MACHINE LEARNING

(Human resource Employee Attrition - Classifier)

Summer Internship Report Submitted in partial fulfillment

of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

Kommera Shresta

221710313024

Under the Guidance of

Mr.

Assistant Professor

GITAM (DEEMED TO BE UNIVERSITY)

VISAKHAPATNAM HYDERABAD BENGALURU

Department Of Computer Science Engineering GITAM
School of Technology
GITAM (Deemed to be University)
Hyderabad-502329
July 2020

DECLARATION

I submit this industrial training work entitled **“**Human resource Employee Attrition - Classifier” to GITAM (Deemed to Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of “**Bachelor of Technology**” in “**Computer Science Engineering**”. I declare that it was carried out independently by me under the guidance of **Mr.,** Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD	Kommera Shresta
Date:	221710313024

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled "Human resource Employee Attrition - Classifier" is being submitted by Kommera Shresta (221710313024) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer Science Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2019-20

It is faithful record work carried out by her at the **Computer Science Engineering Department**, GITAM University Hyderabad Campus under my guidance and supervision.

Mr.

Assistant Professor	Professor and HOD
Department of CSE	Department of CSE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. N. Siva Prasad,** Pro Vice Chancellor, GITAM Hyderabad and **Dr. CH. Sanjay,** Principal, GITAM Hyderabad

I would like to thank respected **Dr.Phani Kumar** Head of the Department of Computer Science Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Mr.** who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Kommera Shresta

221710313024

ABSTRACT

Machine learning algorithms are used to predict the values from the data set by splitting the data set in to train and test and building Machine learning algorithms models of higher accuracy to predict the values is the primary task to be performed on Human resource Employee Attrition.

To get a better understanding and work on a strategical approach for attrition of employee, I have adapted the view point of looking at left and for further deep understanding of the problem, I have taken satisfaction_level, last_evaluation, number_projects, average_monthly_hours , time_spent_company, work_accident, promotion_last_5years, Departments, Salary and my primary objective of this case study was to look up the factor left in Dataset.

Different classifiers are used for checking attrition of employee and the results of those classifiers are compared for identifying the best detection model.

Table of Contents:

LIST of Figures

CHAPTER 1

MACHINE LEARNING

1.1 INTRODUCTION:

Machine Learning (ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence (AI).

1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and “more items to consider” and “get yourself a little something” on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today’s data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that’s in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

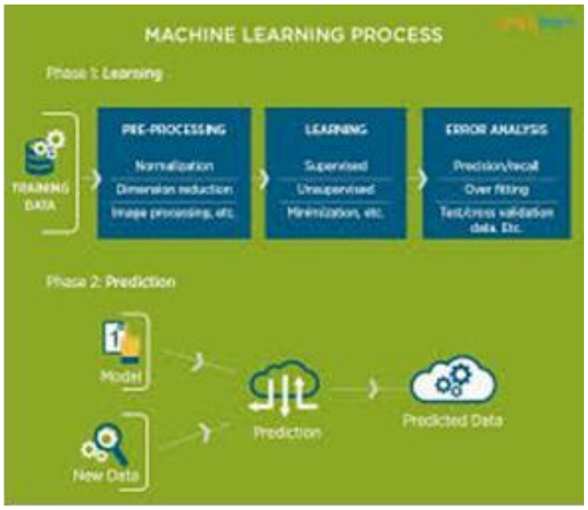


Figure 1.2: The Process Flow

1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

1.4.1 Supervised Learning:

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification

1.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

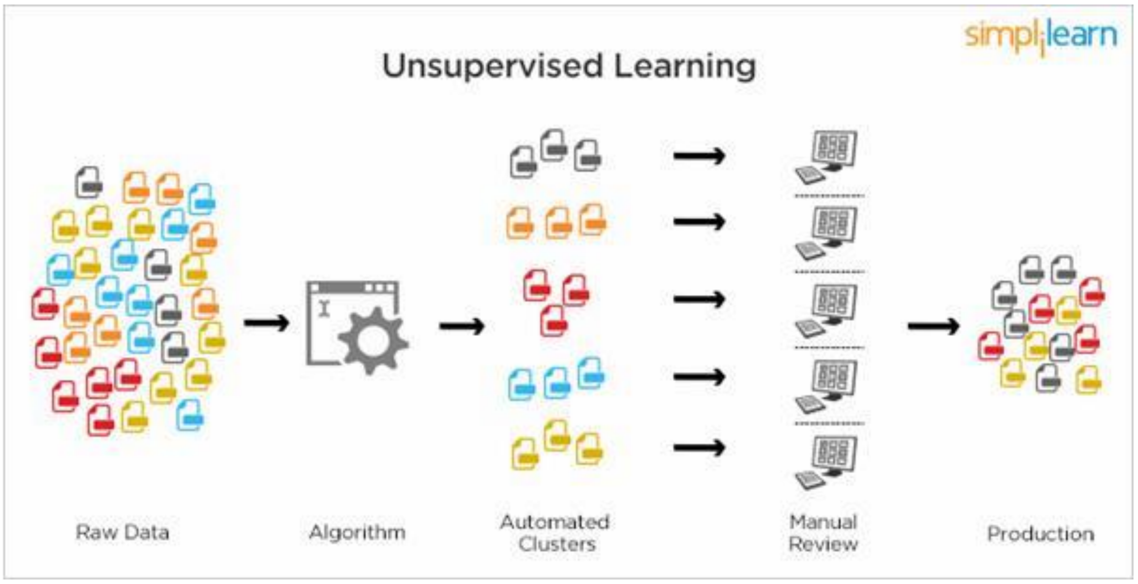


Figure 1.4.2: Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.

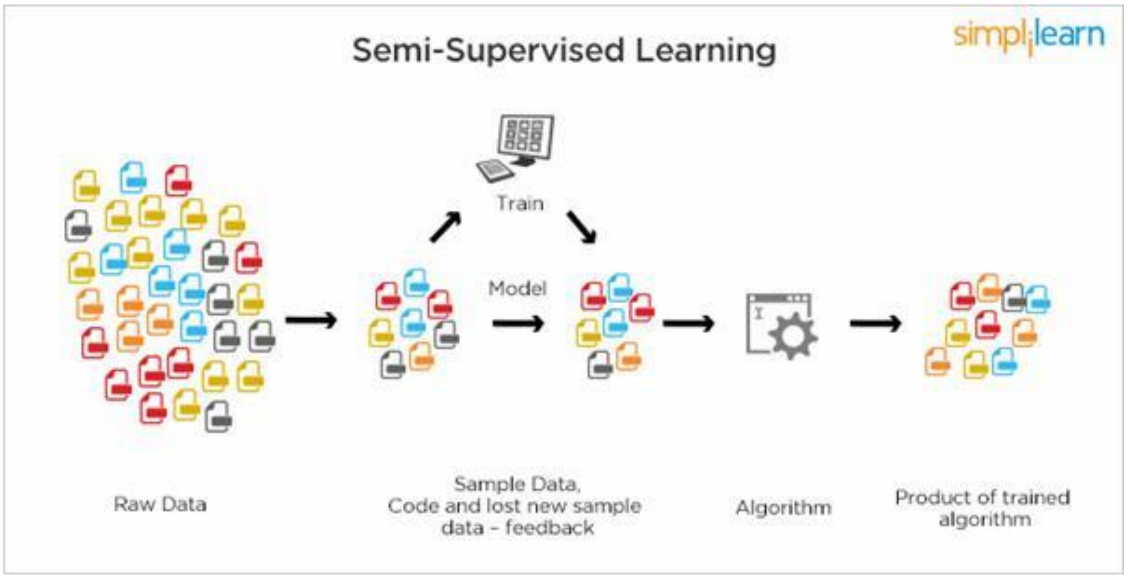

Semi-Supervised Learning
simplilearn
Train
Model
Raw Data
Sample Data, Code and lost new sample data - feedback
Algorithm
Product of trained algorithm

Figure 1.4.3: Semi Supervised Learning

1.5 RELATION BETWEEN DATA MINING, MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2

PYTHON

Basic programming language used for machine learning is: PYTHON

2.1 INTRODUCTION TO PYHTON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general-purpose programming language that is often applied in scripting roles
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

2.2 HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7, it is generally called as python3

2.3 FEATURES OF PYTHON:

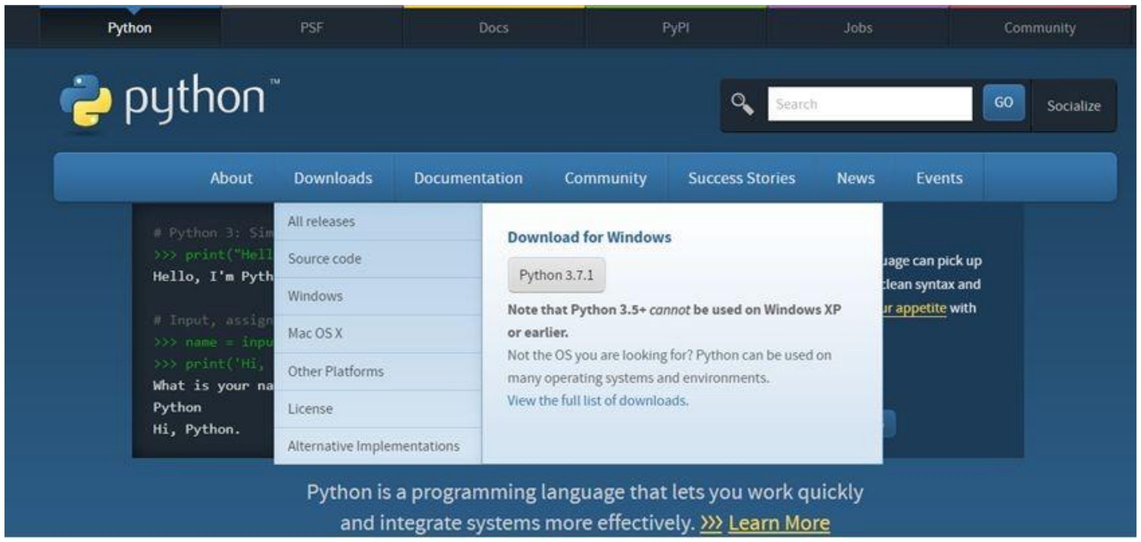
- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

2.4.1 Installation (using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.



```
# Python 3: Sim
>>> print("Hell
Hello, I'm Pyth

# Input, assign
>>> name = inpu
>>> print('Hi,
What is your na
Python
Hi, Python.
```

Download for Windows

Note that Python 3.5+ *cannot* be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

age can pick up
lean syntax and
r appetite with

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Figure 2.4.1 : Python download

2.4.2 Installation (using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.
- In WINDOWS:
- In windows
- Step 1: Open Anaconda.com/downloads in web browser.
- Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
- Step 3: select installation type (all users)
- Step 4: Select path (i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
- Step 5: Open jupyter notebook (it opens in default browser)

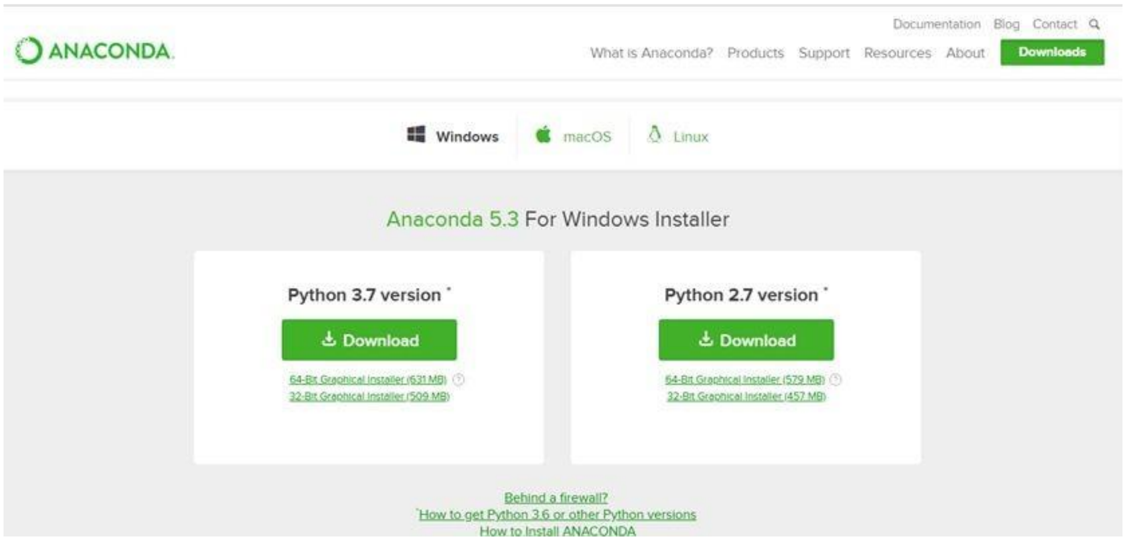

ANACONDA.
Documentation Blog Contact
What is Anaconda? Products Support Resources About Downloads
Windows macOS Linux
Anaconda 5.3 For Windows Installer
Python 3.7 version
Download
64-Bit Graphical Installer (631 MB)
32-Bit Graphical Installer (509 MB)
Python 2.7 version
Download
64-Bit Graphical Installer (579 MB)
32-Bit Graphical Installer (457 MB)
Behind a firewall?
How to get Python 3.6 or other Python versions
How to Install ANACONDA

Figure 2.4.2.1: Anaconda download

localhost:8888/tree
jupyter
Quit Logout
Files Running Clusters
Select items to perform actions on them. View Edit
Upload New
Name Last Modified File size

Figure 2.4.2.2: Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
    - Numbers
    - Strings
    - Lists

- Tuples
- Dictionary

2.5.1 Python Numbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

2.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

2.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets

([]).

- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

2.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

2.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

2.6 PYTHON FUNCTION:

2.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses

The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

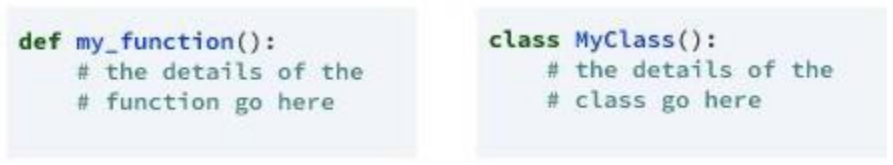
2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

2.7 PYTHON USING OOP's CONCEPTS:

2.7.1 Class:

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.
- Defining a Class:
    - We define a class in a very similar way how we define a function.
    - Just like a function ,we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is indented like a functions body is.



```
def my_function():
    # the details of the
    # function go here
```

```
class MyClass():
    # the details of the
    # class go here
```

Fig 2.7.1: Defining a Class

2.7.2 __init__method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores: init ().

CHAPTER 3

CASE STUDY

3.1 PROBLEM STATEMENT:

Analyze employee attrition. Find out why employees are leaving the company, and learn to predict who will leave the company.

3.2 DATA SET:

The given data set consists of the following parameters:

1. satisfaction_level
2. last_evaluation
3. number_projects
4. average_monthly_hours
5. time_spent_company
6. work_accident
7. promotion_last_5years
8. Departments
9. Salary
10. left

3.3 OBJECTIVE OF THE CASE STUDY:

First, we will visualize the insights from the Human resource attrition and after successful training. Finally, we will evaluate the performance of our classifier using several evaluation metrics. Here, you can predict who, and when an employee will terminate the service. Employee attrition is expensive, and incremental improvements will give significant results. It will help us in designing better retention plans and improving employee satisfaction.

CHAPTER 4

MODEL BUILDING

4.1 PREPROCESSING OF THE DATA:

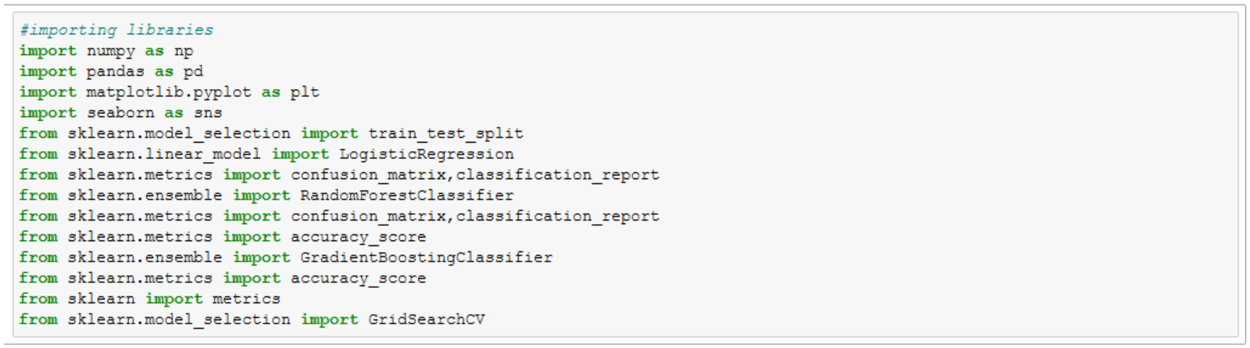
- Preprocessing of the data actually involves the following steps:

4.1.1 GETTING THE DATASET:

- We can get data from dataset or can get from client

4.1.2 IMPORTING THE LIBRARIES:

- We have to import the libraries as per the requirement of the algorithm.

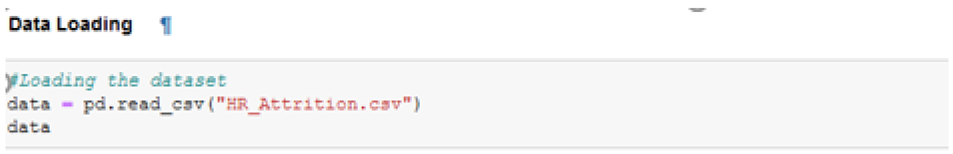


```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import accuracy_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
```

Fig 4.1.2: Importing Libraries

4.1.3 IMPORTING THE DATA-SET:

- Pandas in python provide an interesting method read_csv(). The read_csv function reads the entire dataset from a comma separated values file and we can assign it to a DataFrame to which all the operations can be performed. It helps us to access each and every row as well as columns and each and every value can be access using the dataframe. Any missing value or NaN value have to be cleaned.
- We are loading the data by using read_csv.



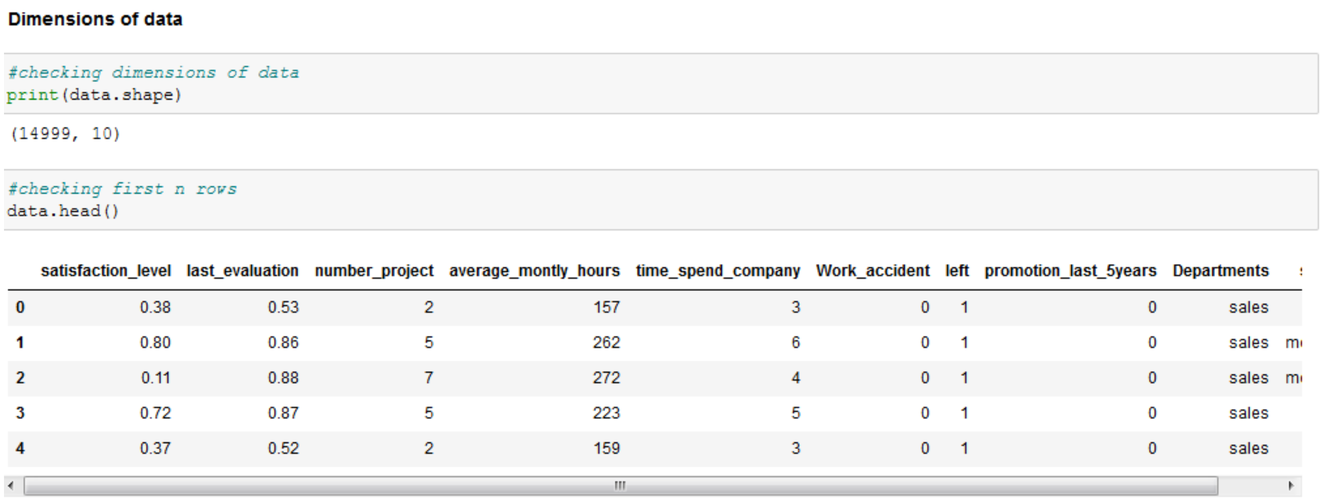
Data Loading ¶

```
#Loading the dataset
data = pd.read_csv("HR_Attrition.csv")
data
```

Fig 4.1.3: Reading the dataset

4.1.4 Shape

- The shape property returns a tuple representing the dimensionality of the Data Frame. The format of shape would be (rows, columns).
- We are looking at shape of dataset and top four columns.



Dimensions of data

```
#checking dimensions of data
print(data.shape)
```

```
(14999, 10)
```

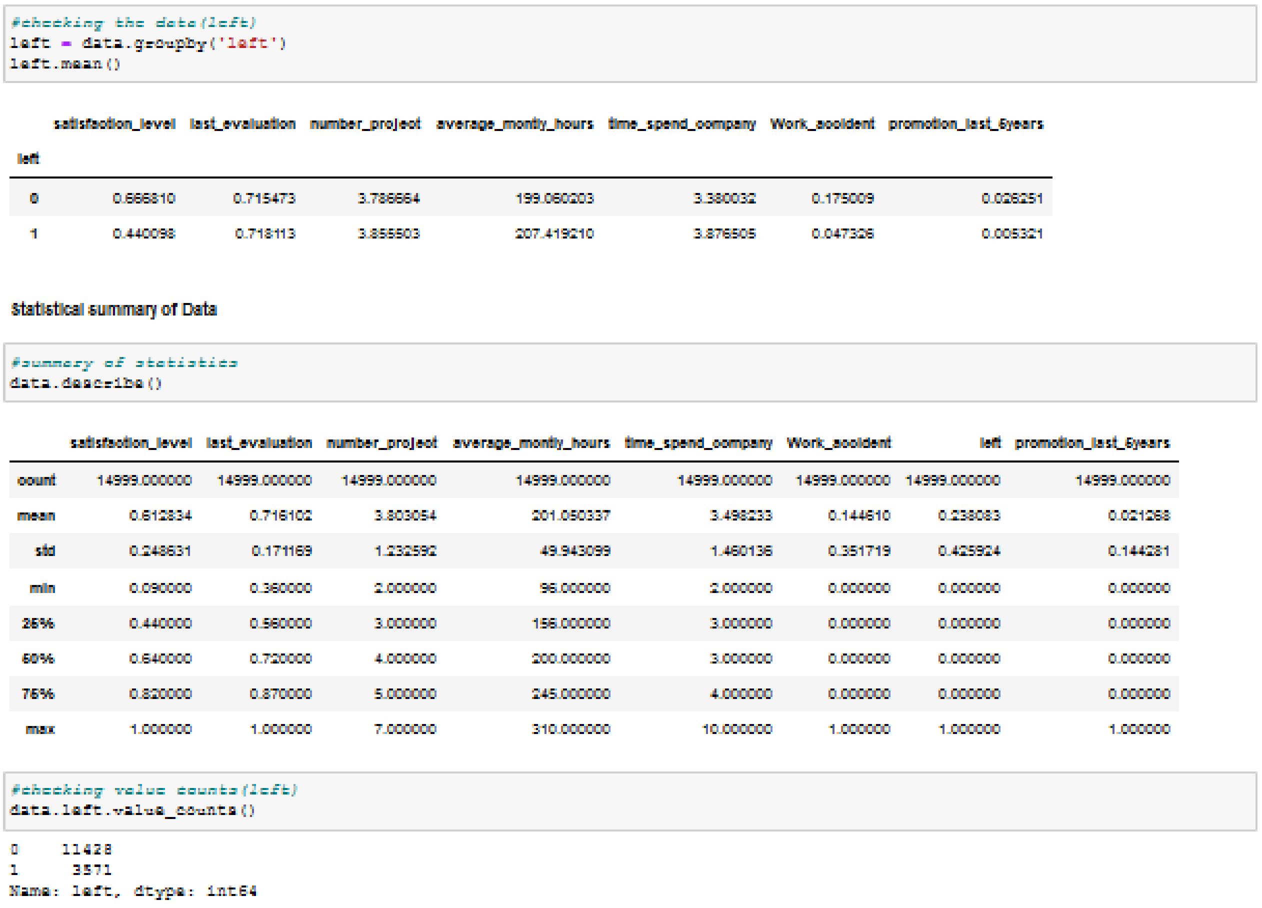
```
#checking first n rows
data.head()
```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Departments
0	0.38	0.53	2	157	3	0	1	0	sales
1	0.80	0.86	5	262	6	0	1	0	sales
2	0.11	0.88	7	272	4	0	1	0	sales
3	0.72	0.87	5	223	5	0	1	0	sales
4	0.37	0.52	2	159	3	0	1	0	sales

Fig 4.1.4: shape of data

4.1.5 Statistical Analysis:

- Total Number or "N", Mean, Median, Mode and Standard Deviation are used to describe your data.
- The Total Number or "N" is the number of observations made.
- Mean: This is the average of the data. Adding the values of all of the observations and dividing the total by the total number of observations or "N".
- Median: This is the middle value of the observations.
- Mode: This is the most frequent observation**.**



```
#checking the data(left)
left = data.groupby('left')
left.mean()
```

left	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	promotion_last_5years
0	0.666810	0.715473	3.786664	199.060203	3.380032	0.175009	0.026251
1	0.440098	0.718113	3.855503	207.419210	3.876505	0.047326	0.005321

Statistical summary of Data

```
#summary of statistics
data.describe()
```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000
mean	0.612834	0.716102	3.803054	201.050337	3.498233	0.144610	0.238083	0.021268
std	0.248631	0.171169	1.232592	49.943099	1.460136	0.351719	0.425924	0.144281
min	0.090000	0.360000	2.000000	96.000000	2.000000	0.000000	0.000000	0.000000
25%	0.440000	0.560000	3.000000	156.000000	3.000000	0.000000	0.000000	0.000000
50%	0.640000	0.720000	4.000000	200.000000	3.000000	0.000000	0.000000	0.000000
75%	0.820000	0.870000	5.000000	245.000000	4.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	7.000000	310.000000	10.000000	1.000000	1.000000	1.000000

```
#checking value counts(left)
data.left.value_counts()
```

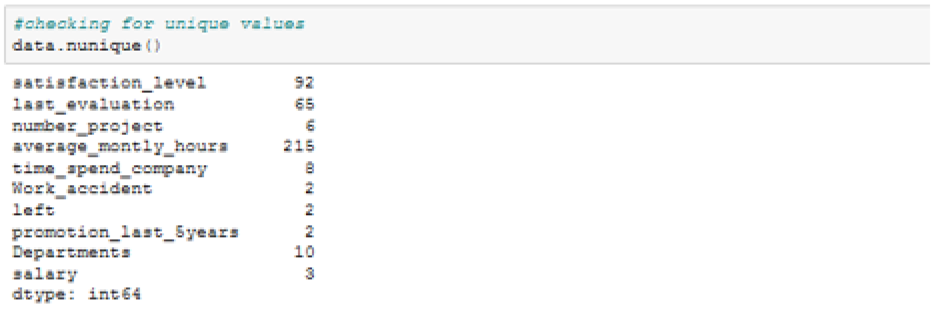
```
0    11428
1     3571
Name: left, dtype: int64
```

Fig 4.1.5: analysis of data

4.1.6 Unique:

- The unique() function is used to find the unique elements of an array. Returns the sorted unique elements of an array.

- We are checking for any unique values.



```
#checking for unique values
data.nunique()
```

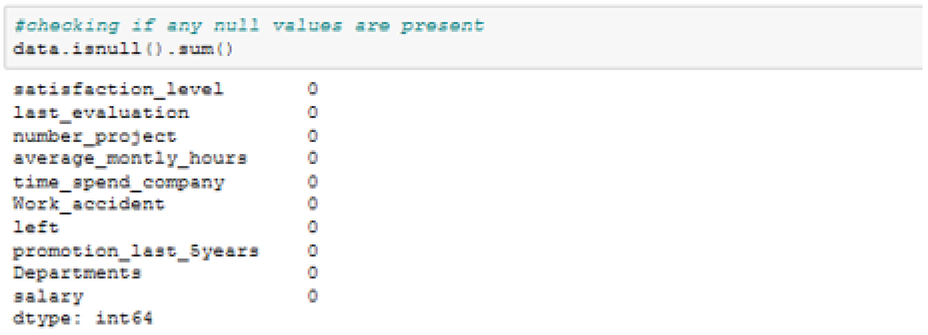
```
satisfaction_level        92
last_evaluation           65
number_project             6
average_montly_hours     215
time_spend_company         8
Work_accident              2
left                       2
promotion_last_5years      2
Departments               10
salary                     3
dtype: int64
```

Fig 4.1.6: checking unique values

4.1.7 Isnull:

- isnull() function detect missing values in the given series object. It returns a boolean same-sized object indicating if the values are NA

- We are checking for any null values are present.If any null are found we use fillna() and imputation to fill columns with a data.



```
#checking if any null values are present
data.isnull().sum()
```

```
satisfaction_level       0
last_evaluation          0
number_project           0
average_montly_hours     0
time_spend_company       0
Work_accident            0
left                     0
promotion_last_5years    0
Departments              0
salary                   0
dtype: int64
```

Fig 4.1.7: checking null values

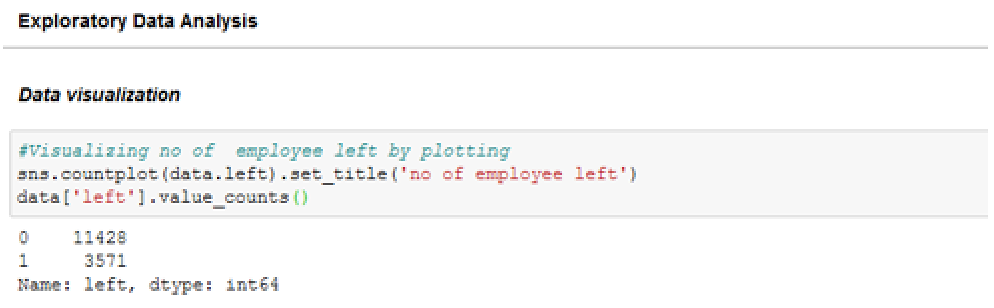
5.Generating Plots

5.1 -Visualize the data between Target and the Features

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for **barplot()**, so you can compare counts across nested variables.

Input data can be passed in a variety of formats, including:

- Vectors of data represented as lists, numpy arrays, or pandas Series objects passed directly to the x, y, and/or hue parameters.
- A “long-form” DataFrame, in which case the x, y, and hue variables will determine how the data are plotted.
- A “wide-form” DataFrame, such that each numeric column will be plotted.
- An array or list of vectors.



Exploratory Data Analysis

Data visualization

```
#Visualizing no of  employee left by plotting
sns.countplot(data.left).set_title('no of employee left')
data['left'].value_counts()
```

```
0    11428
1     3571
Name: left, dtype: int64
```

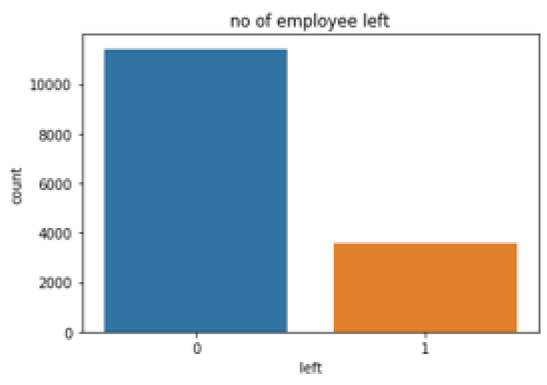


Fig 5.1: Data visualize

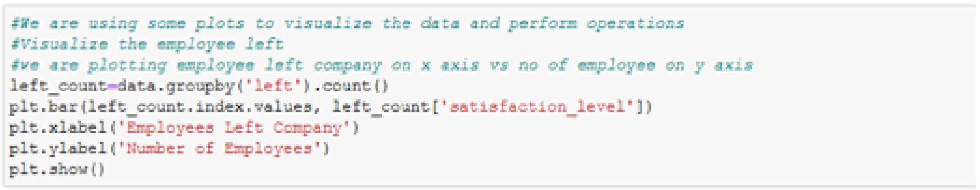
- we will check the total number of employee attrition. And we observe 3571 employees left the company.

5.2 Imbalance data

Imbalance data distribution is an important part of machine learning workflow. An imbalanced dataset means instances of one of the two classes is higher than the other, in another way, the number of observations is not the same for all the classes in a classification dataset.

5.3 - Visualize the data between all the Features

- A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.
- A bar graph shows comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value.
- Matplotlib API provides the **bar()** function that can be used in the MATLAB style use as well as object oriented API.
- We are using some plots to visualize the data and perform operations.



```
#We are using some plots to visualize the data and perform operations
#Visualize the employee left
#we are plotting employee left company on x axis vs no of employee on y axis
left_count=data.groupby('left').count()
plt.bar(left_count.index.values, left_count['satisfaction_level'])
plt.xlabel('Employees Left Company')
plt.ylabel('Number of Employees')
plt.show()
```

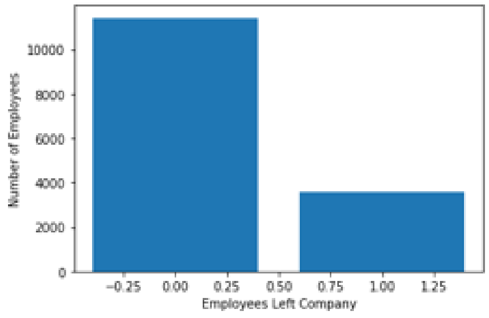


Fig 5.3.1 Employee left vs No of Employee

In this plot, we will visualize the number of employees left company.And we can observe no of employee left is 23 % of the total employment.

```
#Visualize the No of projects
#we are plotting number of projects on x axis vs no of employee on y axis
num_projects=data.groupby('number_project').count()
plt.bar(num_projects.index.values, num_projects['satisfaction_level'])
plt.xlabel('Number of Projects')
plt.ylabel('Number of Employees')
plt.show()
```

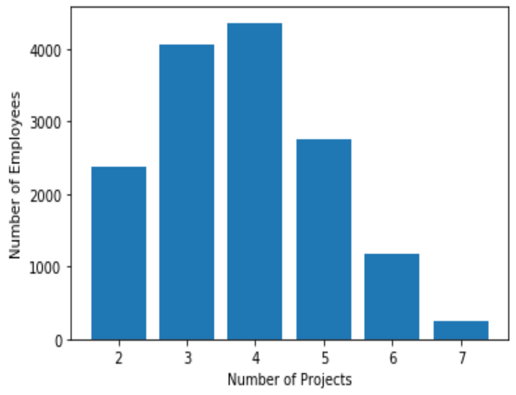
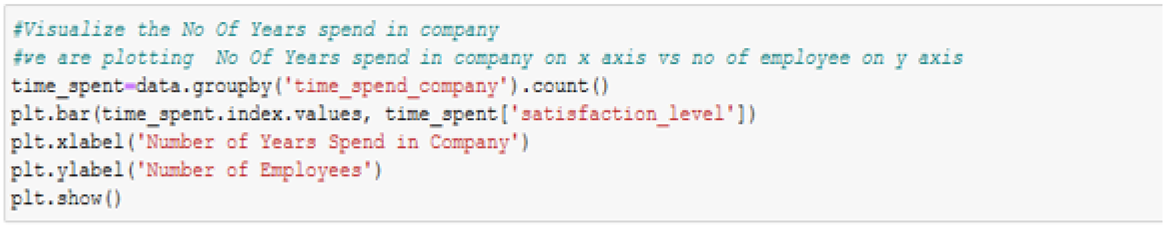


Fig 5.3.2: Number of Project vs No of Employee

In this plot, we will visualize the number of projects done. And we can observe that most of the employee is doing the project from 3-5.



```
#Visualize the No Of Years spend in company
#we are plotting  No Of Years spend in company on x axis vs no of employee on y axis
time_spent=data.groupby('time_spend_company').count()
plt.bar(time_spent.index.values, time_spent['satisfaction_level'])
plt.xlabel('Number of Years Spend in Company')
plt.ylabel('Number of Employees')
plt.show()
```

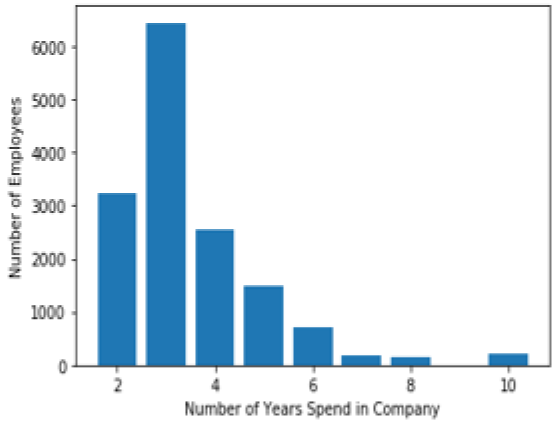


Fig 5.3.3 No of years spent vs No of Employees

In this plot, we will visualize the number of years spend in company. And we can observe that Most of the employee experience between 2-4 years. Also, there is a massive gap between 3 years and 4 years experienced employee.

```
#Visualizing departments by plotting
sns.countplot(data.Departments).set_title('Departments')
data['Departments'].value_counts()
plt.xticks(rotation=90)
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), <a list of 10 Text xticklabel objects>)
```

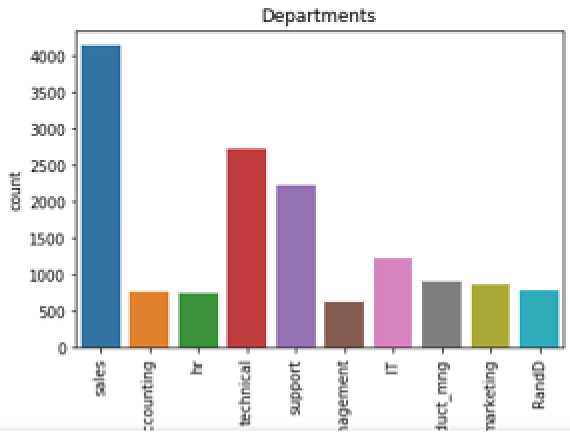


Fig 5.3.4 : Departments vs Count

In this plot, we will visualize the sales department is having maximum number of employee followed by technical and support.

```
#Visualizing no of employee salary by plotting
sns.countplot(data.salary).set_title('no of employee')
data['salary'].value_counts()
```

```
high      1237
Name: salary, dtype: int64
```

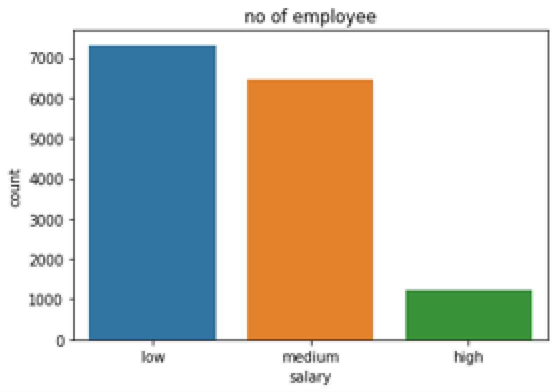
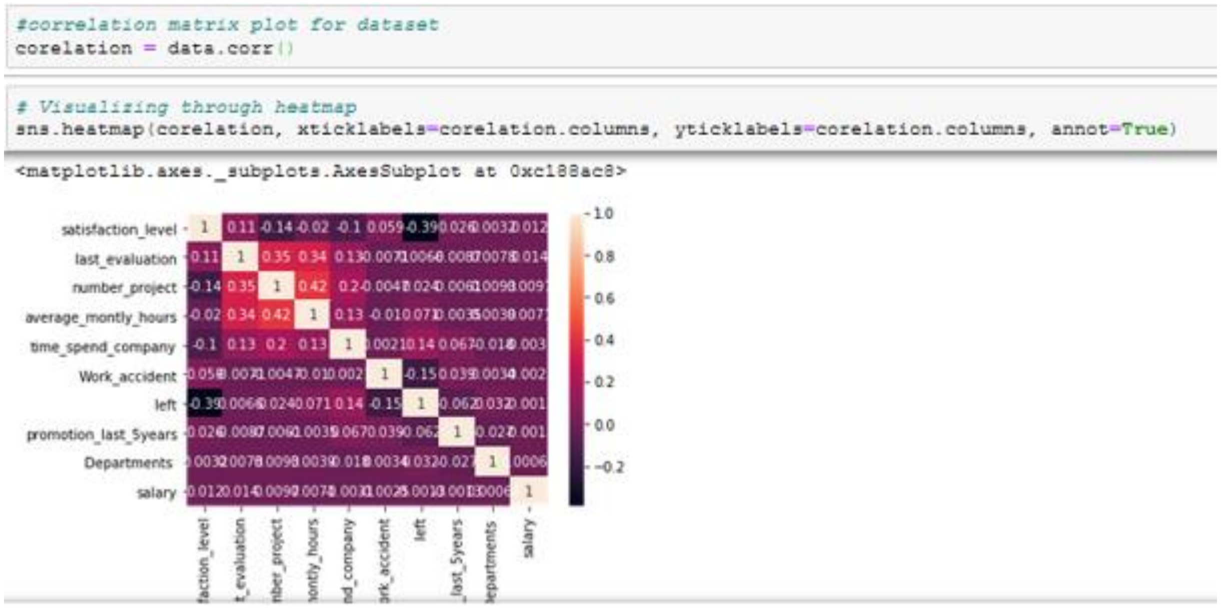


Fig 5.3.5: Salary vs Count

In this plot, we will visualize the number of employee salary.And we can observe that,the employees are getting salary either medium or low.

5.4 Correlation

- A heat map (or heatmap) is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions. The variation in color may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space.



```
#correlation matrix plot for dataset
corelation = data.corr()
```

```
# Visualizing through heatmap
sns.heatmap(corelation, xticklabels=corelation.columns, yticklabels=corelation.columns, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xc188ac8>
```

Fig 5.4: Heat Map

From above Map we can observe,

- Coefficient value = 1 − It represents full positive correlation between variables.
- Coefficient value = -1 − It represents full negative correlation between variables.
- Coefficient value = 0 − It represents no correlation at all between variables.

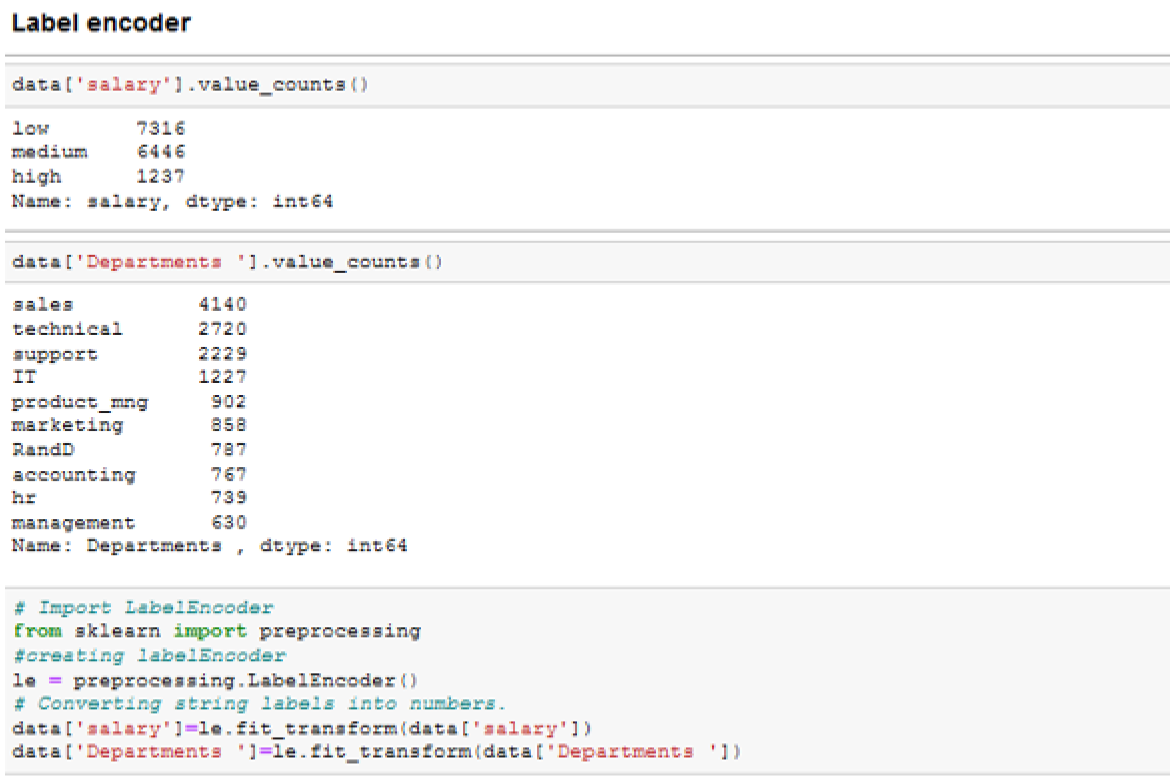
5.5 Categorical Encoding

Typically, any structured dataset includes multiple columns – a combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text. That's essentially the case with Machine Learning algorithms too . That's primarily the reason we need to convert categorical columns to numerical columns so that a machine learning algorithm understands it. This process is called categorical encoding**.**

5.5.1 Label Encoding

Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

- We are applying Label Encoder for salary and department columns.



Label encoder

```
data['salary'].value_counts()
```

```
low       7316
medium    6446
high      1237
Name: salary, dtype: int64
```

```
data['Departments '].value_counts()
```

```
sales          4140
technical      2720
support        2229
IT             1227
product_mng     902
marketing       858
RandD           787
accounting      767
hr              739
management      630
Name: Departments , dtype: int64
```

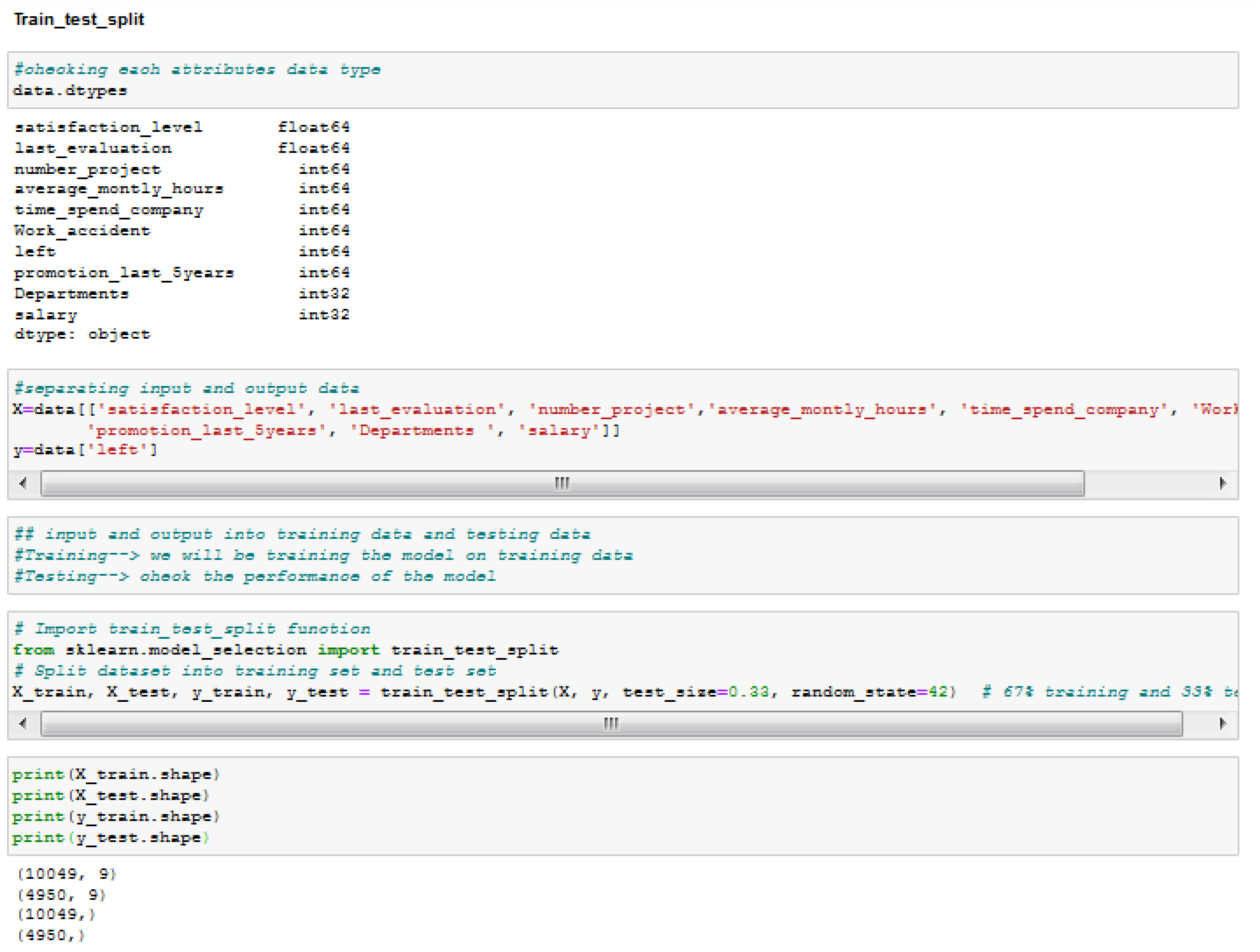
```
# Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
data['salary']=le.fit_transform(data['salary'])
data['Departments ']=le.fit_transform(data['Departments '])
```

Fig 5.5.1: Label Encoder

6.TRAINING THE MODEL:

- Splitting the data : after the preprocessing is done then the data is split into train and test sets
- In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set'. An important point to note is that during training the classifier only uses the training set . The test set must not be used during training the classifier. The test set will only be available during testing the classifier.
- training set - a subset to train a model.(Model learns patterns between Input and Output)
- test set - a subset to test the trained model.(To test whether the model has correctly learnt)
- The amount or percentage of Splitting can be taken as specified (i.e. train data = 75% , test data =25% or train data = 80% , test data= 20%)
- First we need to identify the input and output variables and we need to separate the input set and output set
- In scikit learn library we have a package called model_selection in which train_test_split method is available .we need to import this method
- This method splits the input and output data to train and test based on the percentage specified by the user and assigns them to four different variables(we need to mention the variables)

- We are Spliting arrays or matrices into random train and test subsets .



Train_test_split

```
#checking each attributes data type
data.dtypes
```

```
satisfaction_level       float64
last_evaluation          float64
number_project             int64
average_montly_hours       int64
time_spend_company         int64
Work_accident              int64
left                       int64
promotion_last_5years      int64
Departments                int32
salary                     int32
dtype: object
```

```
#separating input and output data
X=data[['satisfaction_level', 'last_evaluation', 'number_project','average_montly_hours', 'time_spend_company', 'Work
       'promotion_last_5years', 'Departments ', 'salary']]
y=data['left']
```

```
## input and output into training data and testing data
#Training--> we will be training the model on training data
#Testing--> check the performance of the model
```

```
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)  # 67% training and 33% te
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

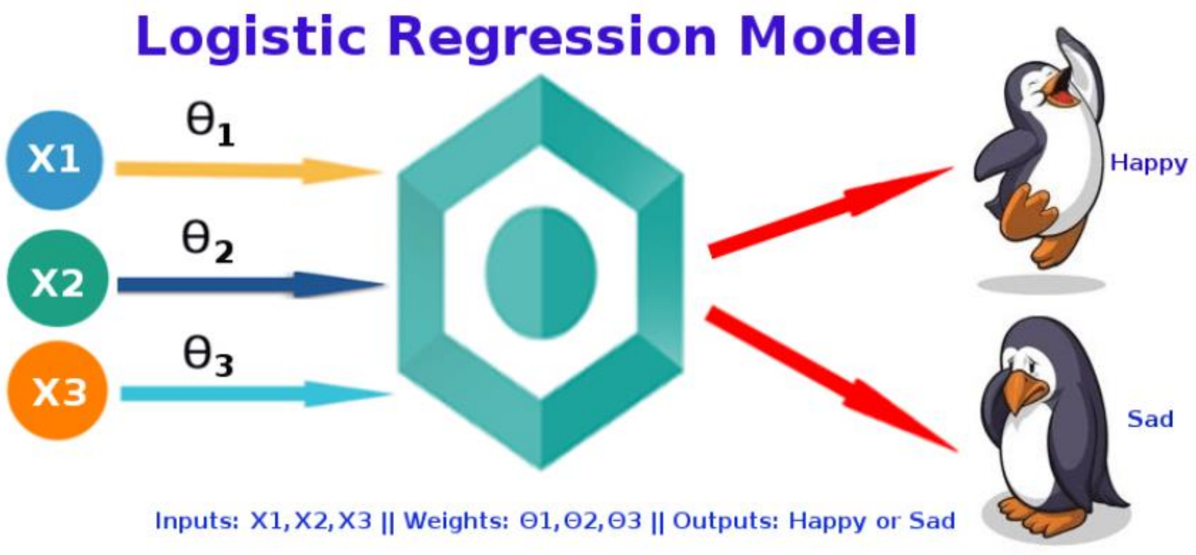
```
(10049, 9)
(4950, 9)
(10049,)
(4950,)
```

Fig 6 Train_Test_Split

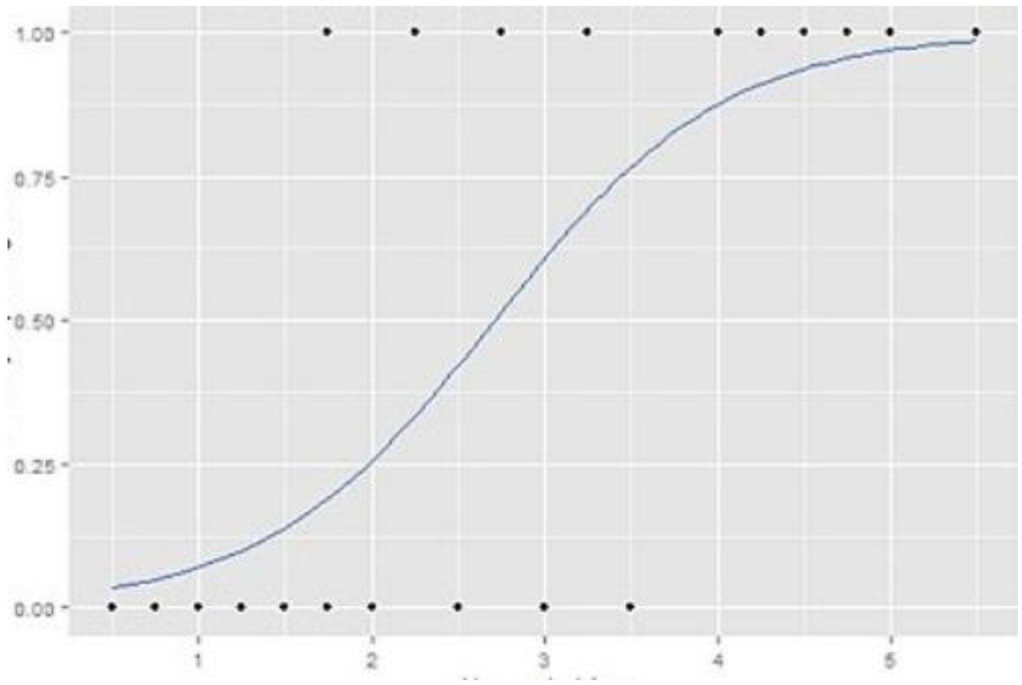
CHAPTER 7

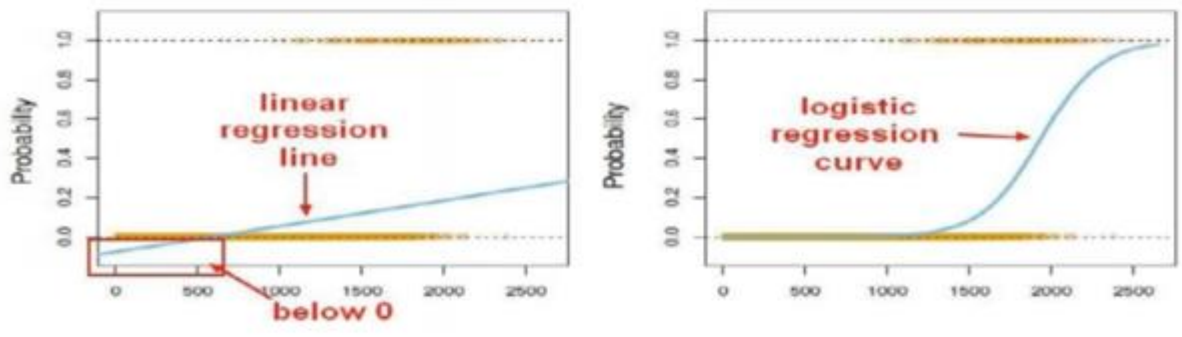
MODEL BUILDING AND EVALUATION:

7.1 Logistic Regression


Logistic Regression Model
X1
X2
X3
θ1
θ2
θ3
Happy
Sad
Inputs: X1,X2,X3 || Weights: Θ1,Θ2,Θ3 || Outputs: Happy or Sad

- Logistic Regression is used when the dependent variable(target) is categorical.
- Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis and it predicts the probability
- Example: Yes or No, get a disease or not, pass or fail, defective or non-defective, etc.,
- Also called a classification algorithm, because we are classifying the data. It predicts the probability associated with each dependent variable category.

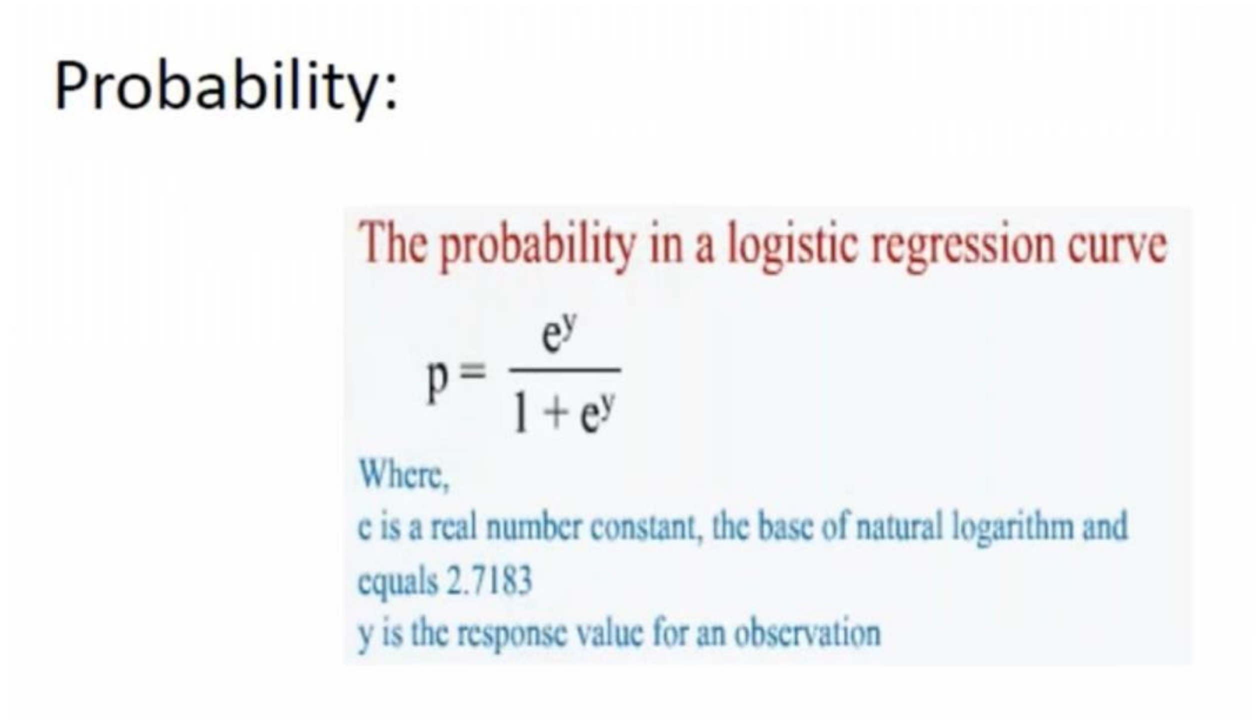




z=b0+b1(x1)+b2(x2)+b3(x3)

But, when we use the above equation to calculate probability, we would get values less than 0 as well as greater than 1. That doesn't make any sense. So, we need to use such an equation which always gives values between 0 and 1, as we desire while calculating the probability.

Out of the equation we are going to calculate the probabilities of the categories.



Probability:

The probability in a logistic regression curve

$$p = \frac{e^y}{1 + e^y}$$

Where,

e is a real number constant, the base of natural logarithm and equals 2.7183

y is the response value for an observation

The final step is to assign class labels (0 or 1) to our predicted probabilities.

If p is less than 0.5, we conclude the predicted output is 0 and if p is greater than 0.5, you conclude the output is 1.

Methods:

Binary or Binomial

In such a kind of classification, a dependent variable will have only two possible types either 1 and 0. For example, these variables may represent success or failure, yes or no, win or loss etc.

Multinomial

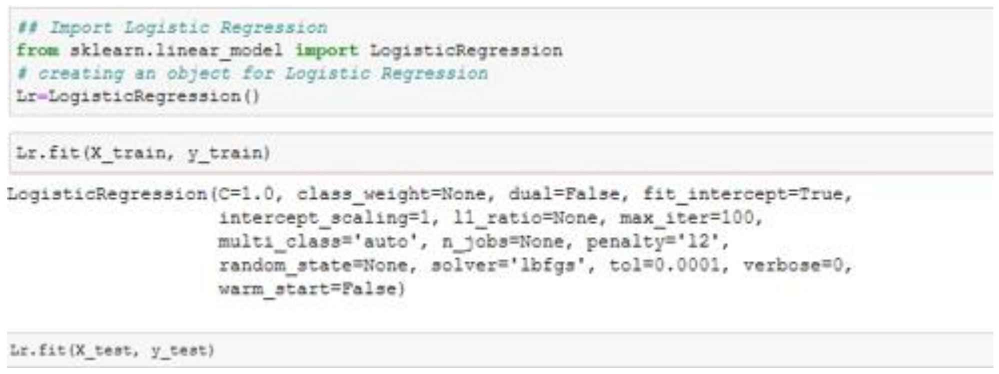
In such a kind of classification, dependent variable can have 3 or more possible unordered types or the types having no quantitative significance. For example, these variables may represent "Type A" or "Type B" or "Type C".

Ordinal

In such a kind of classification, dependent variable can have 3 or more possible *ordered* types or the types having a quantitative significance. For example, these variables may represent "poor" or "good", "very good", "Excellent" and each category can have the scores like 0,1,2,3.

7.1.1 Train and Test Models:

- Importing Logistic Regression from packages of linear_model.
- The training dataset is used to prepare a model, to train it.
- We pretend the test dataset is new data where the output values are withheld from the algorithm. We gather predictions from the trained model on the inputs from the test dataset and compare them to the withheld output values of the test set.



```
## Import Logistic Regression
from sklearn.linear_model import LogisticRegression
# creating an object for Logistic Regression
Lr=LogisticRegression()
```

```
Lr.fit(X_train, y_train)
```

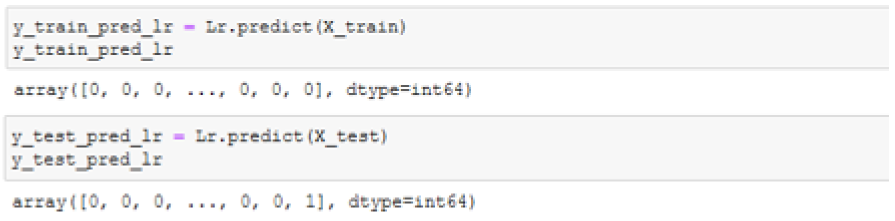
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
Lr.fit(X_test, y_test)
```

Fig 7.1.1: Train the model

7.1.2 Predict Analytics

Predictive analytics uses historical data to predict future events. Typically, historical data is used to build a mathematical model that captures important trends. That predictive model is then used on current data to predict what will happen next, or to suggest actions to take for optimal outcomes.



```
y_train_pred_lr = Lr.predict(X_train)
y_train_pred_lr
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

```
y_test_pred_lr = Lr.predict(X_test)
y_test_pred_lr
```

```
array([0, 0, 0, ..., 0, 0, 1], dtype=int64)
```

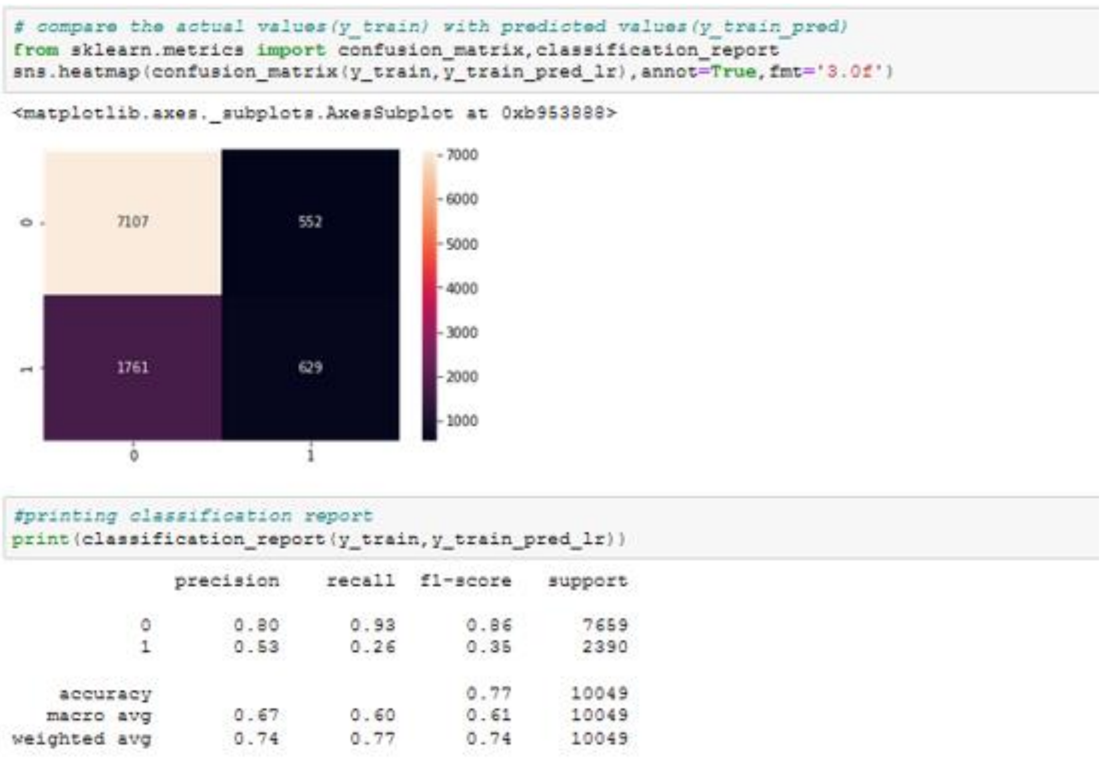
Fig 7.1.2: Predict Values

7.1.3 Classification report and Confusion Matrix:

A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report as shown below.

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

- In this we observe precision, recall, f1 score, accuracy, macro avg, weighted avg in
- Classification Report for both train and test.
- In Confusion Matrix we observe True Positive, True Negative, False Positive, False Negative for both train and test.



```
# compare the actual values(y_train) with predicted values(y_train_pred)
from sklearn.metrics import confusion_matrix,classification_report
sns.heatmap(confusion_matrix(y_train,y_train_pred_lr),annot=True,fmt='3.0f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xb953888>
```

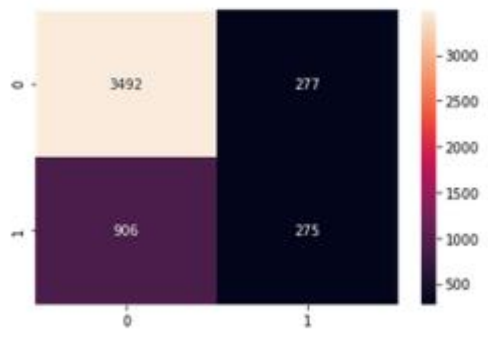
```
#printing classification report
print(classification_report(y_train,y_train_pred_lr))
```

```
              precision    recall  f1-score   support

           0       0.80      0.93      0.86      7659
           1       0.53      0.26      0.35      2390

    accuracy                           0.77     10049
   macro avg       0.67      0.60      0.61     10049
weighted avg       0.74      0.77      0.74     10049
```

Fig 7.1.3.1: Classification Report

```
# compare the actual values(y_test) with predicted values(y_test_pred)
from sklearn.metrics import confusion_matrix,classification_report
sns.heatmap(confusion_matrix(y_test,y_test_pred_lr),annot=True,fmt='3.0f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xc2f4648>
```



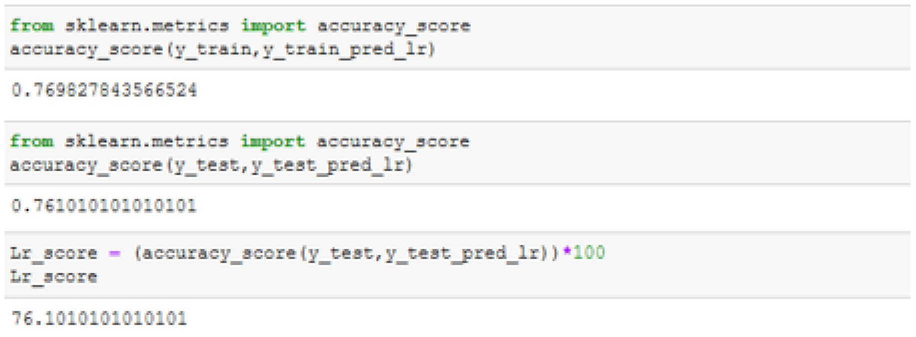
```
#printing classification report
print(classification_report(y_test,y_test_pred_lr))
```

```
              precision    recall  f1-score   support

           0       0.79      0.93      0.86      3769
           1       0.50      0.23      0.32      1181

    accuracy                           0.76      4950
   macro avg       0.65      0.58      0.59      4950
weighted avg       0.72      0.76      0.73      4950
```

Fig 7.1.3.2: Classification Report

7.1.4 Accuracy score

Accuracy is one metric for evaluating classification models. Informally, **accuracy** is the fraction of predictions our model got right.



```
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_train_pred_lr)
```

```
0.769827843566524
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_test_pred_lr)
```

```
0.761010101010101
```

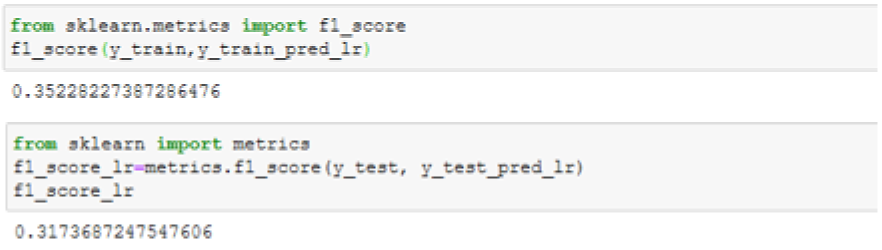
```
Lr_score = (accuracy_score(y_test,y_test_pred_lr))*100
Lr_score
```

```
76.1010101010101
```

Fig 7.1.4: Accuracy Score

7.1.5 F1 Score

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal.



```
from sklearn.metrics import f1_score
f1_score(y_train,y_train_pred_lr)
```

```
0.35228227387286476
```

```
from sklearn import metrics
f1_score_lr=metrics.f1_score(y_test, y_test_pred_lr)
f1_score_lr
```

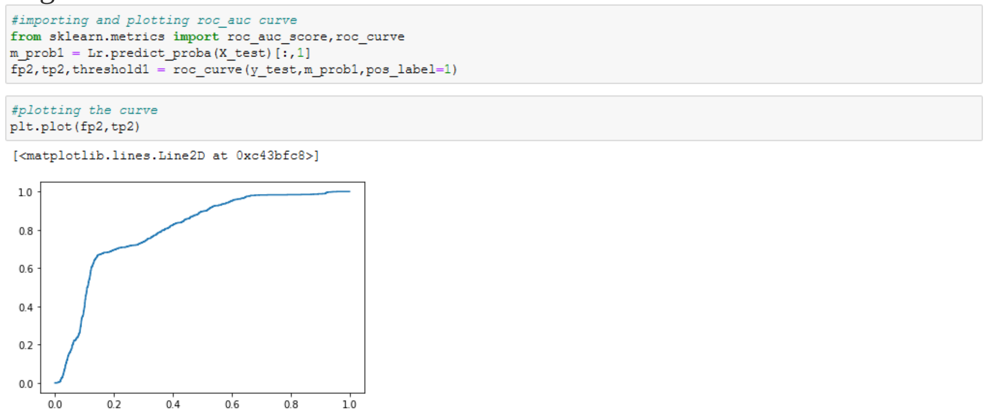
```
0.3173687247547606
```

Fig 7.1.5: F1 score

7.1.6 ROC-AUC Score

- An ROC curve (Receiver Operating Characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:
- True Positive Rate
- False Positive Rate

- The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.
- The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.



```
#importing and plotting roc_auc curve
from sklearn.metrics import roc_auc_score,roc_curve
m_prob1 = Lr.predict_proba(X_test)[:,1]
fp2,tp2,threshold1 = roc_curve(y_test,m_prob1,pos_label=1)
```

```
#plotting the curve
plt.plot(fp2,tp2)
```

```
[<matplotlib.lines.Line2D at 0xc43bfc8>]
```

Fig 7.1.6: ROC_AUC Curve

- In this we are plotting roc_auc curve for Logistic Regression. And we observe accuracy is 76%.

7.2 Random Forest:

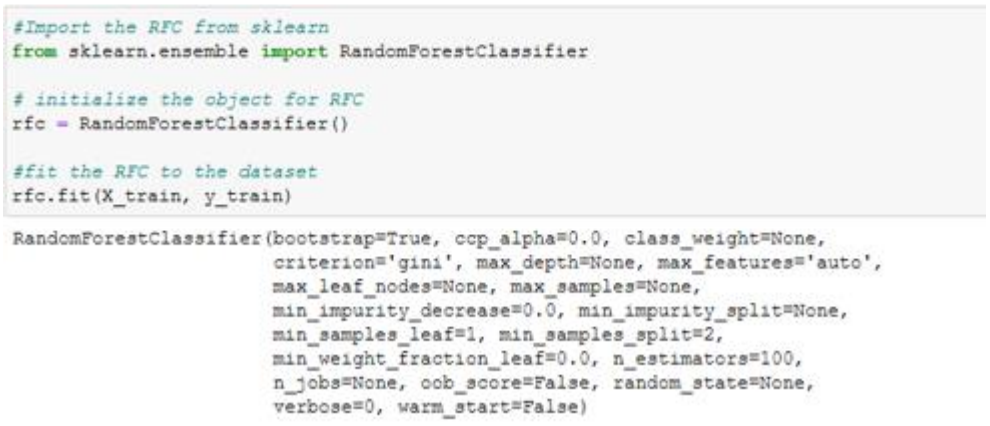
Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below).

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. Random Forest, which is nothing but Bagging, you draw random bootstrap samples and also random subset of features. And thus because of random feature selection, the trees are not correlated and are faster

Random Forest Classifier is an ensemble algorithm, which creates a set of decision trees from a randomly selected subset of the training set, which then aggregates the votes from different decision trees to decide the final class of the test object.

7.2.1 Train and Test Models:

- Importing RandomForestClassifierfrom packages of Ensemble.
- The training dataset is used to prepare a model, to train it.
- We pretend the test dataset is new data where the output values are withheld from the algorithm. We gather predictions from the trained model on the inputs from the test dataset and compare them to the withheld output values of the test set.



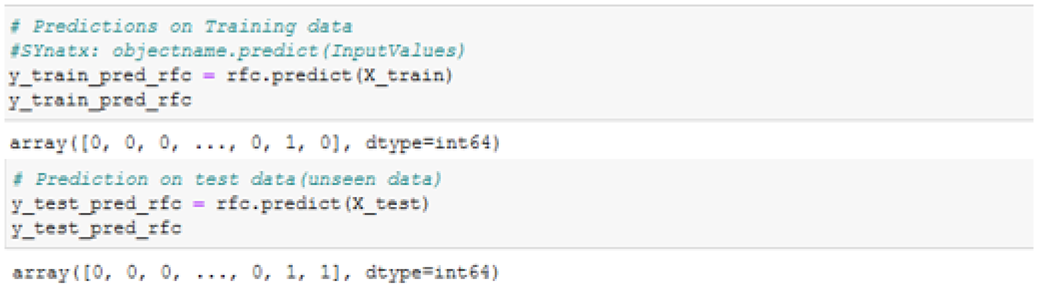
```
#Import the RFC from sklearn
from sklearn.ensemble import RandomForestClassifier

# initialize the object for RFC
rfc = RandomForestClassifier()

#fit the RFC to the dataset
rfc.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Fig 7.2.1: Train the model

7.2.2 Predict Analytics

Predictive analytics uses historical data to predict future events. Typically, historical data is used to build a mathematical model that captures important trends. That predictive model is then used on current data to predict what will happen next, or to suggest actions to take for optimal outcomes.



```
# Predictions on Training data
#SYnatx: objectname.predict(InputValues)
y_train_pred_rfc = rfc.predict(X_train)
y_train_pred_rfc
```

```
array([0, 0, 0, ..., 0, 1, 0], dtype=int64)
```

```
# Prediction on test data(unseen data)
y_test_pred_rfc = rfc.predict(X_test)
y_test_pred_rfc
```

```
array([0, 0, 0, ..., 0, 1, 1], dtype=int64)
```

Fig 7.2.2: Predict Values

7.2.3 Classification report and Confusion Matrix:

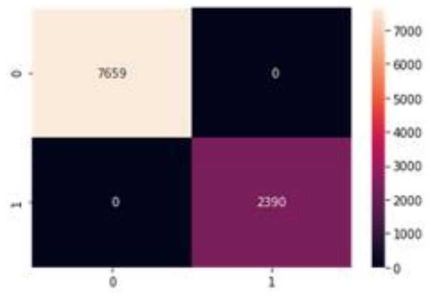
A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report as shown below

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

- In this we observe precision, recall, f1 score, accuracy, macro avg, weighted avg in Classification Report for both train and test.
- In Confusion Matrix we observe True Positive, True Negative, False Positive, False Negative for both train and test.

```
# compare the actual values(y_train) with predicted values(y_train_pred)
from sklearn.metrics import confusion_matrix,classification_report
sns.heatmap(confusion_matrix(y_train,y_train_pred_rfc),annot=True,fmt='3.0f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xc39c6c8>
```



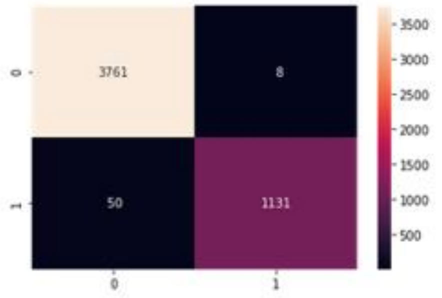
```
#printing classification report
print(classification_report(y_train,y_train_pred_rfc))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7659
           1       1.00      1.00      1.00      2390

    accuracy                           1.00     10049
   macro avg       1.00      1.00      1.00     10049
weighted avg       1.00      1.00      1.00     10049
```

Fig 7.2.3.1: Classification Report

```
# compare the actual values(y_test) with predicted values(y_test_pred)
from sklearn.metrics import confusion_matrix,classification_report
sns.heatmap(confusion_matrix(y_test,y_test_pred_rfc),annot=True,fmt='3.0f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xc44f288>
```



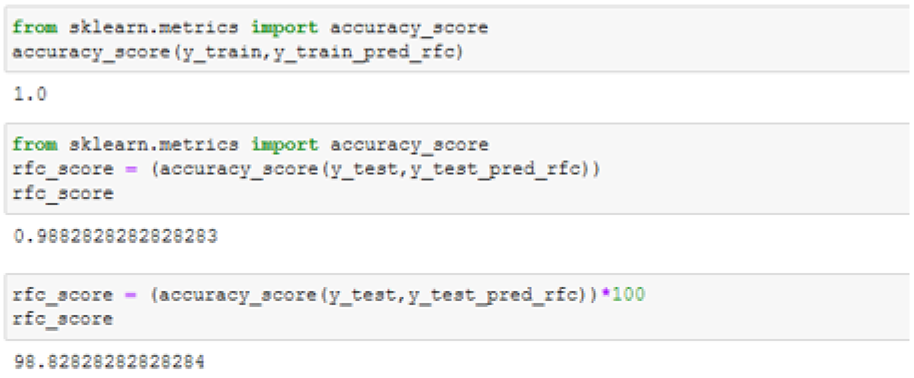
```
#printing classification report
print(classification_report(y_test,y_test_pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      3769
           1       0.99      0.96      0.98      1181

    accuracy                           0.99      4950
   macro avg       0.99      0.98      0.98      4950
weighted avg       0.99      0.99      0.99      4950
```

Fig 7.2.3.2: Classification Report

7.2.4 Accuracy score

Accuracy is one metric for evaluating classification models. Informally, **accuracy** is the fraction of predictions our model got right.



```
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_train_pred_rfc)
```

```
1.0
```

```
from sklearn.metrics import accuracy_score
rfc_score = (accuracy_score(y_test,y_test_pred_rfc))
rfc_score
```

```
0.9882828282828283
```

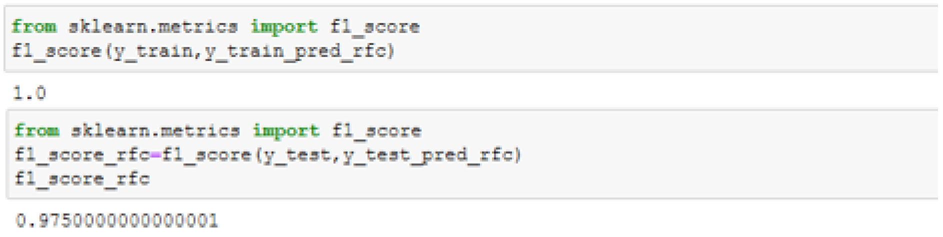
```
rfc_score = (accuracy_score(y_test,y_test_pred_rfc))*100
rfc_score
```

```
98.82828282828284
```

Fig 7.2.4: Accuracy Score

7.2.5 F1 Score

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal.



```
from sklearn.metrics import f1_score
f1_score(y_train,y_train_pred_rfc)
```

```
1.0
```

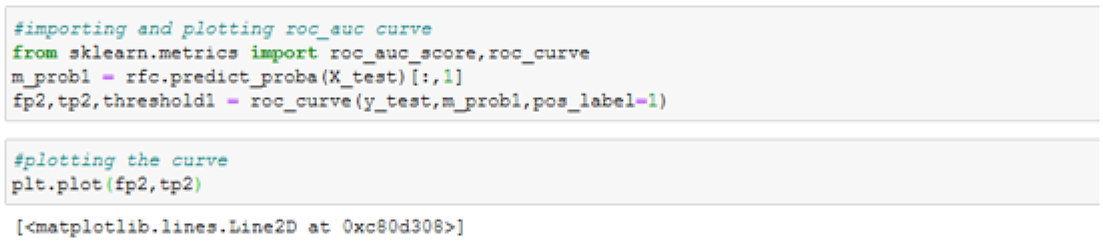
```
from sklearn.metrics import f1_score
f1_score_rfc=f1_score(y_test,y_test_pred_rfc)
f1_score_rfc
```

```
0.9750000000000001
```

Fig 7.2.5: F1 score

7.2.6 ROC-AUC Score

- An ROC curve (Receiver Operating Characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:
- True Positive Rate
- False Positive Rate
  - The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.
  - The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.



```
#importing and plotting roc_auc curve
from sklearn.metrics import roc_auc_score,roc_curve
m_prob1 = rfc.predict_proba(X_test)[:,1]
fp2,tp2,threshold1 = roc_curve(y_test,m_prob1,pos_label=1)
```

```
#plotting the curve
plt.plot(fp2,tp2)
```

```
[<matplotlib.lines.Line2D at 0xc80d308>]
```

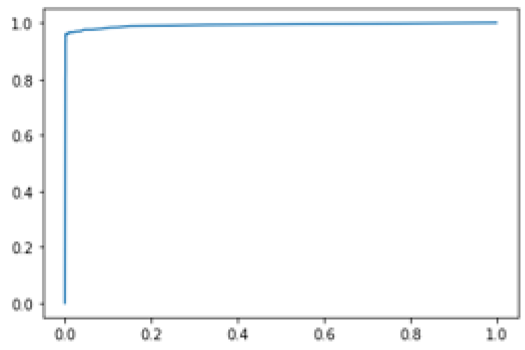


Fig 7.2.6: ROC_AUC Curve

- In this we are plotting roc_auc curve for Random Forest Classifier. And we observe 98% Accuracy score.

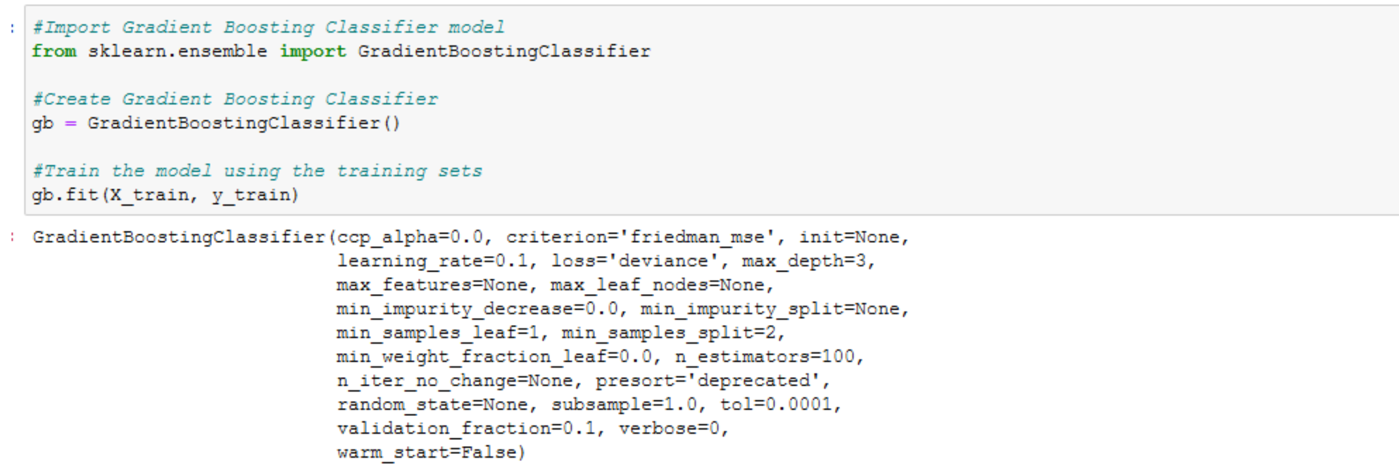
7.3 Gradient boosting:

Gradient boosting explained. Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model in order to minimize the error.

7.3.1 Train and Test Models:

- Importing Gradient Boosting from packages of Ensemble.
- The training dataset is used to prepare a model, to train it.

- We pretend the test dataset is new data where the output values are withheld from the algorithm. We gather predictions from the trained model on the inputs from the test dataset and compare them to the withheld output values of the test set.



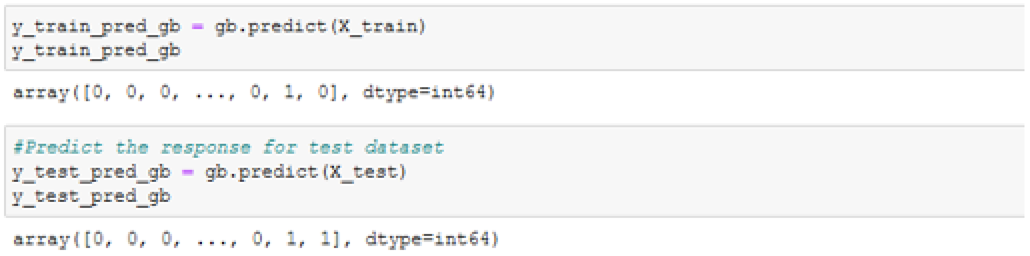
```
#Import Gradient Boosting Classifier model
from sklearn.ensemble import GradientBoostingClassifier

#Create Gradient Boosting Classifier
gb = GradientBoostingClassifier()

#Train the model using the training sets
gb.fit(X_train, y_train)
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

Fig 7.3.1: Train the model

7.3.2 Predict Analytics

Predictive analytics uses historical data to predict future events. Typically, historical data is used to build a mathematical model that captures important trends. That predictive model is then used on current data to predict what will happen next, or to suggest actions to take for optimal outcomes.



```
y_train_pred_gb = gb.predict(X_train)
y_train_pred_gb
```

```
array([0, 0, 0, ..., 0, 1, 0], dtype=int64)
```

```
#Predict the response for test dataset
y_test_pred_gb = gb.predict(X_test)
y_test_pred_gb
```

```
array([0, 0, 0, ..., 0, 1, 1], dtype=int64)
```

Fig 7.3.2: Predict Values

7.3.3 Classification report and Confusion Matrix:

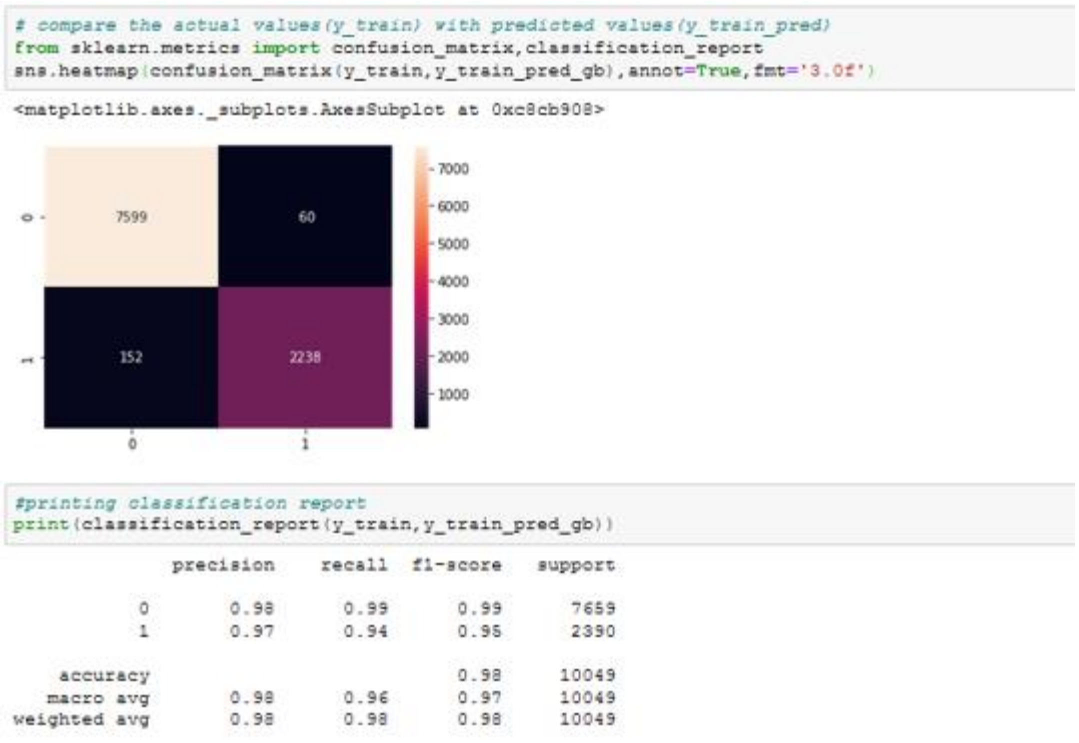
A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report as shown below.

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

- In this we observe precision, recall, f1 score, accuracy, macro avg, weighted avg in

  Classification Report for both train and test.

- In Confusion Matrix we observe True Positive, True Negative, False Positive, False Negative for both train and test.



```
# compare the actual values(y_train) with predicted values(y_train_pred)
from sklearn.metrics import confusion_matrix,classification_report
sns.heatmap(confusion_matrix(y_train,y_train_pred_gb),annot=True,fmt='3.0f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xc8cb908>
```

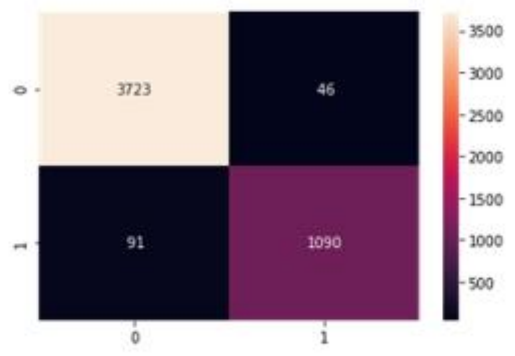
```
#printing classification report
print(classification_report(y_train,y_train_pred_gb))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      7659
           1       0.97      0.94      0.95      2390

    accuracy                           0.98     10049
   macro avg       0.98      0.96      0.97     10049
weighted avg       0.98      0.98      0.98     10049
```

Fig 7.3.3.1: Classification Report

```
# compare the actual values(y_test) with predicted values(y_test_pred)
from sklearn.metrics import confusion_matrix,classification_report
sns.heatmap(confusion_matrix(y_test,y_test_pred_gb),annot=True,fmt='3.0f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0xc98b948>
```



```
#printing classification report
print(classification_report(y_test,y_test_pred_gb))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      3769
           1       0.96      0.92      0.94      1181

    accuracy                           0.97      4950
   macro avg       0.97      0.96      0.96      4950
weighted avg       0.97      0.97      0.97      4950
```

Fig 7.3.3.2: Classification Report

7.3.4 Accuracy score

Accuracy is one metric for evaluating classification models. Informally, **accuracy** is the fraction of predictions our model got right.

```
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_train_pred_gb)
```

```
0.978903373469997
```

```
from sklearn.metrics import accuracy_score
gb_score = (accuracy_score(y_test,y_test_pred_gb))
gb_score
```

```
0.9723232323232324
```

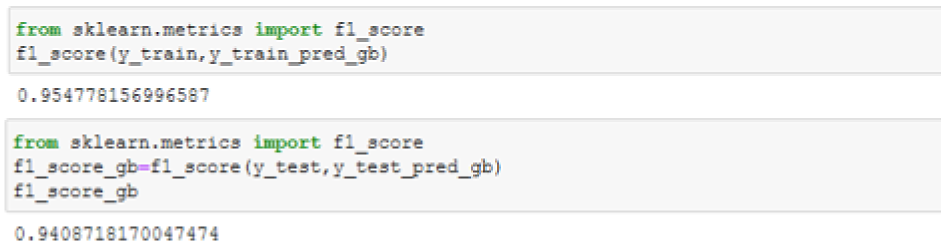
```
gb_score = (accuracy_score(y_test,y_test_pred_gb))*100
gb_score
```

```
97.23232323232324
```

Fig 7.3.4: Accuracy Score

7.3.5 F1 Score

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal.

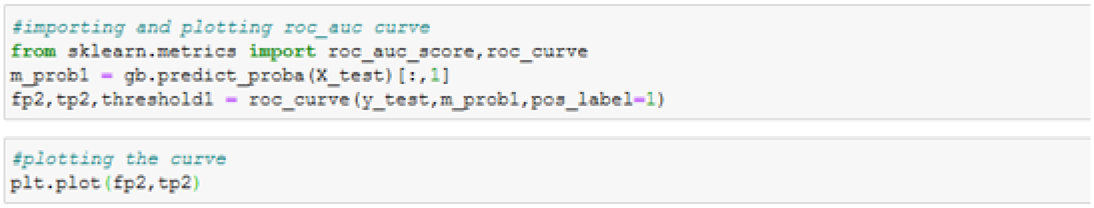


```
from sklearn.metrics import f1_score
f1_score(y_train,y_train_pred_gb)

0.954778156996587

from sklearn.metrics import f1_score
f1_score_gb=f1_score(y_test,y_test_pred_gb)
f1_score_gb

0.9408718170047474
```

Fig 7.3.5: F1 score

7.3.6 ROC-AUC Score

- An ROC curve (Receiver Operating Characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:
  - True Positive Rate
  - False Positive Rate
- The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.
- The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.



```
#importing and plotting roc_auc curve
from sklearn.metrics import roc_auc_score,roc_curve
m_prob1 = gb.predict_proba(X_test)[:,1]
fp2,tp2,threshold1 = roc_curve(y_test,m_prob1,pos_label=1)
```

```
#plotting the curve
plt.plot(fp2,tp2)
```

```
[<matplotlib.lines.Line2D at 0xcdc4908>]
```

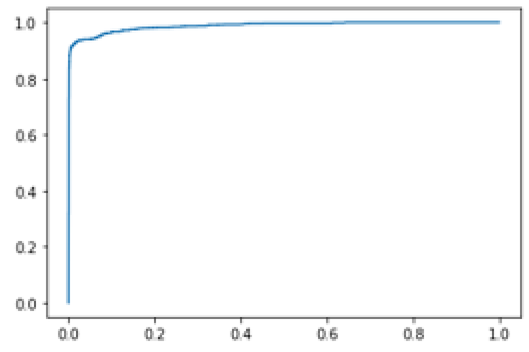


Fig 7.3.6: ROC_AUC Curve

- In this we are plotting roc_auc curve for Gradient Boosting. And we observe 97% Accuracy score.

CHAPTER 8

Hyper Parameter Tuning

A hyperparameter is a parameter whose value is set before the learning process begins. Hyperparameter tuning is also tricky in the sense that there is no direct way to calculate how a change in the hyperparameter value will reduce the loss of your model, so we usually resort to experimentation. This starts with us specifying a range of possible values for all the hyperparameters. Now, this is where most get stuck, what values you are going to try, and to answer that question, you first need to understand what these hyperparameters mean and how changing a hyperparameter will affect your model architecture, thereby try to understand how your model performance might change.

The next step after you define the range of values is to use a hyperparameter tuning method, there's a bunch, the most common and expensive being Grid Search

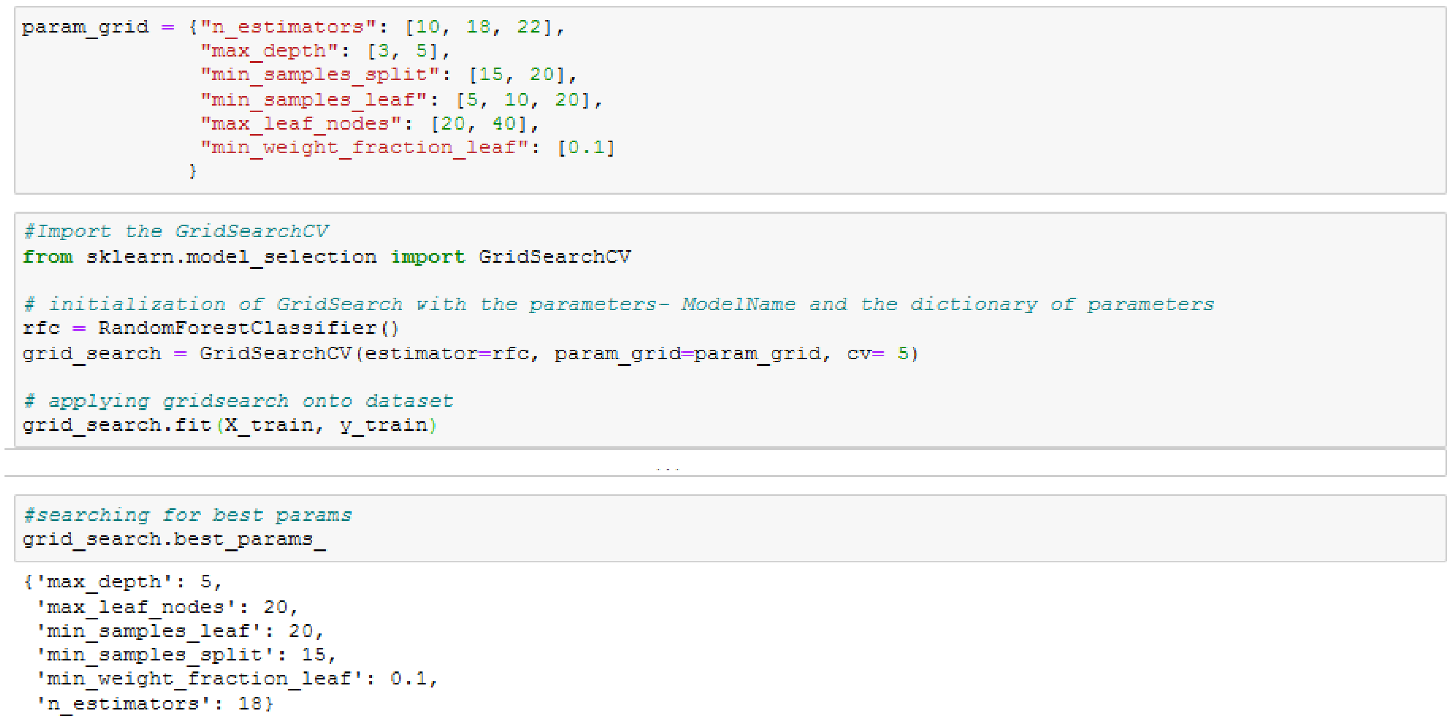
8.1. GridSearchCV

What is grid search?

Grid search is a traditional way to perform hyperparameter optimization. It works by searching exhaustively through a specified subset of hyperparameters.

Grid search is the process of performing hyper parameter tuning in order to determine the optimal values for a given model. This is significant as the performance of the entire model is based on the hyper parameter values specified.

Using sklearn's GridSearchCV, we first define our grid of parameters to search over and then run the grid search.



```
param_grid = {"n_estimators": [10, 18, 22],
              "max_depth": [3, 5],
              "min_samples_split": [15, 20],
              "min_samples_leaf": [5, 10, 20],
              "max_leaf_nodes": [20, 40],
              "min_weight_fraction_leaf": [0.1]
             }
```

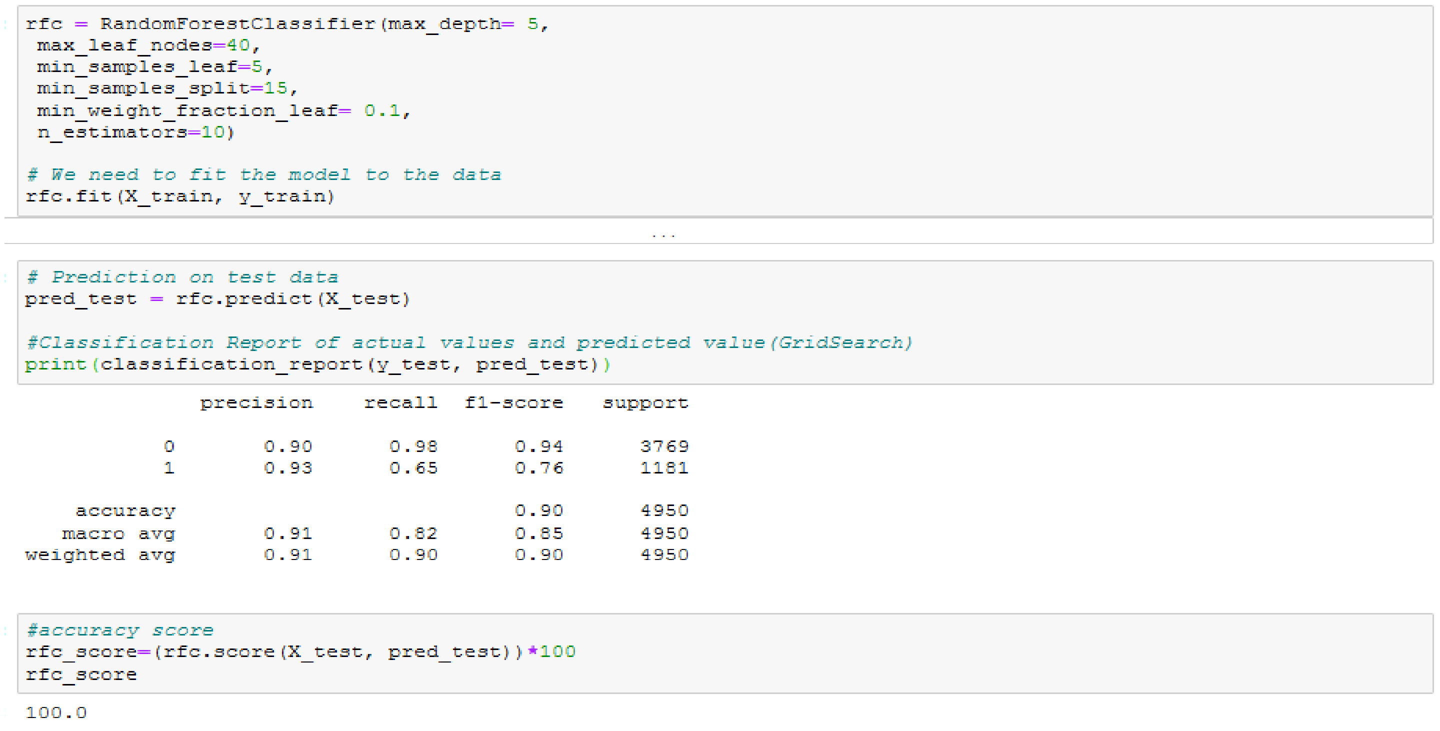
```
#Import the GridSearchCV
from sklearn.model_selection import GridSearchCV

# initialization of GridSearch with the parameters- ModelName and the dictionary of parameters
rfc = RandomForestClassifier()
grid_search = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)

# applying gridsearch onto dataset
grid_search.fit(X_train, y_train)
```

```
...
```

```
#searching for best params
grid_search.best_params_
```

```
{'max_depth': 5,
 'max_leaf_nodes': 20,
 'min_samples_leaf': 20,
 'min_samples_split': 15,
 'min_weight_fraction_leaf': 0.1,
 'n_estimators': 18}
```

Fig 8.1.1: Hyper parameter for RandomForestClassifier



```
rfc = RandomForestClassifier(max_depth= 5,
 max_leaf_nodes=40,
 min_samples_leaf=5,
 min_samples_split=15,
 min_weight_fraction_leaf= 0.1,
 n_estimators=10)

# We need to fit the model to the data
rfc.fit(X_train, y_train)
```

```
...
```

```
# Prediction on test data
pred_test = rfc.predict(X_test)

#Classification Report of actual values and predicted value(GridSearch)
print(classification_report(y_test, pred_test))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      3769
           1       0.93      0.65      0.76      1181

    accuracy                           0.90      4950
   macro avg       0.91      0.82      0.85      4950
weighted avg       0.91      0.90      0.90      4950
```

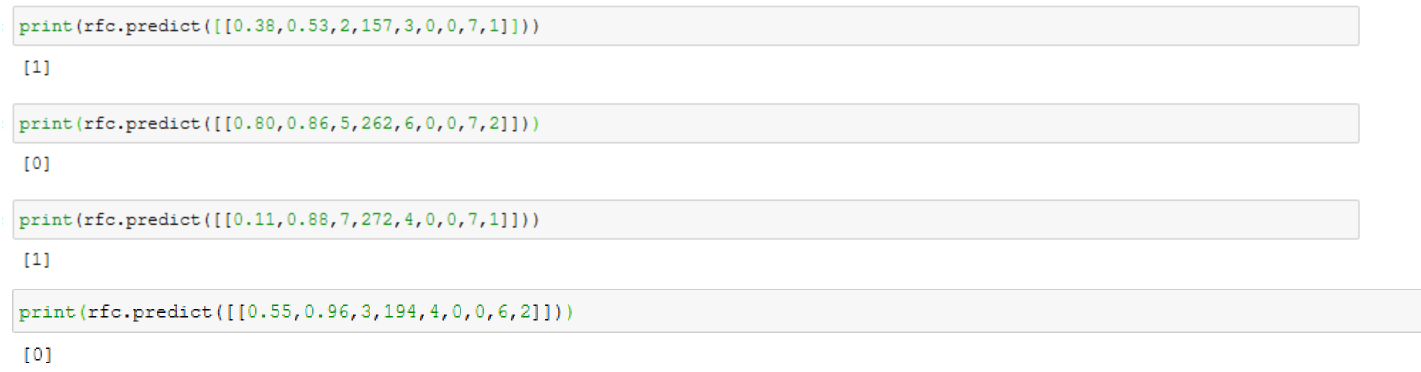
```
#accuracy score
rfc_score=(rfc.score(X_test, pred_test))*100
rfc_score
```

```
100.0
```

Fig 8.1.2: classification Report

CHAPTER 9

Predicting Attrition For Random Forest Classifier



```
print(rfc.predict([[0.38,0.53,2,157,3,0,0,7,1]]))
```

```
[1]
```

```
print(rfc.predict([[0.80,0.86,5,262,6,0,0,7,2]]))
```

```
[0]
```

```
print(rfc.predict([[0.11,0.88,7,272,4,0,0,7,1]]))
```

```
[1]
```

```
print(rfc.predict([[0.55,0.96,3,194,4,0,0,6,2]]))
```

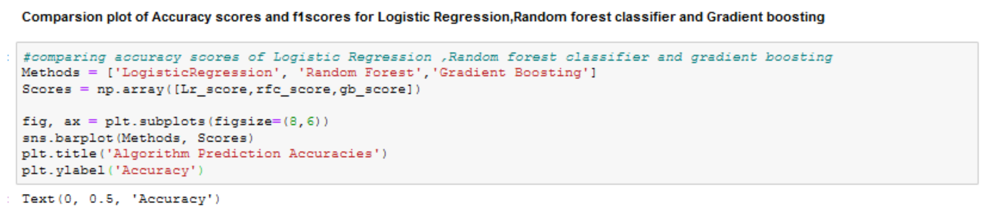
```
[0]
```

Fig 9.1: Predict

In above figure we are predicting attrition per each. In this 0 value stands for NO and 1 value stands for YES the person quits.

CHAPTER 10

Conclusion



Comparsion plot of Accuracy scores and f1scores for Logistic Regression,Random forest classifier and Gradient boosting

```
#comparing accuracy scores of Logistic Regression ,Random forest classifier and gradient boosting
Methods = ['LogisticRegression', 'Random Forest','Gradient Boosting']
Scores = np.array([Lr_score,rfc_score,gb_score])

fig, ax = plt.subplots(figsize=(8,6))
sns.barplot(Methods, Scores)
plt.title('Algorithm Prediction Accuracies')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```

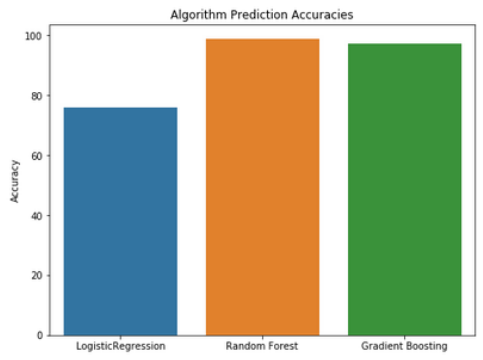


Fig 10.1: comparison plot Accuracy Scores(before gridsearchcsv)

In above plot, we plotted accuracy scores of Logistic Regression, Random forest classifier and gradient boosting. From these plots we can say that accuracy score of Random Forest Classifier is higher than Logistic Regression and Gradient Boosting. So, from this observation we can say that Random Forest Classifier is most preferable.

```
#comparing f1 scores of Logistic Regression ,Random forest classifier and gradient boosting
Methods = ['LogisticRegression', 'Random Forest','Gradient Boosting']
Scores = np.array([f1_score_lr,f1_score_rfc,f1_score_gb])

fig, ax = plt.subplots(figsize=(8,6))
sns.barplot(Methods, Scores)
plt.title('Algorithm Prediction f1 score')
plt.ylabel('f1 scores')
```

```
Text(0, 0.5, 'f1 scores')
```

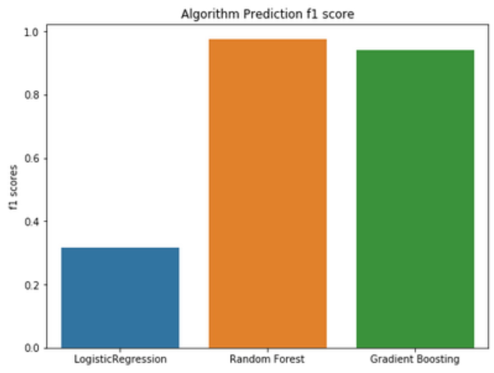


Fig 10.2: comparison plot F1 score(before gridsearchcsv)

In above plot, we plotted f1 scores of Logistic Regression, Random forest classifier and gradient boosting. From this plot we can say that f1 score of Random Forest Classifier is higher than Logistic Regression and Gradient Boosting. So, from this observation we can say that Random Forest Classifier is most preferable

Comparsion plot of Accuracy scores and f1scores for Logistic Regression,Random forest classifier and gradient boosting After Hyper Parameter Tuning (and GridsearchCV)

```
#comparing accuracy scores of Logistic Regression ,Random forest classifier and gradient boosting
Methods = ['LogisticRegression', 'Random Forest','Gradient Boosting']
Scores = np.array([Lr_score,rfc_score,gb_score])

fig, ax = plt.subplots(figsize=(8,6))
sns.barplot(Methods, Scores)
plt.title('Algorithm Prediction Accuracies')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```

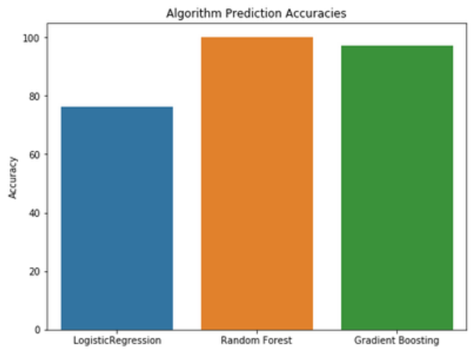
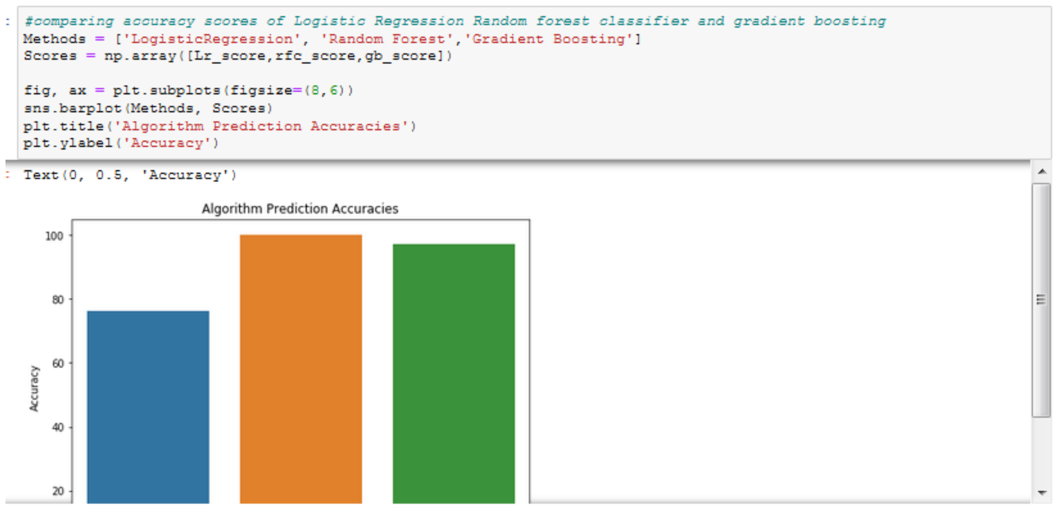


Fig 10.3: Comparison plot Accuracy score (after Hyper parameters Tuning GridsearchCV)

In above plot, we plotted accuracy scores of Logistic Regression, Random forest classifier and gradient boosting after hyper parameter tuning (and GridsearchCV)for best model. From this plot we can say that accuracy score of Random Forest Classifier is higher than Logistic

Regression and gradient boosting. So, from this observation we can say that Random Forest Classifier is most preferable.



```
#comparing accuracy scores of Logistic Regression Random forest classifier and gradient boosting
Methods = ['LogisticRegression', 'Random Forest','Gradient Boosting']
Scores = np.array([Lr_score,rfc_score,gb_score])

fig, ax = plt.subplots(figsize=(8,6))
sns.barplot(Methods, Scores)
plt.title('Algorithm Prediction Accuracies')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```

Fig 10.4: Comparison plot F1 score (after Hyper parameters Tuning GridsearchCV)

In above plot, we plotted f1 scores of Logistic Regression, Random forest classifier and gradient boosting after hyper parameter tuning (and GridsearchCV) for best model. From this plot we can say that f1 score of Random Forest Classifier is higher than Logistic Regression and gradient boosting. So, from this observation we can say that Random Forest Classifier is most preferable.

It is concluded after performing thorough Exploratory Data analysis which include Stats models which are computed to get accuracy which are computed to get a clear understanding of the data set (which parameter has most abundant effect on the study case) ,after seeing accuracy and comparison plot before and after grid searchcv of Logistic Regression ,Random forest classifier and gradient boosting and its come to point that using of Random forest classifier is better rather than Logistic Regression and gradient boosting.

Reference

https://github.com/sumathi16/Datasets/blob/master/Human_Resources_Employee_Attrition.csv

https://moodle.dspsinstitute.com/

https://github.com/shrestareddy/AIandML-Project