OULU UNIVERSITY OF
APPLIED SCIENCES

Nibesh Khadka

# GENERAL MACHINE LEARNING PRACTICES USING PYTHON

# GENERAL MACHINE LEARNING PRACTICES USING PYTHON

Nibesh Khadka
Bachelor’s Thesis
Spring 2019
Information Technology
Oulu University of Applied Sciences

# ABSTRACT

Oulu University of Applied Sciences
Information Technology

Author: Nibesh Khadka

Title of the bachelor’s thesis: General Machine Learning Practices Using Python
Supervisor: Kari Jyrkkä
Term and year of completion: Spring 2019 Number of pages: 80

Machine Learning (ML), is a process of teaching an algorithm to learn, feeding data. Algorithms try to find patterns from data to generalise a rule or relation to predict future unseen instances. Some of the popular results of machine learning systems are Google Translate, YouTube’s Video Recommendation System, Movie Recommendation System.

The aim of the thesis was to introduce the readers to the concept of ML and phases in the ML model development process as well as their implementation in the Python programming language.

Different categories of ML with examples, typical phases in an ML modelling has been explained in theory in the thesis. In addition, Pre-processing Data, Training and Testing Models, Optimization of Model in the Supervised Machine Learning has been further elaborated using Linear Regression and K-Nearest Neighbor (KNN) as examples in python programming.

The paper, as a result, can act as a starting tool and conventional guide to beginner ML practitioner for any ML algorithms other than presented in the paper. People with knowledge of python can further benefit for the conventions and alternatives presented in the coding section of the paper.

Keywords: Machine Learning, ML Phases, Supervised Machine Learning, Python Programming, Guide, Linear Regression, KNN

# PREFACE

## Acknowledge

I would like to thank my supervisor Kari Jyrkkä, a senior lecturer at Oulu University of Applied Sciences for his continuous support and feedbacks which helped me improve my paper.

## Who is this paper for?

The paper is for beginner level Machine Learning Practitioners and Machine Learning Enthusiasts. Programmers with knowledge in python programming can get the most out of the course. However, regardless of background paper will still prove helpful as a guide in ML journey of a beginner.

## What is the paper not about?

The thesis is meant to for beginner Level Machine Learning Engineer. However, the thesis is not a tutorial of the Python programming language and its libraries. Even though several modules and methods used in codes are explained to the best of my ability, readers are expected to have a fluent knowledge of Python (version 3) and its packages.

## What Can you do to get the best out of this paper?

The thesis provided links and references to very useful documentation and books that were used during development. It is instructed to the reader to visit those links and read the topics. For the code part, the screenshots of some portions of codes are used and explained in the thesis. However, to get best out of the paper, it is instructed to visit the full version of the code at GitHub from the link provided and run the codes in IDE with the chapters. Since Jupyter Notebook has been used as IDE (Integrated Development Environment), it will be most efficient to use the same IDE.

## Note

Due to my programming background and experience in more than one programming language, I have used the terms like line and Cell number and Input Line interchangeably those all refer to the same thing a cell in Jupyter Notebook, or sometimes parameter and attributes interchangeably while explaining the codes. Moreover, while checking codes, please do not panic or try to match cell number in codes with pictures in the thesis because, in Juypter Notebook, the number changes every time we run the codes. But instead, follow through comments in codes.

Nibesh Khadka

15.05.2019, Oulu

## CONTENTS

# VOCABULARY

## ABBREVIATIONS

CPU = Central Processing Unit

FP = False Positive

FN = False Negative

KNN = K-Nearest Neighbour

ML = Machine Learning

MAE = Mean Absolute Error

MSE = Mean Squared Error

RMSE = Root Mean Squared Error

SVC = Support Vector Classification

SVM = Support Vector Machine

TN = True Negative

TP= True Positive

## Terms

Bias = ML model performance is not good i.e. very low accuracy, at both training and testing data. This situation is referred to as Highly Biased or underfitting. It indicates the model is not a good fit and/or data is not meaningful enough. During such a situation, it is recommended to use a complex ML algorithm and play through data to make it more meaningful.

Categorical Data / Qualitative Data = Data type that are non-numerical.

Classes = Prediction Values in Classification Algorithms.

Continuous Data / Quantitative Data = Data type that are numerical.

Data-frame = Database represented in columns and rows with labelled columns and indexed rows. It is a Pandas object.

Dependent Variable = Features column or target column to be predicted.

Domain Knowledge = Expertise in a field. For instance, ML engineer working for Nokia 5g network projects can benefit from knowledge about internet networking processes.

Dummy-Variable Trap = It’s a scenario created due to multicollinearity between column. In such a situation on the column can help in the prediction of another one so, there is no need for an algorithm to learn anything. To avoid such a situation, one of the multiple dummy variables created during conversion from categorical to dummy numerical variables is dropped. After dropping one column, our algorithm is forced to keep on guessing.

Features / Variables = A column in data frame. For instance, age column, sex column, price column.

Feature Engineering = Processing raw data to make it ready to be used by algorithms.

Independent Variable = All the feature columns except the to be predicted columns, that affect the result of prediction, are the independent column.

Multivariate = More than one variable.

Outliers = Data in a column that is exceptional from its siblings, in ML outliers can affect results negatively. For instance, A number 1000000 in the list of numbers less than 1000. (1, 200, 300, 999, 100, 25, 56, 465, 789, 1000000).

Parameters = Input of the Estimators or algorithms, some of which can be determined by the practitioner and some cannot. They determine the result.

Univariate = Single features or variable.

Variance = It is the situation when the model's performance is exceptionally good for training data, but poor results are achieved for unseen or test data. The situation indicates the use of complex ML model unnecessarily. It is also referred to as High Variance or Overfitting. The model tries to fit each data point rather than generalizing underlying pattern between points.

# 1 INTRODUCTION

Human beings, can easily distinguish the tree from rock, can learn languages, learn to read and write, learn to drive. Everyday performance gets better and better. But how is this happening, we learn, from our surroundings, from trails and success, from failures, from companions. Every time a task is performed our brain receives feedback, learns from feedbacks and saves them for future references.

Similarly, Machine Learning (ML) is a process of teaching an artificial system to learn through enormous amounts of data. It is crucial to remember the end goal of machine learning exercise is to produce a self-sufficient model for a specific task, hence the focus is to teach a model on how to learn than to perform.

With the development in technologies, several programming languages have been adopted by machine learning engineers and data scientist to solve machine learning problems. Some of the popular programming languages used are Python, R, Java. Python is one of the popular programming languages for machine learning practices.

Python is a high level, a general-purpose programming language developed in 1991. Python has been used in several platforms such as desktop applications, server-side scripting, machine learning, data analysis, web development. [1]

The thesis will walk through several phases in Machine Learning. It explains the ML and ML phases carried out to obtain an optimal ML model in brief. The thesis introduces several packages available in the Python programming language used for different ML Phases through examples of two supervised ML model development in code. However, the codes and packages are more conventions than rules. In coding, there can always be different methods and steps to perform the same operation.

# 2 MACHINE LEARNING

Machine Learning is a field of computer science, that uses statistical algorithms and techniques to teach an artificial system an ability to learn feeding a huge amount of data. [2]

Field of machine learning is huge and widespread, but there are some popular use cases successfully touching the lives of billions of people.

1. Recommendation System: YouTube suggests your videos based on your personal history, likes and dislikes. Netflix users get movie recommendation based on user profile and histories.
2. Spam Detection: Gmail has a separate folder that says Spam, which holds all spam emails. This is a brilliant application of machine learning saving people from frauds.
3. Google Translate: It detects language automatically and translates to the language of choosing instantly.
4. Speech Recognition: Siri in Apple products, Alexa from Amazon are products of deep learning which is the field of machine learning.

As shown in figure 1, the field of machine learning can be classified into three categories. [2]

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

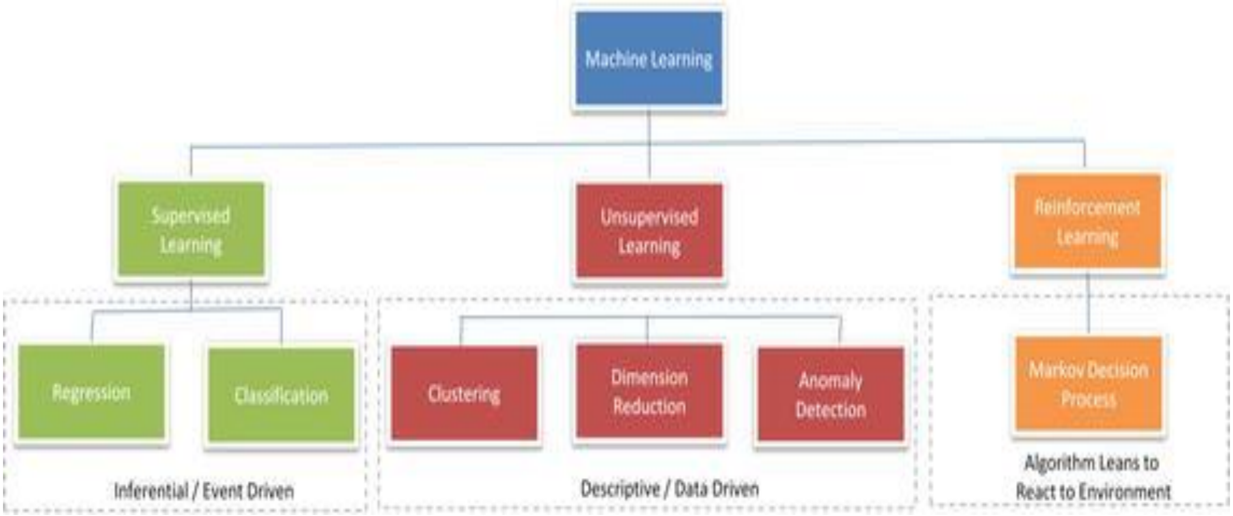


*FIGURE 1. Types of machine learning* [3]

### 2.1 Supervised Learning

Supervised learning in ML is the process in which prediction classes are already known. In supervised learning, a learning algorithm analyses rules and patterns using label input data, a group of dependent and independent variables, during training and generalise rules to deduce future unseen instances of data [4]. For instance, if the sky is cloudy and the clouds are dark, the weather is more likely to be rainy than sunny.

Supervised Learning has been further divided into two sub-types depending upon the desired data type of prediction, Regression and Classification

#### 2.1.1 Regression

ML algorithms that are used to predict continuous data-type are categorized as regression learning algorithms. For instance, prediction of stock prices, real state housing prices. [5]

Some examples of regression algorithms are Linear Regression, Polynomial Regression, Support Vector Machine (SVM).

**Linear Regression**

Linear Regression is the type of learning algorithms where the dependent and independent variable has linear relations.  Figure 2 demonstrates a plot between training data and the prediction of the model. A linear slope is created between prediction and training data, where the difference is the error.

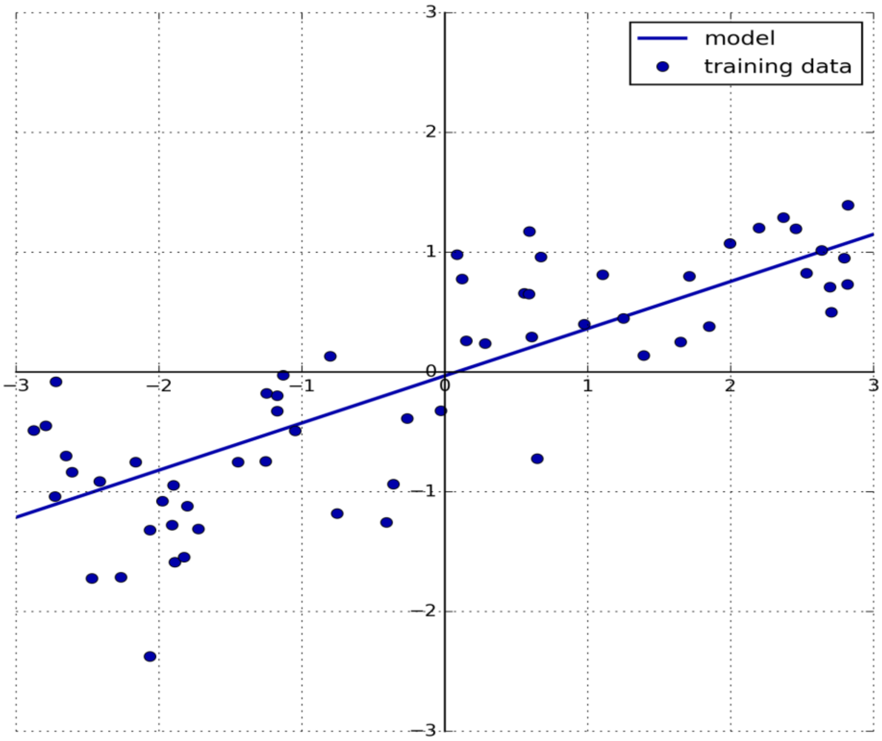


*FIGURE 2. An example of linear regression* [6]

Mathematical formula for simple linear regression has been provided in formula 1.

*Formula 1. Formula for calculating simple linear regression*

*y= m*x + c*

*y = predicted value,*

*m = coefficient,*

*x = independent variable*

*c = constant or intercept.*

For multivariate linear regression equation can expand as

*Formula 2. Formula for calculating multilinear regression*

$y = m_1 * x_1 + m_2 * x_2 + \ldots + m_n * x_n + c$

*n = the number of independent columns.*

Hence, in terms of simple linear regression, it can be interpreted as:

*y = sum of m*x of n number of columns + c*

### 2.1.2 Classification

Classification in ML is used to predict categorical data types (Vocabulary - Terms – Categorical Features). For instance, prediction of weather, directions. Some examples of classification algorithms are Random Forest Classification, Decision Tree, Support Vector Classification (SVC), Logistic Regression. Logistic Regression, unlike the name suggests, is a classification algorithm. [5]

**K-Nearest Neighbor**

K-Nearest Neighbor divides a data point among given categories based on the distance between the new data point and previously assigned data points. Figure 3 is displaying a selection process in KNN through visual aid.

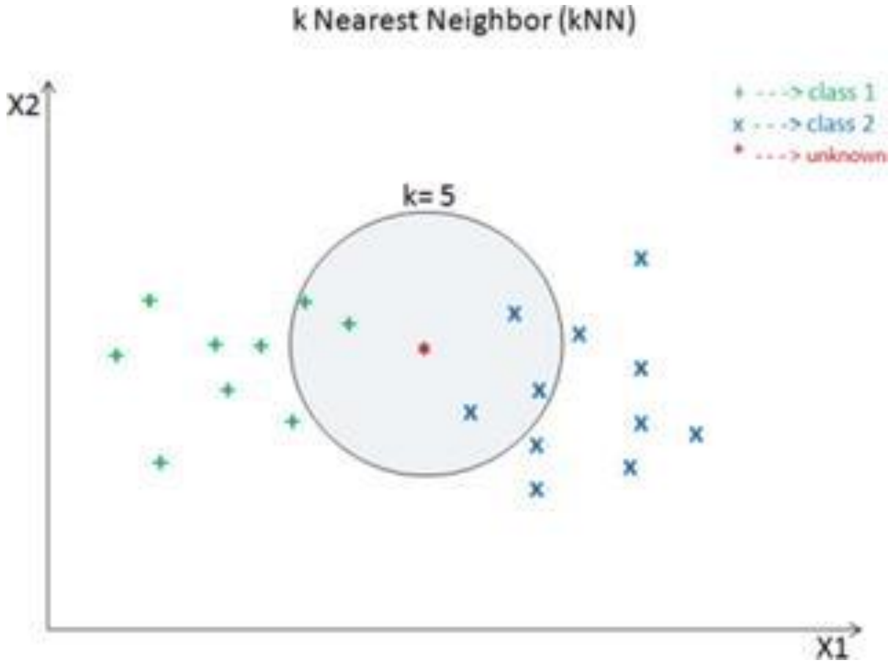


*FIGURE 3. K-nearest neighbor* [7]

So, how does it work?

Typically, a K-Nearest Neighbors selection process can be explained in the following steps:

1. The algorithm takes the new input data point (red point in figure 3), calculates the distance between the new data point and its neighbor data points (blue and green points in figure 3). Distance calculation metrics can be chosen, default one in sklearn’s KNN algorithm is “Euclidean Distance” metric.
2. Then, the KNN algorithm, chooses a K number of data points among all previously assigned data points, fitting the distance metric. For instance, if k is 5, and distance metric is “Euclidean Distance”, it will pick 5 data points (among blue and green points in figure 3), closest to new data points in a straight line.
3. Among the 5 chosen points, the category (dependent class 1 and 2 in figure 3), consisting of the highest number of chosen data points or neigh-

bors gets assigned the new data point. For instance, in figure 3 the number of blue points is 3 and green in 2 which are the K = 5 closest points represented inside the circle. So, the redpoint falls into Class 2 in figure 3.

### 2.2 Unsupervised Learning

Unsupervised Learning is a category in ML where prediction class are unknown, and prediction is done based on an analysed pattern from features available. There are situations in real life where features available are unclear or meaningless, in such scenario, unsupervised learning algorithms are used. In simple words, if you do not know, what are you predicting then use unsupervised learning techniques [8]. Some examples of clustering algorithms are Clustering, Anomaly Detection, Dimension Reduction.

#### 2.2.1 Clustering

As the name suggests, in clustering, predictions are clustered together in several groups based on the patterns recognized from the relationship between independent variables. K-Means Clustering, Agglomerative Hierarchical Clustering are few examples of clustering algorithms. [8]

### Reinforcement Learning

Reinforcement Learning is feedback-based learning, where algorithms receive feedbacks from the environment and adjust the parameters that affect the result. Some examples of the Reinforcement learning techniques are the Markov decision process, Q-learning, Temporal Difference methods, Monte-Carlo methods. [9]

So, Machine learning is a process of teaching machine how to learn by feeding a large amount of relevant data. ML can be divided into three categories, Supervised Learning, Unsupervised Learning and Reinforcement learning. The method is Supervised learning when dependent variables are labelled. The method is Unsupervised learning when dependent variables are unlabelled, so it is the responsibility of algorithms to group prediction in several classes. Further in the coming chapters, details of Supervised ML phases are explained and implemented using python programming language.

# 3 PYTHON FOR MACHINE LEARNING

Python is a high level, a general-purpose programming language developed in 1991. Python has been used in several platforms such as desktop applications, server-side scripting, machine learning, data analysis, web development. [1]

### Python Modules and Packages

Python modules are files in python with “.py" extensions. Packages are collections of modules and packages are often referred to as namespaces [10]. Modules and Packages are prewritten set of codes aimed for reuse to reduce the workload of developers.

## 3.1 Python Packages for Machine Learning

Python has been used widely in the field of ML. There are several packages in python used for data analysis and machine learning, using all or some of them are mostly personal preference and experience. Some of them are as follows:

### 3.1.1 NumPy

NumPy abbreviation for numerical python is a package for scientific computation. It is used for creation and manipulation of multi-dimensional arrays, mathematical operations like mean, medians, sum, maximum and minimum values. [11]

### 3.1.2 Pandas

In machine learning, data fetched from sources are converted into data frames and further processed by desire. For python, pandas is an open source library used for creation, manipulation of data-frames. [12]

### 3.1.3 Scikit Learn

Scikit-Learn also is known as sklearn, is open source package for python, being used for data mining, data analysis, machine learning and data science. The sklearn package offers various machine learning algorithms and tools for estimation, pre-processing, splitting and many more. [13]

### 3.1.4 Matplotlib

Matplotlib is a python library for data visualization. Many statistical diagrams for instance histogram, graph, scatterplot can be drawn using matplotlib. Data Exploration is an essential part of machine learning, where data visualization libraries like Matplotlib are used. [14]

### 3.1.5 Seaborn

Seaborn is a data visualization library based on Matplotlib. Statistical plots like heatmap, violin plot, time series can be constructed easily with seaborn. [15]

This chapter was a very short introduction on Python Programming language and the packages available for ML. Further in chapter 5, several modules inside these packages have been utilized to elaborate ML phases.

# 4 PHASES IN MACHINE LEARNING

ML is a complicated process, procedures mostly depend on the data, desired output, client or employer. However, steps in achieving a near perfect model are similar. Typical ML phases can be categorized as the following.

1. Fetching Data
2. Data Pre-processing
3. Data Splitting
4. Modelling
5. Model Optimization / Tuning

### Feature Engineering

Sometimes, steps 1 to 3(Data Fetching, Pre-processing and Splitting), in the list are also referred combinedly as feature engineering. In short, Feature engineering is a process where a raw data is processed and remodelled as per need of the project. After the feature engineering data is ready to be fed to an algorithm. Figure 4 demonstrates a lifecycle of data from starting point to a trained model in ML. Feature engineering comprises very important section in journey of data.

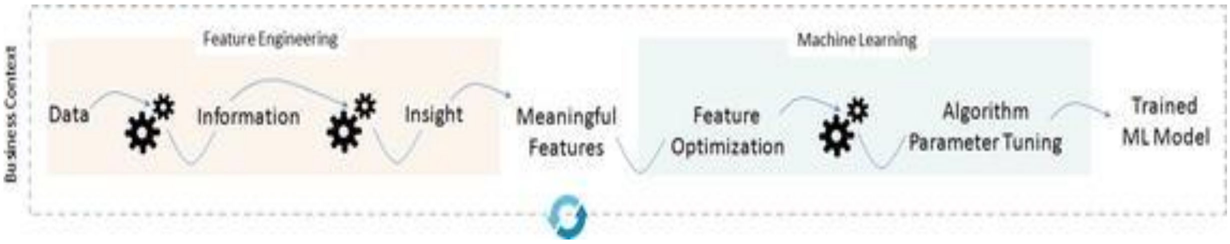


*FIGURE 4. Logical flow of data in machine learning model building* [17]

In a real-world application, the size of data is huge, and type also varies and is generally referred to as big data. Not only size but also data type can be different, data can be in the form of video, audio, text, image. Feature engineering is applied to cut a meaningful portion of data randomly from huge lump, make data more informative, make the whole process faster and less memory intensive for CPU (Central Processing Unit). Hence, feature engineering is an iterative process. Domain knowledge can be a benefit when it comes to data pre-processing. [16]

Basically, after being pre-processed data should be meaningful, free from outliers, and ready to be fed to algorithms. The table 2 is a table achieved from replacing, removing and adding values from table 1 for further processing.

*TABLE 1. Raw data before pre-processing*

| ID | Age | Salary per Year (In US $) |
|---|---|---|
| 1 | NaN | 150,000 |
| 2 | 20 | 60,000 |
| 3 | 30 | 175,000 |
| 4 | NaN | 180,000 |
| 5 | 35 | 120,000 |
| 6 | 110 | 120,000 |

*TABLE 2. Data after pre-processing*

| ID | Age | Salary per Year (In US $) |
|---|---|---|
| 1 | 28 | 150,000 |
| 2 | 20 | 60,000 |
| 3 | 30 | 175,000 |

| | | |
|---|---|---|
| 4 | 28 | 180,000 |
| 5 | 35 | 120,000 |

However, some flaws of table 1 and table 2 are, in real-world data are not this simple and small. Also, if a table has a categorical feature column it should be converted into numerical values since algorithms cannot take categorical values as input.

### 4.1 Fetching Data

Data are often stored in clouds or hardware in companies. As, mentioned earlier data can be in several forms like video, audio, image, texts. In this thesis, all the data being used for experimentation are either freely available in python sklearn libraries or has been created by the author. In python, pandas can be used to read the files in several formats. (Chapter 5, Pre-processing in Python)

### 4.2 Pre-processing

#### 4.2.1 Missing Values

As mentioned in section Feature Engineering, pre-processing is the phase where raw data is processed and cooked to be fed to the algorithm. Often in practice, many values in data are missing and its responsibility of ML engineer to find a solution to that issue either removing the whole row or finding an optimal replacement for the missing value. [18]

Often the mean or the median of a feature column is used as replacement value if the feature is continuous. However, extreme values can affect the quality of the mean or median [18]. For instance,

A set of numbers X= [12,5,6,7,100, 90000, missing_value]

*Formula 3. Formula for calculating mean*

*Mean = Sum of total numbers / Total numbers*

missing_value_1 (Mean of X) = (12+5+6+7+100+90000)/6 =15021.67

If we remove the outlier in the set which is 90000. Then,

missing_value_2 (Mean of X) = (12+5+6+7+100)/5 = 26

The difference is huge, can affect the predictive quality of the model negatively.

Similar can be the case for the median. Hence, outliers should always be removed before performing operations like mean and median. [18]

The mode can be used when the feature is a categorical value. The mode is a statistical method which returns the most occurring value in a list. (Chapter 5, section - Dealing with Missing Categorical Values)

ML prediction method is also used by advanced practitioners. In this method, feature column with missing values are treated as dependent variables and using ML algorithms values are predicted to fill in the missing value. [18]

### 4.2.2 Normalizing Data

Data as raw obtained from various sources are often of various units and unbalanced. Using them without scaling down or neutralizing, can affect the algorithm. For instance, in from table 2, the values in the "Age" column range from 20-30 but values "**Salaries**" column range from 60,000 – 80,000, the scale is quite unbalanced. Since algorithms are some form of mathematical operation of variables (formula 1 and formula 2) the unscaled variable will not produce a concise result. Hence, it's a common practice to normalize continuous features. Normalizing in ML process is done before or after splitting data into training and testing dataset [19]. Two common methods used for normalizing data are as follows:

#### Normalization / Min-Max Scaling

Normalization is the process of scaling numeric values within the range of 0 to 1. It is done by subtracting minimum value in the list by the given value and dividing the difference by the difference between the minimum and maximum value in that list (or feature column). [19]

*Formula 4. Formula for calculating normalization*

*Normalization = (Given Number- Minimum Value) / (Maximum Value – Minimum value)*

Outliers can affect the normalized values since; the calculation involves upper and lower margin in the list. Hence, outliers should be removed before normalizing a numeric column.

**Standardization**

In standardization, numbers are transformed in a manner where their mean is zero and the standard deviation is 1. Therefore, numbers after application of standardization range from -1 to 1. [19]

*Formula 5. Formula for calculating standardization*

*Standardization = (Given Number In column – Mean of Column) / Standard Deviation of Column*

### 4.2.3 Feature Generation

Some features in a data-frames can be highly correlated and some can be unnecessary from the perspective of the algorithm.

*TABLE 3. Salary of employees in company x*

| ID | Age | Salary per Year (In US $) |
|---|---|---|

|  |  |  |
|---|---|---|
| 1 | 25 | 150,000 |
| 2 | 20 | 60,000 |
| 3 | 30 | 175,000 |
| 4 | 58 | 180,000 |
| 5 | 35 | 120,000 |

In table 3, there are three columns of which ID column is unnecessary from the perspective of the algorithm, it is only helpful for human eyes. Hence, it should be removed before using it for training the algorithm. There are often multiple features in data. Merging or removing the features often requires domain knowledge and experience. (Vocabulary - Terms - Domain knowledge)

### 4.3 Data Splitting

ML algorithms analyse the correlation between features and learn from them to generalize a rule from those patterns. If we train and test our model in the same data, the model will fail to generalize relation which is the sole purpose of ML. Since future instances of test data in the real world after deployment is unknown so, data we already have is divided into train and test set. More the clean processed data to train the better the model. Hence, it is a common practice to divide data into a train set around 70 to 80 percentage and test set around 30 to 20 per cent of the original data. The key is to divide data randomly.

### 4.4 Modelling

Data can be normalized before or after the splitting data to train and test portion. However, the inevitable question arrives i.e. Which algorithm to use?

Figure 5 has been provided by Sklearn that can be used as a guide to solve the query about estimator or algorithm.

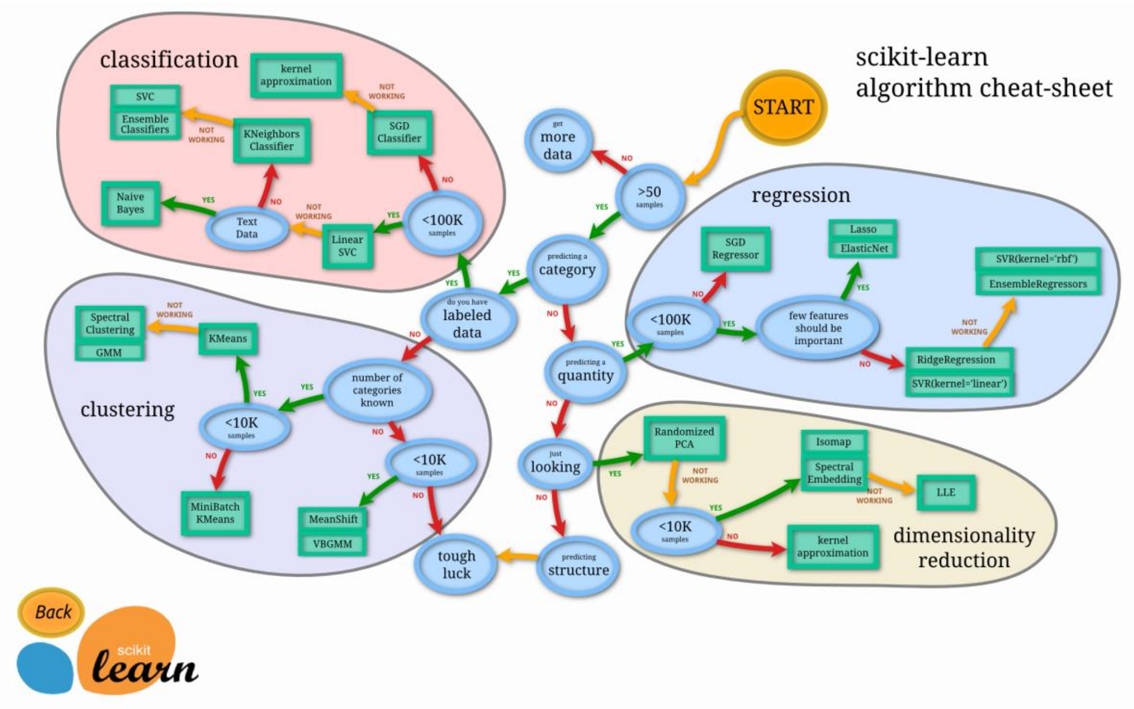


*FIGURE 5. Scikit-learn algorithm cheat-sheet* [20]

So, some key points to consider while choosing an algorithm are:

1. If the dependent feature is of continuous data type, use regression algorithms.
2. If the dependent feature is of categorical data type, use classification algorithms.
3. If feature columns are unknown(unlabelled), and dependent classes cannot be identified use clustering algorithms.
4. Start with a very simple algorithm and climb your way up to more complex algorithms. For instance, if the result is excellent using simple linear or multi-linear regression algorithms do not use polynomial or Support Vector Machine (SVM).
5. Use google search and forums for answers from experts.

After, the decision in type of algorithm, in python sci-kit learn package can be used for the fetching algorithm. The algorithm comes with build in methods to train and predict data.

### 4.5 Model Optimization/ Tuning

After the train and test of the model, several reports can be gathered depending upon the type of algorithms.

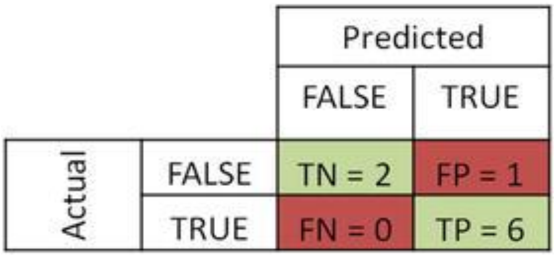
In regression, the common error evaluation metrics are:

1. Mean Absolute Error (MAE): Mean of the absolute value of errors
2. Mean Squared Error (MSE): Mean of squared value of errors.
3. Root Mean Square Error (RMSE): Square root of mean squared errors.

Here, the error is the difference between real value and the predicted value. RMSE is the widely popular one because it’s unit of measurement is the same as the unit of dependent value.

For classification algorithms, confusion matrix and classification report can provide an evaluation report.

1. Confusion Matrix: Figure 6 demonstrates an example of confusion matrix. It is a table consisting of True Positives (TP), True Negative (TN), False Positive (FP) and False Negative (FN). [21]



| | | Predicted | |
|---|---|---|---|
| | | FALSE | TRUE |
| Actual | FALSE | TN = 2 | FP = 1 |
| | TRUE | FN = 0 | TP = 6 |

*FIGURE 6. Confusion matrix in binary classifier* [22]

2. Accuracy can be calculated as:
   *(TP+TN) / (Total Number of Values)* *Formula 6.*

True Positive: When the prediction result is true (or positive or 1) and the actual value is also true (or positive or 1) it is called True Positive.

True Negative: When the prediction result is false (or negative or 0) and the actual value is also false (or negative or 0) it is called True Negative.

False Positive: When the prediction result is True (or positive or 1) but the actual result is False (or negative or 0), it is called False Positive. Also referred to as Type I error.

False Negative: When the prediction result is False (or negative or 0) but the actual result is True (or positive or 1), it is called False Negative. Also referred to as Type II error.

The optimization of the model is done in various ways. Sometimes, we return to point zero i.e. pre-processing, maybe remove a column that seems unnecessary. Other times, internal parameters of an algorithm that can be altered are altered for instance for KNN, the number of K-neighbors parameter or attribute can be altered to achieve better results (Chapter 5, KNN).

### 4.5.1 Elbow Method

Elbow method can be used to determine the optimum number of K in K-Nearest Neighbor or in K-Means Clustering (an unsupervised ML algorithm). In this method, average error (mean of incorrect prediction) from a range of K for instance, 1 to 10 number of k, is plotted in the graph. Theoretically, the line plot obtained is supposed to be in the shape of an elbow. From the plot, the K which gave the lowest error is chosen as the K for KNN. The K is theory is usually the point (sometimes its further below or above) where the elbow bents.

Figure 7 display elbow method in theory created for demonstration. In chapter 5, the elbow method has been used for the KNN algorithm (chapter 5, KNN). According to figure 5, the most suitable value for k is 6 since after that error rate is not decreasing. Using K higher than 6 is not a good idea, it may force our algorithm to fit every outlier that can result in high variance. (Vocabulary – Terms – Bias, Variance)

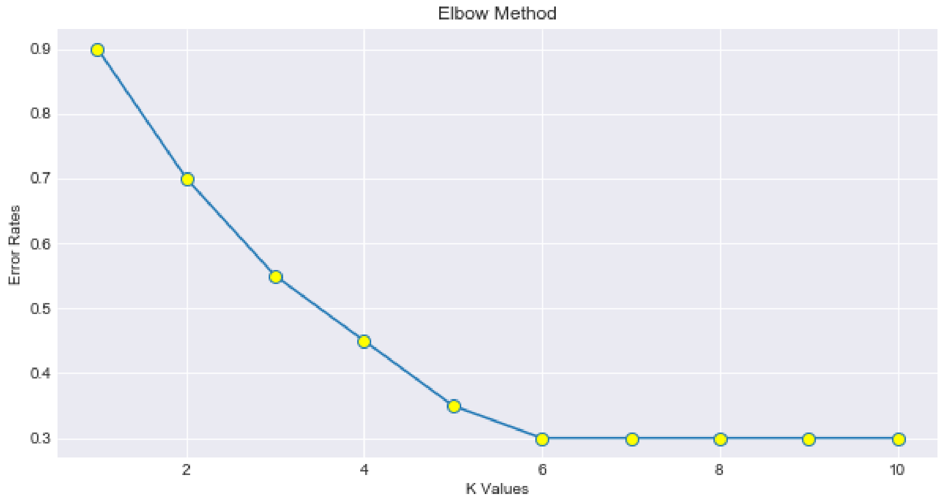


*FIGURE 7. Elbow method* [23] [24]

Classification Report is a table that provides several metrics of classification. Classification Report table can be imported from the metrics module of sklearn. Chapter 5 demonstrates the use of the classification report in KNN. The reference 25 provides information about classification report metric more. [25]

From Classification report some metrics are:

*Formula 7. Formula for calculating precision*

*Precision = True Positives / Predicted Positives = TP / (TP + FP)*

*Formula 8. Formula for calculating recall*

*Recall = True Positives / Actual Positives = TP / (TP + TN)*

More about precision and recall at reference 26. [26]

So, any output is as good as the input. If we can polish our data properly than only we can hope for a very good model. Data is the basic need in a life on the ML model. ML engineer (or Data scientists) is a chef that cleans, cuts and cooks the data. Only after that, the ML algorithm feed with the data and outputs the outcome of an ML model.

# 5 MACHINE LEARNING IN PYTHON

The chapter demonstrates the Supervised ML phases in the Python programming language. Linear Regression and K-Nearest Neighbors are the two algorithms of type regression and classification algorithms respectively.

The chapter uses built-in datasets freely available in Scikit-Learn library. For Regression algorithm, Load Boston Dataset has been used and Iris Dataset for the KNN algorithm. Load Boston is a dataset about real state housing in Boston where we predict Housing prices a regression problem [27]. Iris Dataset is a collection of data about three Species of Iris flower where we try to predict among three species of Iris flowers a classification problem [28]. In Scikit learn, Datasets we import are of Bunch types (dictionary-like objects). Hence, it is required to convert to a data frame for further operations. (figure 21 and figure 22)

**Importing Libraries and Packages**

Packages and Modules need to be installed and imported separately in each file since they are not part of the python download package. Modules and Packages are sometimes imported whole or partial. The main reason behind partial importing is to save memory.

Way to import:

1. import package_name as conventional_short_form

for instance,

```
import numpy as np
```

2. import package_name.family_name as conventional_short_form
   for instance,

```
import matplotlib.pyplot as plt

#It imports pyplot submodule from
matplotlib library as plt.
```

3. Instead of importing the whole package small modules and files are imported whenever required. When working with big size files it helps save memory. Scikit-Learn which is known as sklearn in python coding, importing package can like this.
   **from sklearn.family_name import module_name**
   for instance,

```
from sklearn.model_selection import train_test_split

#It imports the library to split dataframe.
```

Short forms are used to save time and have been adopted worldwide as a rule though it is just a convention.

**Importing Datasets**

Pandas package is used for importing databases. Pandas support several formats like HTML, CSV(Comma Separated Values), TSV (Tab Separated Values). Data is imported and converted into a dataframe object in one step using pandas. [29]

For instance, to import a CSV file,

```
Dataframe_name = pd.read_csv(“Name of file”)
```

## 5.1 Pre-processing in Python

### 5.1.1 Information About Data

The dataset used for pre-processing in this section is fake data. The dataset is supposedly data about the salary of anonymous IT employees in five different cities of Finland. Dataset has four columns. They are as follows:

1. Years_of_Experience = Number of years of experience from employees with values ranging from 0 to 10.
2. Cities= Five cities in Finland. They are 'Oulu', 'Tampere', 'Helsinki', 'Turku' and 'Rovaniemi'.
3. Salaries = Fake Salaries of IT employees ranging between 2000 to 5000.
4. Gender = Gender category with values “M” or “F”. However, this column is almost empty for preprocessing purpose.

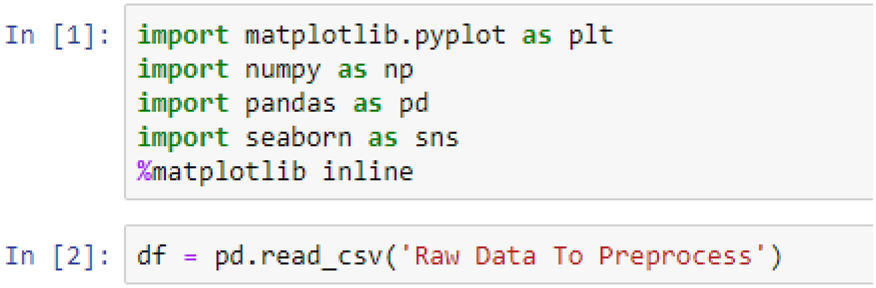
Since the dataset is fake and created randomly the correlation between columns is not meaningful. For instance, the salary of an employee of 2 years of experience can be greater than that of 8 years of experience.

Our intention is to clean data so several values from the columns have been replaced by “**NaN**” (not a number) values. This data has only been created for preprocessing, so dependent or independent are not required to be distinguished. If the reader wants to train an algorithm with this data, may use “Salaries” column for regression or “Cities” column for the classification algorithm.

Tutorial on pandas and data-frame creation is not the actual motive of the thesis. But the reader can visit the code section and follow through the code and comments for extra knowledge. [30] [31].

### 5.1.2 Import the dataset

The full version of the code to preprocess data can be found at GitHub resource provided at reference [32].



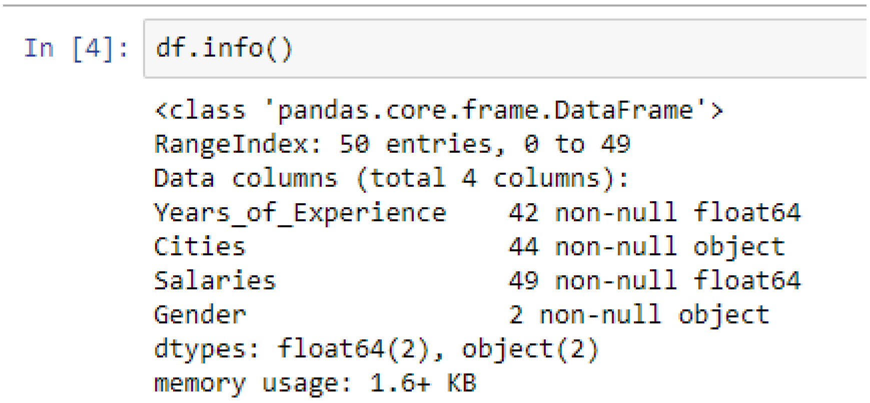
```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        import seaborn as sns
        %matplotlib inline

In [2]: df = pd.read_csv('Raw Data To Preprocess')
```

*FIGURE 8. Import pre-processing data* [32]

In figure 8 cell 1 we import related packages. The last line "**%matplotlib inline"** is used to display plots in a browser in **Jupyter Notebook** IDE (Integrated Development Environment). In line 2, we import our dataset and name of dataframe "df".

### 5.1.3 Info Method

Since data is fake describe the method and its statistical information are not useful. Let us observe the result of info method from figure 9.



```
In [4]: df.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 50 entries, 0 to 49
        Data columns (total 4 columns):
        Years_of_Experience    42 non-null float64
        Cities                 44 non-null object
        Salaries               49 non-null float64
        Gender                 2 non-null object
        dtypes: float64(2), object(2)
        memory usage: 1.6+ KB
```

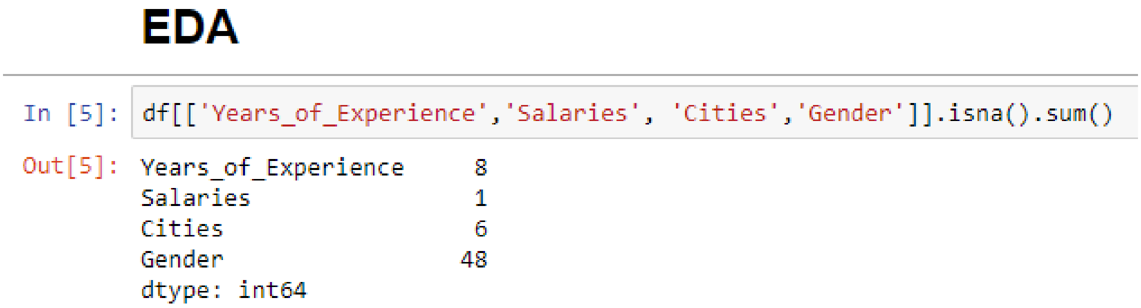
*FIGURE 9. Result of info method on pre-processing data* [33]

Info method is very useful to check datatype of feature columns and their null value number. Let us observe important key points.

1. RangeIndex: It gives a total length which is 50 ranging from 0 to 49.

2. Other Four Columns: Each column in our df has different lengths. For instance, in Cities, it says 44 non-null objects. It means among 50 values, 44 are non-null, 6 are null values and data-type is float64.

Let’s see the null value counts in pandas and display them in seaborn as well.



EDA

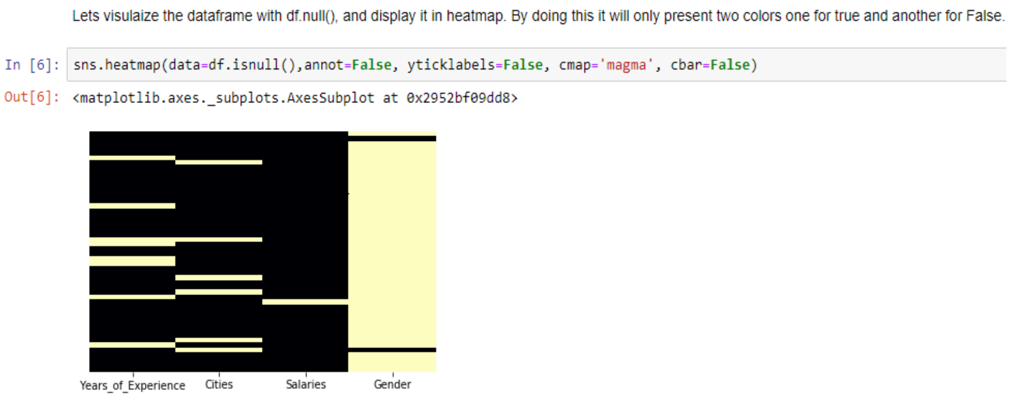
```
In [5]:  df[['Years_of_Experience','Salaries', 'Cities','Gender']].isna().sum()

Out[5]:  Years_of_Experience     8
         Salaries                1
         Cities                  6
         Gender                 48
         dtype: int64
```

*FIGURE 10. Pandas null value count* [34]

In pandas, we can calculate the sum of null values by combining methods as displayed in figure 10.

1. **df[['Years_of_Experience','Salaries', 'Cities','Gender']]**, selects the provided list of columns.
2. **df[['Years_of_Experience', 'Salaries', 'Cities','Gender']].isna()**, checks null values in these columns and provides an answer either True for null and False for non-null values and return a dataframe of given lists.
3. **df[['Years_of_Experience','Salaries', 'Cities','Gender']].isna().sum()**, now calculates the sum of true values and outputs the result as shown in the image.

Similarly, we can also visualize the null value proportion in seaborn heatmap as shown in figure 11.



Lets visulaize the dataframe with df.null(), and display it in heatmap. By doing this it will only present two colors one for true and another for False.

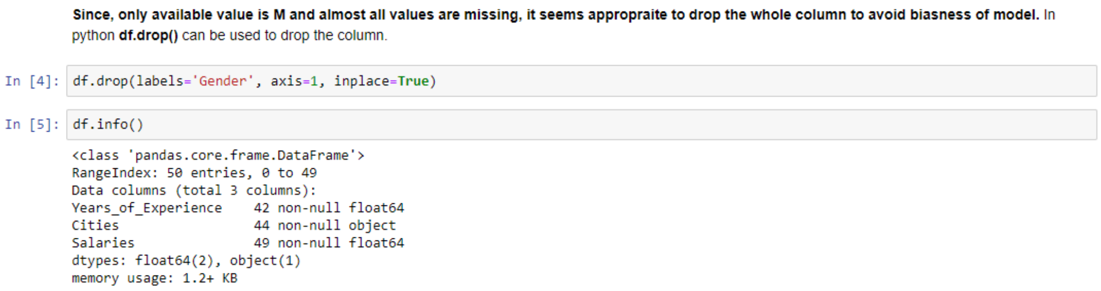
```
In [6]: sns.heatmap(data=df.isnull(),annot=False, yticklabels=False, cmap='magma', cbar=False)
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x2952bf09dd8>
```

*FIGURE 11. Null value count visualization in seaborn* [35]

Let’s observe figure 11, “**sns.heatmap(data=df.isnull(),annot=False, yticklabels=False, cmap='magma', cbar=False)**”  in steps:

1. **df.isnull()**, returns a dataframe like df.isna(), with values True for null and False for non-null values.
2. **annot=False, yticklabels=False, cbar=False,** just for not displaying data values because we are not looking for correlation like in figure 24.
3. In the figure, we are seeing four columns in black and yellow. Black is for non-null values and yellow is for null-values.

We can conclude that the Gender column of all is almost empty from both heatmap and Pandas null count. Since Gender column is almost empty we will drop it instead of filling random values by using pandas drop method as displayed in figure 12.



**Since, only available value is M and almost all values are missing, it seems appropraite to drop the whole column to avoid biasness of model.** In python **df.drop()** can be used to drop the column.

```
In [4]: df.drop(labels='Gender', axis=1, inplace=True)

In [5]: df.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 50 entries, 0 to 49
        Data columns (total 3 columns):
        Years_of_Experience    42 non-null float64
        Cities                 44 non-null object
        Salaries               49 non-null float64
        dtypes: float64(2), object(1)
        memory usage: 1.2+ KB
```

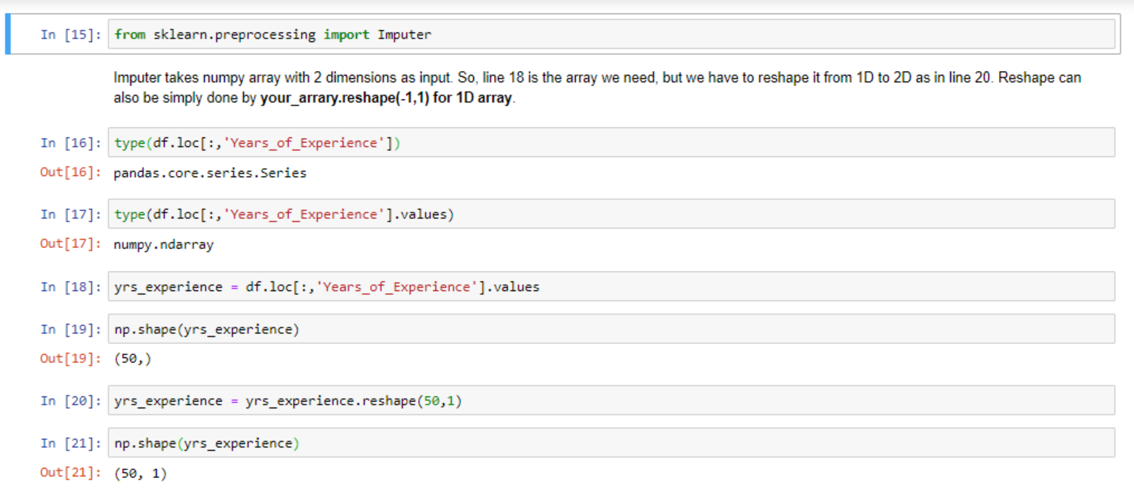
*FIGURE 12. Info on data* after dropping gender column [36]

In line 4 we drop the column and 5 we check the new df info without Gender column.

### 5.1.4 Dealing with Missing Continuous Values

### Imputing Using Sklearn

Python provides Imputer method in the Pre-processing module. We will impute column “**Years_of_Experience**” from our data.



```
In [15]: from sklearn.preprocessing import Imputer
```

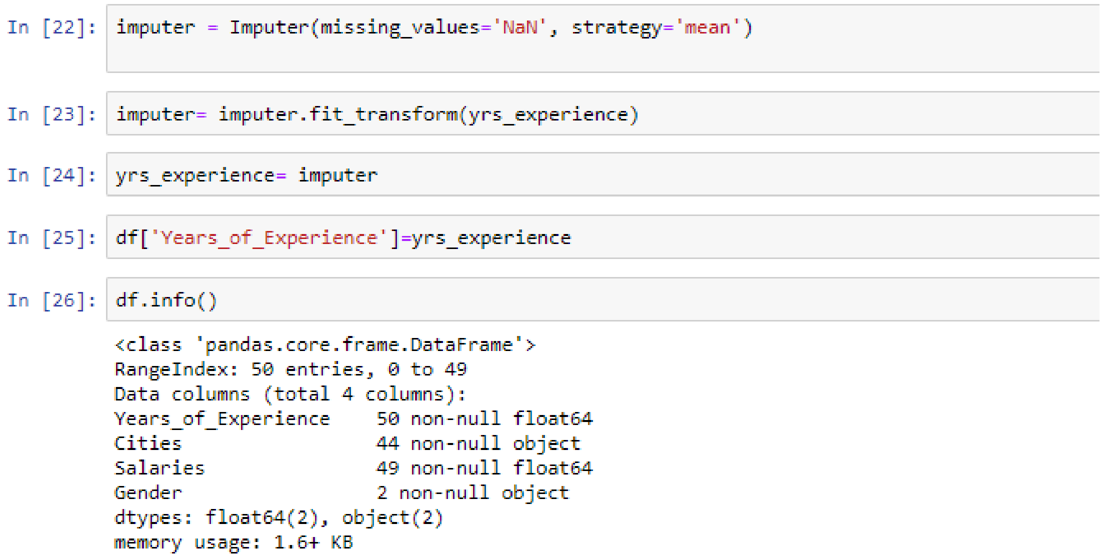
Imputer takes numpy array with 2 dimensions as input. So, line 18 is the array we need, but we have to reshape it from 1D to 2D as in line 20. Reshape can also be simply done by **your_array.reshape(-1,1) for 1D array**.

```
In [16]: type(df.loc[:,'Years_of_Experience'])
Out[16]: pandas.core.series.Series

In [17]: type(df.loc[:,'Years_of_Experience'].values)
Out[17]: numpy.ndarray

In [18]: yrs_experience = df.loc[:,'Years_of_Experience'].values

In [19]: np.shape(yrs_experience)
Out[19]: (50,)

In [20]: yrs_experience = yrs_experience.reshape(50,1)

In [21]: np.shape(yrs_experience)
Out[21]: (50, 1)
```

*FIGURE 13. Setting up for imputing* [37]

In figure 13, we set up data for imputing. In line 15 we import out Imputer function.Imputer function takes a 2D (2 Dimensions) array as input. So, we check the type of our column type in 16, which is a series, a Pandas object. However, in line 17 we can see that values can be extracted as an array. “**Numpy.ndarray**” means a NumPy array with n dimension where n is a number of dimensions. As mentioned earlier array must be 2D so, we create a variable named “**yrs_experience”** and assign our array in line 18. In line 19, “**np.shape()**” provides shape of our array which is 1D(1 dimension). After that, in line 20 “**name_of_array.reshape()**”, a standard method provided by NumPy has been used to reshape our 1D array to 2D [38]. The result of our operation is being displayed in line 21.

In figure 14, line 22 we begin the process by impute initialization.



```
In [22]: imputer = Imputer(missing_values='NaN', strategy='mean')

In [23]: imputer= imputer.fit_transform(yrs_experience)

In [24]: yrs_experience= imputer

In [25]: df['Years_of_Experience']=yrs_experience

In [26]: df.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 50 entries, 0 to 49
         Data columns (total 4 columns):
         Years_of_Experience    50 non-null float64
         Cities                 44 non-null object
         Salaries               49 non-null float64
         Gender                 2 non-null object
         dtypes: float64(2), object(2)
         memory usage: 1.6+ KB
```

*FIGURE 14. Imputing column using imputer* [39]

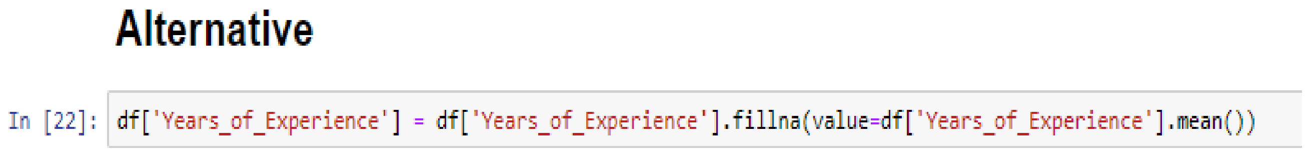
Let's de-structure figure 14

1. We create variable name imputer and initialize Imputer. The attribute "**strategy**" is most important to note in line 22. The strategy basically means what is the statistical operation we will use to replace missing values in the column of implementation. As mentioned in section "Missing Values" of chapter 4, we use operations such as mean, median, mode to replace missing values. So, it is important to provide such information to the imputer method while initializing the method. For this continuous feature column, the mean strategy has been used. Check Imputer documentation for all possible attributes and values.
2. Line 23 fits and transforms our "**yrs_experience**" variable into imputer at once. This means all the missing values have been replaced by the mean of the same columns. It is important to note while calculating mean only non-null values are included.
3. In line 24, we reassign our "**yrs_experience**" value to our imputed values. Similarly, in line 25, we reassign our column "**Years_of_Experience**" in our data-frame new imputed values without missing values. Line

26 is the proof of our successful operation because the length of “Years of Experience” column is 50 with non-null values.

### Alternative Imputing Using Pandas

It is also possible to use pandas for imputing [40]. The whole imputing process from sklearn can be replaced by the line in figure 15, where the missing values have been replaced using mean.



Alternative

```
In [22]: df['Years_of_Experience'] = df['Years_of_Experience'].fillna(value=df['Years_of_Experience'].mean())
```

*FIGURE 15. Imputing using pandas* [41]

Let us de-structure the code.

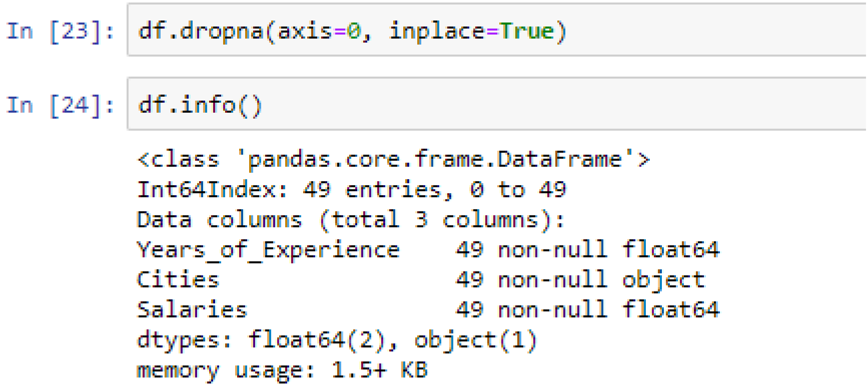
1. **df['Years_of_Experience'].fillna()**, checks either the column has null-values.
2. **df['Years_of_Experience'].mean()**, is a function that calculates the mean of given numerical column only using non-null values.
3. **df['Years_of_Experience'].fillna(value=df['Years_of_Experience'].mean())**, fillna() if finds null values replaces them with the mean obtained from process mentioned in step 2. However, not only mean, the value attribute can take a string or any integers (check to deal with missing categorical values in Pandas).

At the end, which one to choose between Imputer and Pandas.fillna(). From the authors perspective, it is about the experience and comfort of the practitioner.

Last continuous column in df is “**Salaries”**. From figure 9 (info method), it can be agreed that Salaries column has only one value missing. There are two things that can be done here.

1. The value can either be replaced with mean or median using above Imputer or Pandas fillna().
2. We can drop the row with null values.

For versatility, let's drop the row.



```
In [23]: df.dropna(axis=0, inplace=True)

In [24]: df.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 49 entries, 0 to 49
         Data columns (total 3 columns):
         Years_of_Experience    49 non-null float64
         Cities                 49 non-null object
         Salaries               49 non-null float64
         dtypes: float64(2), object(1)
         memory usage: 1.5+ KB
```

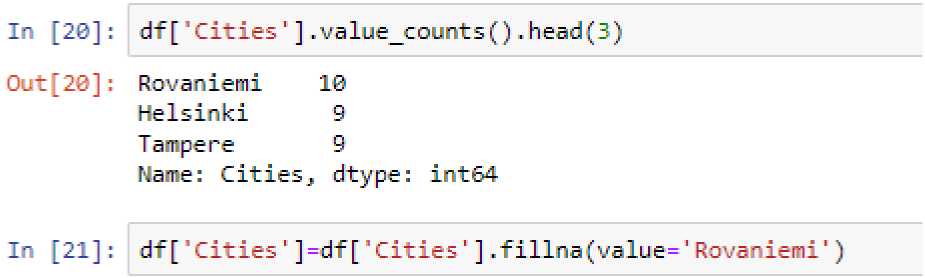
*FIGURE 16. Drop a row with a null value in salaries column* [42]

In figure 16, Pandas “**dropna()**” method drops the null values if found [43]. The most important attribute is the axis. It determines whether dropping row (axis = 0) or column (axis = 1). Line 24 displays the new information about our dataframe with 49 rows with 0 null values. The whole method iterates through our dataframe searching for a null value. In our df, only null value now is in Salaries column so, it drops the row with a null value in Salaries column.

### 5.1.5 Dealing with Missing Categorical Values

Using Mode is a viable option for replacing categorical variables. In our dataset, we do have a categorical column named “Cities”. The mode is the most repeated value in a list. For strings, there is no built-in function provided in python (for integer check scipy.stats.mode package). But the pandas “**value_counts()**” can be used to find most repeated values as well. [44]

In figure 17, value_counts() has been used for counting the total value counts of each value in the City feature column. From line 20 city we find that Rovaniemi is the most repeated value in the feature column. In line 21 fillna() method has been used to replace null values with Rovaniemi.



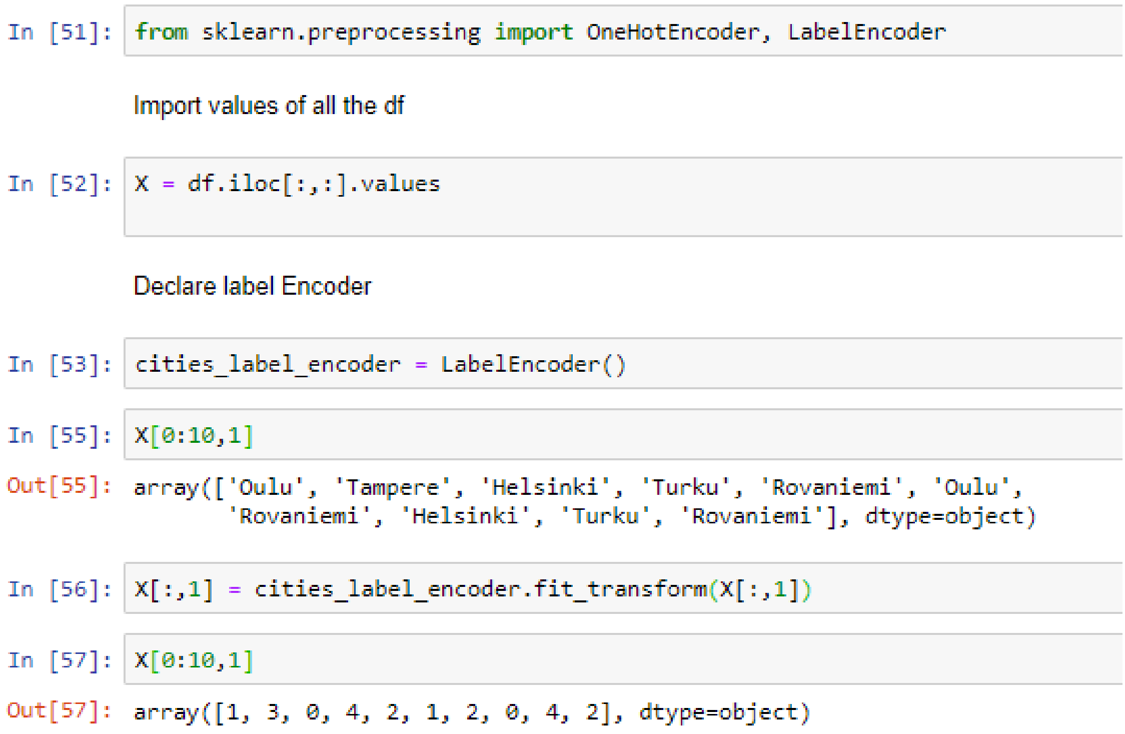
```
In [20]: df['Cities'].value_counts().head(3)

Out[20]: Rovaniemi    10
         Helsinki      9
         Tampere       9
         Name: Cities, dtype: int64

In [21]: df['Cities']=df['Cities'].fillna(value='Rovaniemi')
```

*FIGURE 17. Use of value_count() for mode* [45]

### OneHotEncoder and LabelEncoder

Preprocessing is not yet complete. The feature column “Cities” column is still a categorical value. ML algorithm can only process numerical values. Preprocessing module in python has two methods name LableEncoder and OnehotEncoder. [46][47]

Figure 18 demonstrates Label Encoding procedure.



```
In [51]: from sklearn.preprocessing import OneHotEncoder, LabelEncoder
```

Import values of all the df

```
In [52]: X = df.iloc[:,:].values
```

Declare label Encoder

```
In [53]: cities_label_encoder = LabelEncoder()

In [55]: X[0:10,1]

Out[55]: array(['Oulu', 'Tampere', 'Helsinki', 'Turku', 'Rovaniemi', 'Oulu',
                'Rovaniemi', 'Helsinki', 'Turku', 'Rovaniemi'], dtype=object)

In [56]: X[:,1] = cities_label_encoder.fit_transform(X[:,1])

In [57]: X[0:10,1]

Out[57]: array([1, 3, 0, 4, 2, 1, 2, 0, 4, 2], dtype=object)
```

*FIGURE 18. Label encoding* [48]

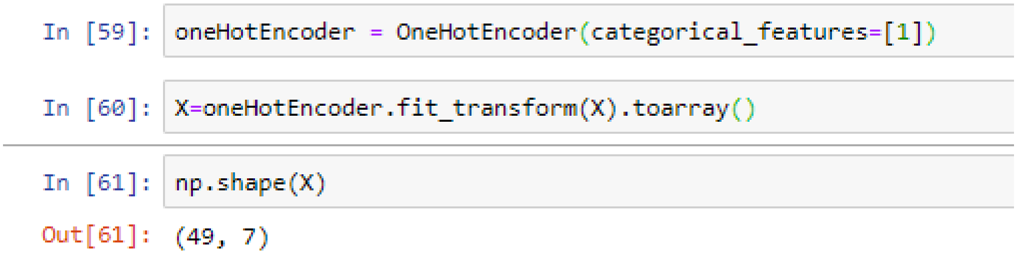
Line 51, we import both OneHotEncoder and LabelEncoder. Label Encoder requires the array data type as input, not the dataframe. Hence, we extract the values of our dataframe, all the rows and column in next line 52. After that, we initialize our LabelEncoder with name "**cities_label_encoder**" in line 53. Line 54 checks the values of the city before encoding which is a column in 1 in position (check dataframe). In Line 56 we fit and transform our cities into a numeric column with "**fit_trainsfrom()**" and also reassign column. Line 57 proofs the success of our operation.

The LabelEncoder changes categorical values to numerical values. Since ML algorithm does not know our encoded labels are categorical values changed into numerical basically a dummy variable, they can treat them as an actual number. If that happens they might rank cities by order according to their value. For instance, Turku is ranked highest with its value 4 and Helsinki lowest with its value 0 (line 55, and 57 of figure 18).

Hence, to prevent that situation we use OneHotEncoder. The OneHotEncoder creates one column for each encoded label i.e. one column for each city's encoded label. The key thing is since our categorical dummies are like true and false rather than points. The onehotencoder takes care of that by providing values to newly create columns of either 0 or 1. For instance, after one-hot-encoding,

1. 'Oulu' will be a separate column with value 1 or 0.
2. All the rows before encoding which have cities with values 'Oulu' will now have the value of 1 as it emphasizes true
3. All the rows before encoding which did not have cities with values 'Oulu' will now have a value of 0 as it emphasizes false.
4. Same is the case for each city in our case. For our 5 cities in df, it will create 5 columns

Figure 19 demonstrates the process of one-hot encoding [47]. Line 59 is for initialization. The attribute "**categorical_features**" is for the position of the column that is to encode. In our case, the position of the "**Cities**" column is 1.



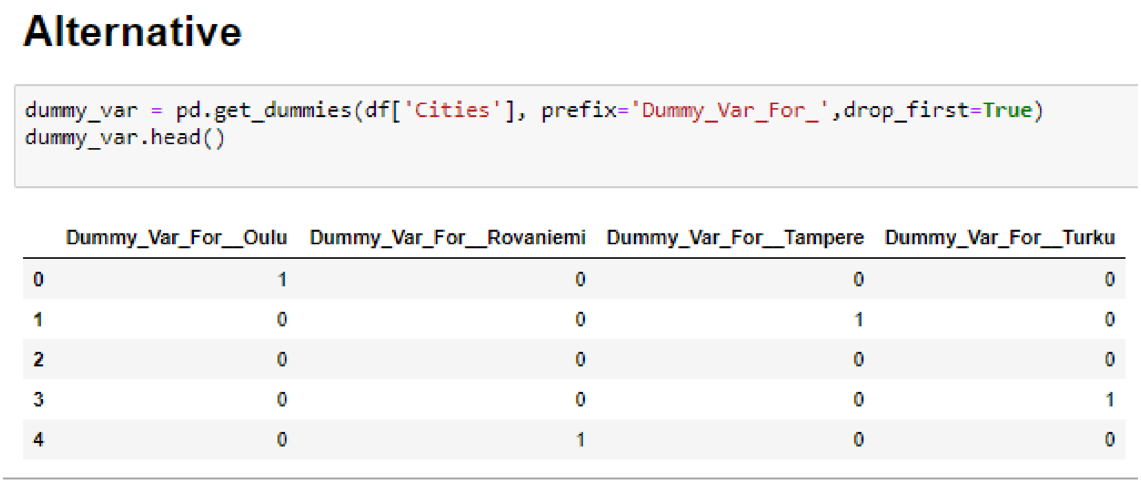
```
In [59]: oneHotEncoder = OneHotEncoder(categorical_features=[1])

In [60]: X=oneHotEncoder.fit_transform(X).toarray()

In [61]: np.shape(X)
Out[61]: (49, 7)
```

*FIGURE 19. One hot encode procedure* [49]

In Line 60 we fit and transform our label encoded X. The important thing to notice is that this time input is whole array, not a single column unlike in LabelEncoder. "**oneHotEncoder.fit_transform(X)**" returns matrix object which is then again changed to an array by "**toarray()**" function of Scipy [50]. Now our categorical variable has been converted to a dummy continuous variable.

**Alternative For Label Encoding And OneHotEncoder**

Pandas “**get_dummies()**” method can also create dummy variables from categorical variables. Get Dummies returns a newly created dataframe with the number of columns equal to the number of values in the category column. [51]



Alternative

```
dummy_var = pd.get_dummies(df['Cities'], prefix='Dummy_Var_For_',drop_first=True)
dummy_var.head()
```

| | Dummy_Var_For__Oulu | Dummy_Var_For__Rovaniemi | Dummy_Var_For__Tampere | Dummy_Var_For__Turku |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 |

*FIGURE 20. Pandas get dummies* [52]

Figure 20 demonstrates implementation of “**get_dummies()**”. We create a new dataframe “**dummy_var**”.

The first attribute is the column to be converted. Second attribute “**prefix**" is to prefix the newly created column with the given value. The third attribute "**drop_first**” is set to true. This will drop the first column created after the conversion of cities. Notice the output has four cities and Helsinki is missing. It is done to prevent the dummy variable trap (Vocabulary- Terms- Dummy Variable Trap). For OneHotEncoder we must manually drop a column after an operation.

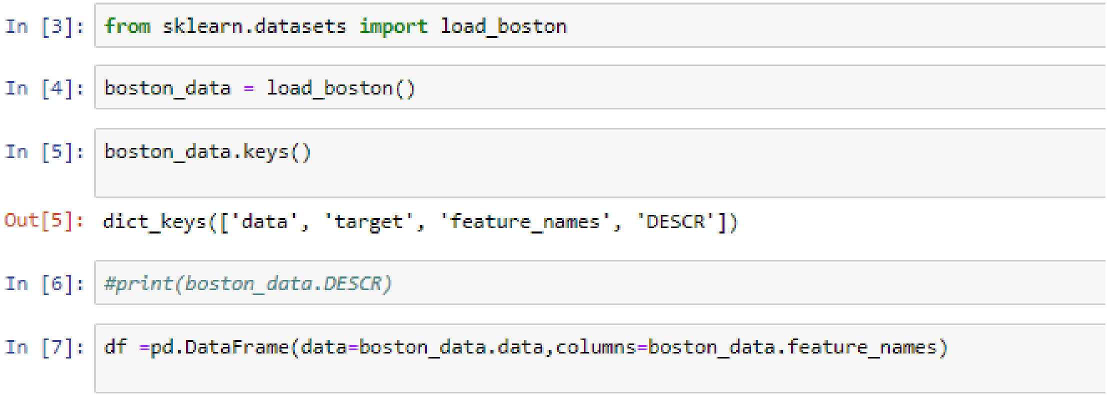
Now our data is clean and free of missing values and categorical variables any other procedure like normalizing, more exploration, splitting of data has been demonstrated further in Linear Regression and KNN section with different datasets. The reader can practice those methods in this data for learning purpose. [82] [83]

### 5.2 Linear Regression

Scikit Learn provides several datasets to practice. All the datasets available in sci-kit learn package is available under module name datasets. For linear regression load_boston which is dataset about housing price, has been used to demonstrate ML phases. [27] [54]

### 5.2.1 Loading and Creating Dataframe:

Figure 21 displays a screenshot of codes to load boston_data although the process is similar for any dataset available in the datasets module of sci-kit-learn package.

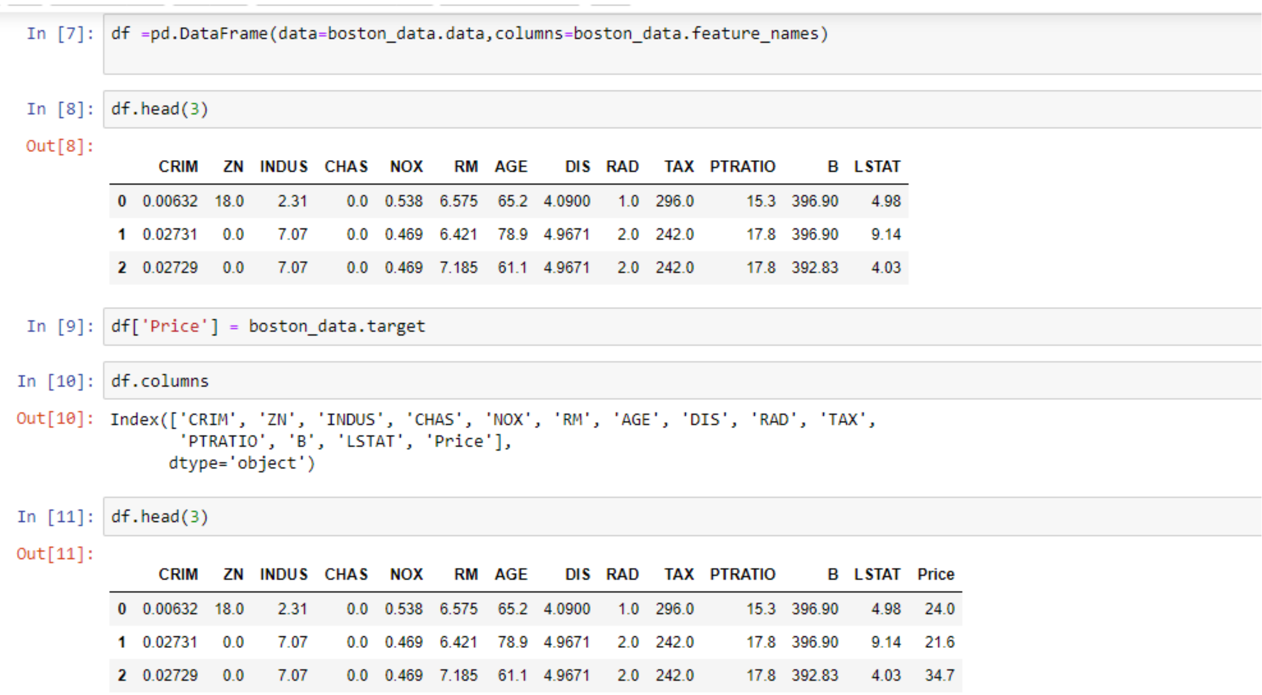


```
In [3]: from sklearn.datasets import load_boston

In [4]: boston_data = load_boston()

In [5]: boston_data.keys()

Out[5]: dict_keys(['data', 'target', 'feature_names', 'DESCR'])

In [6]: #print(boston_data.DESCR)

In [7]: df =pd.DataFrame(data=boston_data.data,columns=boston_data.feature_names)
```

*FIGURE 21. Importing load boston dataset* [53]

In [3], which can be read as cell number 3, or input line 3, imports dataset name **load_boston** from **datasets** module of the sklearn library. Input line 4, is to instantiate the imported library, the "**boston_data**" name is an arbitrary name and can be anything, however, is recommended to provide a meaningful name. Input line 5, display list of keys available in the boston_data since boston_data is dictionary object type in python. Input line 6: prints the description of Boston housing data. It has been commented out in code to reduce the frame of the screenshot. The difference to notice here is between the codes, **boston_data.DESCR VS. print(boston_data.DESCR).** Both print the same result however, later one takes care of indents, line space, alignments and other things to make it more readable.  The reader can uncomment and run the code to read the description or visit the reference 54. [54]

Figure 22 demonstrates the process for creating a data frame from the Boston data from figure 21.



```
In [7]: df =pd.DataFrame(data=boston_data.data,columns=boston_data.feature_names)

In [8]: df.head(3)
Out[8]:
```

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | 4.98 |
| **1** | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | 9.14 |
| **2** | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | 4.03 |

```
In [9]: df['Price'] = boston_data.target

In [10]: df.columns
Out[10]: Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
                'PTRATIO', 'B', 'LSTAT', 'Price'],
               dtype='object')

In [11]: df.head(3)
Out[11]:
```

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | Price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | 4.98 | 24.0 |
| **1** | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | 9.14 | 21.6 |
| **2** | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | 4.03 | 34.7 |

*FIGURE 22. Creating dataframe from boston dataset* [55]

In dataset module, it is imperative to understand, data key has values only for independent features, names of the columns are in columns key. Line 7 provides relevant values to attributes to function “pd.Dataframe()” and creates dataframe object name **df**.

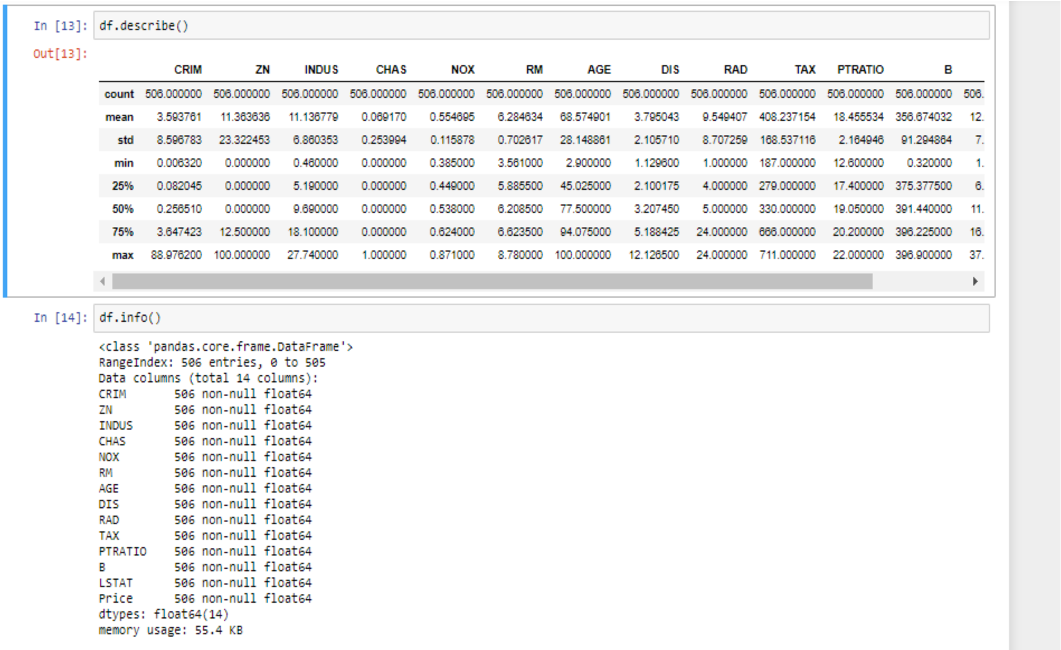
In line 8, we can see the header and 3 rows of **df** dataframe. Still, the dependent part which is the price column is missing. Since, as mentioned in the previous paragraph, we only provided independent features. In datasets module, target key is consists of dependent variable (Vocabulary- Terms – Dependent Variable and Independent Variable). By running line 9, the price column is added to the dataframe “**df**” with the relevant target value. The “**Price**” column is the target column we must predict for this regression problem. Hence from according to formula 1 and formula 2 in the case of Load Boston Data, y is the “**Price**", x's are all other independent variables. Line 10 and 11, confirms the addition of the “**PRICE**” column to df object.

### 5.3 Understanding the Dataset

Exploring and getting proper knowledge of dataset (feature engineering) is one of the most important parts of ML.

#### Pandas Describe and Info

Pandas library has “**describe()**” and info method that helps in data frame analysis. Describe function performs statistical operations and displays value for each numerical column (figure 23).



```
In [13]: df.describe()
Out[13]:
```

|  | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506. |
| **mean** | 3.593761 | 11.363636 | 11.136779 | 0.069170 | 0.554695 | 6.284634 | 68.574901 | 3.795043 | 9.549407 | 408.237154 | 18.455534 | 356.674032 | 12. |
| **std** | 8.596783 | 23.322453 | 6.860353 | 0.253994 | 0.115878 | 0.702617 | 28.148861 | 2.105710 | 8.707259 | 168.537116 | 2.164946 | 91.294864 | 7. |
| **min** | 0.006320 | 0.000000 | 0.460000 | 0.000000 | 0.385000 | 3.561000 | 2.900000 | 1.129600 | 1.000000 | 187.000000 | 12.600000 | 0.320000 | 1. |
| **25%** | 0.082045 | 0.000000 | 5.190000 | 0.000000 | 0.449000 | 5.885500 | 45.025000 | 2.100175 | 4.000000 | 279.000000 | 17.400000 | 375.377500 | 6. |
| **50%** | 0.256510 | 0.000000 | 9.690000 | 0.000000 | 0.538000 | 6.208500 | 77.500000 | 3.207450 | 5.000000 | 330.000000 | 19.050000 | 391.440000 | 11. |
| **75%** | 3.647423 | 12.500000 | 18.100000 | 0.000000 | 0.624000 | 6.623500 | 94.075000 | 5.188425 | 24.000000 | 666.000000 | 20.200000 | 396.225000 | 16. |
| **max** | 88.976200 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.780000 | 100.000000 | 12.126500 | 24.000000 | 711.000000 | 22.000000 | 396.900000 | 37. |

```
In [14]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
Price      506 non-null float64
dtypes: float64(14)
memory usage: 55.4 KB
```

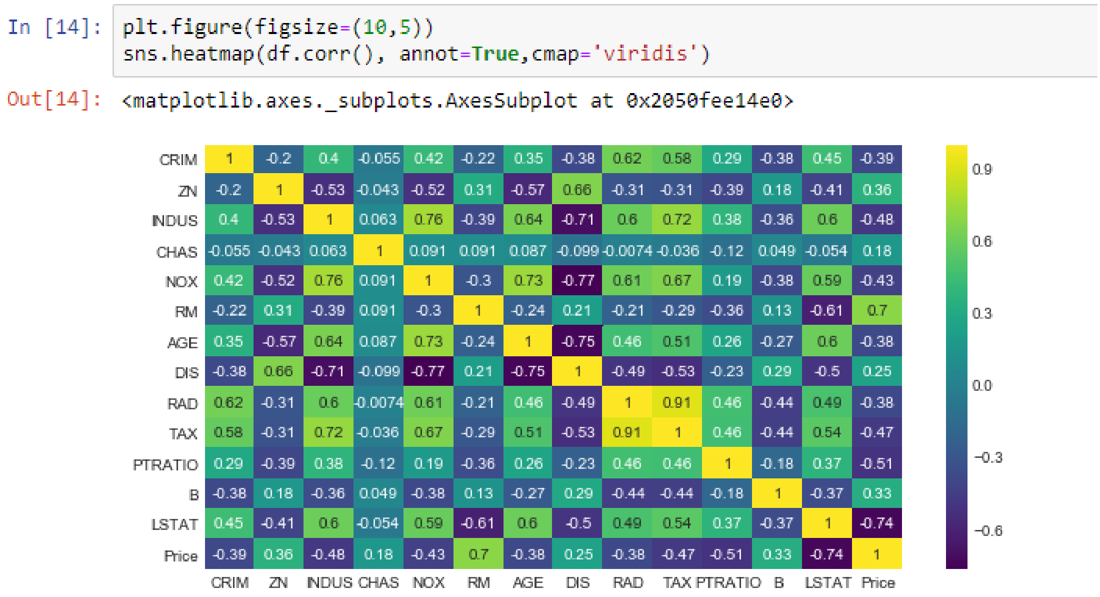
*FIGURE 23. Pandas describe and info* [56]

For instance, percentage values are first quartile (25%), median (50%) and third quartile (75%) in statistics. Describe method can be helpful to see the dispersion of data, detect outliers. [57]

The “info()” method prints information about a data frame, data types of columns including index, and the memory usage (figure 7). We can see columns with null values with info methods. Boston data, however, is already free from any null values. [58]

### Using Pyplot and Seaborn

Graphical representation of data can elaborate on the relation of the database. Let’s, create a heatmap, from the correlation between features.



```
In [14]: plt.figure(figsize=(10,5))
         sns.heatmap(df.corr(), annot=True,cmap='viridis')

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x2050fee14e0>
```

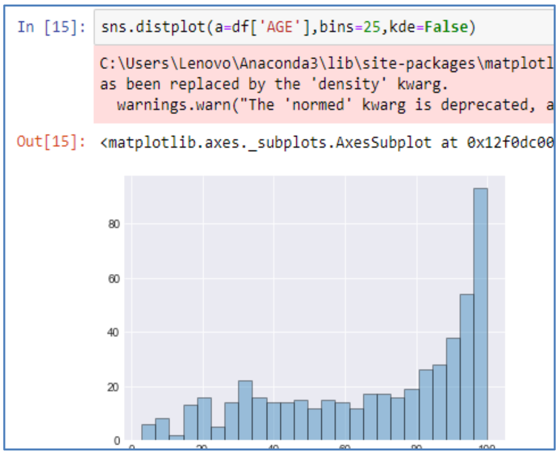
| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | Price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CRIM | 1 | -0.2 | 0.4 | -0.055 | 0.42 | -0.22 | 0.35 | -0.38 | 0.62 | 0.58 | 0.29 | -0.38 | 0.45 | -0.39 |
| ZN | -0.2 | 1 | -0.53 | -0.043 | -0.52 | 0.31 | -0.57 | 0.66 | -0.31 | -0.31 | -0.39 | 0.18 | -0.41 | 0.36 |
| INDUS | 0.4 | -0.53 | 1 | 0.063 | 0.76 | -0.39 | 0.64 | -0.71 | 0.6 | 0.72 | 0.38 | -0.36 | 0.6 | -0.48 |
| CHAS | -0.055 | -0.043 | 0.063 | 1 | 0.091 | 0.091 | 0.087 | -0.099 | -0.0074 | -0.036 | -0.12 | 0.049 | -0.054 | 0.18 |
| NOX | 0.42 | -0.52 | 0.76 | 0.091 | 1 | -0.3 | 0.73 | -0.77 | 0.61 | 0.67 | 0.19 | -0.38 | 0.59 | -0.43 |
| RM | -0.22 | 0.31 | -0.39 | 0.091 | -0.3 | 1 | -0.24 | 0.21 | -0.21 | -0.29 | -0.36 | 0.13 | -0.61 | 0.7 |
| AGE | 0.35 | -0.57 | 0.64 | 0.087 | 0.73 | -0.24 | 1 | -0.75 | 0.46 | 0.51 | 0.26 | -0.27 | 0.6 | -0.38 |
| DIS | -0.38 | 0.66 | -0.71 | -0.099 | -0.77 | 0.21 | -0.75 | 1 | -0.49 | -0.53 | -0.23 | 0.29 | -0.5 | 0.25 |
| RAD | 0.62 | -0.31 | 0.6 | -0.0074 | 0.61 | -0.21 | 0.46 | -0.49 | 1 | 0.91 | 0.46 | -0.44 | 0.49 | -0.38 |
| TAX | 0.58 | -0.31 | 0.72 | -0.036 | 0.67 | -0.29 | 0.51 | -0.53 | 0.91 | 1 | 0.46 | -0.44 | 0.54 | -0.47 |
| PTRATIO | 0.29 | -0.39 | 0.38 | -0.12 | 0.19 | -0.36 | 0.26 | -0.23 | 0.46 | 0.46 | 1 | -0.18 | 0.37 | -0.51 |
| B | -0.38 | 0.18 | -0.36 | 0.049 | -0.38 | 0.13 | -0.27 | 0.29 | -0.44 | -0.44 | -0.18 | 1 | -0.37 | 0.33 |
| LSTAT | 0.45 | -0.41 | 0.6 | -0.054 | 0.59 | -0.61 | 0.6 | -0.5 | 0.49 | 0.54 | 0.37 | -0.37 | 1 | -0.74 |
| Price | -0.39 | 0.36 | -0.48 | 0.18 | -0.43 | 0.7 | -0.38 | 0.25 | -0.38 | -0.47 | -0.51 | 0.33 | -0.74 | 1 |

*FIGURE 24. Heatmap of features correlation* [59]

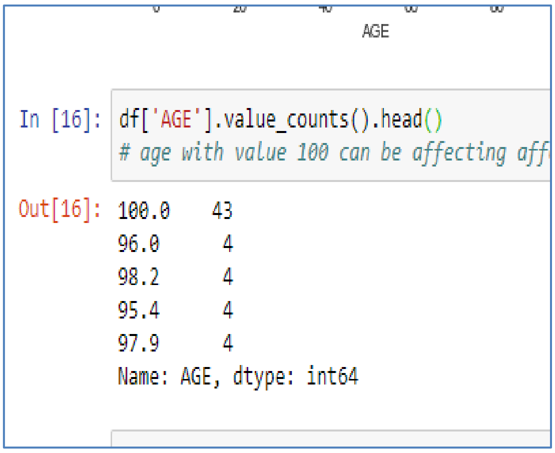
Correlation values represent the interrelation between features with values ranging from -1 to 1. Higher the value higher the correlation and vice-versa, positive interprets positive relation (increment in one increase the other) and negative indicates negative relation (decrement in one decrease the other).

Based on the heat map, it seems the feature “**RM**” has a high correlation with the "**Price**” column in a positive way, whereas **B** has a very low correlation with all other features. Also, “**CRIM**” which is related to the crime rate affects price in a negative way as it should.

Let’s observe figure 25 and 26, both display the same information, former in graphical form and later in numerical. It can be concluded that the age column (**df[‘AGE’]** from figure 25 and 26) has unbalanced value 100, which is highly repeated in comparison to other values.



```
In [15]: sns.distplot(a=df['AGE'],bins=25,kde=False)

C:\Users\Lenovo\Anaconda3\lib\site-packages\matplotl
as been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, a

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x12f0dc00
```



```
In [16]: df['AGE'].value_counts().head()
         # age with value 100 can be affecting aff

Out[16]: 100.0    43
         96.0      4
         98.2      4
         95.4      4
         97.9      4
         Name: AGE, dtype: int64
```

*FIGURE 25. Age count plot* [60]

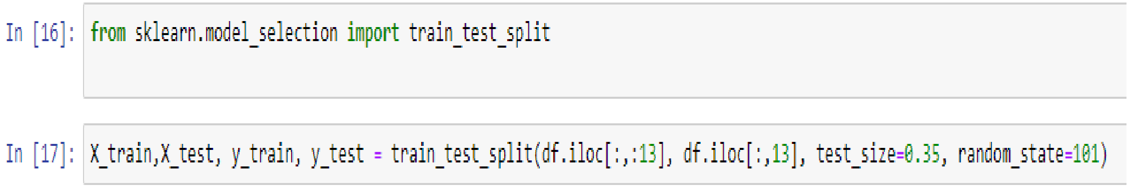
*FIGURE 26. Age value count* [61]

Other graphical representations that can be used to explore and understand the relations between data are Joint plot, pair plot, KDE plot, Dist plot (histogram) from Seaborn. Seaborn is built on top of matplotlib, hence, it is always possible to use the “**PyPlot”** only. If the reader is interested more in learning exploration of data in python visit the link at reference 62 for a page at Kaggle.com.

### 5.3.1 Splitting Data

The sklearn package provides a module name "**model_selection**” with submodule “**tain_test_split”** that can be used to split the data. [63]

Figure 27 represents a portion of code used to split our Boston data. The train_test_split submodule divides the data into independent features for training (X_train), independent features for testing (X_test), dependent features for training (y_train) and dependent data for testing (y_test). The important thing to note is that order of initiating these four variables. The naming of datasets are arbitrary however, it is a convention of prefix train independent dataset with capital “X” followed by suffix “_train” and “_test” for training and testing data respectively. Similarly, for dependent or target features are named as the prefix of small “y” followed by suffixes “_train” and “_test”.



```
In [16]: from sklearn.model_selection import train_test_split

In [17]: X_train,X_test, y_train, y_test = train_test_split(df.iloc[:,:13], df.iloc[:,13], test_size=0.35, random_state=101)
```

*FIGURE 27. Data splitting* [64]

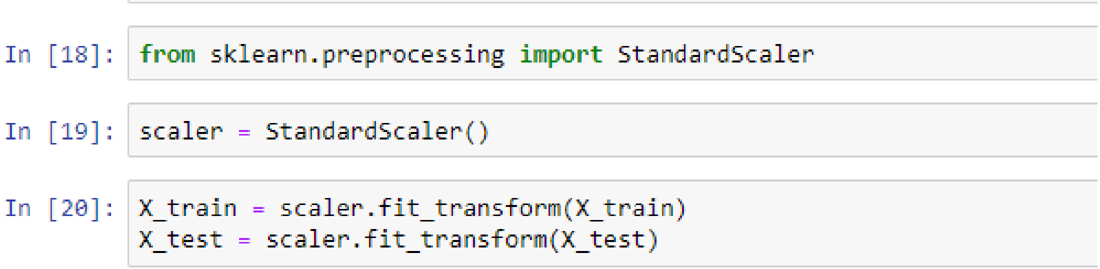
Pandas offer several methods for subsetting or slicing data frames.

1. Iloc: It is integer based locating method. [65]
2. Loc: It is the Label (name of column and index) based locating method. [66]
3. Cross Section (xs): It takes the key argument in the multilevel index. [67]

Boston data has been divided into train and test data with test data consisting of 35 per cent of the original data extracted randomly. The **df.iloc[ : , :13]** returns all rows and independent feature columns i.e. 0 to 12 columns from beginning since in python indexing starts from 0 not 1 and **df.iloc[ : ,13]** returns our dependent feature column “Price”. The attribute “**random_state**” provides the same value through random every time code is run in a session. It is done for consistency. [63]

### 5.3.2 Scaling Data

As mentioned in Chapter 4, data must be normalized before training. Figure 28 displays how X_train and X_test were scaled using "StandardScaler" provided by a python in the module “**preprocessing**”. [68]



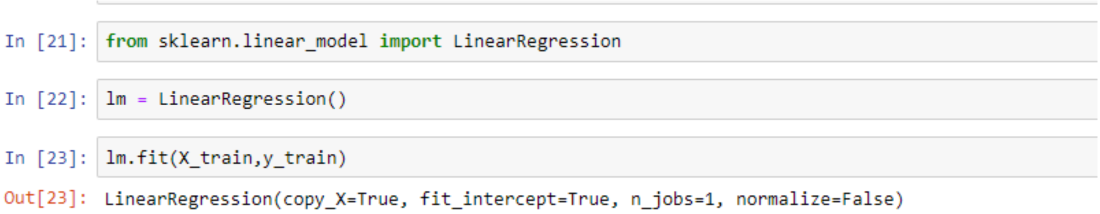
```
In [18]: from sklearn.preprocessing import StandardScaler

In [19]: scaler = StandardScaler()

In [20]: X_train = scaler.fit_transform(X_train)
         X_test = scaler.fit_transform(X_test)
```

*FIGURE 28. Data scaling* [69]

Sometimes “**fit_transform()**” method is also performed in two steps. Using “**fit()**” and then “**transform()**” in next line. Both steps provide the same result. [68]

### 5.3.3 Train Model

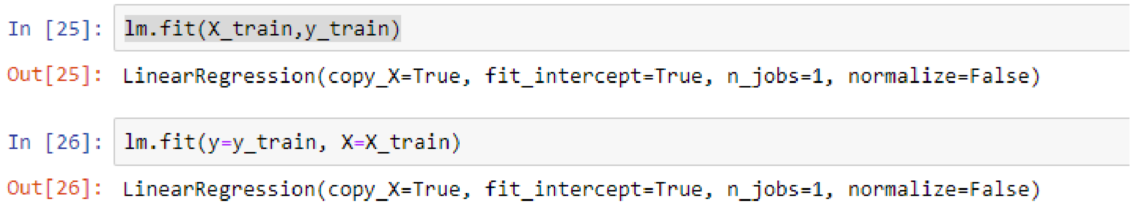
Sklearn has the module name “**linear_model**” which is consist of “**LinearRegression**” model which is being imported as shown in figure 29. [70]



```
In [21]: from sklearn.linear_model import LinearRegression

In [22]: lm = LinearRegression()

In [23]: lm.fit(X_train,y_train)

Out[23]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

*FIGURE 29. Import and train linear model* [71]

After importing any model, it is important to initialize the model, Input cell 22, initializes linear regression model with name “**lm**” which is an arbitrary name. Model is trained by feeding training data in input cell 23, with method "**fit()**”, which takes two inputs, independent feature “X_train” and dependent feature “y_train” of training data. The output 23, is the linear regression model that has been trained. Other parameters like **copy_x**, **fit_intercept** are default parameters that were not changed during initialization. [70]



```
In [25]: lm.fit(X_train,y_train)
Out[25]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [26]: lm.fit(y=y_train, X=X_train)
Out[26]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

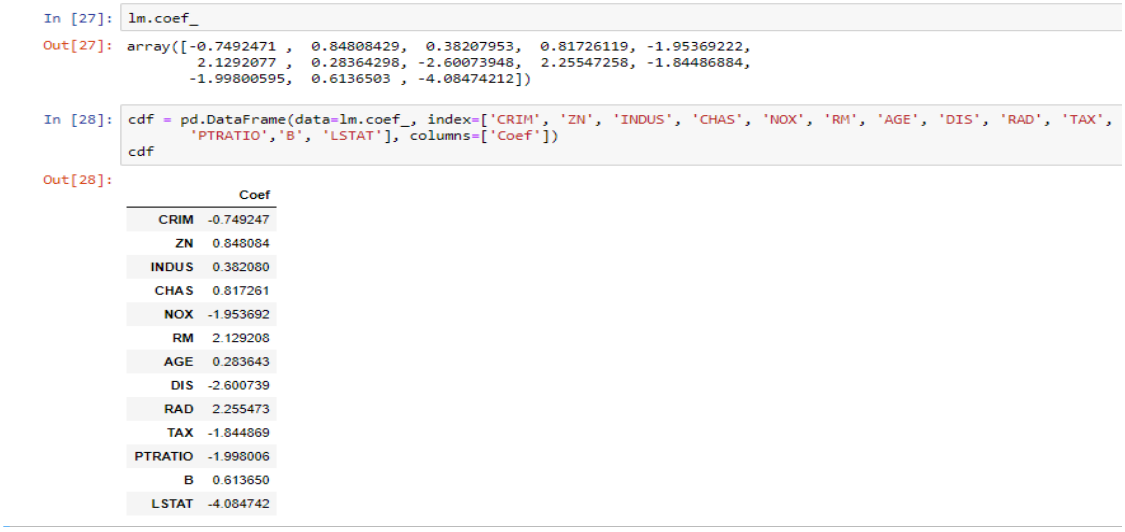
*FIGURE 30. Default vs custom attribute assignment order* [72]

Figure 30 is for a demonstration that the order of the input matters, when the input is not ordered its important to relate them to their proper attributes or else error will occur.

### Exploring Results of Train

Result of the training model can be explored using several methods provided by the linear model package. [70]

In figure 31 coefficient of linear models has been converted to dataframe for better understanding.



```
In [27]: lm.coef_
Out[27]: array([-0.7492471 ,  0.84808429,  0.38207953,  0.81726119, -1.95369222,
                 2.1292077 ,  0.28364298, -2.60073948,  2.25547258, -1.84486884,
                -1.99800595,  0.6136503 , -4.08474212])

In [28]: cdf = pd.DataFrame(data=lm.coef_, index=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
                'PTRATIO','B', 'LSTAT'], columns=['Coef'])
         cdf
Out[28]:
```

| | Coef |
|---|---|
| CRIM | -0.749247 |
| ZN | 0.848084 |
| INDUS | 0.382080 |
| CHAS | 0.817261 |
| NOX | -1.953692 |
| RM | 2.129208 |
| AGE | 0.283643 |
| DIS | -2.600739 |
| RAD | 2.255473 |
| TAX | -1.844869 |
| PTRATIO | -1.998006 |
| B | 0.613650 |
| LSTAT | -4.084742 |

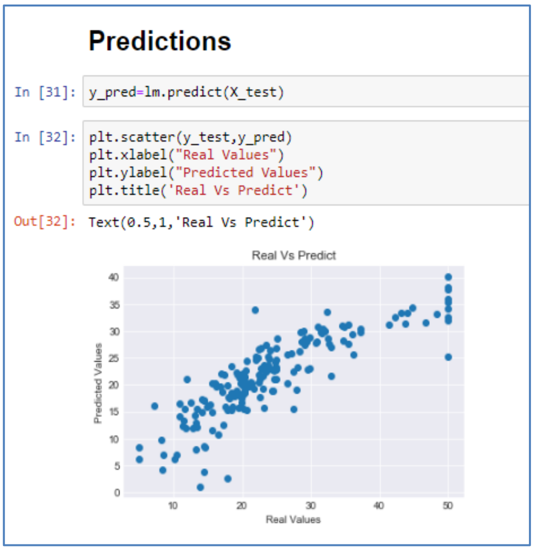
*FIGURE 31. Linear model coefficient in dataframe* [73]

It is imperative to realize algorithm takes one whole row as input or multiple rows as one input if specified for instance, common in Neural Networks which is out of the scope of this thesis. But algorithms do not take one column at once as input. From the formula of linear regression, it implies that coefficient plays a detrimental role in the value of y. Negative coefficient causes in decrease value of prediction value and vice versa. The sign of coefficient and their effects are the same for all algorithms. For instance, in our case, column **CRIM** has a coefficient, “ **-0.749247**” which is 1 unit, hence, every decrease in 1 unit of CRIM causes a decrease in housing price.

Now for the intercept i.e. “**c**” in our algorithm (formula 1 and formula 2), “**lm.intercept_**”, though not in picture however, has been calculated in full code section. When intercept is zero the line is passing through the origin or vice-versa.

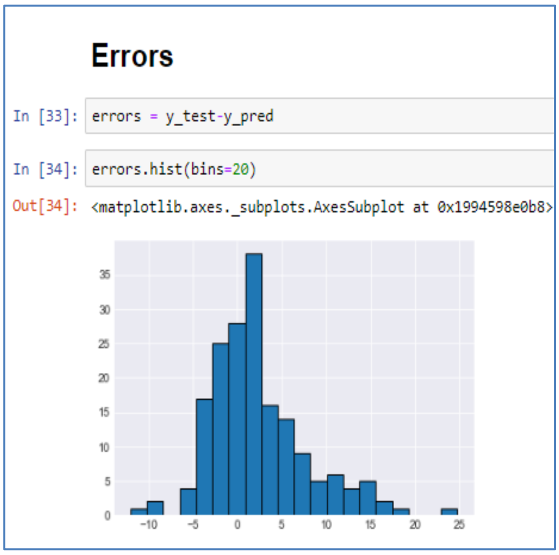
### 5.3.4 Predictions and Errors

The “**predict()**” method provided by linear regression can be used for value prediction which takes X_test as input. In figure 16, Plot “**Real Vs Predict**” displays the scatter plot of real and predicted values for Boston data. The almost straight slope has been observed in the scatter plot can be portrayed as the correct choice of using a regression model.  The second part of the figure 32, errors, which is the difference between real and predicted values, is displayed as a histogram. The bell-shaped graph also is known as “Bell curve” or “Gaussian Distribution” implies that outliers are not present in the data.



Predictions

```
In [31]: y_pred=lm.predict(X_test)

In [32]: plt.scatter(y_test,y_pred)
         plt.xlabel("Real Values")
         plt.ylabel("Predicted Values")
         plt.title('Real Vs Predict')

Out[32]: Text(0.5,1,'Real Vs Predict')
```



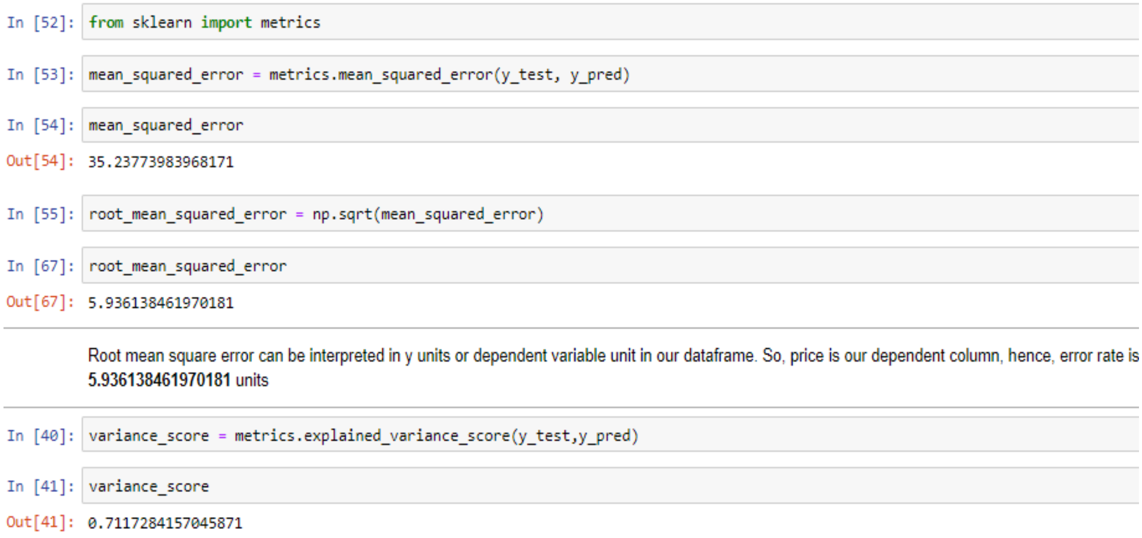
Errors

```
In [33]: errors = y_test-y_pred

In [34]: errors.hist(bins=20)

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x1994598e0b8>
```

*FIGURE 32. Predictions, errors and distribution* [74] [75]

### 5.3.5 Evaluation

Prediction of the linear model for the Boston data has been evaluated in figure 33.



```
In [52]: from sklearn import metrics

In [53]: mean_squared_error = metrics.mean_squared_error(y_test, y_pred)

In [54]: mean_squared_error
Out[54]: 35.23773983968171

In [55]: root_mean_squared_error = np.sqrt(mean_squared_error)

In [67]: root_mean_squared_error
Out[67]: 5.936138461970181
```

Root mean square error can be interpreted in y units or dependent variable unit in our dataframe. So, price is our dependent column, hence, error rate is **5.936138461970181** units

```
In [40]: variance_score = metrics.explained_variance_score(y_test,y_pred)

In [41]: variance_score
Out[41]: 0.7117284157045871
```

*FIGURE 33. Evaluation of regression model* [76]

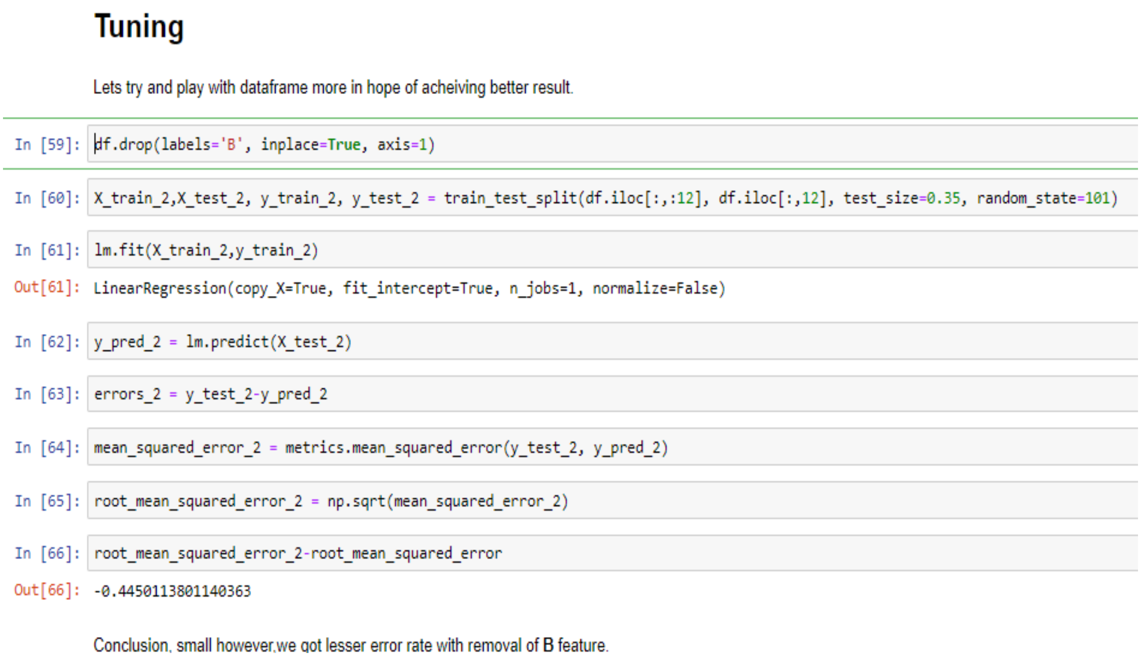
“**Metrics**" sub-module provides method “**mean_squared_error()**” to calculate mean square error [77 ]. In cell 55, the  root mean square error has been calculated using NumPy’s built-in function called “sqrt()”. [78]

Explained Variance Score can be interpreted as quality grading of the regression model. Higher the score better the model and vice-versa. [79]

### 5.3.6 Optimization

It is time to get back to the preprocessing phase again and observe what are the changes possible to data for better the model if we decide to use the same model. Sometimes, if the evaluation implies that our model is not a good fit for this data, during such case replacing the model with a more complex model is a good choice too.

In the case of Boston Data, as concluded in figure 24, let's drop feature column "**B**”, and repeat the whole process as demonstrated in figure 34.



## Tuning

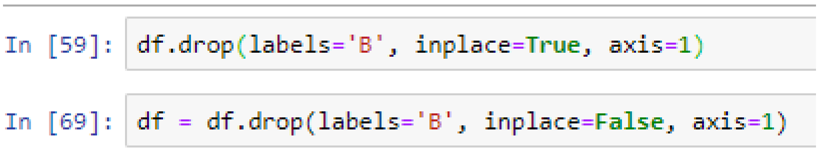
Lets try and play with dataframe more in hope of acheiving better result.

```
In [59]: df.drop(labels='B', inplace=True, axis=1)

In [60]: X_train_2,X_test_2, y_train_2, y_test_2 = train_test_split(df.iloc[:,:12], df.iloc[:,12], test_size=0.35, random_state=101)

In [61]: lm.fit(X_train_2,y_train_2)

Out[61]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [62]: y_pred_2 = lm.predict(X_test_2)

In [63]: errors_2 = y_test_2-y_pred_2

In [64]: mean_squared_error_2 = metrics.mean_squared_error(y_test_2, y_pred_2)

In [65]: root_mean_squared_error_2 = np.sqrt(mean_squared_error_2)

In [66]: root_mean_squared_error_2-root_mean_squared_error

Out[66]: -0.4450113801140363
```

Conclusion, small however,we got lesser error rate with removal of B feature.

*FIGURE 34. Optimizing linear model for boston data* [80]

Pandas “**drop()**” can be used to drop columns. It is imperative to note that “axis” attribute is used for targeting either row or column with values 0 and 1 respectively. Figure 35 shows the different ways of using the drop function.



```
In [59]: df.drop(labels='B', inplace=True, axis=1)

In [69]: df = df.drop(labels='B', inplace=False, axis=1)
```

*FIGURE 35. Inplace parameter use case* [81]

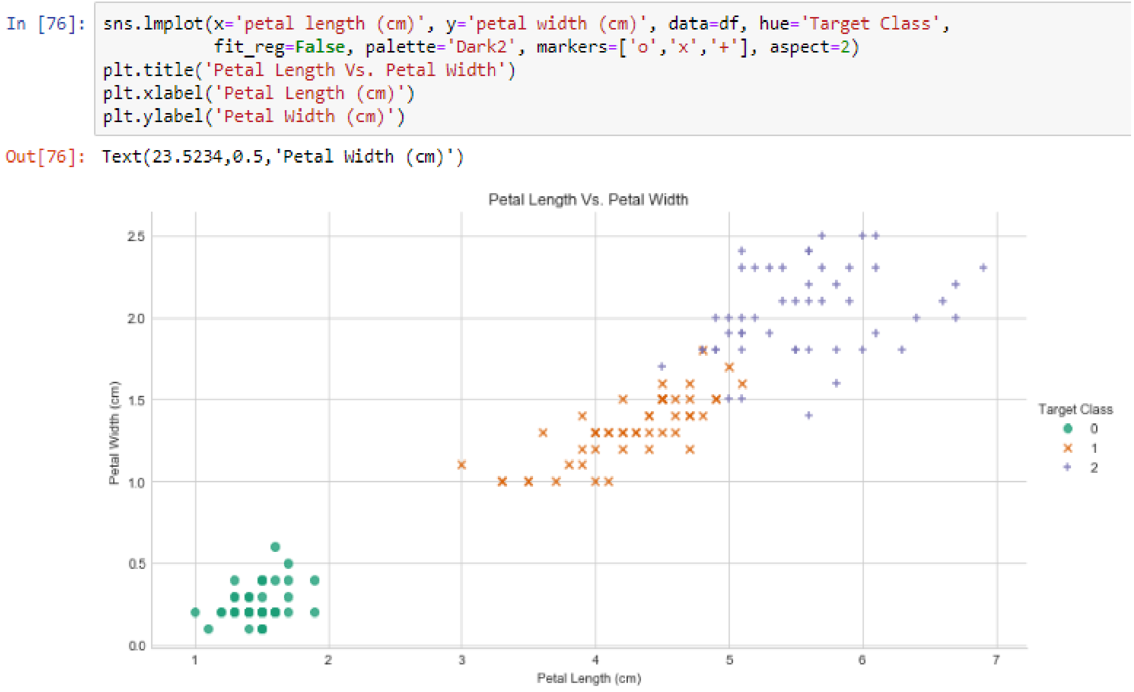
Dropping column “B” had very little but improvement in the error rates since the difference in RMSE in phase 1 and 2 is positive. This still shows there are lots of ways to improve the quality of the model, domain knowledge does play significant roles in tuning.The full version of code for Linear Regression can be found at the link provided by reference 82. [82]

### 5.4 K-Nearest Neighbor

The steps for Importing Iris Dataset and Preparing the dataframe are similar to the Load Boston Datasets that have been demonstrated for Regression Algorithm [82]. Please visit the full code version for KNN in the link provided at reference 83 to see the pre-processing phase. [83]

#### 5.4.1 Understanding Iris Dataset

Figure 36 displays the plot between the petal length and petal width of all three classes. Apart from **lmplot,** pairplot can be used to plot all the numerical columns of data-frame at once to see the relations, but bigger the database more time is consumed for plotting. (see the full version of code for pairplot [83]). [84]



```
In [76]: sns.lmplot(x='petal length (cm)', y='petal width (cm)', data=df, hue='Target Class',
                    fit_reg=False, palette='Dark2', markers=['o','x','+'], aspect=2)
         plt.title('Petal Length Vs. Petal Width')
         plt.xlabel('Petal Length (cm)')
         plt.ylabel('Petal Width (cm)')

Out[76]: Text(23.5234,0.5,'Petal Width (cm)')
```

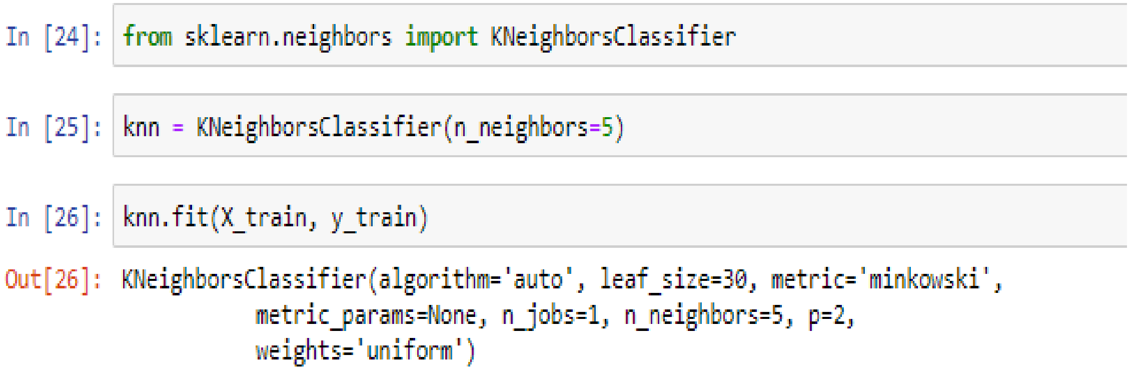
*FIGURE 36. Petal length vs. petal width* [85]

The figure 36 demonstrates that target Class 1 and Class 2 which are “**Versicolor**”, “**Virginica**” (visit, or run full code version or see dataset description from reference) are more similar since they are gathered together. Target Class 0 “**Setosa**”, on the contrary, is quite far and distinguished from other classes. By

Replace the values of x and y with sepal length and width and run code from figure 36, the graph will provide a similar result of correlation.

Splitting and Scaling the data has been done in a similar fashion as Linear Regression. So, let's jump to Training and Testing Data. [83]

### 5.4.2 Train Model



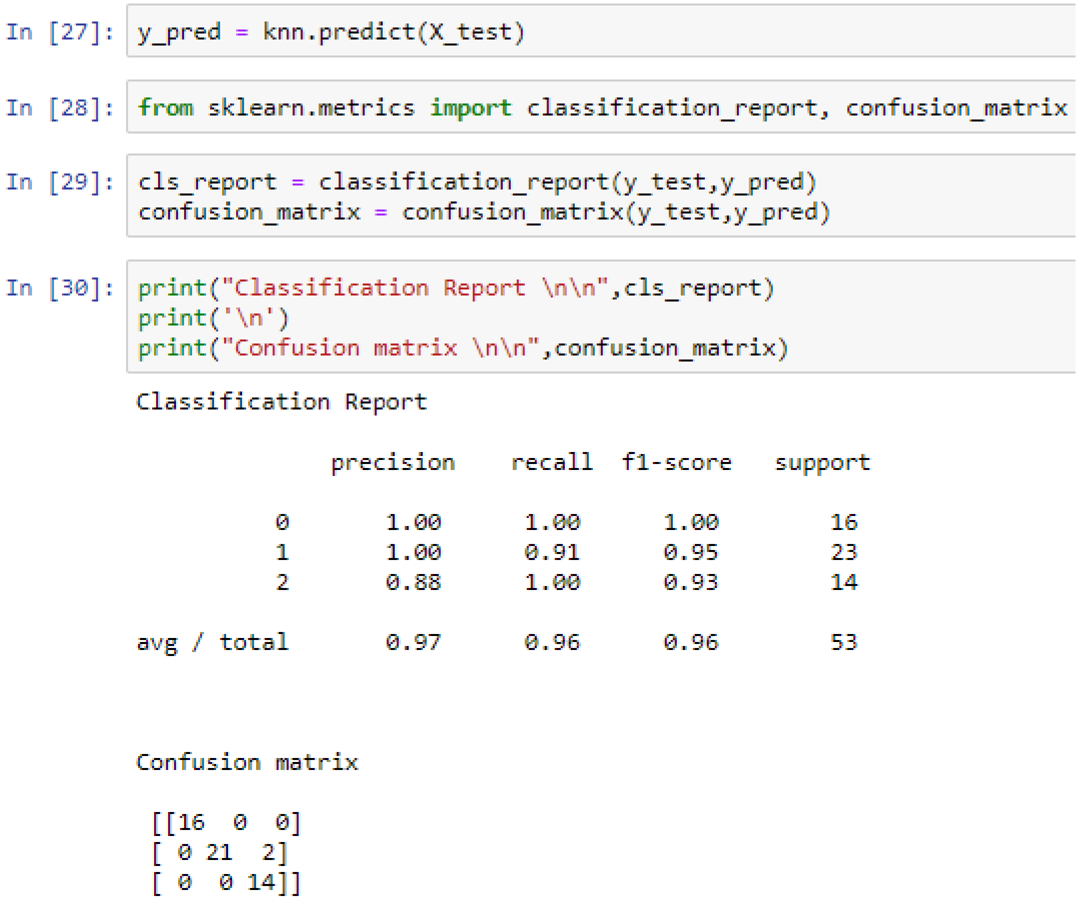
```
In [24]: from sklearn.neighbors import KNeighborsClassifier

In [25]: knn = KNeighborsClassifier(n_neighbors=5)

In [26]: knn.fit(X_train, y_train)

Out[26]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                    weights='uniform')
```

*FIGURE 37. Train knn model* [86]

Let us observe figure 37, “**KNeighorsClassifier**” is the name of the algorithm in “**neighbors**” family of the sklearn package which is KNN algorithm being imported in line 24. The initialization of the model is done in line 25, where the number of neighbors during the first training session is 5, which is also the default value of “**n_neighbors**” parameter. In line 26, the model is fitted with train data and the output 26 display the KNN trained model with all its parameters and their values used by the model.

### 5.4.3 Prediction and Errors

In figure 38, line 27, we declare and predict “**y_pred**” like our regression algorithm. Line 28, we import Classification Report and Confusion Matrix which are being executed at line 29. We print the tables in line 30.



```
In [27]: y_pred = knn.predict(X_test)

In [28]: from sklearn.metrics import classification_report, confusion_matrix

In [29]: cls_report = classification_report(y_test,y_pred)
         confusion_matrix = confusion_matrix(y_test,y_pred)

In [30]: print("Classification Report \n\n",cls_report)
         print('\n')
         print("Confusion matrix \n\n",confusion_matrix)
```

```
Classification Report

              precision    recall  f1-score   support

          0       1.00      1.00      1.00        16
          1       1.00      0.91      0.95        23
          2       0.88      1.00      0.93        14

avg / total       0.97      0.96      0.96        53



Confusion matrix

 [[16  0  0]
 [ 0 21  2]
 [ 0  0 14]]
```

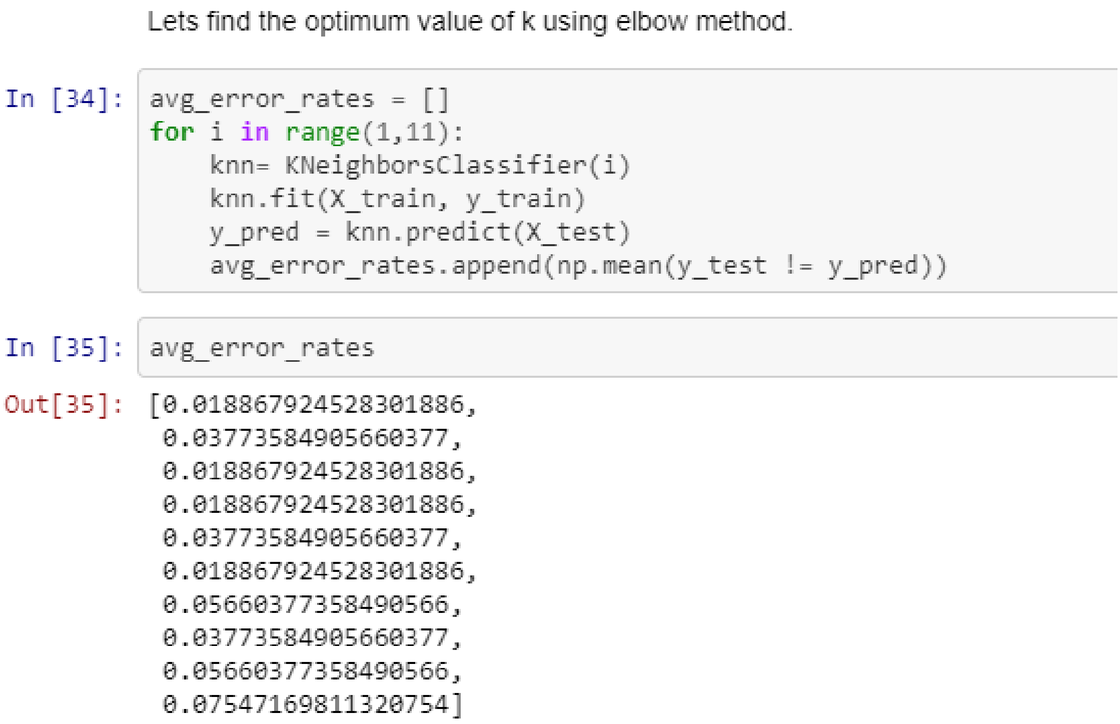
*FIGURE 38. Prediction and error knn model* [87]

Check the index or first column of Classification Report. The values 0,1 and 2 are the classes in the dependent variable or three species of Iris Flowers. Classification report provides a classification report for prediction in each class. And At the bottom row, it gives average each metrics for all classes altogether. By looking at the score our model is already very well trained. Confusion Matrix, where correct predictions are the main diagonal values i.e. in matrix index values at position [0,0], [1,1], [2,2]. And misclassification is just 2 values.

Accuracy of our model is, Accuracy (formula 6) = (16+21+14) / (16+21+14 + 2) = 0.96 = 96%.

### 5.4.4 Elbow Method and Grid Search CV

In figure 39 we calculate list “**avg_error_rates**” in line 34, the result is being outputted in output 35. In classification algorithms, errors are misclassification or cases where prediction values are incorrect from actual values.



Lets find the optimum value of k using elbow method.

```
In [34]: avg_error_rates = []
         for i in range(1,11):
             knn= KNeighborsClassifier(i)
             knn.fit(X_train, y_train)
             y_pred = knn.predict(X_test)
             avg_error_rates.append(np.mean(y_test != y_pred))

In [35]: avg_error_rates

Out[35]: [0.018867924528301886,
          0.03773584905660377,
          0.018867924528301886,
          0.018867924528301886,
          0.03773584905660377,
          0.018867924528301886,
          0.05660377358490566,
          0.03773584905660377,
          0.05660377358490566,
          0.07547169811320754]
```

*FIGURE 39. Error calculation for a range of k* [88]

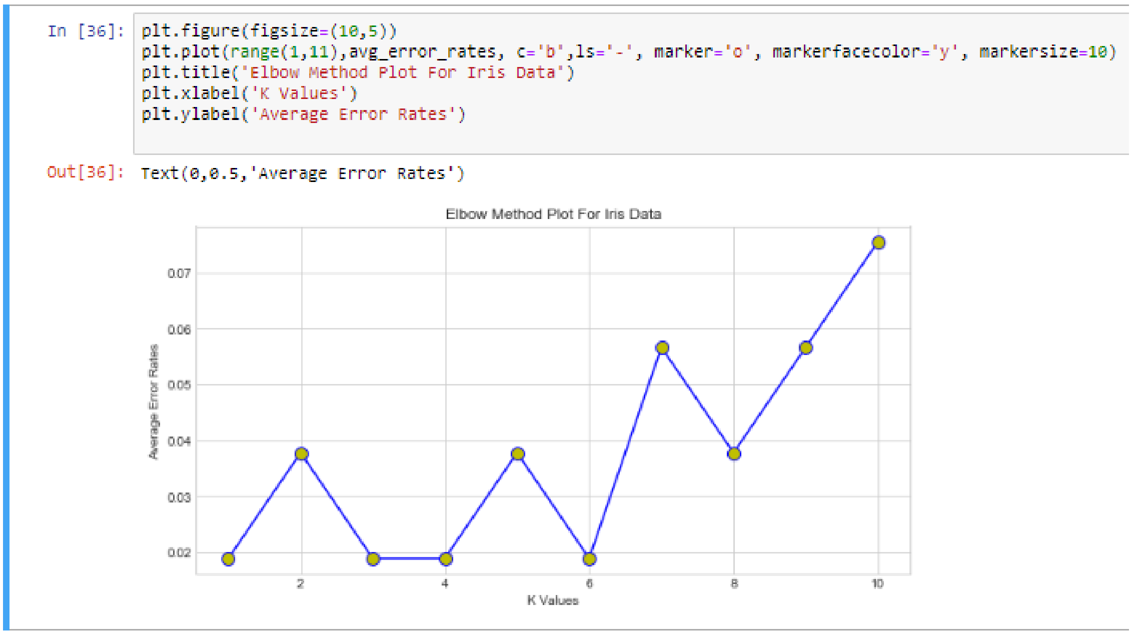
We are calculating errors for values of K from range(1,11) i.e. values 1 to 10. In each iteration of for loop, the KNN algorithm is initialized using the value of “**i**” that goes from 1 to 10. The train and prediction processes are the same as the model train and prediction section. What differs is last code of cell 31. i.e. **avg_error_rates.append(np.mean(y_test != y_pred)).** Let’s divide and explain in section.

1. **y_test != y_pred** checks error i.e. if y_test is not equal to y_pred, if not it returns the value Ture.
2. **np.mean(y_test != y_pred))**, calculates mean of all errors i.e. mean of all cases where the value True was returned. [89]

3. **avg_error_rates.append(np.mean(y_test != y_pred))**, adds each average values to avg_error_rates list which is initially empty.

At the end of iteration of for loop, we get average error rates for algorithms, with k values ranging from 1 to 10 respectively.

Our elbow graph plot in figure 40 does not look like an elbow due to irregular error rates, however, is indeed informative (chapter 4, Elbow Method). In figure 40, error rates are in y-axis in increasing order from bottom to top and k-values are in x-axis in increasing order from left to right. As we can see error rates from different values of K are not that huge but there is a very small difference. The error rates are very small and almost similar when k values are 1,3,4 and 6. When the value of K is 10, the error rate is the maximum among the given values of K. So, 3,4,6 are the values that can be tested to find optimum results. Using 1 as the value of is not a good idea since 1 number of neighbor can never compare the odds of the data point in question with more than one prediction class.



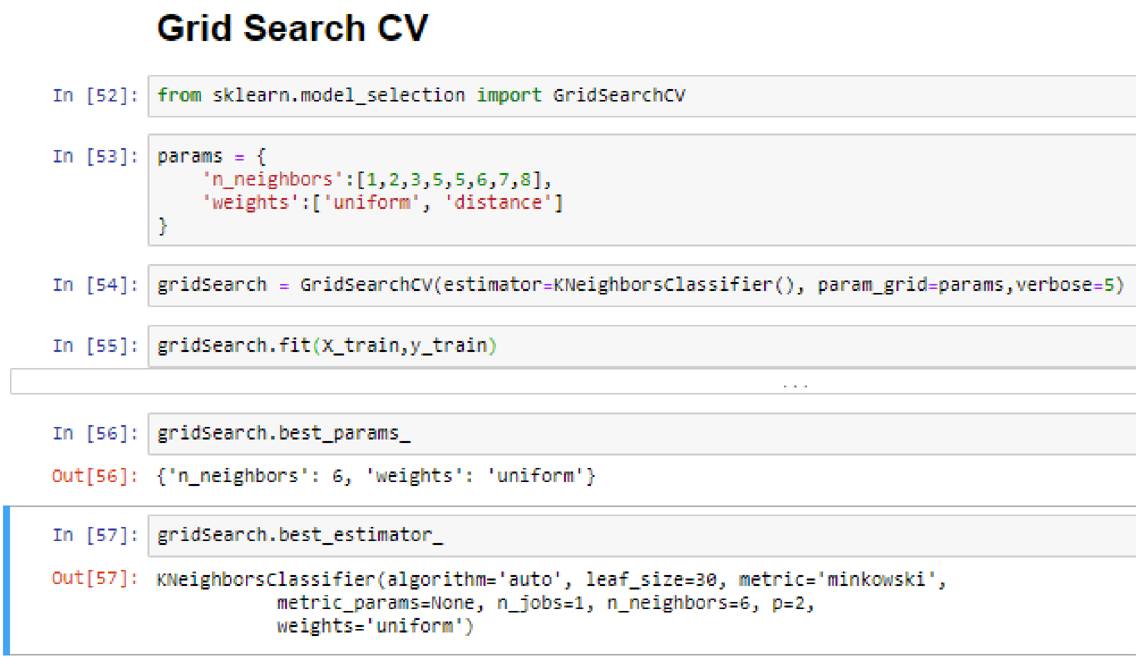
```
In [36]: plt.figure(figsize=(10,5))
         plt.plot(range(1,11),avg_error_rates, c='b',ls='-', marker='o', markerfacecolor='y', markersize=10)
         plt.title('Elbow Method Plot For Iris Data')
         plt.xlabel('K Values')
         plt.ylabel('Average Error Rates')

Out[36]: Text(0,0.5,'Average Error Rates')
```

*FIGURE 40. Elbow method plot for iris dataset* [90]

Grid Search CV is a method of searching best fit for an estimator(algorithm). In grid search, we provide a list of values for different parameters of an algorithm and then iterate through each of them. After the end of execution, GridSearchCV provides the best fit based on the parameters and their value provided for search.

It is imperative to remember,

1. GridSearchCV is for classification algorithms.
2. It can be used for several Classification algorithms.
3. Parameters to input are different for different algorithms. We must check their documentation for parameters.
4. Grid Search uses default values of parameters if options are not provided.
5. In Sklearn, “**model_selection**” module provides “**GridSearchCV**” method. [92]



Grid Search CV

```
In [52]: from sklearn.model_selection import GridSearchCV

In [53]: params = {
             'n_neighbors':[1,2,3,5,5,6,7,8],
             'weights':['uniform', 'distance']
         }

In [54]: gridSearch = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=params,verbose=5)

In [55]: gridSearch.fit(X_train,y_train)

                                       ...

In [56]: gridSearch.best_params_

Out[56]: {'n_neighbors': 6, 'weights': 'uniform'}

In [57]: gridSearch.best_estimator_

Out[57]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=6, p=2,
                    weights='uniform')
```

*FIGURE 41. Grid search cv* [91]

Let’s describe figure 41, from cell to cell. In cell 52 we import GridSearchCV from model_selection module of sklearn.In line 53, we declare dictionary python object name “**params**” with its key and value pairs. The params object has two keys “**n_neighbors**” and “**weights**”. These are the parameters of KNN algorithms. In our experiment values of k_neighbors range from 1 to 8, and weights two metrics “**uniform**” and “**distance**”. In Cell 54 we initialize GridSearchSV, where “**estimator**” takes value of “**KNearestClassifier()**” which is KNN. It is important to note the "**estimator**" parameter can take a list of algorithms, not just one if we want to check other algorithms as well.  The “**parma_grid**” attribute is where we provide our custom object “**params”**. The verbose displays the whole process in the console. [92]

Line 55, we fit the GridSearchCV with our training data. After training, in line 56 we can see the “**best_params_**” a built-in property of GridSearchCv, which gives the best parameters from the list of parameters and values we provided. From the Grid Search, we can say the optimal number of K is 6 for our model.

In line 57, Grid provides the best estimator KNN, since we only provide KNN in the “**estimator**” parameter during initializing GridSearchCV.  As mentioned earlier Grid uses default values for the attributes if their custom value is not provided. If the reader wants to try the result of Grid Search, they may do so using the train and test processes described in the thesis.

# 6 CONCLUSION

The paper answered the following questions or tasks.

1. What is Machine Learning? (Chapter 2)
2. What are the different types of Machine Learning? (Chapter 2)
3. What is supervised Machine Learning? Give some examples. (Chapter 2)
4. What are the different phases in Machine Learning? Explain them. (Chapter 4)
5. Implement Regression and Classification algorithm in a python programming language. (Chapter 5)

“General Machine Learning Practices in Python”, meaning - What are the basic machine learning processes that can be done in Python programming language?

## 6.1 Result

As the name suggests paper aimed to introduce novice users to the world of Machine Learning in Python, which was the whole theme of the paper. To achieve that purpose, it was an absolute necessity to first introduce readers to Machine Learning and phases in ML model development. Then only after that second task of introducing various python packages and their use in ML model development was possible. Hence, the primary theme got divided into two themes as mentioned in Chapter 1 introduction.

The thesis went from introduction of ML to the phases in ML development from Chapter 1 to 4. These chapter act as a base of knowledge which fulfilled our first task. Then the very long chapter 5, made readers aware of several packages and codes to use during several Machine Learning phases which fulfilled our second task.

In conclusion, now, readers can use the python programming to build not only two algorithms (Linear regression and KNN) explained in Chapter 5 but a similar procedure can be carried out for any other supervised learning systems. For unsupervised learning, the most important take away from this paper is the preprocessing phase (chapter 4 and 5). Data is the heart of the ML model; hence its quality is necessary. For that exact reason, the preprocessing phase is very long and detailed in the paper. Moreover, Elbow method can also be used in K-Means Clustering algorithm, a popular unsupervised learning algorithm.

Unfortunately, the paper sets a requirement that the reader should be fluent in python programming to get the best out of the paper. However, if the reader is familiar with other common programming languages for ML, for instance, R and Java then paper at-least helps to provide an idea about how to approach model development.

### 6.2 Future Possibilities

Machine Learning is a huge field where lots of fields overlap, for instance, statistics, business, programming. Many years of relentless approach might not be enough to call yourself an expert in the field.

The reader can start practising ML algorithms other than those provided in paper like Logistic Regression, Decision Trees, SVM (Support Vector Machine) based on the instructions in the paper. For datasets, there are other datasets available in Sklearn package. Moreover, Kaggle.Com is a website for Data Scientist and Machine Learning Engineers where many datasets are available for practice, many experts in the field can provide tips and reviews about your codes. There are also competitions going on all the year round at Kaggle where the rewards can be money or jobs.

The important topics like Bias, Variance, Bias-Variance Trade-Off, Complexity of Algorithms and their effect on Prediction, and many more are not explained in the thesis. Readers can now start diving deeper into the understanding of these topics. Readers who are interested in Mathematics and Intuition of several algorithms can take freely available tutorial of Andrew Ng on YouTube. In his tutorial, he explains intuition about various ML algorithms. [93]

# REFERENCES

[1] Python Tutorial. Python Programming Language. Date of Retrieval 07.03.2019,

https://docs.python.org/3/tutorial/index.html

[2] Swamynathan, M. 2017. Introduction To Machine Learning. Date of Retrieval 07.03.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_2_Chapter.html

[3] Swamynathan, M. 2017. Figure 1. Mastering Machine Learning with Python in Six Steps. Figure 2-9 Types of Machine Learning. Date of Retrieval 09.03.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_2_Chapter.html

[4] Swamynathan, M. 2017. Supervised Learning. Mastering Machine Learning with Python in Six Steps. Date of Retrieval 07.03.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_2_Chapter.html

[5] Sarah Guido, Andreas C. Müller. 2016. Introduction to Machine Learning with Python, 2.1 Classification and Regression. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/ch02.html#classification-and-regression

[6] Guido et al. 2016. Introduction to Machine Learning with Python, Figure 2-11 Predictions of a linear model on the wave dataset. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/ch02.html#classification-and-regression

[7] Swamynathan, M. 2017. Figure 3. Mastering Machine Learning with Python in Six Steps. Figure 3-26 k Nearest Neighbors with k = 5. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

[8] Guido et al. 2016. Introduction to Machine Learning with Python, Chapter 3 Unsupervised Learning and Preprocessing. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/ch03.html#unsupervised-learning-and-preprocessing

[9] Swamynathan, M. 2017. Mastering Machine Learning with Python in Six Steps. Reinforcement Learning. Date of Retrieval 08.03.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_2_Chapter.html

[10] LearnPython. Python Modules and Packages. Date of Retrieval 10.03.2019,

https://www.learnpython.org/en/Modules_and_Packages

[11] Numpy Documentation. Date of Retrieval 10.03.2019,

http://www.numpy.org/

[12] Pandas Documentation. Date of Retrieval 10.03.2019,

https://pandas.pydata.org/

[13] Scikit Learn Documentation. Date of Retrieval 10.03.2019,

https://scikit-learn.org/stable/index.html

[14] Matplotlib Documentation. Date of Retrieval 20.03.2019,

https://matplotlib.org/

[15] Seaborn Documentation. Date of Retrieval 20.03.2019,

https://seaborn.pydata.org/

[16] Swamynathan, M. 2017. Mastering Machine Learning with Python in Six Steps, Feature Engineering. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

[17] Swamynathan, M. 2017. Figure 4. Mastering Machine Learning with Python in Six Steps, Logical Flow of data in Machine Learning model building. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

[18] Swamynathan, M. 2017. Mastering Machine Learning with Python in Six Steps, Dealing With Missing Data. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

[19] Swamynathan, M. 2017. Mastering Machine Learning with Python in Six Steps, Normalizing Data. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

[20] Sklearn. Figure 4. Scikit Learn algorithm cheat-sheet. Date of Retrieval 15.04.2019,

https://scikit-learn.org/stable/_static/ml_map.png

[21] Sklearn, Confusion Matrix. Confusion Matrix Documentation. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

[22] Swamynathan, M. 2017. Figure 6. Mastering Machine Learning with Python in Six Steps,Figure 3-11 Confusion Matrix. Date of Retrieval 25.04.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

[23] Nibesh Khadka, 2019.   Elbow Method in Theory.png. Date of Retrieval 25.04.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/KNN/Elbow%20Method%20In%20Theory.png

[24] Nibesh Khadka, 2019.   Elbow Method Preparation Code. Date of Retrieval 25.04.2019,

https://github.com/nibukdk/Thesis/blob/master/Codes/Prepare%20Elbow%20Method%20Graph.ipynb

[25] Swamynathan, M. 2017. Mastering Machine Learning with Python in Six Steps, Table 3-11 Classification performance matrices. Date of Retrieval 10.05.2019,

https://learning.oreilly.com/library/view/mastering-machine-learning/9781484228661/A434293_1_En_3_Chapter.html

[26] Sklearn, Precision, Recall, and Fscore. Precision_recall_fscore_supprot Documentation. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html#sklearn.metrics.precision_recall_fscore_support

[27] Sklearn, Load Boston Dataset. Details on Load Boston Dataset. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html

[28] Sklearn, Iris Dataset. Details on Load Iris Dataset. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html#sklearn.datasets.load_iris

[29] Pandas, Read CSV, Read_CSV Documentation. Date of Retrieval 10.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

[30] Khadka, 2019. Preparation of Pre-processing Data. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Codes/Preprocessing/DataCreationForPreprocess.ipynb

[31] Khadka, 2019. Raw Data To Pre-process. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Codes/Preprocessing/Raw%20Data%20To%20Preprocess

[32] Khadka, 2019. Full Code For Pre-processing Data Section. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Codes/Preprocessing/Preprocessing.ipynb

[32] Khadka, 2019. Figure 8. Importing Preprocessing Data.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Importing%20Preprocessing%20Data.png

[32] Khadka, 2019. Figure 9. Preprocessing Data Info Method.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Preprocessing%20Data%20Info%20Method.png

[34] Khadka, 2019. Figure 10. Total Null Count Pandas.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Total%20Null%20Count%20Pandas.png

[35] Khadka, 2019. Figure 11. Total Null Display in Seaborn.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Total%20Null%20Display%20in%20Seaborn.png

[36] Khadka, 2019. Figure 12. Drop Gender Column.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Drop%20Gender%20Column.png

[37] Khadka, 2019. Figure 13. Setting Up for Imputing.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Setting%20Up%20For%20Imputing.png

[38] NumPy, Reshape. Reshape Documentation. Date of Retrieval 11.05.2019,

https://docs.scipy.org/doc/numpy/reference/generated/numpy.reshape.html

[39] Khadka, 2019. Figure 14. Imputing Column.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Imputing%20Column.png

[40] Pandas, FillNa. Pandas.fillna Documentation. Date of Retrieval 10.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html

[41] Khadka, 2019. Figure 15. Imputing Using Pandas.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Imputing%20Using%20Pandas.png

[42] Khadka, 2019. Figure 16. Drop the Row.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Drop%20the%20row.png

[43] Pandas, DropNa. Pandas.DropNa Documentation. Date of Retrieval 11.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html

[44] Pandas, Value Counts. Pandas.Series.value_counts Documentation. Date of Retrieval 11.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.value_counts.html

[45] Khadka, 2019. Figure 17. Use Value Count for Mode.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Use%20Value%20Count%20for%20Mode.png

[46] Sklearn, Label Encoder. Sklearn.Preprocessing.LabelEncoder Documentation. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

[47] Sklearn, One Hot Encoder. Sklearn.Preprocessing.OneHotEncoder Documentation. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

[48] Khadka, 2019. Figure 18. Label Encoding.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Label%20Encoding.png

[49] Khadka, 2019. Figure 19. OneHotEncoding.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/OneHotEncoding.png

[50] Scipy, ToArray. Scipy.Sparse.csr_matrix.toarray Documentation. Date of Retrieval 10.05.2019,

https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.toarray.html

[51] Pandas, Get Dummies. Pandas.get_dummies Documentation. Date of Retrieval 11.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html

[52] Khadka, 2019. Figure 20. Pandas Get Dummies.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Preprocessing/Pandas%20Get%20Dummies.png

[53] Khadka, 2019. Figure 21. ImportingLoadBostonDataset.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/ImportingLoadBostonDataset.png

[54] Sklearn, Load_Boston User Guide. Boston Housing Price Dataset User Guide. Date of Retrieval 10.05.2019,

https://scikit-learn.org/stable/datasets/index.html#boston-dataset

[55] Khadka, 2019. Figure 22. Creating Dataframe.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/Creating%20Dataframe.png

[56] Khadka, 2019. Figure 23. Describe and Info.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/Describe%20And%20Info.png

[57] Pandas, Describe. Pandas.Describe Documentation. Date of Retrieval 11.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html

[58] Pandas, Info. Pandas.Info Documentation. Date of Retrieval 11.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.info.html

[59] Khadka, 2019. Figure 24. HeatMapOfFeaturesCorrelation.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/HeatmapOfFeaturesCorrelation.png

[60] Khadka, 2019. Figure 25. AgeCountPlot.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/AgeCountPlot.png

[61] Khadka, 2019. Figure 26. AgeCountValue.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/AgeCountValue.png

[62] Pedro Marcelino, Kaggle. Python Data Exploration Tutorials. Date of Retrieval 15.05.2019,

https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python

[63] Sklearn, Train_Test_Split. Sklearn.Model_Selction.Train_Test_Split. Date of Retrieval 15.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[64] Khadka, 2019. Figure 27. DataSplit.png. Date of Retrieval 14.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/DataSplit.png

[65] Pandas, Iloc. Pandas.Iloc Documentation. Date of Retrieval 15.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.iloc.html

[66] Pandas, Loc. Pandas.loc Documentation. Date of Retrieval 15.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.loc.html

[67] Pandas, XS. Pandas.XS Documentation. Date of Retrieval 15.05.2019,

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.xs.html

[68] Sklearn, Standard-Scaler. Sklearn.Preprocessing.StandardScaler Documentation. Date of Retrieval 15.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[69] Khadka, 2019. Figure 28. ScalingData.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/ScalingData.png

[70] Sklearn, Linear Regression. Sklearn.linear_model.LinearRegression Documentation. Date of Retrieval 15.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[71] Khadka, 2019. Figure 29. ImportAndTrainLinearModel.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/ImportAndTrainLinearModel.png

[72] Khadka, 2019. Figure 30. FitOrderMatters.png. Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/FitOrderMatters.png

[73] Khadka, 2019. Figure 31. LinearModelCoefficient.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/LinearModelCoeffecient.png

[74] Khadka, 2019. Figure 32. Predictions.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/Predictions.png

[75] Khadka, 2019. Figure 32. Errors.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/Errors.png

[76] Khadka, 2019. Figure 33. Evaluation.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/Evaluation.png

[77] Sklearn, Mean Squared Error. Sklearn.Metrics.Mean_Squared_Error Documentation. Date of Retrieval 15.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

[78] NumPy, Square Root. Numpy.Sqrt Documentation. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/Errors.png

[79] Sklearn, Explained Variance Score. Sklearn.Metrics.Explained_Variance_Score Documentation. Date of Retrieval 15.05.2019,

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.explained_variance_score.html

[80] Khadka, 2019. Figure 34. Optimization.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/Optimizaton.png

[80] Khadka, 2019. Figure 35. PandasDropInplaceTruth.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/Regression/PandasDropInplaceTruth.png

[82] Khadka, 2019. Linear Regression In Python Full Code. Simple_Linear_Regression.ipynb . Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Codes/Simple%20Linear%20Regression/Simple_linear_regression.ipynb

[83] Khadka, 2019. K-Nearest Neighbor In Python Full Code. Classification.ipynb. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Codes/Classification/Classification.ipynb

[84] Seaborn, LmPlot. Seaborn.LmPlot Documentation. Date of Retrieval 15.05.2019,

https://seaborn.pydata.org/generated/seaborn.lmplot.html

[85] Khadka, 2019. Figure 36. Petal Length Vs. Petal Width.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/KNN/Petal%20Lenght%20Vs%20Petal%20Width.png

[86] Khadka, 2019. Figure 37. Train KNN Model.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/KNN/Train%20KNN%20Model.png

[87] Khadka, 2019. Figure 38. Predictions and Errors.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/KNN/Predictions%20and%20Errors.png

[88] Khadka, 2019. Figure 39. Error Calculation For A Range of K.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/KNN/Error%20Claclua-tion%20For%20A%20Range%20of%20K.png

[89] NumPy, Mean. NumPy.Mean Documentation. Date of Retrieval 15.05.2019,

https://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html

[90] Khadka, 2019. Figure 40. Elbow Method Plot for Iris Dataset.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/KNN/Elbow%20Method%20Plot%20For%20Iris%20Dataset.png

[91] Khadka, 2019. Figure 41. GridSearchCV.png. Date of Retrieval 15.05.2019,

https://github.com/nibukdk/Thesis/blob/master/Images/KNN/GridSearchCV.png

[92] Thesis Repository at GitHub, Date of Retrieval 10.05.2019,

https://github.com/nibukdk/Thesis

[93] Andrew Ng. Machine Learning – Andrew Ng, Standford University [Full Course]. Date of Retrieval 20.05.2019,

https://www.youtube.com/playlist?list=PLLssT5z_DsK-h9vYZkQkYNWcItqhl-RJLN