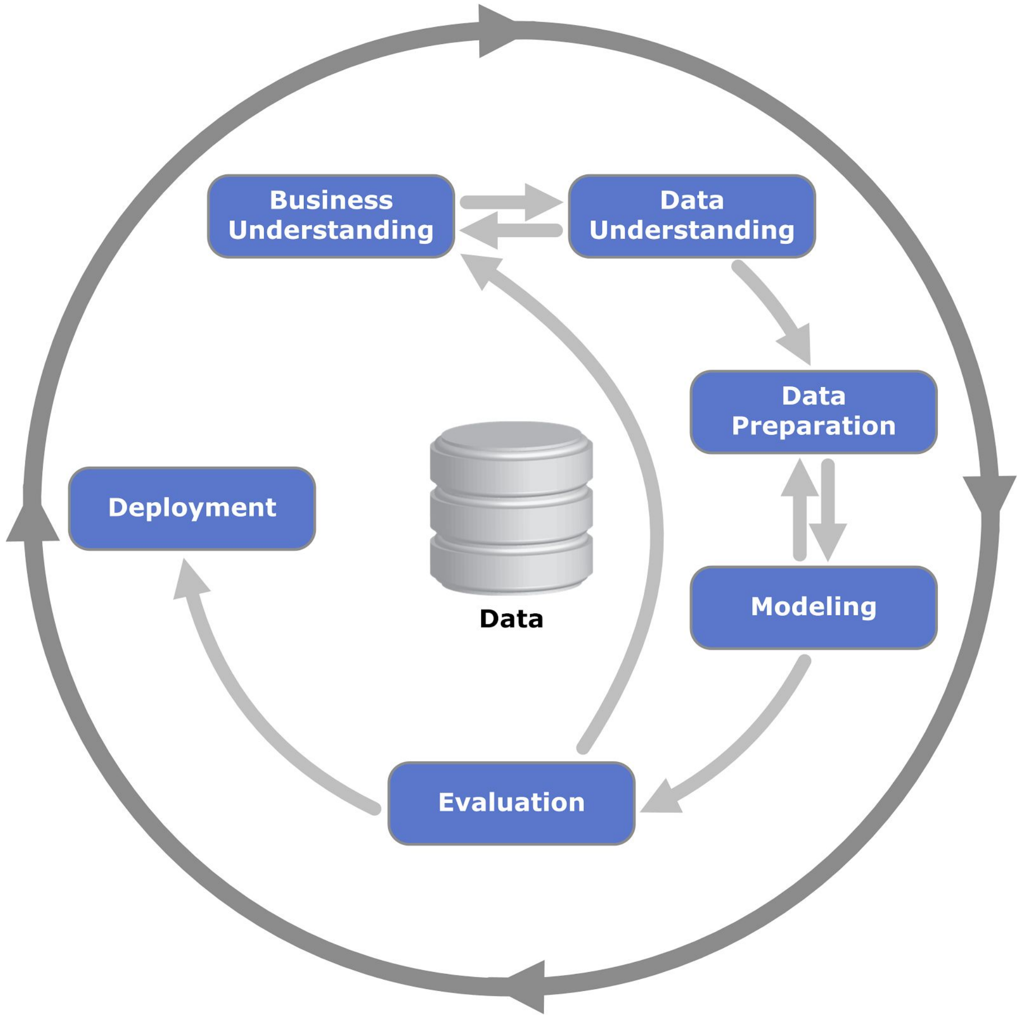
Introduction

This document has been prepared for the CENG 484 data mining course. It contains the necessary steps for processing the data with **Python** and **R**. These two programming languages are explained by comparison. Initially, document shows how to download and install Python, PyCharm, R and RStudio. After the environment is ready, it covers how to read, plot and analyze data. The main purpose of this document is to provide the necessary infrastructure to solve real world problems using mathematical data mining methods.

Data mining is the computational technique that enables us to **find patterns** and learn some rules hidden in data sets. It is an interdisciplinary field with contributions from many areas, such as statistics, machine learning, information retrieval, pattern recognition and bioinformatics. Data mining is widely used in many domains, such as retail, finance, telecommunication and social media. The main techniques for data mining include **classification** and prediction, **clustering**, outlier detection, association rules, sequence analysis, time series analysis and text mining, and also some new techniques such as social network analysis and sentiment analysis.

In real world applications, a data mining process can be broken into **six major phases**: business understanding, data understanding, data preparation, modeling, evaluation and deployment, as defined by the **CRISP-DM** (Cross Industry Standard Process for Data Mining). This phases are shown it the figure below. One of the most important distinguishing issues in data mining is **size**. One has to consider issues like computational **efficiency**, limited **memory** resources, interfaces to databases, etc. All these issues turn data mining into a highly interdisciplinary subject involving tasks

not only of typical data analysts but also of people working with databases, data visualization on high dimensions, etc.



Tools such as RapidMiner, Orange, Weka, Knime and **programming languages** such as Python, R, Julia, Java are widely used for the application of data mining methods. This course focuses on Python and R, the most used languages.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. Python was developed by Guido van Rossum in the late 1980s and early 1990s at the National Research Institute for Mathematics and Computer Science in the

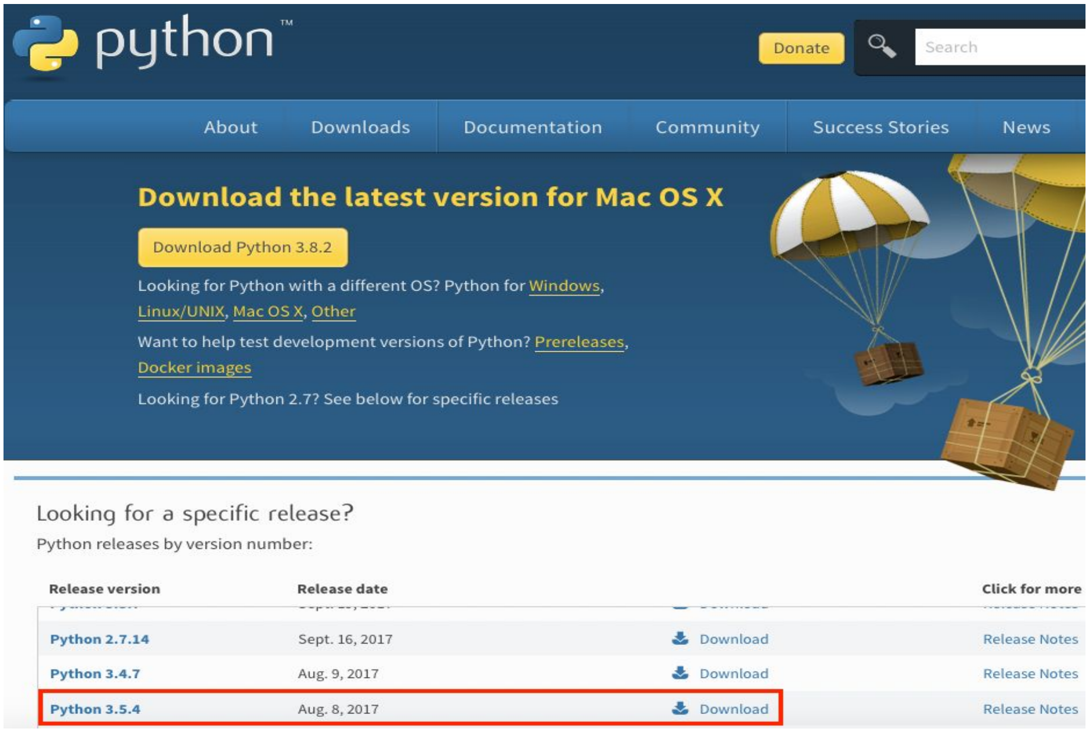
Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available **under the General Public License** (GPL). Python 1.0 was released in November 1994.

R is a programming language and an environment for statistical computing. It is based on the computer language S, developed by John Chambers and others at Bell Laboratories in 1976. R was initially developed in 1996 by Ihaka and Gentleman, both from the University of Auckland, New Zealand. The source code of every R component is freely available for inspection and adaptation. This fact allows you to check and test the reliability of anything you use in R. Many classical and modern **statistical techniques** have been implemented in the R environment. A few of these are built into the **base** R environment, but many are supplied as **packages**.

Part 1 - Preparing the Environment

1.1 Installing Python

Python is available for your system from the website (https://www.python.org/downloads/). In this course, **Python 3.5.4** version will be used, so this version may be selected from the website before downloading.



If the installation is successful, when the command "**python --version**" is written from the console for **Unix** based systems, it gives information about the Python version. **Windows** users will need to set an environment variable to use **Python from the command line** as figures below. Find where you install Python onto your computer; the default location is "C:\Python35".

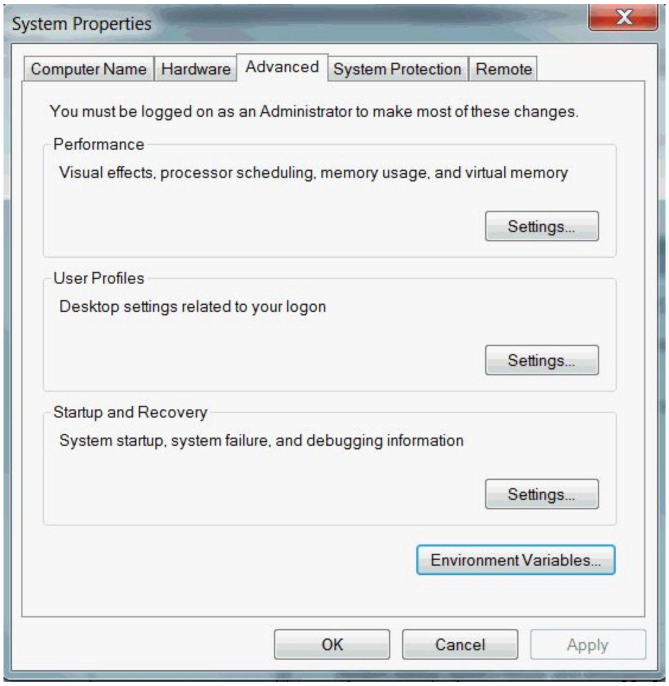
- Right-click the Computer icon and choose Properties,

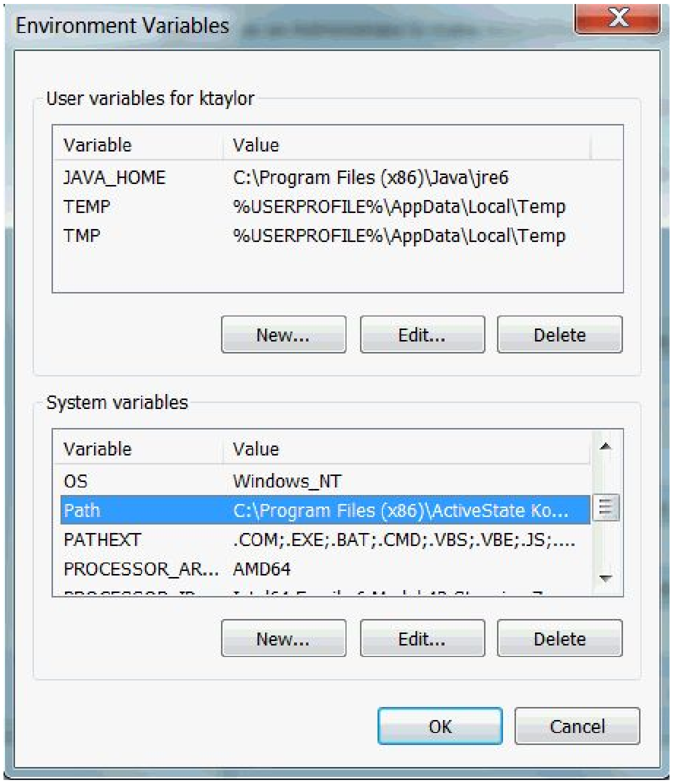
- Choose Advanced system settings,

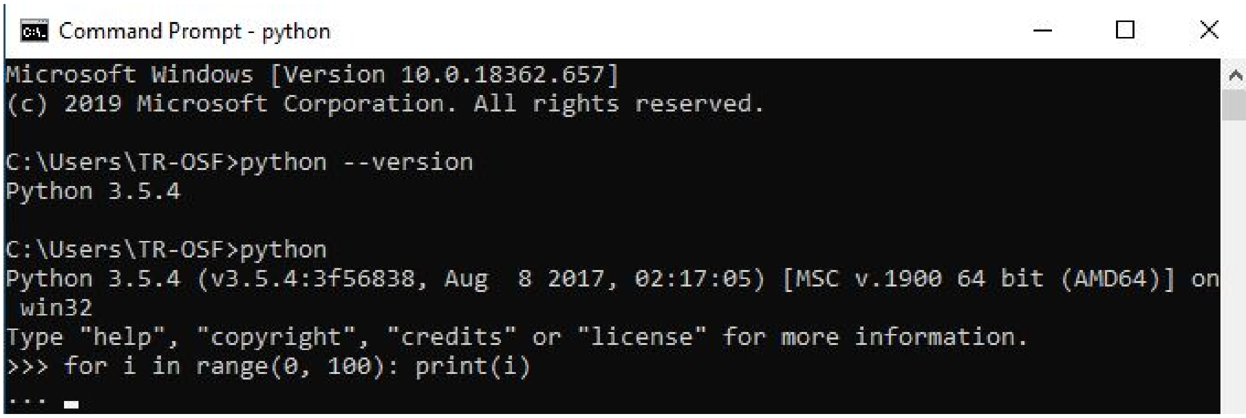
- On the Advanced tab, click Environment Variables,

- Click New to create a new environment variable. Click Edit to modify an existing environment variable,

- Add "C:\Python35" and "C:\Python35\Scripts" to path.

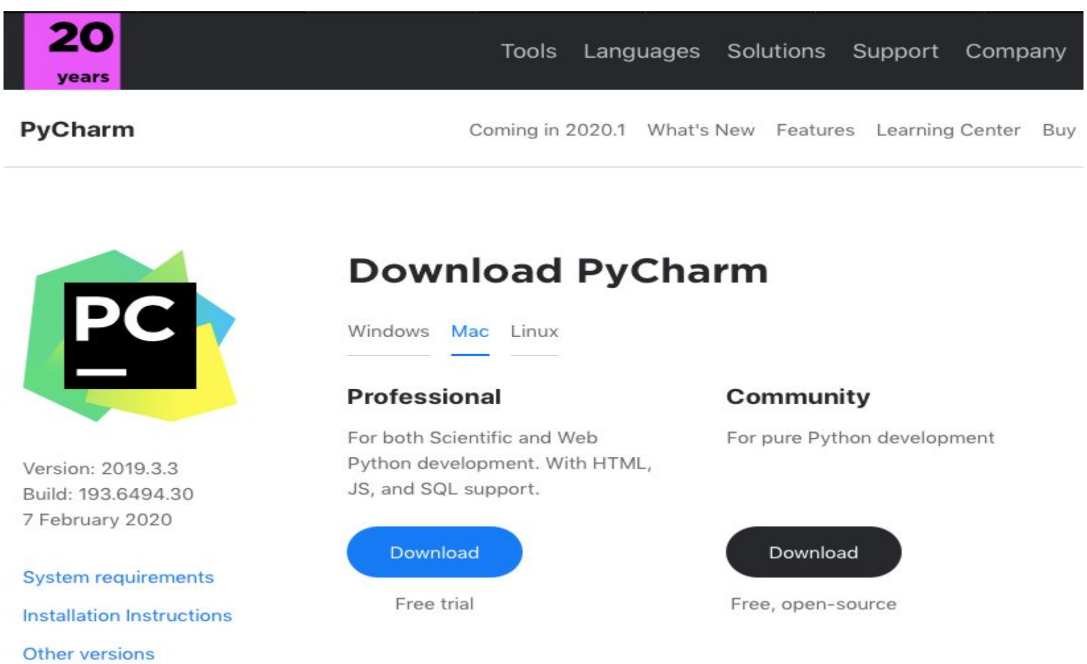

System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your logon
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply


Environment Variables
User variables for ktaylor
Variable
Value
JAVA_HOME
C:\Program Files (x86)\Java\jre6
TEMP
%USERPROFILE%\AppData\Local\Temp
TMP
%USERPROFILE%\AppData\Local\Temp
New...
Edit...
Delete
System variables
Variable
Value
OS
Windows_NT
Path
C:\Program Files (x86)\ActiveState Ko...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_AR...
AMD64
New...
Edit...
Delete
OK
Cancel

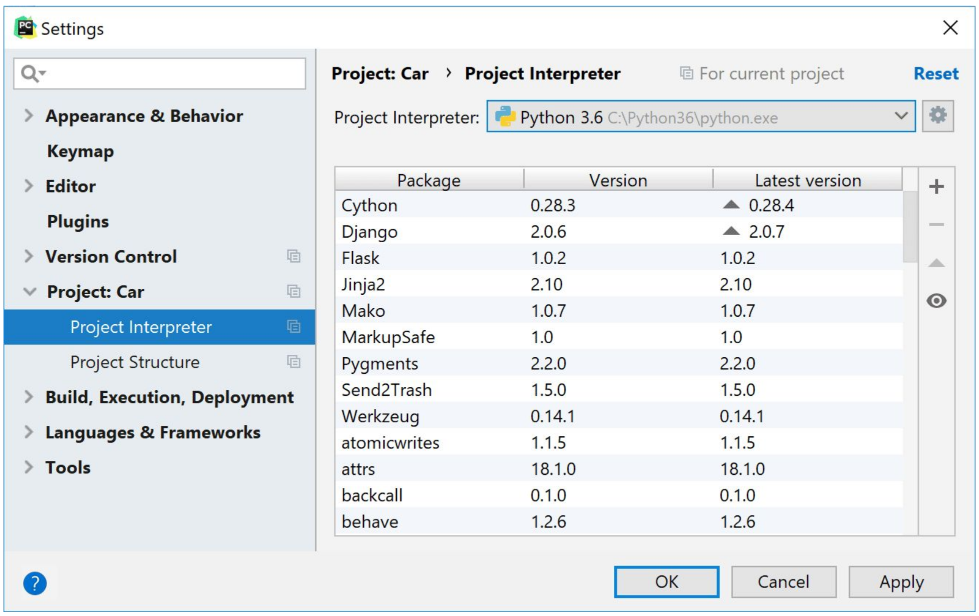


1.2 Installing PyCharm

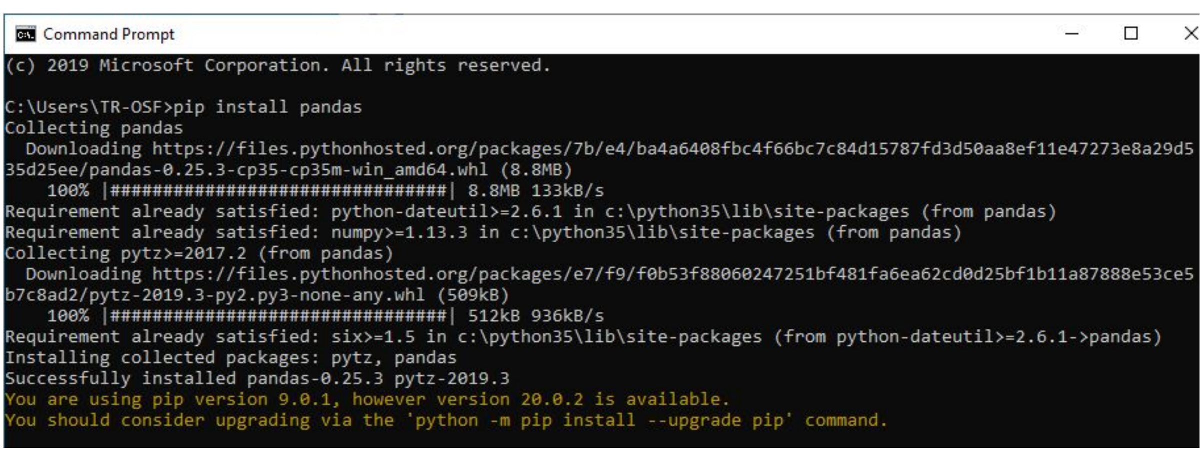
PyCharm is a cross-platform IDE that provides consistent experience on the Windows, macOS, and Linux operating systems. PyCharm is available in three editions: Professional, Community, and Edu. The Community and Edu editions are open-source projects and they are free. In this course, the **PyCharm Community** version will be used since it is sufficient. It can be downloaded from website (https://www.jetbrains.com/pycharm/download/).



If you’re on the Welcome screen, click Create New Project. In this screen you may create virtual environment, if you would like to **use environment** that is **on your computer**, you should select "**Existing interpreter**". To manage **Python packages** for the project interpreter in **PyCharm**, select the **Project Interpreter** page in the project **Settings/Preferences** or select Interpreter Settings in the Python Interpreter widget.

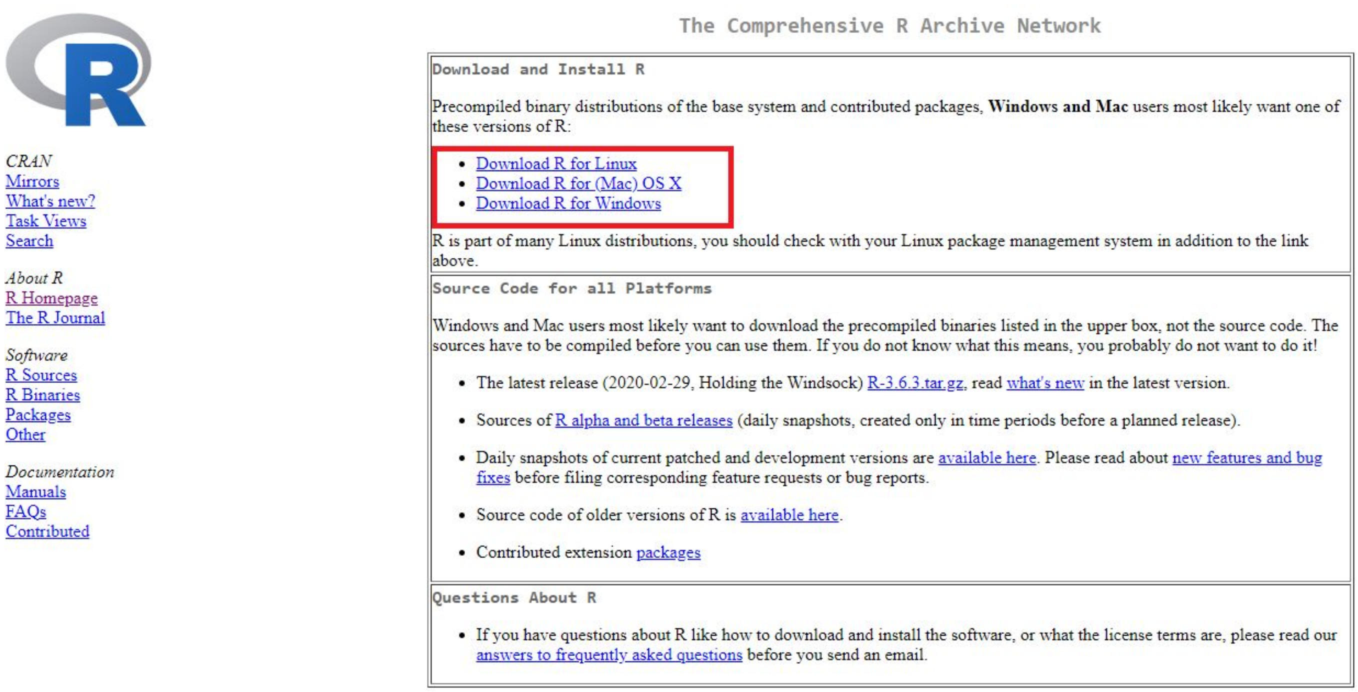


You can use **pip on console** to install packages from the Python Package Index and other indexes, pip is the package installer for Python. You may type package name that you want to install as "**pip install package_name**".



1.3 Installing R

In order to install R in your system, the easiest way is to obtain a **binary distribution** from the R website (https://cran.r-project.org). This site is referred as **CRAN** (Comprehensive R Archive Network). Most users download and install a binary version from the site. Binary version is a different type of version that has been translated (by compilers) into machine language for execution on a given operating system. R is designed to be very **portable**: it will run on Microsoft Windows, Linux, Solaris, Mac OSX, and other operating systems, there are different binary versions for each. For this course, R 3.5.2 should be downloaded as shown in figures below.



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

R for Windows

Subdirectories:

base	Binaries for base distribution. This is what you want to **install R for the first time**.
contrib	Binaries of contributed CRAN packages (for R >= 2.13.x; managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables.
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 2.13.x; managed by Uwe Ligges).
Rtools	Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

R-3.6.3 for Windows (32/64 bit)

Download R 3.6.3 for Windows (83 megabytes, 32/64 bit)
Installation and other instructions
New features in this version

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server. You will need a version of md5sum for windows: both graphical and command line versions are available.

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A stable link which will redirect to the current Windows binary release is
<CRAN MIRROR>/bin/windows/base/release.htm.

Last change: 2020-02-29

Previous Releases of R for Windows

This directory contains previous binary releases of R to run on Windows 95, 98, ME, NT4.0, 2000 and XP or later on Intel/clone chips.

The current release, and links to development snapshots, are available here. Source code for these releases and others is available through the main CRAN page.

In this directory:

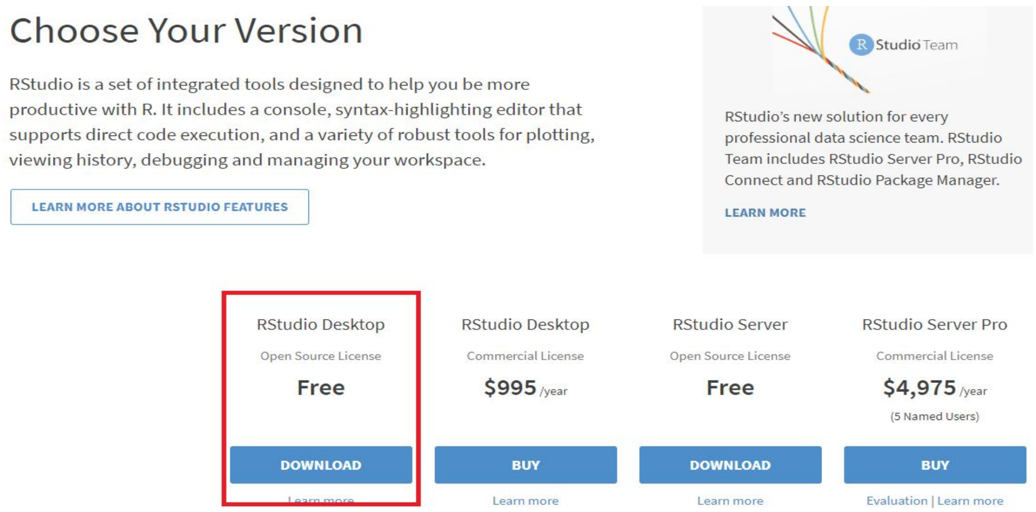
R 3.6.2 (December, 2019)
R 3.6.1 (July, 2019)
R 3.6.0 (April, 2019)
R 3.5.3 (March, 2019)
R 3.5.2 (December, 2018)
R 3.5.1 (July, 2018)
R 3.5.0 (April, 2018)
R 3.4.4 (March, 2018)
R 3.4.3 (November, 2017)
R 3.4.2 (September, 2017)
R 3.4.1 (June, 2017)
R 3.4.0 (April, 2017)
R 3.3.3 (March, 2017)

To run R in **Windows** you simply double-click the **R application** that is called like "R_x64_3.5.2.exe". In **Unix** versions you should **type R** at the operating **system prompt**. Both will bring up the R console with its prompt ">". Whenever you see this **R command prompt**, ">" you can interpret it as R waiting for you to enter a command. You type in the commands at the prompt and then press the enter key to ask R to execute them. This may or may not produce some form of output and then a new prompt appears. At the prompt you may use the arrow keys to browse and edit previously entered commands. This is handy when you want to type commands similar to what you have done before as you avoid typing them again.

If you want to **quit R** you can issue the command "**q()**" at the prompt. You will be asked if you want to save the current workspace. You should answer yes only if you want to resume your current analysis at the point you are leaving it, later on.

1.4 Installing RStudio

RStudio is a free and open source IDE (integrated development environment) for R. It is a very useful and powerful tool for R programming. **RStudio Desktop** open source edition can be downloaded from website as shown in figure below (https://rstudio.com/products/rstudio/download/).



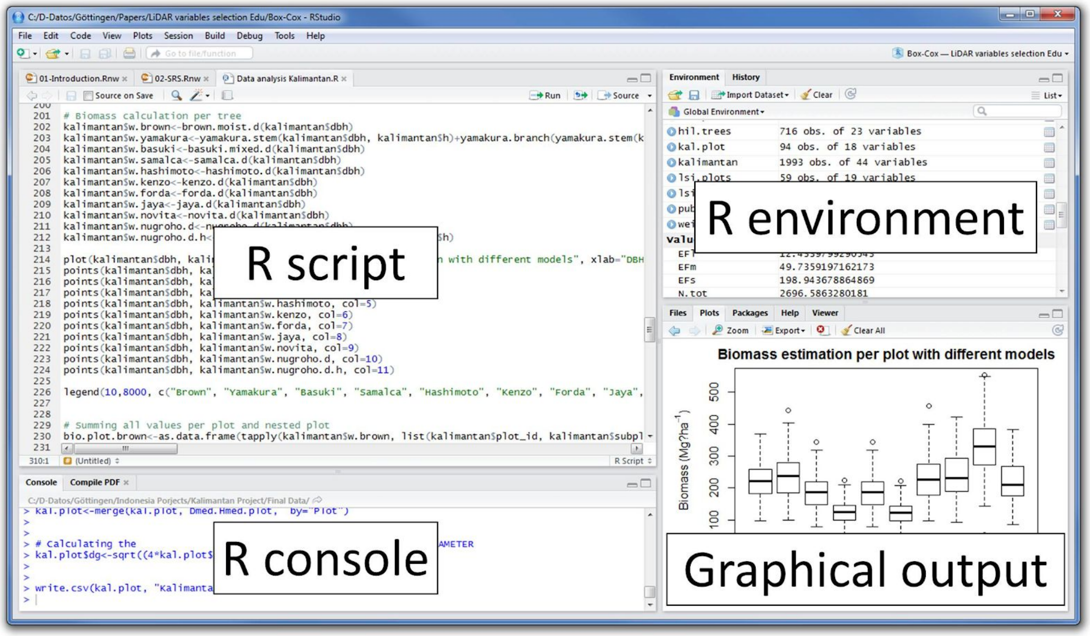
It is free of charge and can run on various operating systems like Windows, Mac and Linux. Your operating system will be automatically recognized by this website. Once the installation of R has completed successfully, run the RStudio installer. When RStudio is launched for the first time, you can see a similar window in figure below. There are **four** panels:

• **Source panel (top left)**, which shows your R source code.

• **Console panel (bottom left)**, which shows outputs and system messages displayed in a normal R console;

• **Environment/History/Presentation panel (top right)**, whose three tabs show respectively all objects and function loaded in R, a history of submitted R code, and Presentations generated with R;

• **Files/Plots/Packages/Help/Viewer panel (bottom right)**, whose tabs show respectively a list of files, plots, R packages installed, help documentation and local web content.

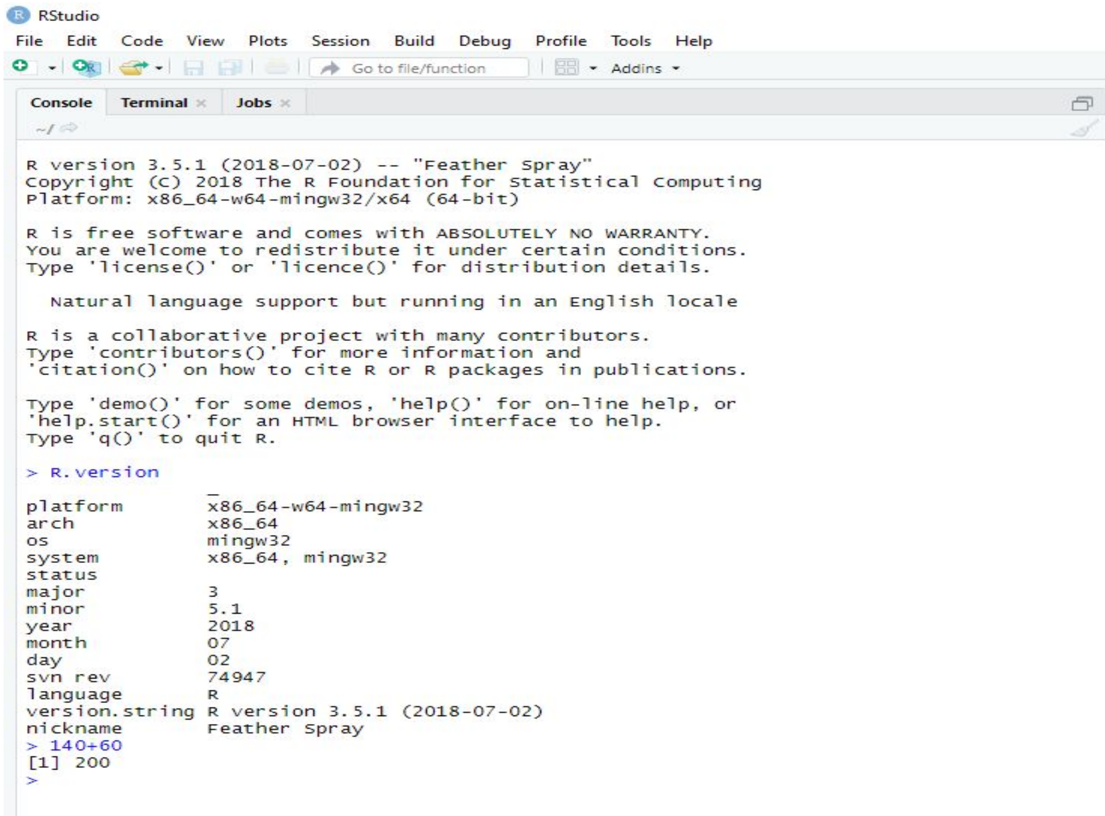


Note: If you cannot see the source panel, you can find it by clicking menu “File”, “New File” and then “R Script”. You can run a line or a selection of R code by clicking the “Run” bottom on top of source panel, or pressing “Ctrl + Enter”.

To create a **new project**, click the“Project” button at the top right corner and then choose “New Project”. After that, select “create project from new directory” and then “Empty Project”. After typing a directory name, which will also be your project name, click “Create Project” to create your project folder and files. After that, create three folders as:

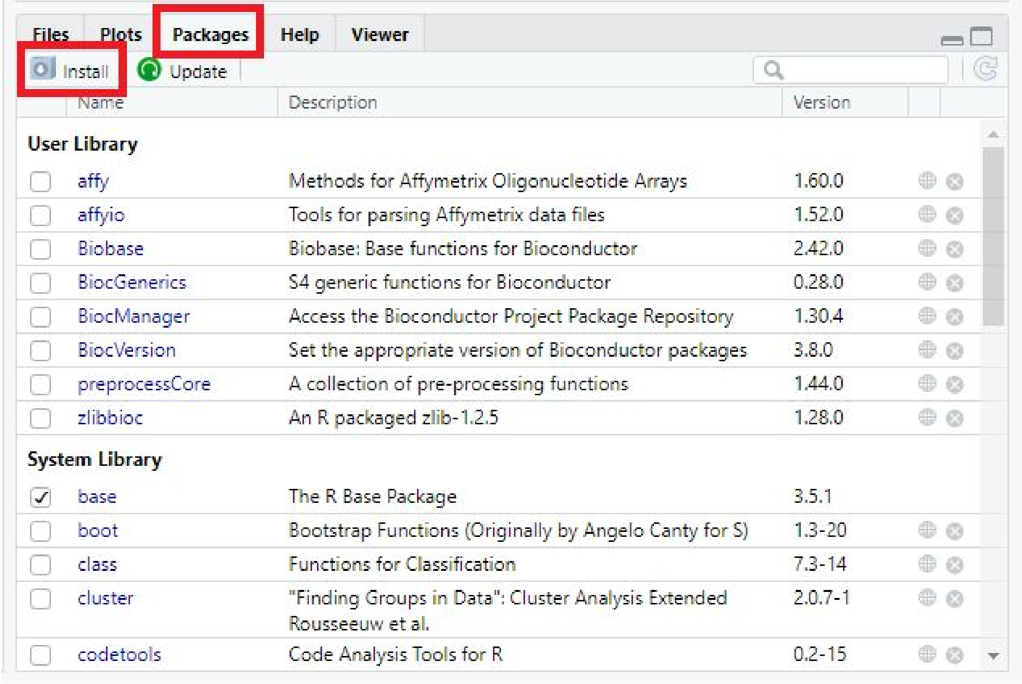
- **code**, where to put your R source code
- **data**, where to put your datasets;
- **figures**, where to put produced diagrams.

"**R.version**" command may be written from the console screen to check the **R version via RStudio**. Some **arithmetic** operations such as “140+60” can be done on the console to check whether R Studio is properly working or not as shown in figure below. The [1] that prefixes the output indicates that this is item 1 in a vector of output.



To install a **R package**, come to the "**Packages**" tab and select "**Install**". Suppose we downloaded the ggplot2 package, "**library (ggplot2)**" command may be written to check that this library is installed. If there is no message like "**Error in library(x)** : there is no package called x", it was successfully installed.

As a second option, the "**install.packages("ggplot2")**" command may be written manually on the console to install packages.



Part 2 - Data Types

2.1 Python

Python built in main data types are: Text (String), Numeric (Integer, Float, Long), Complex, Boolean (True / False).

The most common data structure in Python is known as **list**.

```
a = [1.8, 4.5, 6.5]   #float

b = [1 + 2*i, 3 - 6*i] #complex

d = [23, 44, 65]   #integer

e = [True, False, True, False, True] #boolean
```

Python also includes different data structures, which are **dictionary** and **tuple**. Python **dictionary** is a collection of **key and value pairs** separated by a colon (:).

```
dict = {'no':'20151231212', 'name':'Ali', 'age':20 } # dictionary

print("Student no:", dict['no'])

print("Student name:", dict['name'])

print("Student age:", dict['age'])
```

Iterating over the elements of a **tuple is faster** compared to iterating over a **list**. We can **have tuple** of same type of data items as well as **mixed type** of data items.

```
d = (23, 44, 65) # tuple

e = (23, 60.4, "Ali") # mixed tuple

f = ("Ali", [80, 85, 90]) # mixed tuple
```

2.2 R

Everything you see or create in R is an **object**. R has 5 basic classes of objects: Character, Numeric (Real or Decimal Numbers), Integer (Whole Numbers), Complex, Logical (True / False). The most common data structure in R is known as **vector**. You can create an empty vector using vector(). A vector contains object of same class. You can also create vector using c() or concatenate command.

```
> a <- c(1.8, 4.5)   #numeric

> b <- c(1 + 2i, 3 - 6i) #complex

> d <- c(23, 44)   #integer

> e <- vector("logical", length = 5)
```

R contains **factors** that provide an easy and compact form of handling **categorical (nominal)** data. Factors have **levels** and particularly useful in datasets where you have nominal variables with a fixed number of possible values. R stores these values **as numeric codes** that are considerably more memory efficient.

Suppose you have a vector with the gender of ten individuals. We can transform this vector into a factor:

```
>g <- c("f", "m", "m", "m", "f", "m", "f", "m", "f", "f")

> g <- factor(g)

>g

Output:

 [1] f m m m f m f m f f

Levels: f m
```

To find out **what data types** are, "**class()**" function is used **in R** and "**type()**" function is used **in Python**.

```
> print(class(a)) # R

[1] "numeric"

print(type(a)) # Python

<type 'list'>
```

Part 3 - Reading Data

Public datasets are available in the **R base** and **Python Scikit-Learn** library. You can check built-in datasets in R with "**data()**" command. We will use the **iris flower dataset** as an example. It is often used for testing out machine learning algorithms and visualizations. This dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant. You can show the iris data by typing "**iris**" keyword in R.

In Python:

```python
import pandas as pd
from sklearn import datasets

# This creates a Sklearn bunch
data = datasets.load_iris()
# Convert to Pandas dataframe
iris = pd.DataFrame(data.data, columns=data.feature_names)
```

In R:

```r
# it can be obtained only typing iris
iris
```

You can obtain information about data, show row and column (feature) sizes with **str()** and **dim()** in R, **info()** and **shape** in Python. First 3 and last 3 rows are listed with **head(3)** and **tail(3)** commands.

In Python:

```python
print(iris.info())
print(iris.shape)
print(iris.head(3))
print(iris.tail(3))
```

In R:

```
str(iris)
dim(iris)
head(df, 3)
tail(df, 3)
```

Output:

```
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

> dim(iris)
[1] 150   5

> head(df, 3)
  Transaction_date  Product Price Payment_Type              Name
1      1/2/09 6:17 Product1  1200   Mastercard          carolina
2      1/2/09 4:53 Product1  1200         Visa           Betina Parkville
3     1/2/09 13:08 Product1  1200   Mastercard Federica e Andrea Astoria

> tail(df, 3)
    Transaction_date  Product Price Payment_Type   Name
996      1/1/09 4:24 Product3  7500         Amex Pamela Skaneateles
997     1/8/09 11:55 Product1  1200       Diners  julie
998    1/12/09 21:30 Product1  1200         Visa Julia  Madison
```

If we want to get some **subsets** in the data, we can use the **indexes** or **subset** function in R. If we want to obtain data with the number of sepal length is greater than 5, the code can be written as follows.

In Python:

```
print(len(iris[iris['sepal length (cm)']>5.0]))
```

In R:

```
nrow(iris[iris$Sepal.Length>5.0,])

# with subset
nrow(subset(iris, iris$Sepal.Length>5.0))
```

Output:

```
118
```

Total and **average** values can be calculated with the code below.

In Python:

```
def calc_mean(col_name):

  result = iris[col_name].mean()

  return result

def calc_mean_2(col_name):

  result = iris[col_name].sum() / len(iris[col_name])

  return result

print(calc_mean("sepal width (cm)"))
print(calc_mean_2("sepal width (cm)"))
```

In R:

```
calc_mean <- function(col_name){

  result <- mean(col_name)

  return(result)
}
```

```
calc_mean_2 <- function(col_name){

  result <- sum(col_name) / length(col_name)

  return(result)
}

calc_mean(iris$Sepal.Width)
calc_mean_2(iris$Sepal.Width)
```

Output:

```
3.057333
3.057333
```

Comma-separated values (CSV) data is widely used in data mining. **Download** the Amazon sales **CSV data** which was prepared for this course, called **AmazonSales.csv**. A command like the following can be written in both R and Python to read the data. Data will be obtained in **dataframe** structure.

Exercise 1. Read the data separately in Python and R.

In Python:

In R:

Exercise 2. How many countries are there in the data? Write the output and commands separately in Python and R.

In Python:

In R:

Output:

Exercise 3. Find the total sales amount for each product. Write the output and commands separately in Python and R.

In Python: In R: Output:

Exercise 4. Find the total fee charged with Mastercard. Write the output and commands separately in Python and R.

In Python: In R: Output:

Part 4 - Visualization Data

Visualization may be required to understand the data. In this way, information is obtained about data. **Matplotlib** library is widely used to visualize data with **Python** whereas **base** plotting functions can be used for visualizing with **R**. **Box plot** is a type of graph that shows how the values in the data are **spread out**. Data is expressed in **quarters** as minimum, first quartile (25%), median, third quartile (50%), and maximum values.

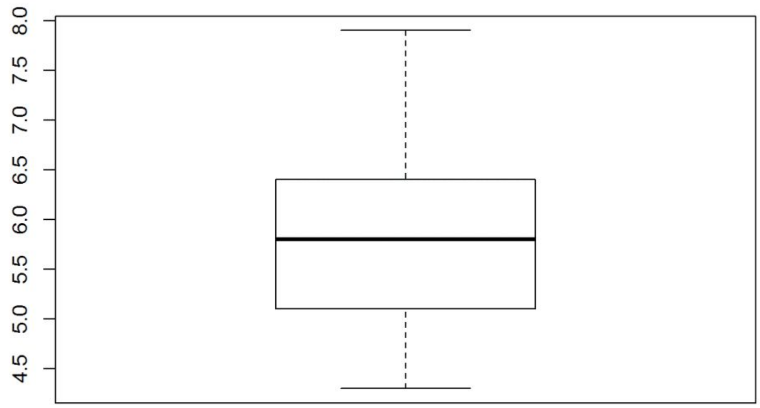
In Python:

```
import matplotlib.pyplot as plt

img=plt.boxplot(iris['sepal length (cm)'])
plt.show(img)
```

In R:

```
boxplot(iris$Sepal.Length)
```

Output:



Scatter plot is used to determine the relationship between two different variables. It can be seen whether there is a direct **relationship between** the **two variables** and how strong this relationship is. In addition, **changes in time series** data can be observed by "**type**" information in R and “**plot()**” function in Python.

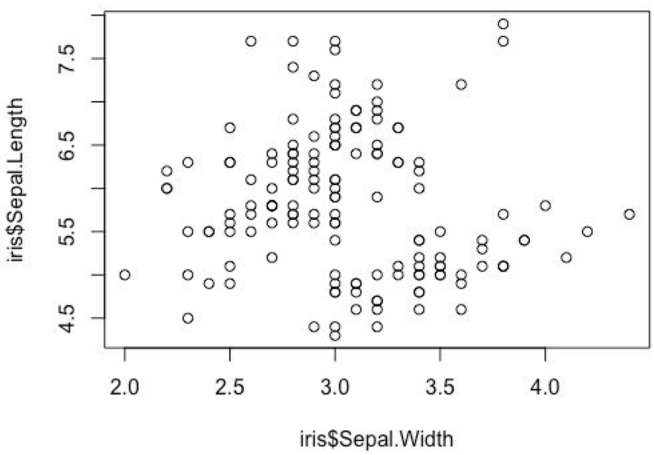
In Python:

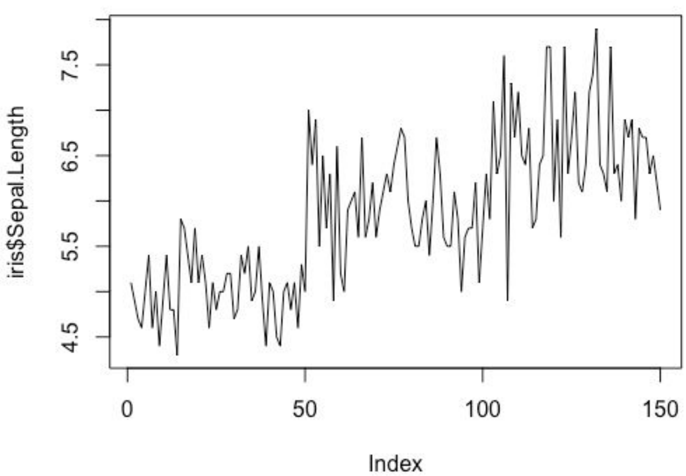
```
import matplotlib.pyplot as plt

img=plt.scatter(iris["sepal width (cm)"], iris["sepal length (cm)"])
plt.show(img)

# Time series plot
img=plt.plot(iris["sepal length (cm)"])
plt.show(img)
```

In R:

```
plot(iris$Sepal.Width,iris$Sepal.Length)

# Time series plot
plot(iris$Sepal.Length, type = 'l')
```

Output:





Histogram plot is used to determine the **distribution** and **frequency** of the data. It divides values into **groups** and returns the **frequency**.

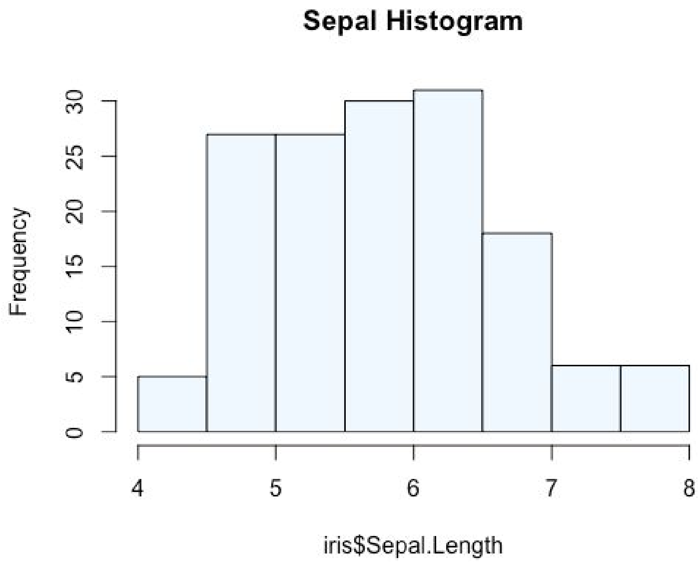
In Python:

```
import matplotlib.pyplot as plt

img=plt.hist(iris["sepal length (cm)"])
plt.title('Sepal Histogram')
plt.show()
```

In R:

```
hist(iris$Sepal.Length, col="aliceblue", main="Sepal Histogram")
```

Output:



Exercise 5. Draw scatter prices by product and compare prices from Amazon Sales CSV file. Write the output and commands separately in Python and R.

In Python:

In R:

Output:

Exercise 6. Describe how you would create visualizations to display information that describes the following types of systems.

(a) Computer **networks**. Be sure to include both the static aspects of the network, such as connectivity, and the dynamic aspects, such as traffic.

(b) The **distribution** of specific plant and animal species around the world for a specific moment in time.

(c) The use of computer **resources**, such as processor time, main memory, and disk, for a set of benchmark database programs.

(d) The change in **occupation** of workers in a particular country over the last thirty years. Assume that you have yearly information about each person that also includes gender and level of education.

Answer: a) b) c) d)

Part 5 - Applying Data Mining

In this section, we will write Python and R code to **classify** species in the iris dataset. In the reading data section, the subject of how the data will be obtained is mentioned. The data obtained should be prepared for the application of mathematical methods. **Features** and **classes** should be determined before performing classification. Iris dataset consists of **3 labels** (setosa, virginica, versicolor), **4 features** (sepal length cm, sepal width cm, petal length cm, petal width cm), **150 samples**.

In Python:

```python
from sklearn import datasets

# Load built-in dataset
iris = datasets.load_iris()

x = iris.data
y = iris.target
y_labels = iris.target_names

print(y)
print(y_labels)
```

In R:

```r
data(iris)

x <- iris[1:4]
y <- iris$Species
y_labels <- levels(iris$Species)

print(y)
print(y_labels)
```

Output:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
['setosa' 'versicolor' 'virginica']
```

Standardization or **normalization** will be applied to make an entire set of values have a particular property. If **different variables** are to be combined in some way, then such a transformation is often necessary to avoid having a variable with **large values dominate** the results of the calculation. For example; The **Gaussian normalization** (val = (x-mean(x))/std(x)) creates a new variable that has a mean of 0 and a standard deviation of 1. **Min-max** normalization is applied in the codes below.

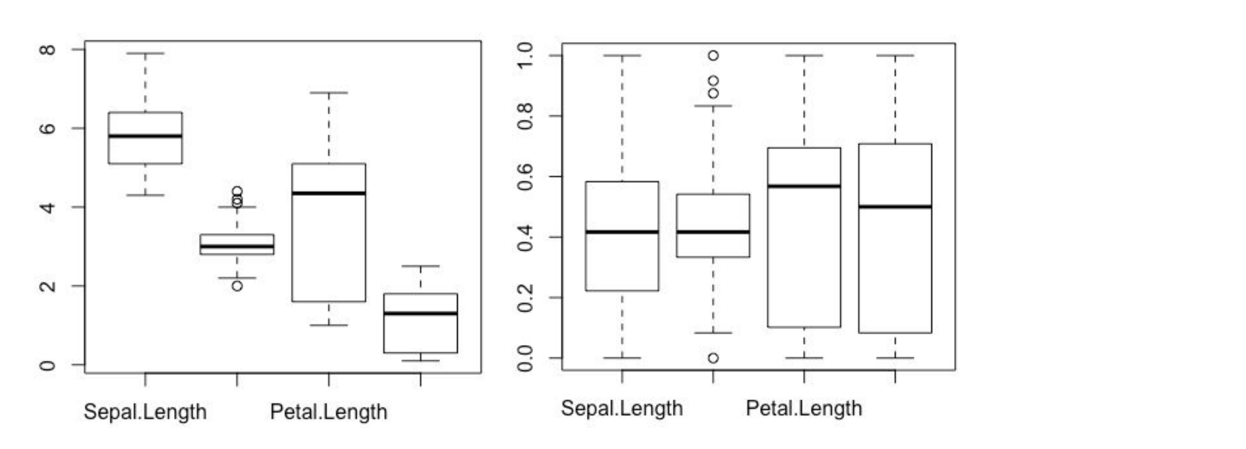
In Python:

```
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

min_max_scaler = MinMaxScaler()
x_norm = min_max_scaler.fit_transform(x)

img=plt.boxplot(x)
plt.show()
img=plt.boxplot(x_norm)
plt.show()
```

In R:

```
# Build your min-max function
normalize <- function(x) {
  num <- x - min(x)
  denom <- max(x) - min(x)
  return (num/denom)
}

# Normalize the data
x_norm <- as.data.frame(lapply(x, normalize))

# Check the normalization
boxplot(x)
boxplot(x_norm)
```

Output:



Data will be divided into training and test sets to perform training and **evaluate the performance** of the models developed.

In Python:

```
from sklearn.model_selection import train_test_split

# set the random state to make reproducible
x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=.25, random_state=123)

# Check data size
print(x_train.shape)
print(x_test.shape)
```

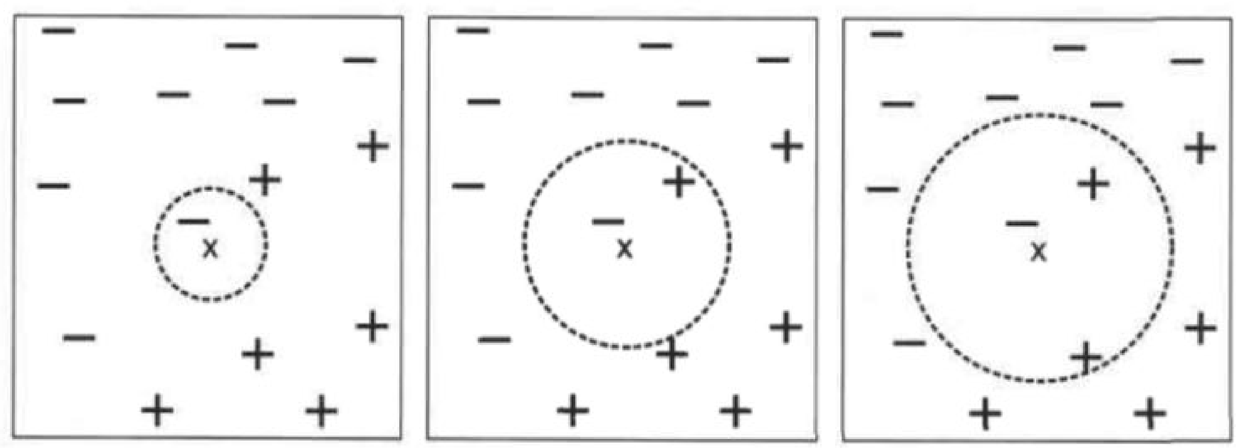
In R:

```
# set the seed to make reproducible
set.seed(123)

train_ind <- sample(seq_len(nrow(x_norm)), size = floor(0.75 * nrow(x_norm)))

x_train = x_norm[train_ind,]
x_test = x_norm[-train_ind,] # all indices except train indexes
y_train = y[train_ind]
y_test = y[-train_ind]

# Check data size
dim(x_train)
dim(x_test)
```

Output:

(112, 4)
(38, 4)

KNN (K-Nearest-Neighbor) model will be implemented in two language. The algorithm computes the **distance** (or similarity) between each test example and all the training examples to determine its nearest-neighbor list. We will use the **knn()** function in **R** and **KNeighborsClassifier()** function in **Python**, both of them use the **Euclidian distance** measure in order to find the k-nearest neighbours to your new, unknown instance. Here, the **k parameter** is one that you set yourself.



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

In Python:

```
from sklearn import neighbors
from sklearn.metrics import accuracy_score, confusion_matrix

classifier=neighbors.KNeighborsClassifier(n_neighbors=5)

classifier.fit(x_train,y_train)

iris_pred = classifier.predict(x_test)

print(accuracy_score(y_test, iris_pred))
```

```
print(confusion_matrix(y_test, iris_pred))
```

In R:

```
install.packages("Metrics") # For accuracy
install.packages("caret") # For confusion matrix
install.packages('e1071', dependencies=TRUE) # For caret

library(class) # For knn()
library(Metrics)
library(caret)

iris_pred <- knn(train = x_train, test = x_test, cl = y_train, k=5)
iris_pred

accuracy(iris_pred, y_test)

# Confusion matrix
table(iris_pred, y_test)

# Confusion matrix with overall statistics
confusionMatrix(iris_pred, y_test)
```

Output:

```
0.9736842105263158

[[16  0  0]
 [ 0  7  1]
 [ 0  0 14]]
```

Part 6 - Assignment

In this assignment, the **following tasks** will be done:

- Write the most appropriate answers to the **exercises (Exercise 1, 2, 3, 4, 5, 6)** in the document.
- In addition, you will download the **breast cancer dataset** and write the Python and R code that can analyse the data.
- Which **visualization** techniques would be more appropriate for analyzing this data (breast cancer)? Apply many different visualization techniques.
- What should we do to **find outliers** in the data? Explain and apply your solution.
- You should **decide** whether **pre-processing** is necessary for this data. Explain which technique is appropriate for the data.
- What **type of data mining** (classification, clustering, etc.) you think would be relevant? Apply the data mining technique that you decide is relevant.

Notes:

- You can download the dataset as a CSV file from **UCI** (University of California, Irvine), "wdbc.data" will be used for analysing (https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data).
- Please upload your **Python, R codes** and **your report** (with answers to questions from **exercises and tasks**) to CMS until **20 March 2020** 23:00. You should upload a zip file, and file name "Student_Number_Name.zip".

References

- R and Data Mining: Examples and Case Studies, Yanchang Zhao, 2015.
- Data Mining with R, Learning with Case Studies, Chapman & Hall/CRC Data Mining and Knowledge Discovery Series SERIES EDITOR, Vipin Kumar.
- Learning Data Mining with Python, Second Edition, Robert Layton Book.
- http://www.cs.ukzn.ac.za/~hughm/dm/content/slides01.pdf
- https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf
- https://shiring.github.io/r_vs_python/2017/01/22/R_vs_Py_post
- https://www.datacamp.com/community/tutorials/machine-learning-in-r
- https://www.analyticsvidhya.com/blog/2016/02/complete-tutorial-learn-data-science-scratch/
- https://docs.python.org/3/library/datatypes.html
- https://www.tutorialspoint.com/python3/python_tutorial.pdf
- https://www.jetbrains.com/help/pycharm/installation-guide.html
- https://beginnersbook.com/