MACHINE LEARNING
(CHURN PREDICTION)

Summer Internship Report Submitted in partial fulfillment of the requirement for undergraduate degree of

Bachelor of Technology
In
COMPUTER SCIENCE AND ENGINEERING
By
DOMAKONDA SAI KRISHNA
221710307022

Under the Guidance of

GITAM
(DEEMED TO BE UNIVERSITY)
(Estd. u/s 3 of the UGC Act, 1956)
VISAKHAPATNAM ✸ HYDERABAD ✸ BENGALURU
Accredited by NAAC with A$^+$ Grade

Department Of COMPUTER SCIENCE AND ENGINEERING
GITAM School of Technology
GITAM (Deemed to be University) Hyderabad-502329
July 2020

DECLARATION

I submit this industrial training work entitled “CHURN PREDICTION” 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 and Engineering”. I declare that it was carried out independently by me under the guidance of , 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 DOMAKONDA SAI KRISHNA
DATE:29-07-2020 221710307022

GITAM (DEEMED TO BE UNIVERSITY)
Hyderabad-502329, India
Dated:29-07-2020

CERTIFICATE

This is to certify that the Industrial Training Report entitled "CHURN PREDICTION" is being submitted by DOMAKONDA SAI KRISHNA (221710307022) 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 him at the Computer Science & Engineering Department, GITAM University Hyderabad Campus under my guidance and supervision.

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 **Prof.N.Seeta Ramaiah**, Principal, GITAM Hyderabad

I would like to thank respected **Mr.S.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 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.

DOMAKONDA SAI KRISHNA
221710307022

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. The primary task is to build the models of higher accuracy . As companies increase their efforts in retaining customers, being able to predict accurately ahead of time, whether a customer will churn in the foreseeable future is an extremely powerful tool for any company. Customer churn is a major problem and one of the most important concerns for large companies. Therefore, finding factors that increase customer churn is important to take necessary actions to reduce this churn. The main contribution of our work is to develop a churn prediction model which predicts customers who are most likely subject to churn. The model developed in this work uses machine learning techniques such as LogisticRegression, RandomForestClassifier and KNeighborsClassifier and are compared to know which one among them predicts with most accuracy.

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

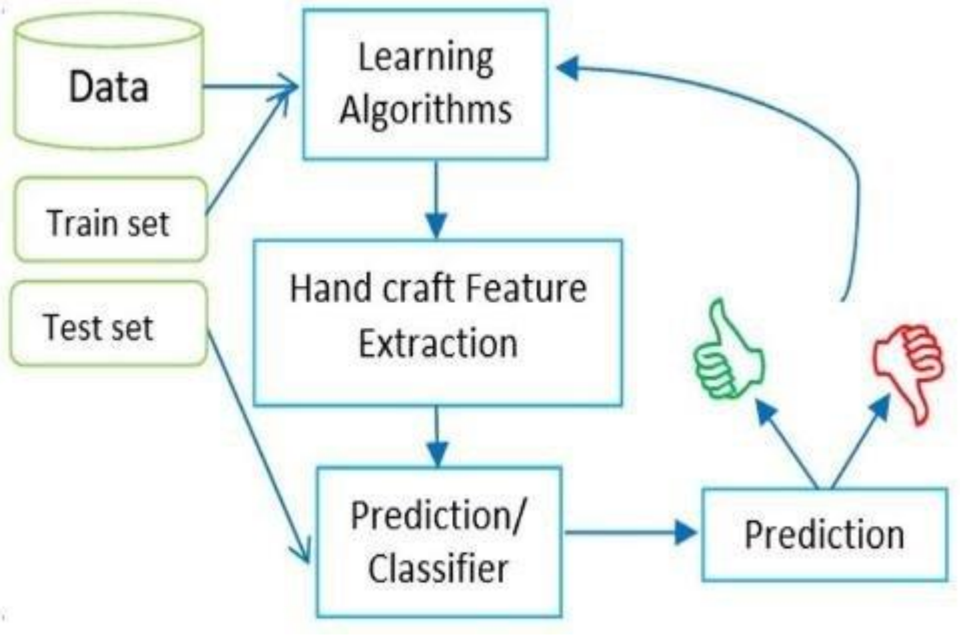


Figure 1.2 1 : The Process Flow

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.

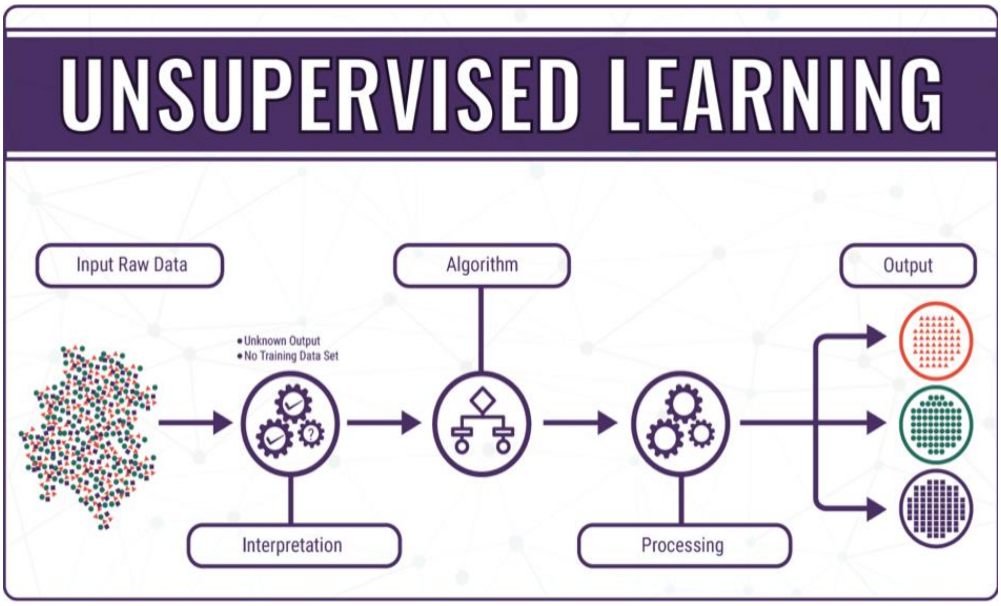


Figure 1.4.2 1: Unsupervised Learning

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.

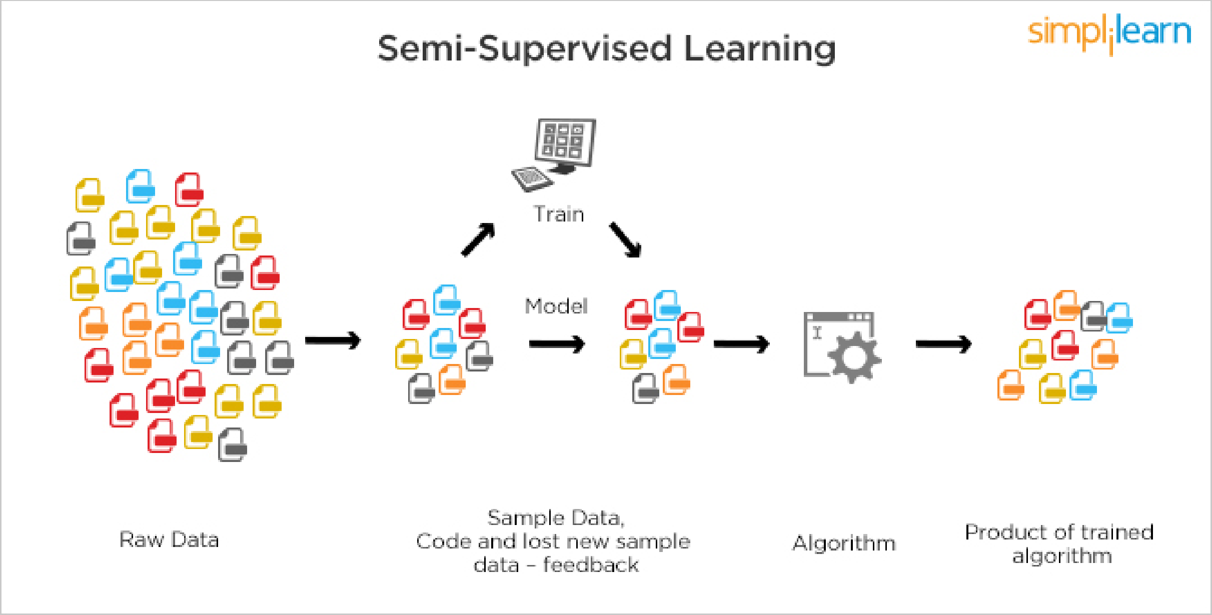


Figure 1.4.3 1: Semi Supervised Learning

CHAPTER 2

INFORMATION ABOUT DEEP LEARNING

2.1 Importance of Deep Learning:

Deep learning recently returned to the headlines when google’s AlphaGo program crushed Lee Sedol, one of the highest-ranking Go players in the word. Google has invested heavily in deep learning and AlphaGo is just their latest deep learning project to make the news. Google's search engine, voice recognition system and self-driving cars all rely heavily on deep learning. They've used deep learning networks to build a program that picks out an alternative still from a YouTube video to use as a thumbnail. Late last year Google announced smart reply, a deep learning network that writes short email responses for you. Deep learning is clearly powerful, but it also may seem somewhat mysterious.

2.2 Uses of Machine Learning:

The value of machine learning technology has been recognized by companies across several industries that deal with huge volumes of data. By leveraging insights obtained from this data, companies are able work in an efficient manner to control costs as well as get an edge over their competitors. This is how some sectors / domains are implementing machine learning

Financial Services

Companies in the financial sector are able to identify key insights in financial data as well as prevent any occurrences of financial fraud, with the help of machine learning technology. The technology is also used to identify opportunities for investments and trade. Usage of cyber surveillance helps in identifying those individuals or institutions which are prone to financial risk, and take necessary actions in time to prevent fraud.

Marketing and Sales

Companies are using machine learning technology to analyze the purchase history of their customers and make personalized product recommendations for their next purchase. This ability to capture, analyze, and use customer data to provide a personalized shopping experience is the future of sales and marketing.

Government

Government agencies like utilities and public safety have a specific need FOR Ml, as they have multiple data sources, which can be mined for identifying useful patterns and insights. For example sensor data can be analyzed to identify ways to minimize costs and increase efficiency. Furthermore, ML can also be used to minimize identity thefts and detect fraud.

Healthcare

With the advent of wearable sensors and devices that use data to access health of a patient in real time, ML is becoming a fast-growing trend in healthcare. Sensors in wearable provide real-time patient information, such as overall health condition, heartbeat, blood pressure and other vital parameters. Doctors and medical experts can use this information to analyze the health condition of an individual, draw a pattern from the patient history, and predict the occurrence of any ailments in the future. The technology also empowers medical experts to analyze data to identify trends that facilitate better diagnoses and treatment.

Transportation

Based on the travel history and pattern of traveling across various routes, machine learning can help transportation companies predict potential problems that could arise on certain routes, and accordingly advise their customers to opt for a different route. Transportation firms and delivery organizations are increasingly using machine learning technology to carry out data analysis their customers make smart decisions when they travel. and data modeling to make informed decisions and help

Oil and Gas

This is perhaps the industry that needs the application of machine learning the most. Right from analyzing underground minerals and finding new energy sources to streaming oil distribution, ML applications for this industry are vast and are still expanding.

2.3 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 discovered 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 3

PYTHON

Basic programming language used for machine learning is : PYTHON

3.1 INTRODUCTION TO PYTHON:

- 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.

3.2 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

3.3 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.

3.4 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.

3.4.1 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.

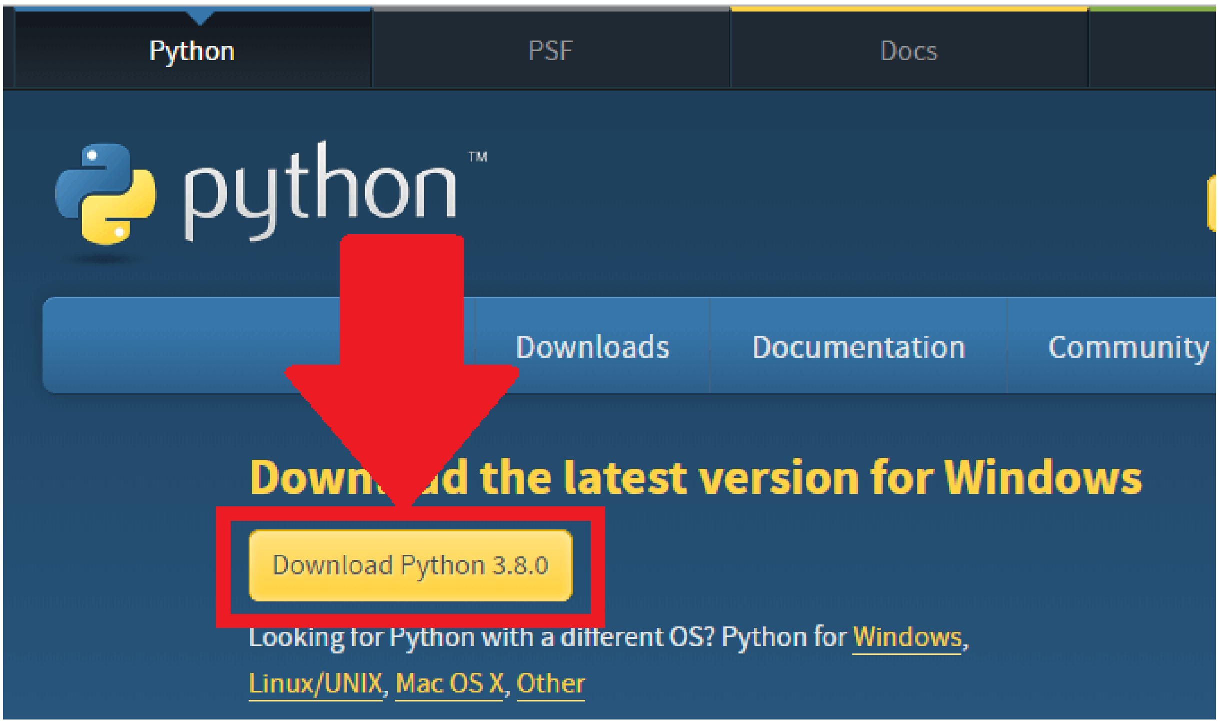

Python
PSF
Docs
python™
Downloads
Documentation
Community
Download the latest version for Windows
Download Python 3.8.0
Looking for Python with a different OS? Python for Windows,
Linux/UNIX, Mac OS X, Other

Figure 3.4.1 1: Python download

3.4.2 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 that 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-bits graphic 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)

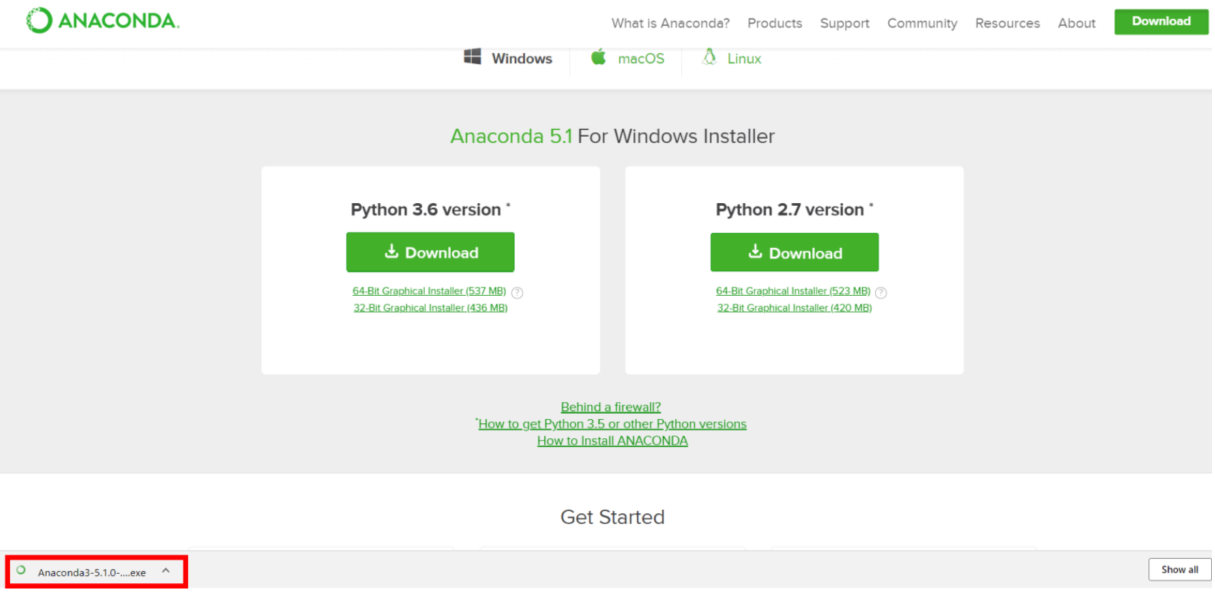


Figure 3.4.2 1 : Anaconda download

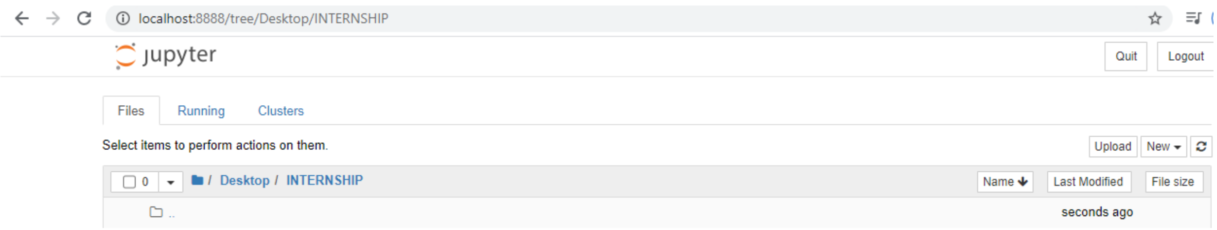


Figure 3.4.2 2 : Jupyter notebook

3.5 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 –

o Numbers

o Strings

o Lists

o Tuples

o Dictionary

3.5.1 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).

3.5.2 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.

3.5.3 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 types.
- 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.

3.5.4 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.

3.5.5 Python Dictionary:

- Python's dictionaries are a 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.

3.6 PYTHON FUNCTION:

3.6.1 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.

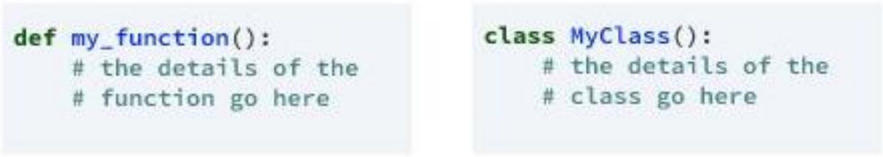
3.6.2 Calling a Function:

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.

3.7 PYTHON USING OOPs CONCEPTS:

3.7.1 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
```

Figure 3.7 1:Defining a Class

3.7.2 __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 4

CASE STUDY

PROJECT NAME-CHURN PREDICTION

4.1 PROBLEM STATEMENT:

To Predict the Customer Churn based on various variables.

Packages used:

1.Pandas

2.Numpy

3. Matplotlib.pyplot

4.Seaborn

5.sklearn.model_selection.train_test_split

6. from sklearn.utils.resample

7. from sklearn.preprocessing.MinMaxScaler

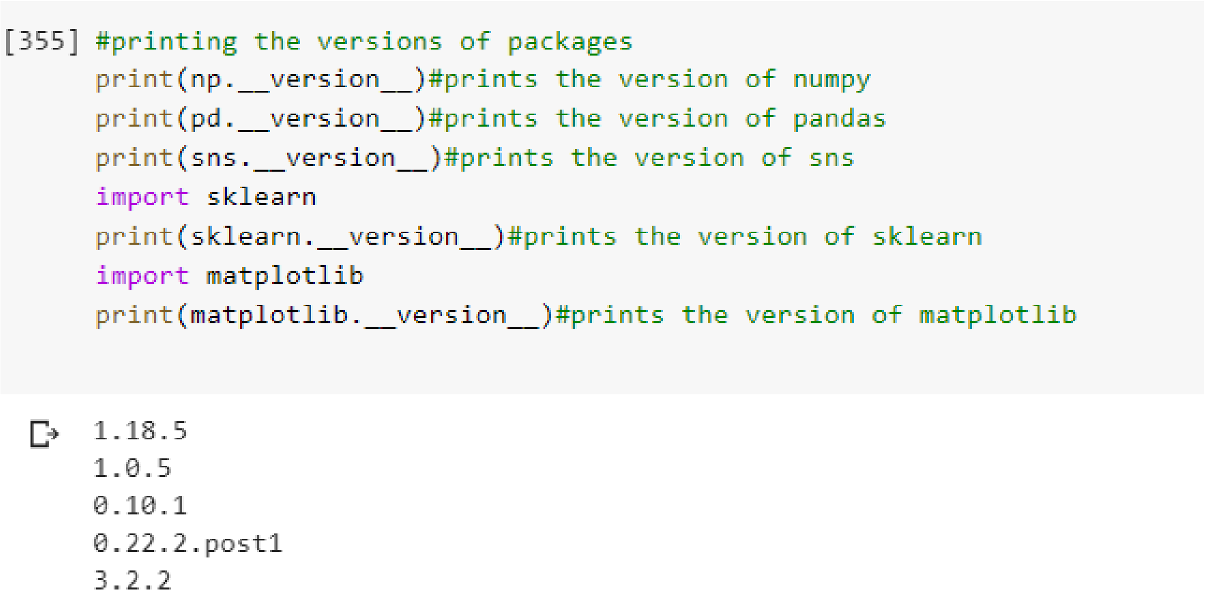
8.sklearn.metrics.classification_report,accuracy_score,confusion_matrix

9.sklearn.linear_model.LogisticRegression

10.sklearn.ensemble.RandomForestClassifier

11. from sklearn.neighbors.KNeighborsClassifier

Versions Of Packages:



```
[355] #printing the versions of packages
      print(np.__version__)#prints the version of numpy
      print(pd.__version__)#prints the version of pandas
      print(sns.__version__)#prints the version of sns
      import sklearn
      print(sklearn.__version__)#prints the version of sklearn
      import matplotlib
      print(matplotlib.__version__)#prints the version of matplotlib
```

```
1.18.5
1.0.5
0.10.1
0.22.2.post1
3.2.2
```

Figure 4.1 1:Versions of packages

Algorithms used:

1. K-NEAREST NEIGHBORS CLASSIFIER ALGORITHM

2.LOGISTIC REGRESSION ALGORITHM

3.RANDOM FOREST CLASSIFIER ALGORITHM

4.2 DATA SET:

The given data set consists of the following parameters:

1.customerID -ID of the customer

2.gender-Gender of the customer

3.SeniorCitizen-Whether a customer is a senior citizen or not

4.Partner-Does a customer have a partner or not

5.Dependents -Whether the customer has dependents or not

6.tenure-Tenure of the customer’s active subscription

7.PhoneService-Whether the customer has a phone service or not

8.MultipleLines-Whether the customer’s phone service has multiple lines or not

9.InternetService-Whether the customer has an internet service if yes which type

10.OnlineSecurity- Whether the customer has an online security or not

11.OnlineBackup- Whether the customer has an online backup or not

12.DeviceProtection- Whether the customer has a device protection or not

13.TechSupport- Whether the customer has a Tech Support or not

14.StreamingTV-Whether the customer streams tv or not

15.StreamingMovies-Whether the customer streams movies or not

16.Contract-Contract opted by the customer

17.PaperlessBilling-Whether the customer chose paperless billing or not

18.PaymentMethod-Type of payment method used by the customer

19.MonthlyCharges-Monthly charges of the subscription chosen by the customer

20.TotalCharges-Total charges spent by the customer

21.Churn-This is the target variable

4.3 OBJECTIVE OF THE CASE STUDY:

To build a machine learning model which identifies the customers who are most likely to churn.

CHAPTER 5

DATA PREPROCESSING

5.1 PREPROCESSING OF THE DATA:

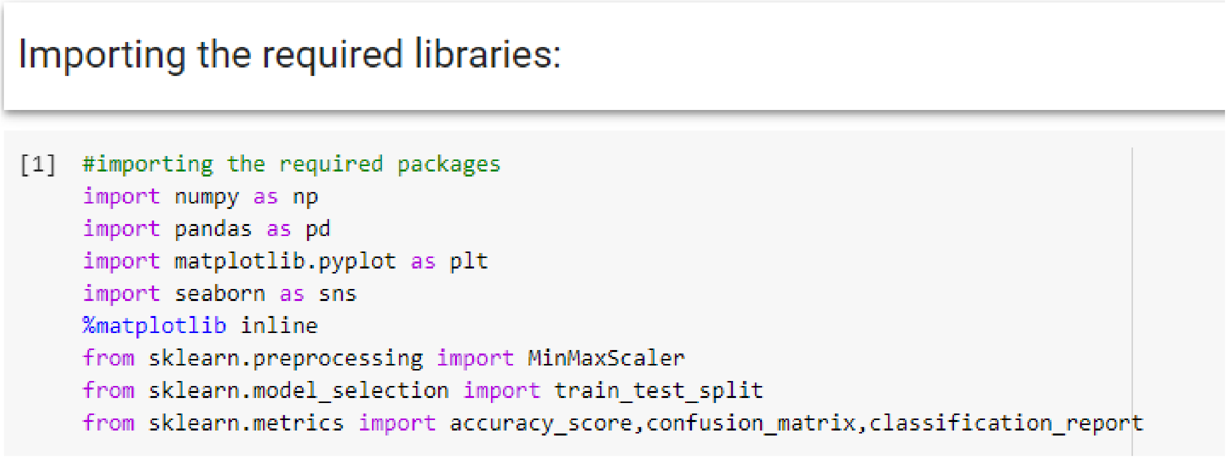
Preprocessing of the data actually involves the following steps:

5.1.1 GETTING THE DATASET:

We can get the data set from the database or we can get the data from client.

5.1.2 IMPORTING THE LIBRARIES:

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



Importing the required libraries:

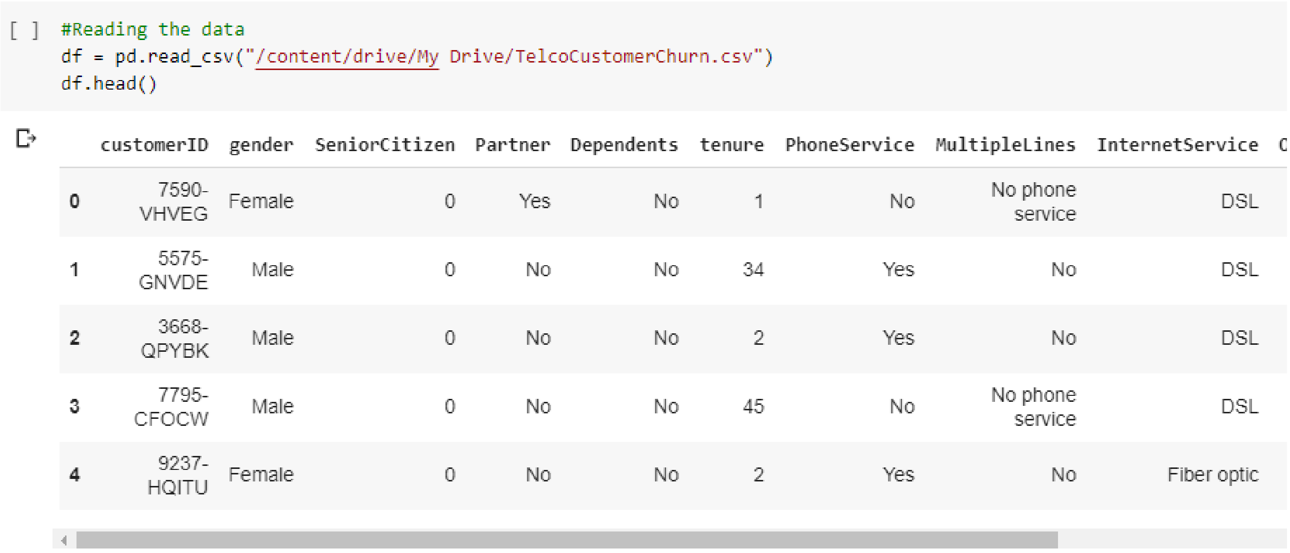
```
[1]  #importing the required packages
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     %matplotlib inline
     from sklearn.preprocessing import MinMaxScaler
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

Figure 5.1.2 1: Importing Libraries

5.1.3 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.

Importing the dataset:



```
#Reading the data
df = pd.read_csv("/content/drive/My Drive/TelcoCustomerChurn.csv")
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	C
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	

Figure 5.1.3 1 : Reading the dataset

5.1.4 HANDLING MISSING VALUES:

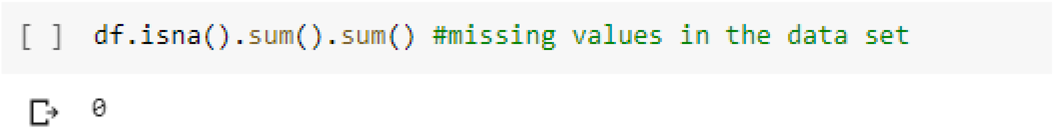
Missing values can be handled in many ways using some inbuilt methods:

(a)dropna()

(b)fillna()

(c)interpolate()

(d)mean imputation and median imputation



```
df.isna().sum().sum() #missing values in the data set
```

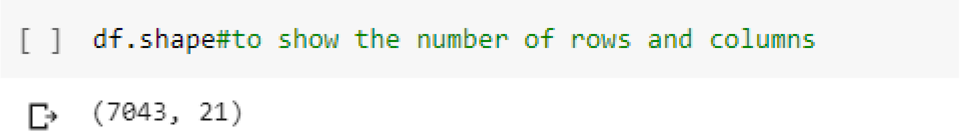
```
0
```

Figure 5.1.4 1:Checking for missing values

Since there were no missing values,we can proceed with the further steps.

5.1.5 STATISTICAL ANALYSIS:

- The number of rows and columns of the dataset can be found using data.shape()

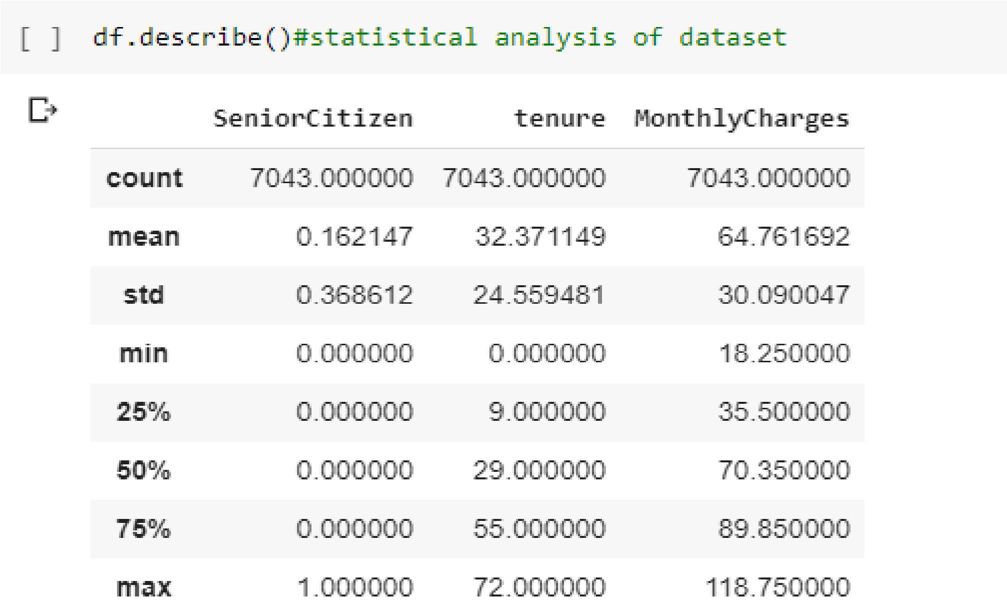


```
[ ] df.shape#to show the number of rows and columns
```

```
(7043, 21)
```

Figure 5.1.5 1: Shape of the data

- Our data set contains 7043 rows and 21 columns
- We need to describe the data.This can be done as follows:



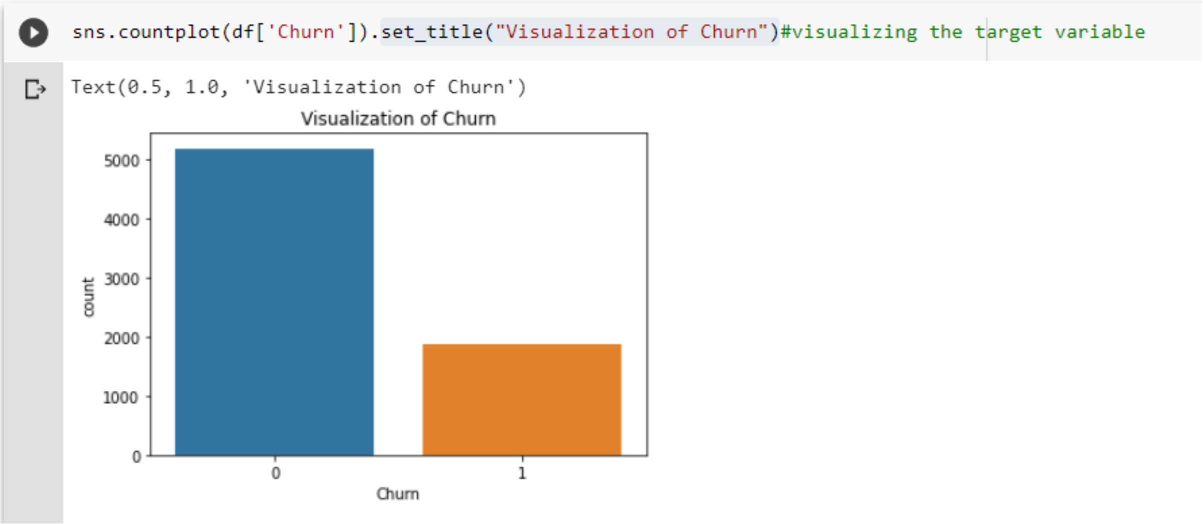
```
[ ] df.describe()#statistical analysis of dataset
```

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

Figure 5.1.5 2: Description of Data

5.1.6 GENERATING PLOTS:

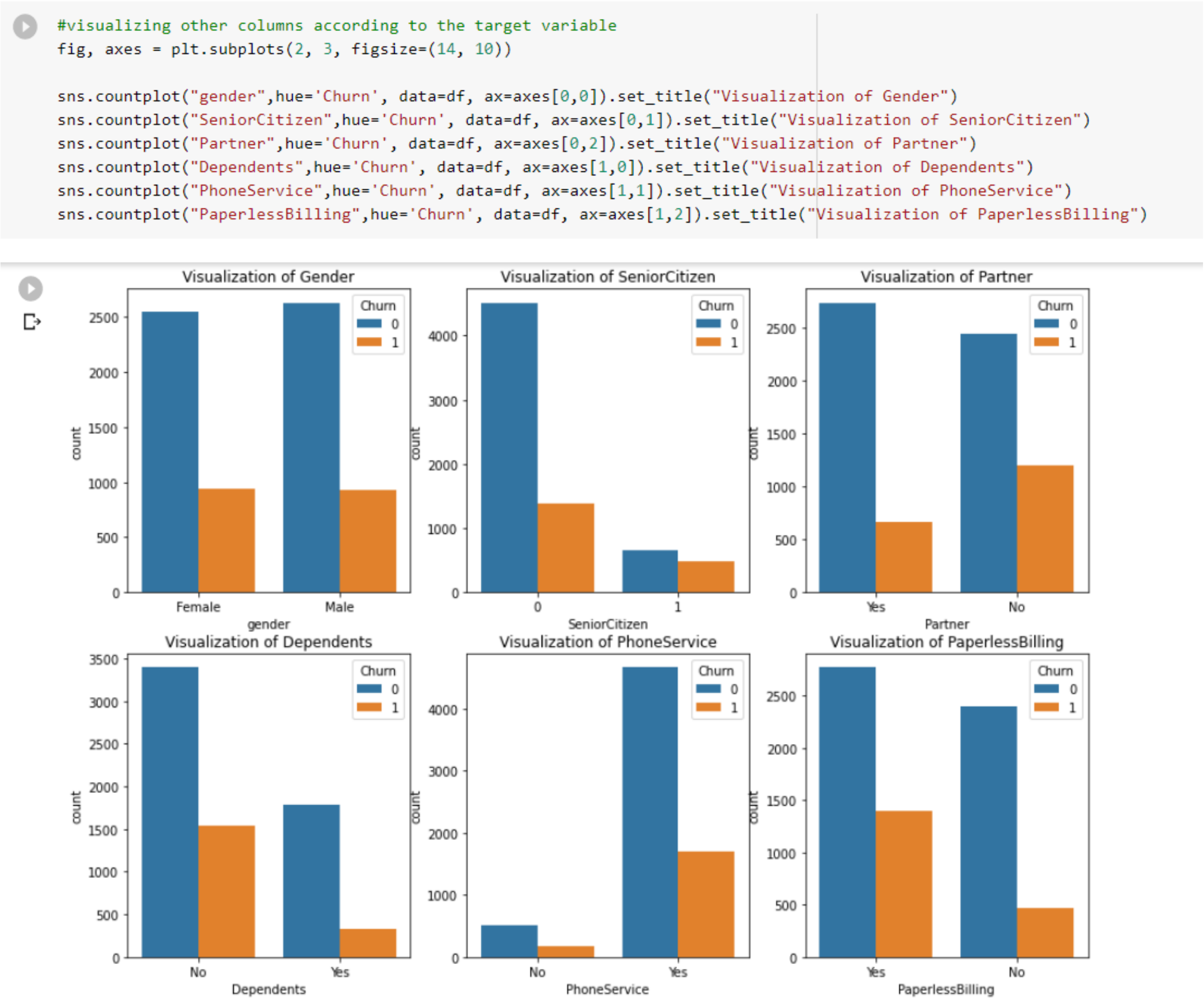
Visualization of the target column i.e. the CHURN column using COUNTPLOT .



```
sns.countplot(df['Churn']).set_title("Visualization of Churn")#visualizing the target variable
```

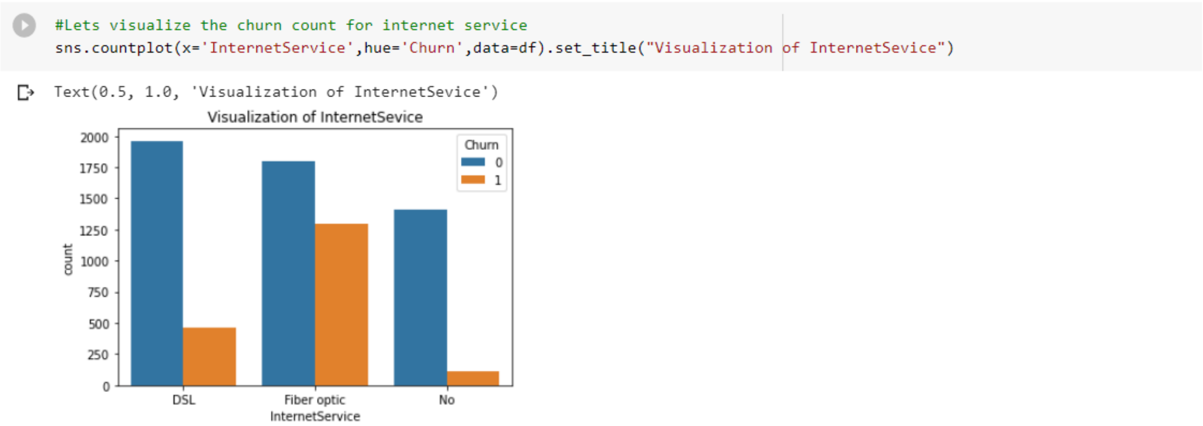
```
Text(0.5, 1.0, 'Visualization of Churn')
```

Figure 5.1.6 1: Visualization of the target variable i. e. Churn



```
#visualizing other columns according to the target variable
fig, axes = plt.subplots(2, 3, figsize=(14, 10))

sns.countplot("gender",hue='Churn', data=df, ax=axes[0,0]).set_title("Visualization of Gender")
sns.countplot("SeniorCitizen",hue='Churn', data=df, ax=axes[0,1]).set_title("Visualization of SeniorCitizen")
sns.countplot("Partner",hue='Churn', data=df, ax=axes[0,2]).set_title("Visualization of Partner")
sns.countplot("Dependents",hue='Churn', data=df, ax=axes[1,0]).set_title("Visualization of Dependents")
sns.countplot("PhoneService",hue='Churn', data=df, ax=axes[1,1]).set_title("Visualization of PhoneService")
sns.countplot("PaperlessBilling",hue='Churn', data=df, ax=axes[1,2]).set_title("Visualization of PaperlessBilling")
```

Figure 5.1.6 2:visualization of columns(1)

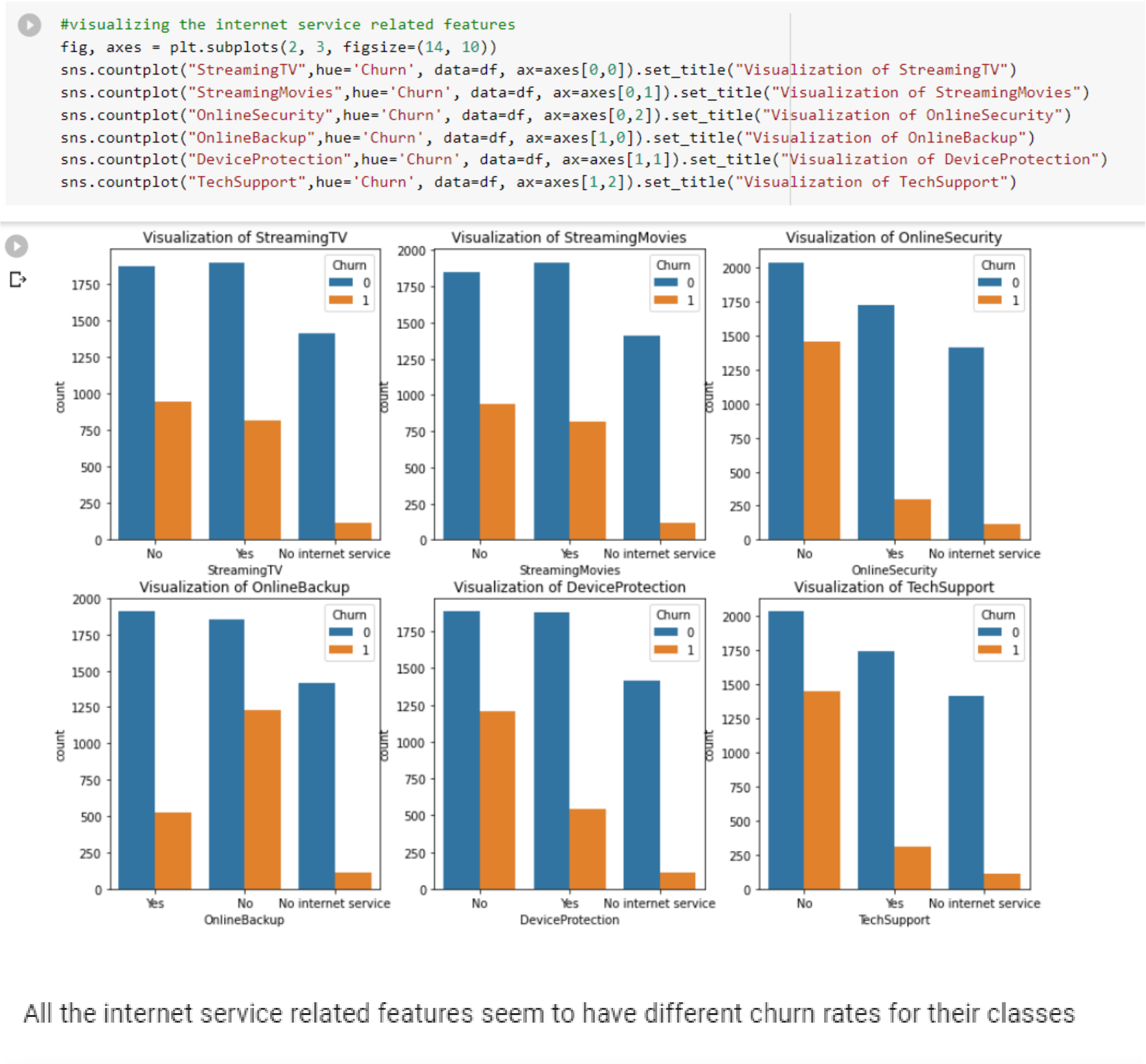


```
#Lets visualize the churn count for internet service
sns.countplot(x='InternetService',hue='Churn',data=df).set_title("Visualization of InternetSevice")
```

```
Text(0.5, 1.0, 'Visualization of InternetSevice')
```

We can see that the Internet Service column is definitely important in predicting the churn rate. customers with fiber optic internet service are much likely to churn than other customers although there is not a big difference in the number of customers with DSL and fiber optic

Figure 5.1.6 3:visualization of columns(2)

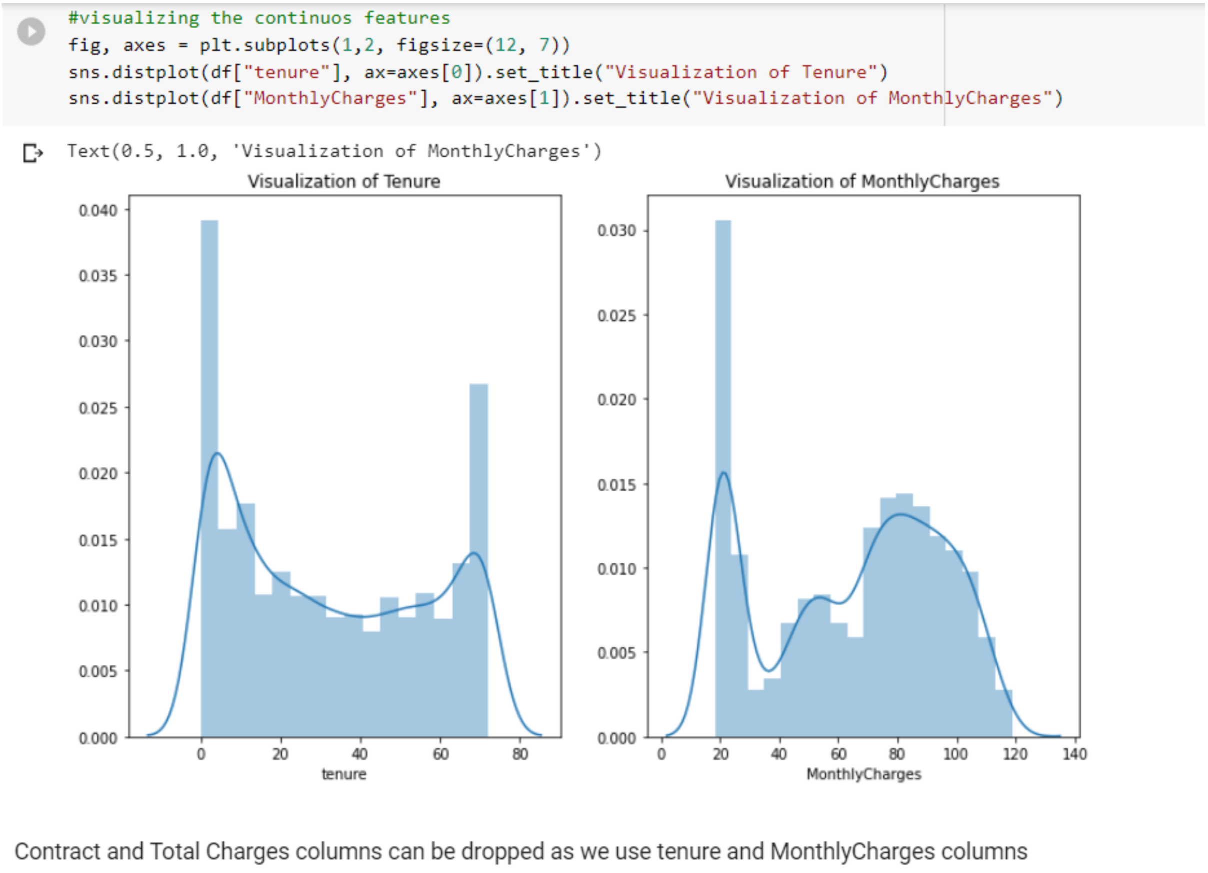


```
#visualizing the internet service related features
fig, axes = plt.subplots(2, 3, figsize=(14, 10))
sns.countplot("StreamingTV",hue='Churn', data=df, ax=axes[0,0]).set_title("Visualization of StreamingTV")
sns.countplot("StreamingMovies",hue='Churn', data=df, ax=axes[0,1]).set_title("Visualization of StreamingMovies")
sns.countplot("OnlineSecurity",hue='Churn', data=df, ax=axes[0,2]).set_title("Visualization of OnlineSecurity")
sns.countplot("OnlineBackup",hue='Churn', data=df, ax=axes[1,0]).set_title("Visualization of OnlineBackup")
sns.countplot("DeviceProtection",hue='Churn', data=df, ax=axes[1,1]).set_title("Visualization of DeviceProtection")
sns.countplot("TechSupport",hue='Churn', data=df, ax=axes[1,2]).set_title("Visualization of TechSupport")
```

Visualization of StreamingTV
Visualization of StreamingMovies
Visualization of OnlineSecurity
Visualization of OnlineBackup
Visualization of DeviceProtection
Visualization of TechSupport
Churn
0
1
count
0
250
500
750
1000
1250
1500
1750
2000
No
Yes
No internet service
StreamingTV
StreamingMovies
OnlineSecurity
OnlineBackup
DeviceProtection
TechSupport

All the internet service related features seem to have different churn rates for their classes

Figure 5.1.6 4:visualization of columns(3)



```
#visualizing the continuos features
fig, axes = plt.subplots(1,2, figsize=(12, 7))
sns.distplot(df["tenure"], ax=axes[0]).set_title("Visualization of Tenure")
sns.distplot(df["MonthlyCharges"], ax=axes[1]).set_title("Visualization of MonthlyCharges")
```

```
Text(0.5, 1.0, 'Visualization of MonthlyCharges')
```

Contract and Total Charges columns can be dropped as we use tenure and MonthlyCharges columns

Figure 5.1.6 5:visualization of columns(4)

5.1.7 ENCODING CATEGORICAL DATA:

● Machine Learning models are based on equations, we need to replace the text by numbers. So that we can include the numbers in the equations.

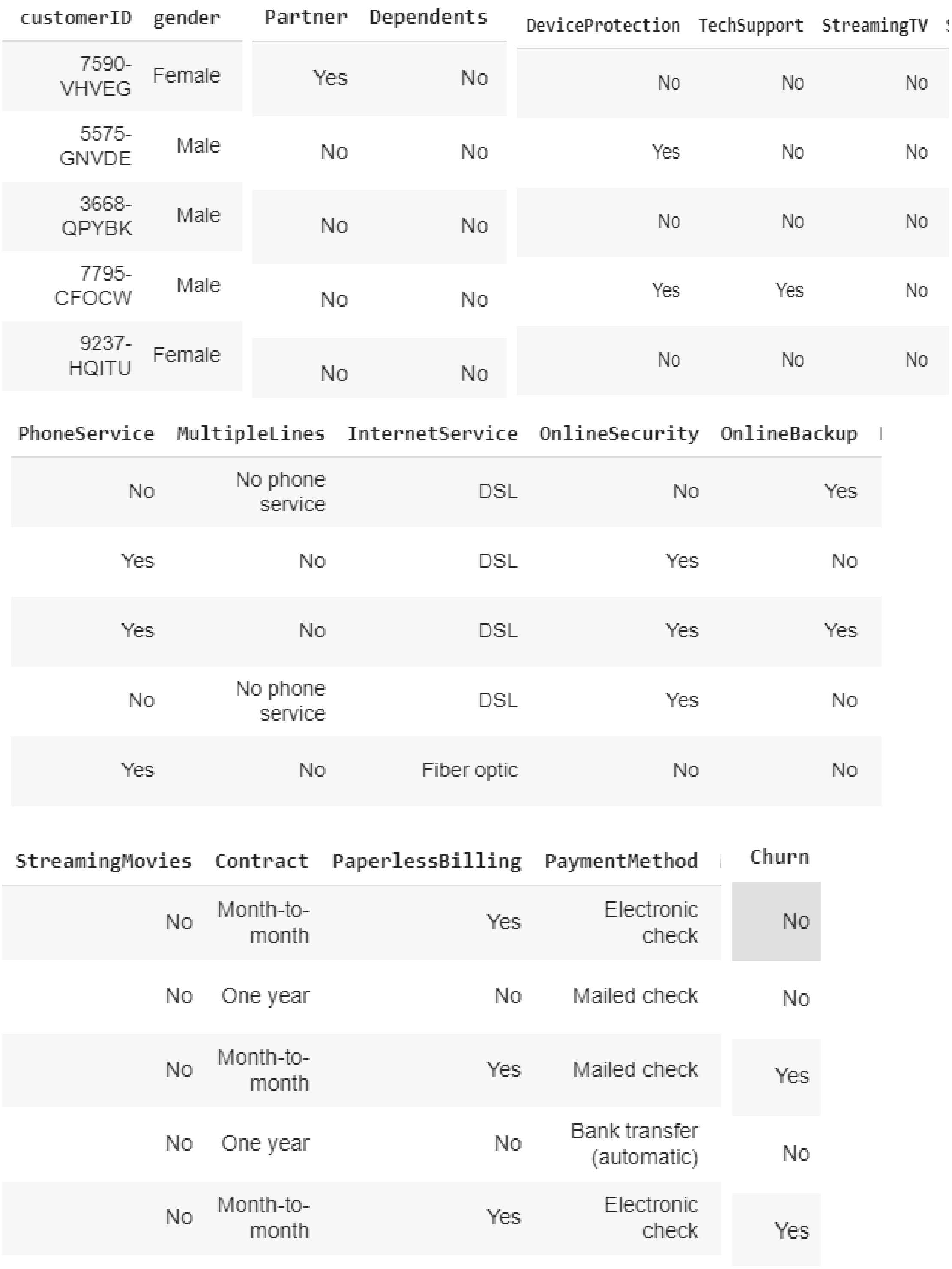
● Categorical Variables are of two types: Nominal and Ordinal

● Nominal: The categories do not have any numeric ordering in between them.They don't have any ordered relationship between each of them.

Examples: Male or Female, any colour

● Ordinal: The categories have a numerical ordering in between them.

Example: Graduate is less than Post Graduate, Post Graduate is less than Ph.D. customer satisfaction survey, high low medium.



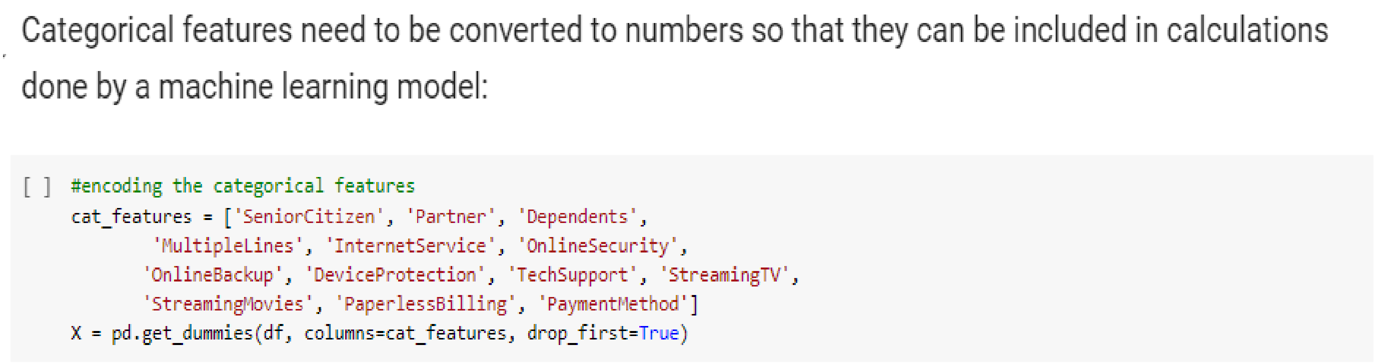
customerID	gender	Partner	Dependents	DeviceProtection	TechSupport	StreamingTV
7590-VHVEG	Female	Yes	No	No	No	No
5575-GNVDE	Male	No	No	Yes	No	No
3668-QPYBK	Male	No	No	No	No	No
7795-CFOCW	Male	No	No	Yes	Yes	No
9237-HQITU	Female	No	No	No	No	No

PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup
No	No phone service	DSL	No	Yes
Yes	No	DSL	Yes	No
Yes	No	DSL	Yes	Yes
No	No phone service	DSL	Yes	No
Yes	No	Fiber optic	No	No

StreamingMovies	Contract	PaperlessBilling	PaymentMethod	Churn
No	Month-to-month	Yes	Electronic check	No
No	One year	No	Mailed check	No
No	Month-to-month	Yes	Mailed check	Yes
No	One year	No	Bank transfer (automatic)	No
No	Month-to-month	Yes	Electronic check	Yes

Figure 5.1.7 1:Categorical data

I used getdummies method to encode categorical variables in my dataset



Categorical features need to be converted to numbers so that they can be included in calculations done by a machine learning model:

```
[ ]  #encoding the categorical features
     cat_features = ['SeniorCitizen', 'Partner', 'Dependents',
             'MultipleLines', 'InternetService', 'OnlineSecurity',
            'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
            'StreamingMovies', 'PaperlessBilling', 'PaymentMethod']
     X = pd.get_dummies(df, columns=cat_features, drop_first=True)
```

Figure 5.1.7 2:encoding categorical variables

CHAPTER 6

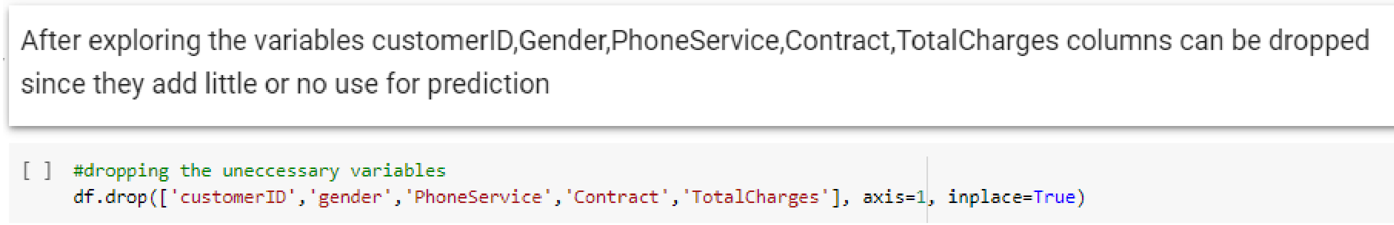
FEATURE SELECTION

6.1 Select relevant features for the analysis:

The uneccessary features in my dataset are Customerid, gender, phoneservice, totalcharges, contract.

These features are considered does not effect my target column.Rest of the columns are all relevant to the target column.

6.2 Drop irrelevant features:



After exploring the variables customerID,Gender,PhoneService,Contract,TotalCharges columns can be dropped since they add little or no use for prediction

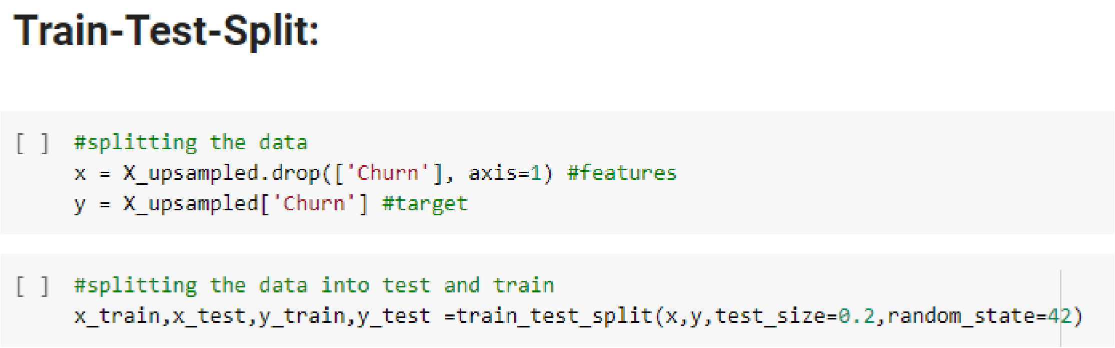
```
#dropping the uneccessary variables
df.drop(['customerID','gender','PhoneService','Contract','TotalCharges'], axis=1, inplace=True)
```

Figure 6.2 1:dropping irrelevant variables

6.3 TRAIN-TEST SPLIT:

- 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 testset 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 mode 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
- Here we have taken the test size as 0.2 indicating train data =80% and testdata=20%



Train-Test-Split:

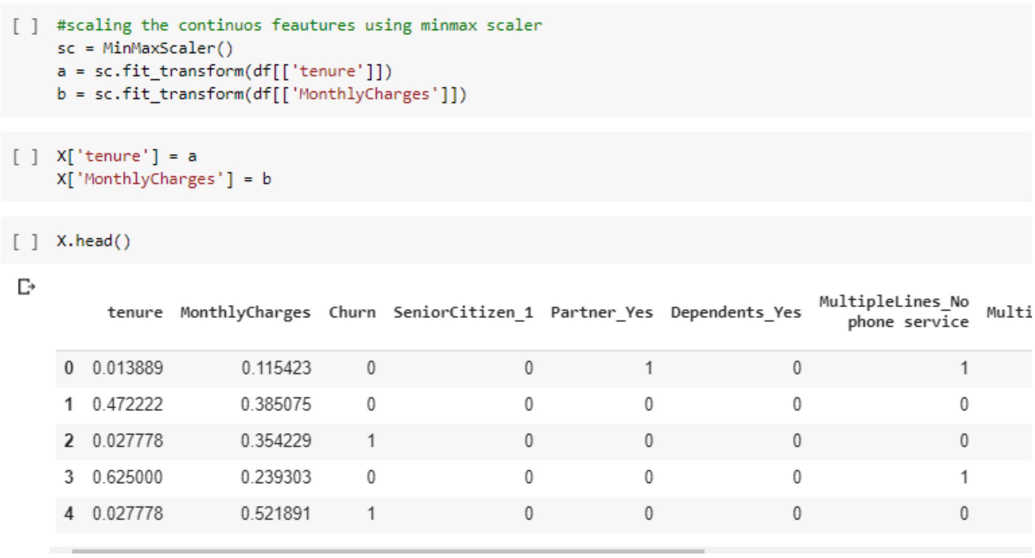
```
[ ]  #splitting the data
     x = X_upsampled.drop(['Churn'], axis=1) #features
     y = X_upsampled['Churn'] #target
```

```
[ ]  #splitting the data into test and train
     x_train,x_test,y_train,y_test =train_test_split(x,y,test_size=0.2,random_state=42)
```

Figure 6.3 1: importing train_test_split

6.4 Feature Scaling:

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values



```
[ ]  #scaling the continuos feautures using minmax scaler
     sc = MinMaxScaler()
     a = sc.fit_transform(df[['tenure']])
     b = sc.fit_transform(df[['MonthlyCharges']])
```

```
[ ]  X['tenure'] = a
     X['MonthlyCharges'] = b
```

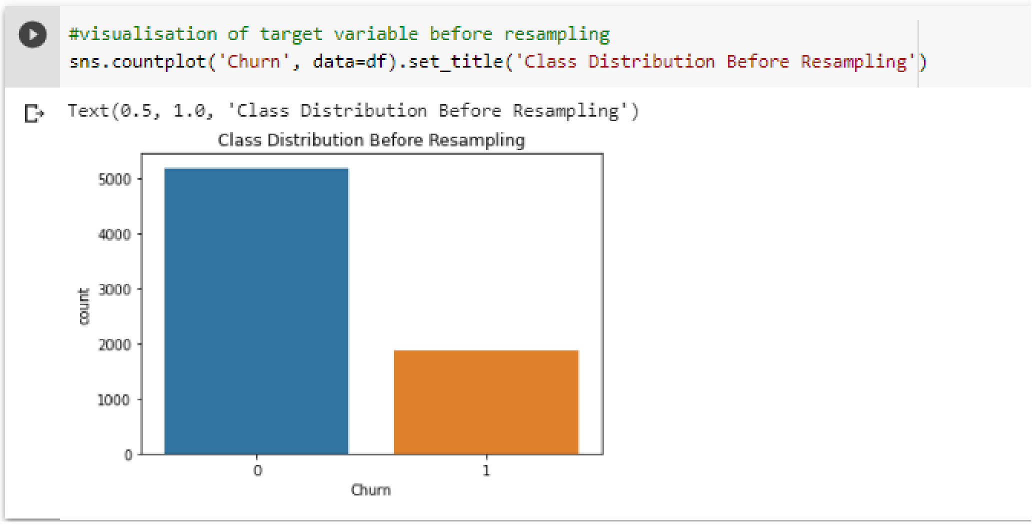
```
[ ]  X.head()
```

	tenure	MonthlyCharges	Churn	SeniorCitizen_1	Partner_Yes	Dependents_Yes	MultipleLines_No phone service	Multi
0	0.013889	0.115423	0	0	1	0	1	
1	0.472222	0.385075	0	0	0	0	0	
2	0.027778	0.354229	1	0	0	0	0	
3	0.625000	0.239303	0	0	0	0	1	
4	0.027778	0.521891	1	0	0	0	0	

Figure 6.4 1:Scaling

6.5 RESAMPLING

Upsampling or downsampling should be done to the data if the target variable has an imbalanced class distribution.Our data needs to be upsampled orelse it will affect the machine learning model negatively.

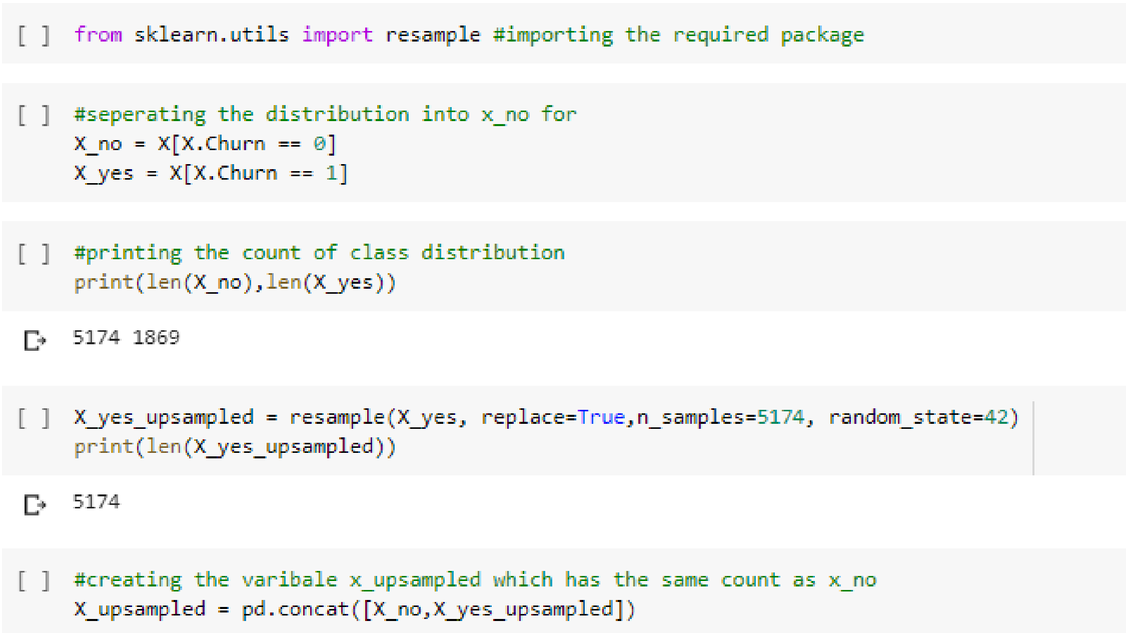


```
#visualisation of target variable before resampling
sns.countplot('Churn', data=df).set_title('Class Distribution Before Resampling')
```

```
Text(0.5, 1.0, 'Class Distribution Before Resampling')
```

Figure 6.5 1:class distribution before upsampling

Up-sampling is the process of randomly duplicating observations from the minority class in order to reinforce its signal. There are several heuristics for doing so, but the most common way is to simply resample with replacement.



```
[ ]  from sklearn.utils import resample #importing the required package
```

```
[ ]  #seperating the distribution into x_no for
     X_no = X[X.Churn == 0]
     X_yes = X[X.Churn == 1]
```

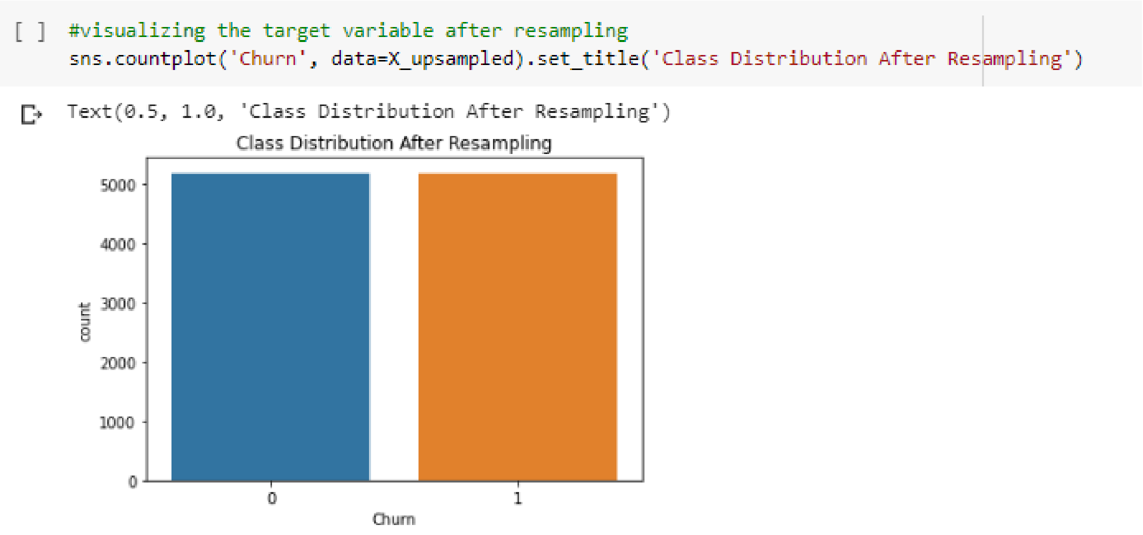
```
[ ]  #printing the count of class distribution
     print(len(X_no),len(X_yes))
```

```
5174 1869
```

```
[ ]  X_yes_upsampled = resample(X_yes, replace=True,n_samples=5174, random_state=42)
     print(len(X_yes_upsampled))
```

```
5174
```

```
[ ]  #creating the varibale x_upsampled which has the same count as x_no
     X_upsampled = pd.concat([X_no,X_yes_upsampled])
```



```
[ ]  #visualizing the target variable after resampling
     sns.countplot('Churn', data=X_upsampled).set_title('Class Distribution After Resampling')
```

```
Text(0.5, 1.0, 'Class Distribution After Resampling')
```

Figure 6.5 2:class distribution after upsampling

CHAPTER 7

MODEL BUILDING AND EVALUATION

In our project,we are going to use 3 different algorithms to train and predict the data.Then,we find out the optimum algorithm that best fits our data and gives maximum accuracy.

The three algorithms which we will be using are:

1. LOGISTIC REGRESSION ALGORITHM
2. K-NEAREST NEIGHBORS CLASSIFICATION ALGORITHM
3. RANDOM FOREST CLASSIFICATION ALGORITHM

7.1 LOGISTIC REGRESSION:

7.1.1 BRIEF ABOUT THE ALGORITHM

Logistic regression is another technique of machine learning from the field of statistics.It is the go-to method for binary classification problems (problems with two class values).Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

There are three main types of logistic regression:

1. **binomial:** target variable can have only 2 possible types: “0” or “1” which may represent “win” vs “loss”, “pass” vs “fail”, “dead” vs “alive”, etc.
2. **multinomial:** target variable can have 3 or more possible types which are not ordered(i.e. types have no quantitative significance) like “disease A” vs “disease B” vs “disease C”.
3. **ordinal:** it deals with target variables with ordered categories. For example, a test score can be categorized as:“very poor”, “poor”, “good”, “very good”. Here, each category can be given a score like 0, 1, 2, 3.

Parameters:

o **Penalty**:{‘l1’, ‘l2’, ‘elasticnet’, ‘none’}, default=’l2’

Used to specify the norm used in the penalization. The ‘newton-cg’, ‘sag’ and ‘lbfgs’ solvers support only l2 penalties. ‘elasticnet’ is only supported by the ‘saga’ solver. If ‘none’ (not supported by the liblinear solver), no regularization is applied.New in version 0.19: l1 penalty with SAGA solver (allowing ‘multinomial’ + L1)

o **Dual**:bool, default=False

□ Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer dual=False when n_samples > n_features.

o **Tol**:float, default=1e-4

Tolerance for stopping criteria.

o **C:**float, default=1.0

□ Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

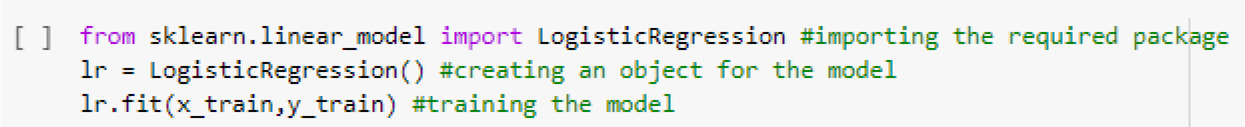
7.1.2 TRAIN THE MODEL

First we import the ‘LogisticRegression’ method from the ‘sklearn.linear_model’ package .

Then we create an object called LogisticRegression() and store it in a variable called ‘lr’

Next we apply the algorithm to the data and fit the data using the syntax:

objectname.fit(input,output)

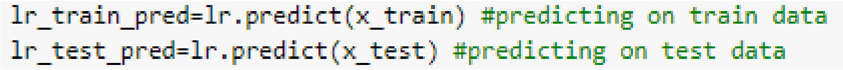


```
[ ]  from sklearn.linear_model import LogisticRegression #importing the required package
     lr = LogisticRegression() #creating an object for the model
     lr.fit(x_train,y_train) #training the model
```

Figure 7.1.2 1:Importing Logistic Regression Method and fitting the model

7.1.3 MAKE PREDICTIONS

Next , we need to predict the data for both training data and testing data and compare the actual value(y_train and y_test) with the model predicted values.

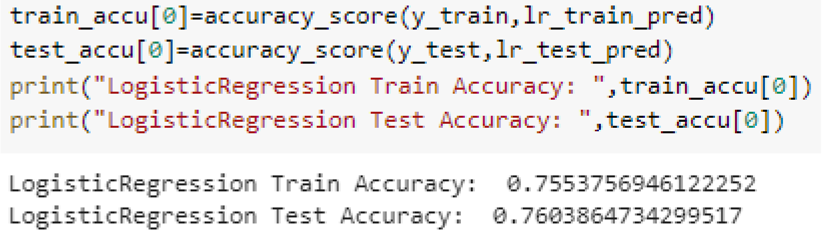


```
lr_train_pred=lr.predict(x_train) #predicting on train data
lr_test_pred=lr.predict(x_test) #predicting on test data
```

Figure 7.1.3 1:Prediction on train and test data(Logistic Regression)

7.1.4 PREDICTIONS FROM RAW DATA

My model has achieved an accuracy score of 75% for train data and 76% for testing data as input



```
train_accu[0]=accuracy_score(y_train,lr_train_pred)
test_accu[0]=accuracy_score(y_test,lr_test_pred)
print("LogisticRegression Train Accuracy: ",train_accu[0])
print("LogisticRegression Test Accuracy: ",test_accu[0])
```

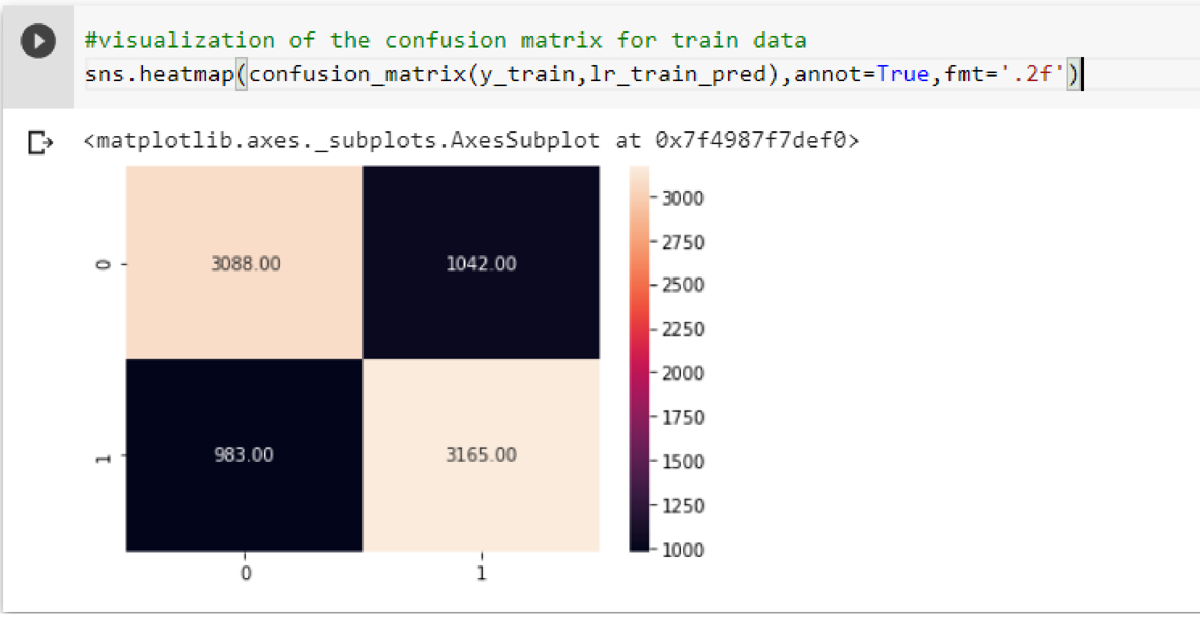
```
LogisticRegression Train Accuracy:  0.7553756946122252
LogisticRegression Test Accuracy:  0.7603864734299517
```

Figure 7.1.4 1:train and test accuracies

For Train Data:

To compare the actual values of y_train with the predicted values of lr_train_pred we use a confusion matrix .

The confusion matrix is imported from sklearn.metrics package.

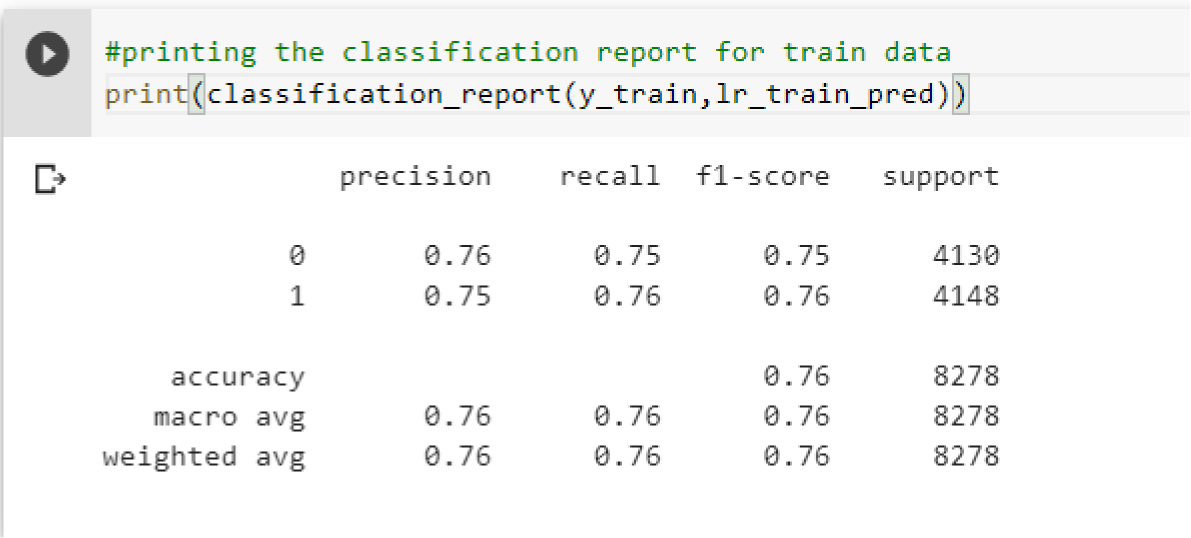


```
#visualization of the confusion matrix for train data
sns.heatmap(confusion_matrix(y_train,lr_train_pred),annot=True,fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4987f7def0>
```

Figure 7.1.4 2:Confusion matrix for train data(Logistic Regression)

The classification report for train data is printed as:



```
#printing the classification report for train data
print(classification_report(y_train,lr_train_pred))
```

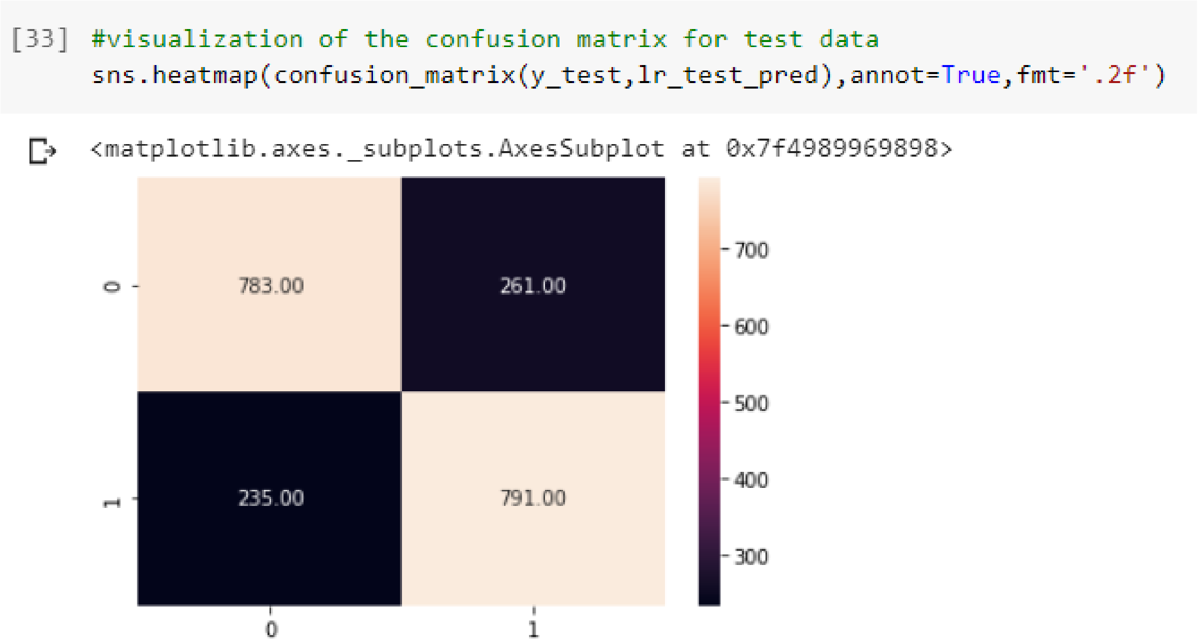
```
              precision    recall  f1-score   support

           0       0.76      0.75      0.75      4130
           1       0.75      0.76      0.76      4148

    accuracy                           0.76      8278
   macro avg       0.76      0.76      0.76      8278
weighted avg       0.76      0.76      0.76      8278
```

Figure 7.1.4 3:Classification report of train data(Logistic Regression)

We can conclude that the accuracy score of Logistic Regression on train data is **0.75**

For Test Data:

The Confusion matrix for predictions against test data

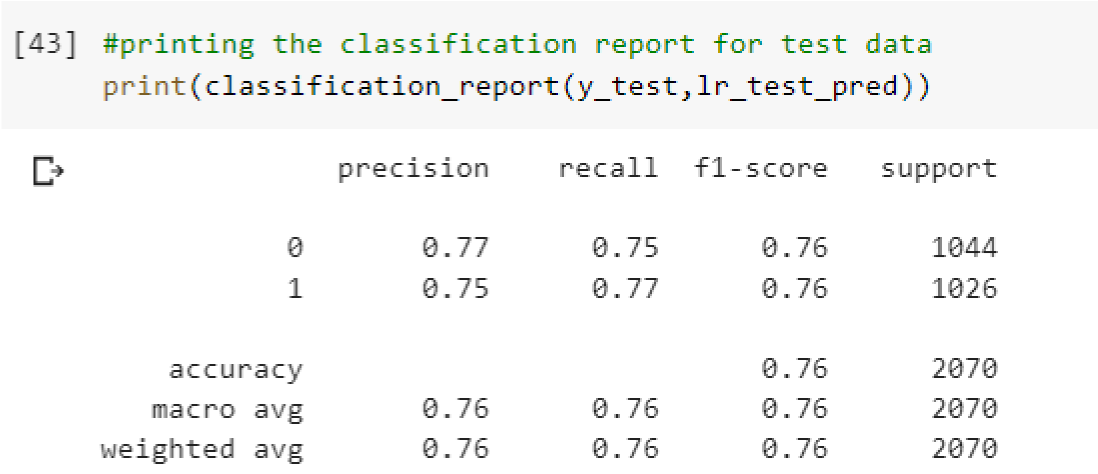


```
[33] #visualization of the confusion matrix for test data
     sns.heatmap(confusion_matrix(y_test,lr_test_pred),annot=True,fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4989969898>
```

Figure 7.1.4 4:Confusion matrix for test data(LinearRegression)

The Classification report for test data:



```
[43] #printing the classification report for test data
     print(classification_report(y_test,lr_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.75      0.76      1044
           1       0.75      0.77      0.76      1026

    accuracy                           0.76      2070
   macro avg       0.76      0.76      0.76      2070
weighted avg       0.76      0.76      0.76      2070
```

Figure 7.1.4 5:classification report for test data(Linear Regression)

We can conclude that the accuracy score of Logistic Regression on train data is **0.76**

7.2 RANDOM FOREST CLASSIFICATION:

7.2.1 BRIEF ABOUT THE ALGORITHM

The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.The random forest algorithm is a supervised learning model; it uses labeled data to “learn” how to classify unlabeled .

Random forests has a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.

How does the algorithm work?

It works in four steps:

1. Select random samples from a given dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Select the prediction result with the most votes as the final prediction.

Parameters:

n_estimators:int, default=100

The number of trees in the forest.

Criterion:{“gini”, “entropy”}, default=”gini”

The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Note: this parameter is tree-specific.

max_depth:int, default=None

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split:int or float, default=2

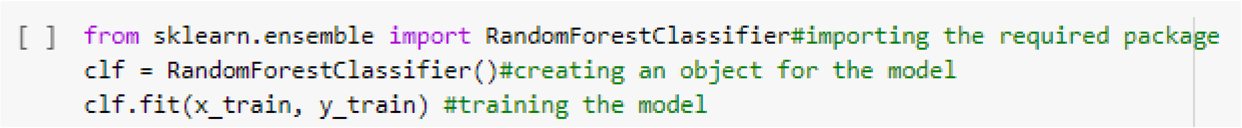
The minimum number of samples required to split an internal node

7.2.2 TRAIN THE MODEL

First we import the ‘RandomForestClassifier’ method from the ‘sklearn.ensemble’ package Then we create an object called RandomForestClassifier() and store it in a variable called ‘clf’

Next we apply the algorithm to the data and fit the data using the syntax

objectname.fit(input,output).



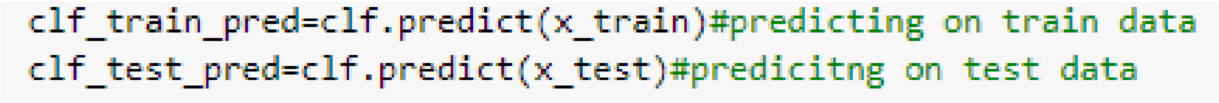
```
from sklearn.ensemble import RandomForestClassifier#importing the required package
clf = RandomForestClassifier()#creating an object for the model
clf.fit(x_train, y_train) #training the model
```

Figure 7.2.2 1:Importing Random Forest Classifier Method and fitting the model

7.2.3 MAKE PREDICTIONS

Next , we need to predict the data for both training data and testing data and compare the actual value(y_train and y_test) with the model predicted values.

For prediction on train and testdata:

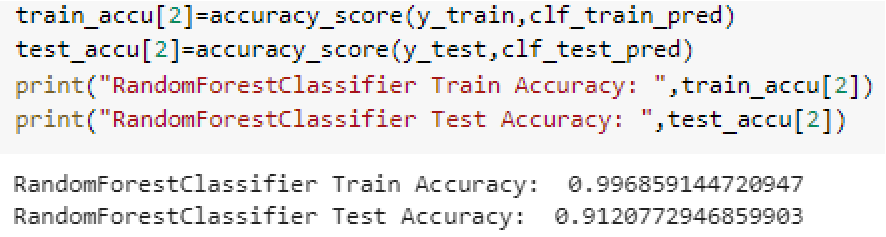


```
clf_train_pred=clf.predict(x_train)#predicting on train data
clf_test_pred=clf.predict(x_test)#predicitng on test data
```

Figure 7.2.3 1:Prediction on train data and test data (Random Forest Classifier)

7.2.4 PREDICTIONS FROM RAW DATA

My model has achieved an accuracy score of 99% for train data and 91% for testing data as input



```
train_accu[2]=accuracy_score(y_train,clf_train_pred)
test_accu[2]=accuracy_score(y_test,clf_test_pred)
print("RandomForestClassifier Train Accuracy: ",train_accu[2])
print("RandomForestClassifier Test Accuracy: ",test_accu[2])
```

```
RandomForestClassifier Train Accuracy:  0.996859144720947
RandomForestClassifier Test Accuracy:  0.9120772946859903
```

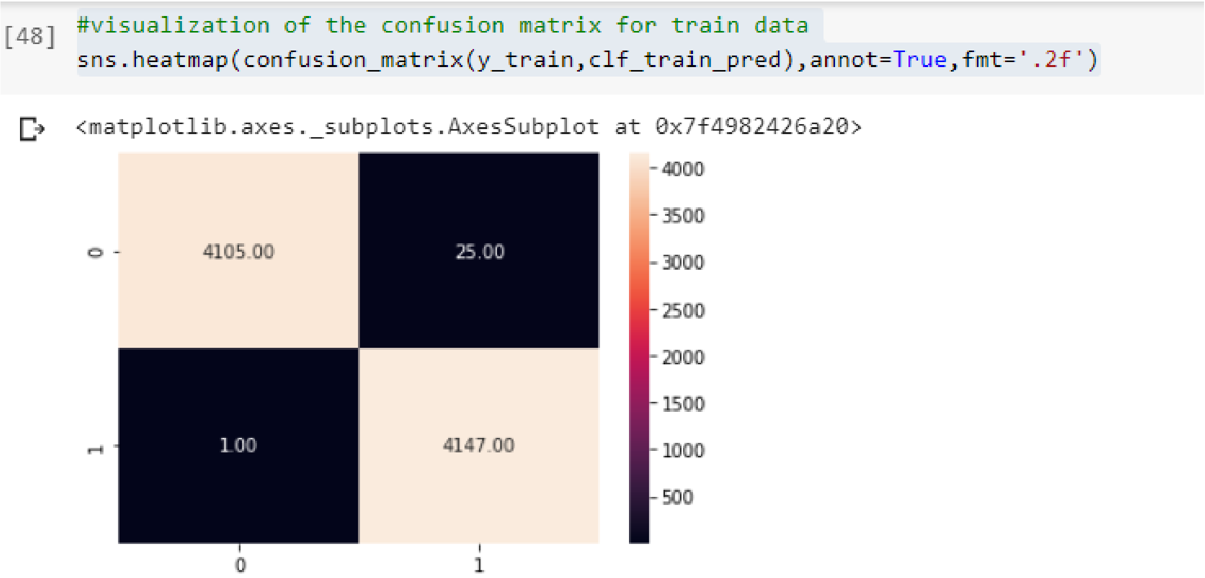
Figure 7.2.4 1:Accuracy for train data and test data(Random Forest Classification)

For Train Data:

To compare the actual values of y_train with the predicted values of clf_train_pred we use a confusion matrix .

The confusion matrix is imported from sklearn.metrics package.

To visualize the confusion matrix , we use heatmap

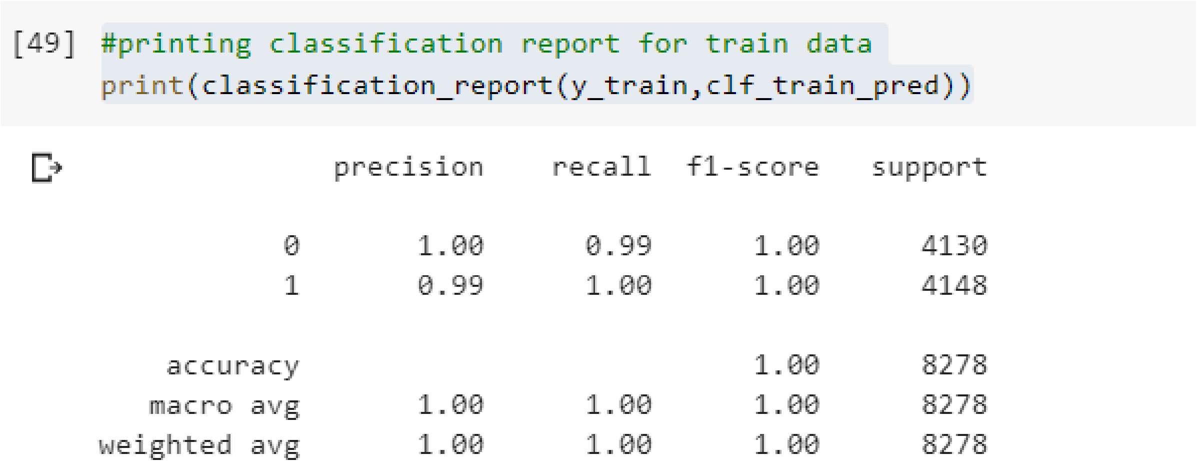


```
[48] #visualization of the confusion matrix for train data
     sns.heatmap(confusion_matrix(y_train,clf_train_pred),annot=True,fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4982426a20>
```

Figure 7.2.4 2Confusion matrix for train data(Random Forest Classifier)

The classification report for train data is printed as:



```
[49] #printing classification report for train data
     print(classification_report(y_train,clf_train_pred))
```

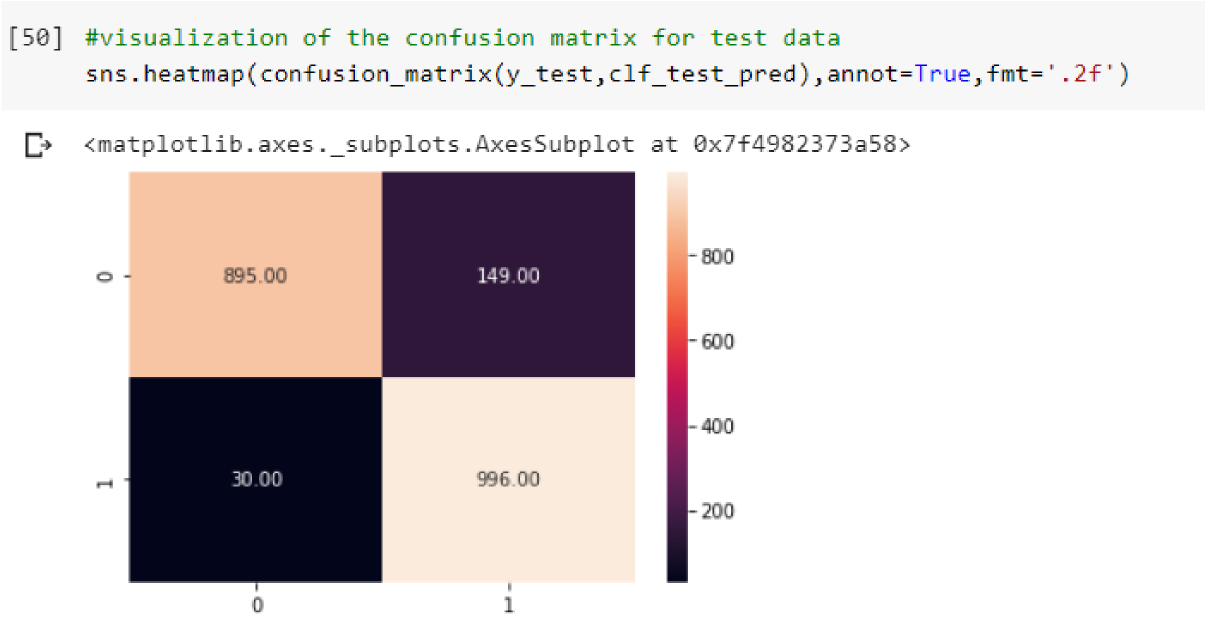
```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      4130
           1       0.99      1.00      1.00      4148

    accuracy                           1.00      8278
   macro avg       1.00      1.00      1.00      8278
weighted avg       1.00      1.00      1.00      8278
```

Figure 7.2.4 3classification report for train data(Random Forest Classification)

We can conclude that the accuracy score of Random Forest Classifier on train data is **0.99**

For Test Data:

The Confusion matrix for predictions against test data

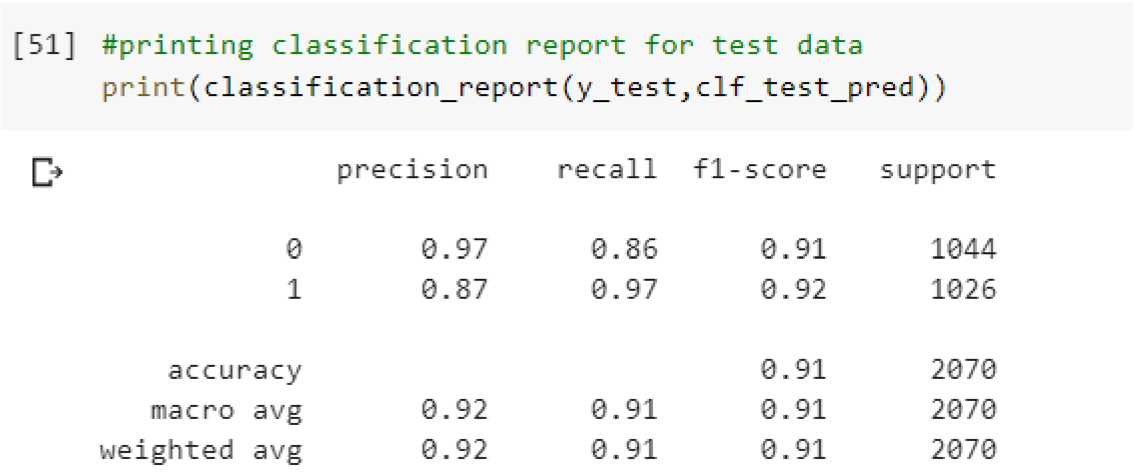


```
[50] #visualization of the confusion matrix for test data
     sns.heatmap(confusion_matrix(y_test,clf_test_pred),annot=True,fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4982373a58>
```

Figure 7.2.4 4:Confusion matrix for test data(Random Forest Classifier)

The classification report for test data



```
[51] #printing classification report for test data
     print(classification_report(y_test,clf_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.86      0.91      1044
           1       0.87      0.97      0.92      1026

    accuracy                           0.91      2070
   macro avg       0.92      0.91      0.91      2070
weighted avg       0.92      0.91      0.91      2070
```

Figure 7.2.4 5:classification report for test data(Random Forest Classification)

We can conclude that the accuracy score of Random Forest Classifier on train data is **0.91**

7.3 K-NEAREST NEIGHBORS CLASSIFIER

7.3.1 BRIEF ABOUT THE ALGORITHM

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well −

- **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.

- **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it doesn’t assume anything about the underlying data.

As we know K-nearest neighbors (KNN) algorithm can be used for both classification as well as regression

K-nearest neighbors (KNN) algorithm uses ‘feature similarity’ to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps –

1. For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

2. Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

3.For each point in the test data do the following −

- **3.1** − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.

- **3.2** − Now, based on the distance value, sort them in ascending order

- **3.3** − Next, it will choose the top K rows from the sorted array.

- **3.4** − Now, it will assign a class to the test point based on most frequent class of these rows.

Parameters:

n_neighborsint, default=5

Number of neighbors to use by default for kneighbors queries.

weights{‘uniform’, ‘distance’} or callable, default=’uniform’

weight function used in prediction. Possible values:

‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.

‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

[callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm{‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’}, default=’auto’

Algorithm used to compute the nearest neighbors:

‘ball_tree’ will use BallTree

‘kd_tree’ will use KDTree

‘brute’ will use a brute-force search.

‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to fit method.

leaf_sizeint, default=30

Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

pint, default=2

Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metricstr or callable, default=’minkowski’

the distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric

metric_paramsdict, default=None

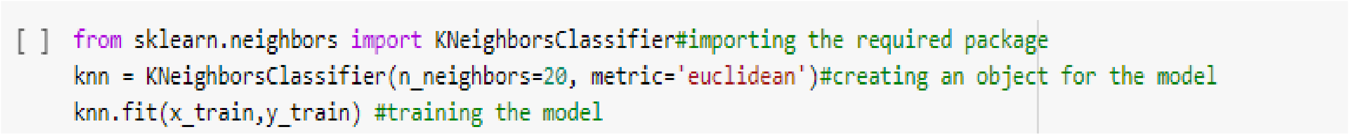
Additional keyword arguments for the metric function.

n_jobsint, default=None

The number of parallel jobs to run for neighbors search. -1 means using all processors.

7.3.2 TRAIN THE MODEL

First we import the ‘KNeighborsClassifier’ method from the ‘sklearn.neighbors’ package Then we create an object called KNeighborsClassifier() and store it in a variable called ‘knn’



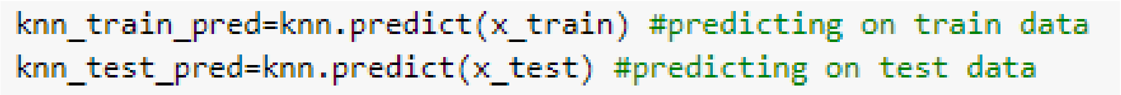
```
from sklearn.neighbors import KNeighborsClassifier#importing the required package
knn = KNeighborsClassifier(n_neighbors=20, metric='euclidean')#creating an object for the model
knn.fit(x_train,y_train) #training the model
```

Figure 7.3.2 1:Importing KNeighbors classifier Method and fitting the model

7.3.3 MAKE PREDICTIONS

Next , we need to predict the data for both training data and testing data and compare the actual value(y_train and y_test) with the model predicted values.

For prediction on train and testdata:

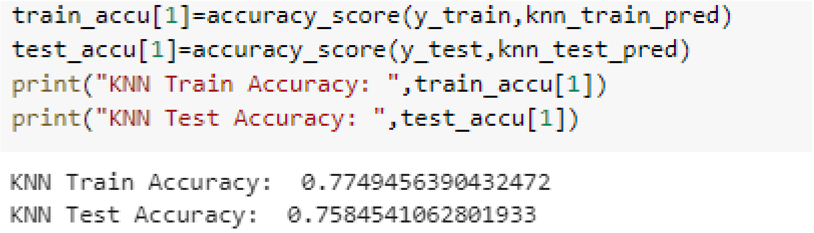


```
knn_train_pred=knn.predict(x_train) #predicting on train data
knn_test_pred=knn.predict(x_test) #predicting on test data
```

Figure 7.3.3 1:Prediction on train data and test data (KNeighbors Classifier)

7.3.4 PREDICTIONS FROM RAW DATA

My model has achieved an accuracy score of 77% for train data and 75% for testing data as input



```
train_accu[1]=accuracy_score(y_train,knn_train_pred)
test_accu[1]=accuracy_score(y_test,knn_test_pred)
print("KNN Train Accuracy: ",train_accu[1])
print("KNN Test Accuracy: ",test_accu[1])
```

```
KNN Train Accuracy:  0.7749456390432472
KNN Test Accuracy:  0.7584541062801933
```

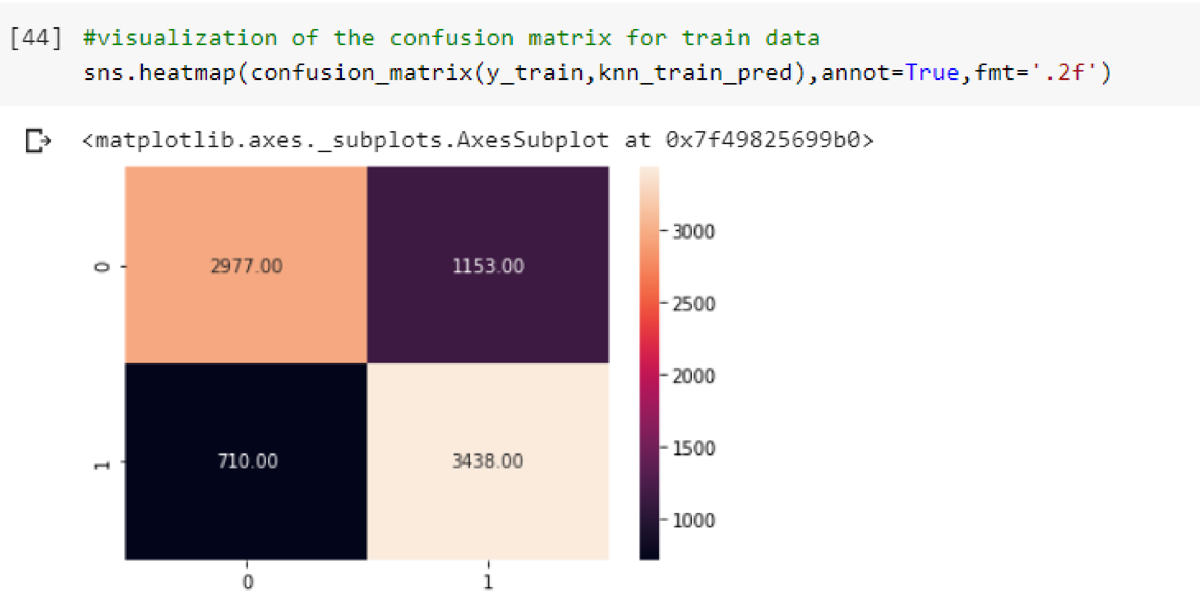
Figure 7.3.4 1:Accuracy for train data and test data(KNeighbors Classification)

For Train Data:

To compare the actual values of y_train with the predicted values of knn_train_pred we use a confusion matrix .

The confusion matrix is imported from sklearn.metrics package.

To visualize the confusion matrix , we use heatmap

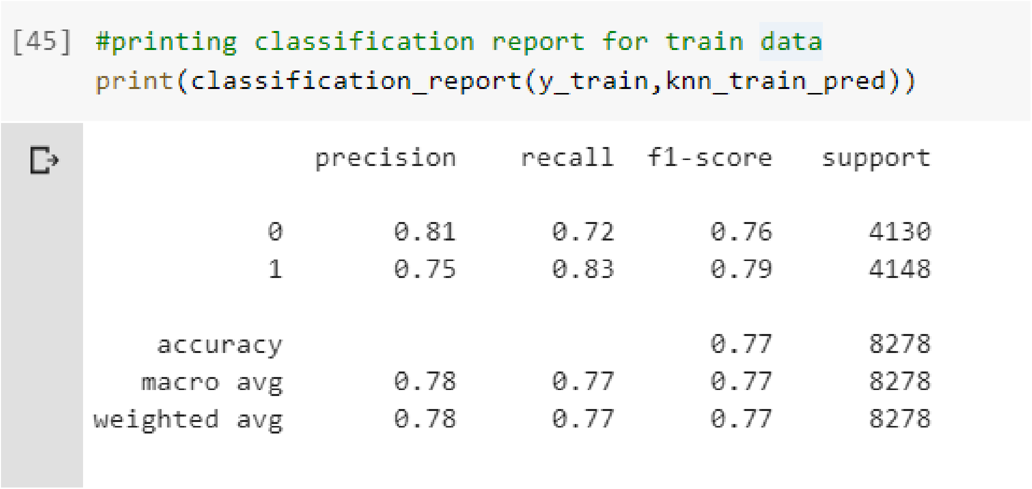


```
[44] #visualization of the confusion matrix for train data
     sns.heatmap(confusion_matrix(y_train,knn_train_pred),annot=True,fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49825699b0>
```

Figure 7.3.4 2:Confusion matrix for train data(KNeighbors Classifier)

The classification report for train data is printed as:



```
[45] #printing classification report for train data
     print(classification_report(y_train,knn_train_pred))
```

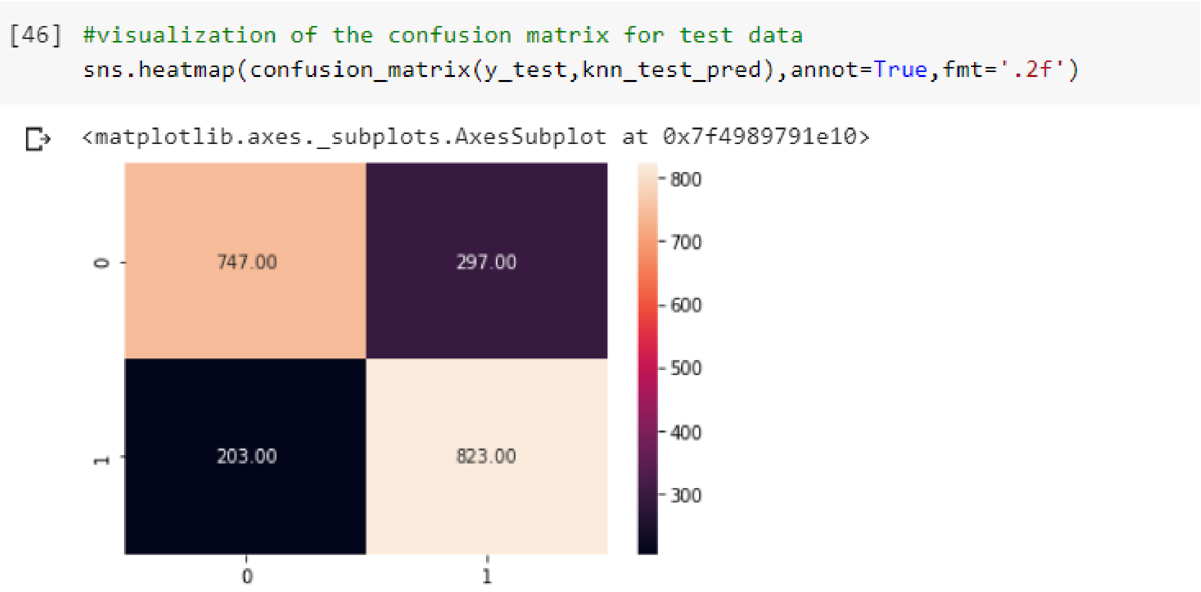
```
              precision    recall  f1-score   support

           0       0.81      0.72      0.76      4130
           1       0.75      0.83      0.79      4148

    accuracy                           0.77      8278
   macro avg       0.78      0.77      0.77      8278
weighted avg       0.78      0.77      0.77      8278
```

Figure 7.3.4 3:classification report for train data(KNeighbors Classification)

We can conclude that the accuracy score of K-Nearest Neighbors Classifier on train data is **0.77**

For Test Data:

The Confusion matrix for predictions against test data

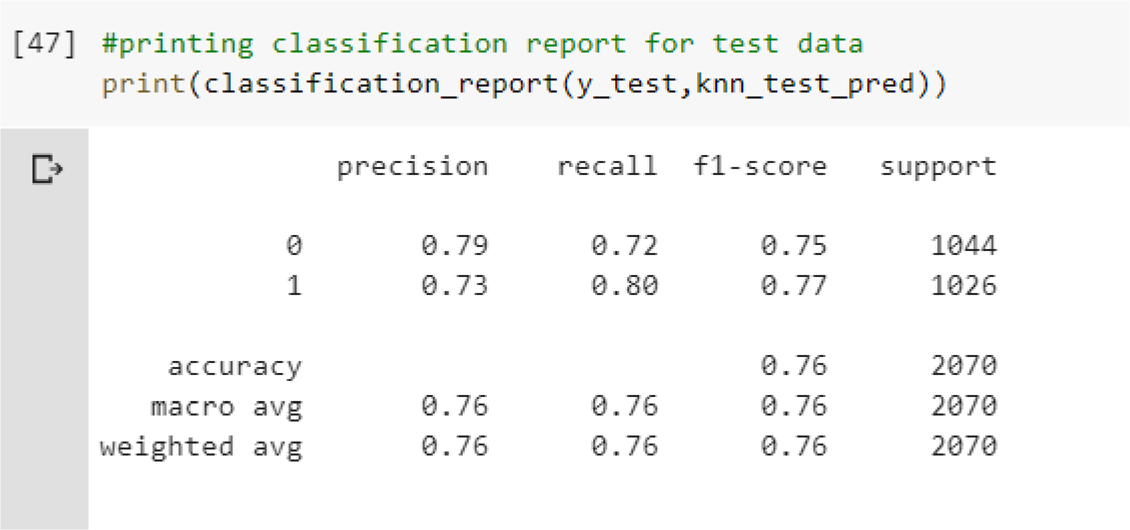


```
[46] #visualization of the confusion matrix for test data
     sns.heatmap(confusion_matrix(y_test,knn_test_pred),annot=True,fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4989791e10>
```

Figure 7.3.4 4:Confusion matrix for test data(KNeighbors Classifier)

The classification report for test data:



```
[47] #printing classification report for test data
     print(classification_report(y_test,knn_test_pred))
```

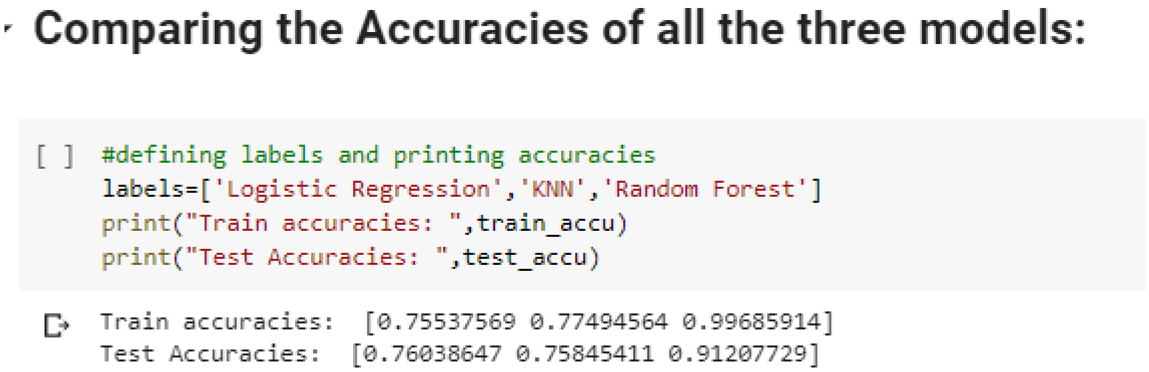
```
              precision    recall  f1-score   support

           0       0.79      0.72      0.75      1044
           1       0.73      0.80      0.77      1026

    accuracy                           0.76      2070
   macro avg       0.76      0.76      0.76      2070
weighted avg       0.76      0.76      0.76      2070
```

Figure 7.3.4 5:classification report for test data(KNeighbors Classification)

We can conclude that the accuracy score of K-Nearest Neighbors Classifier on test data is **0.75**

CHAPTER 8

COMPARING THE PERFORMANCE OF ALL THE MODELS



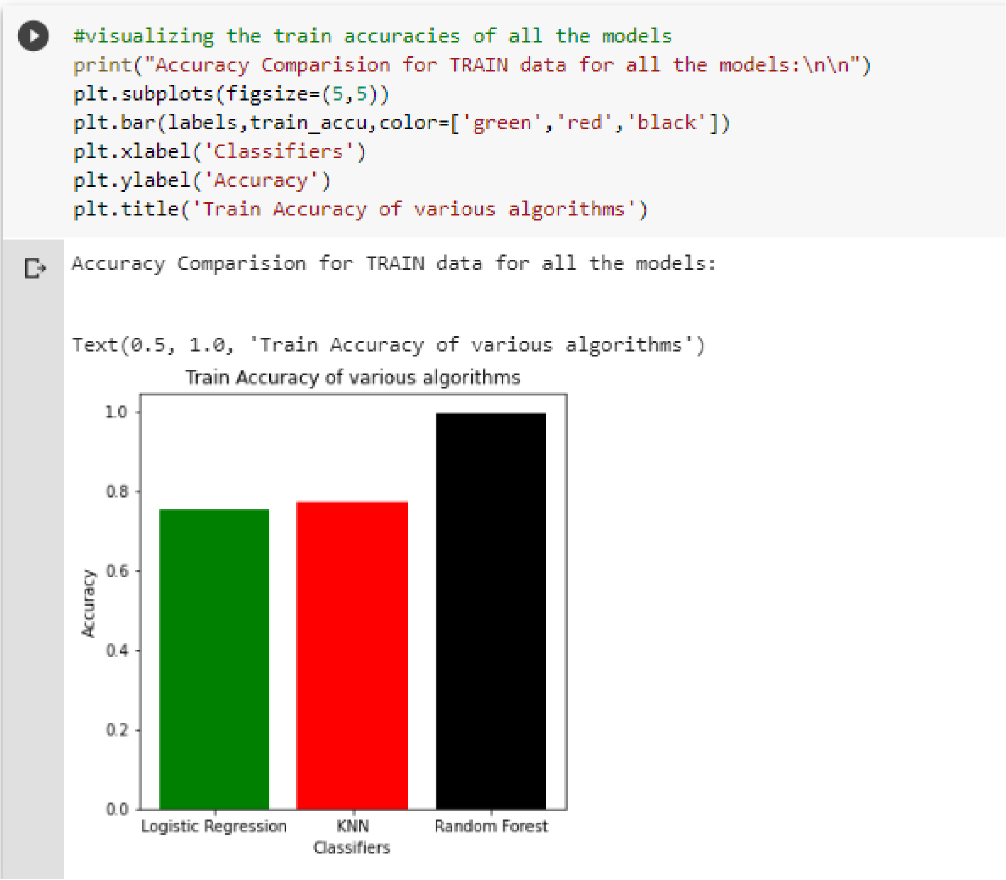
Comparing the Accuracies of all the three models:

```
[ ]  #defining labels and printing accuracies
     labels=['Logistic Regression','KNN','Random Forest']
     print("Train accuracies: ",train_accu)
     print("Test Accuracies: ",test_accu)
```

```
Train accuracies:  [0.75537569 0.77494564 0.99685914]
Test Accuracies:  [0.76038647 0.75845411 0.91207729]
```

Figure 8 1:Accuracies of all 3 models

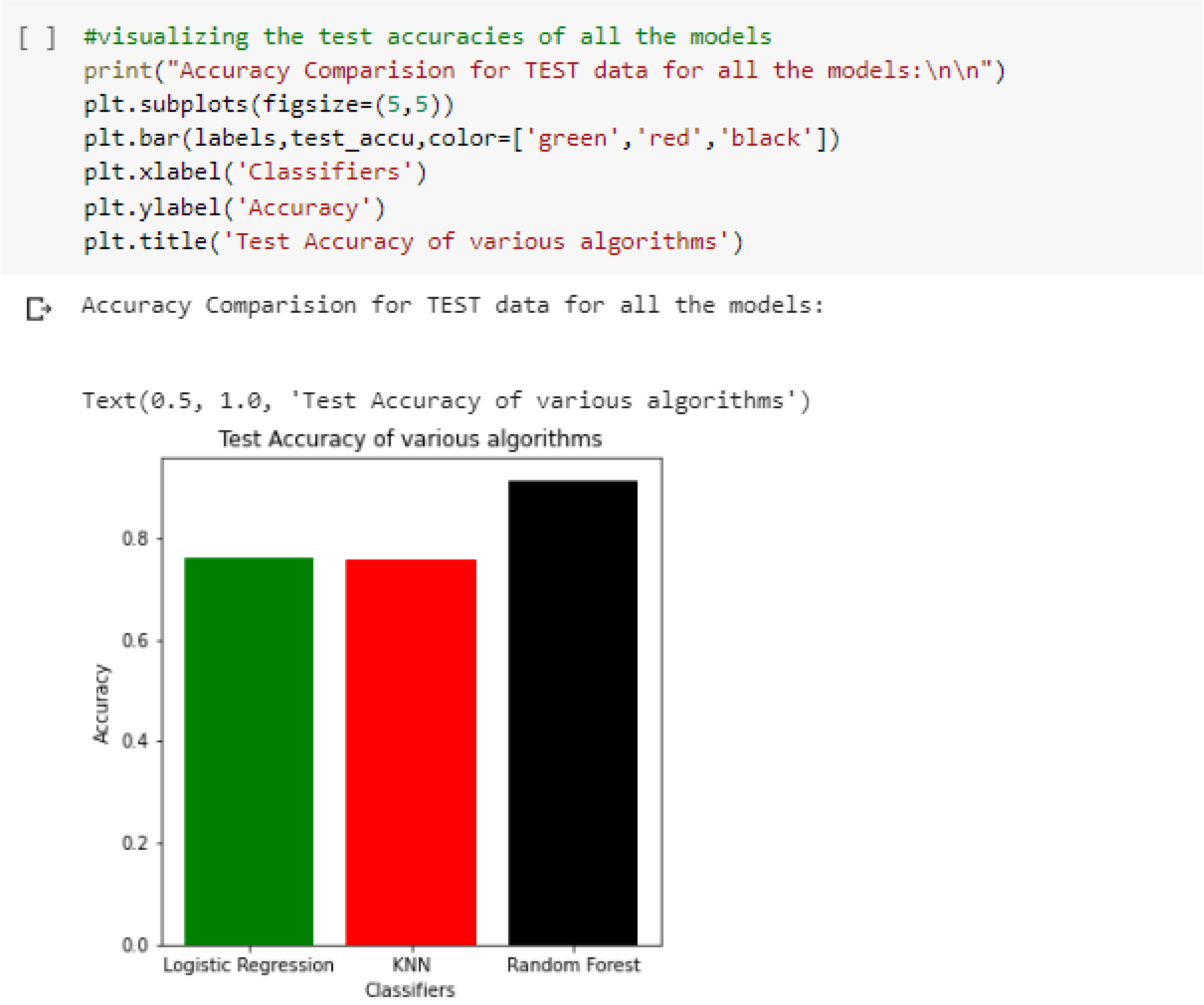
On comparing the accuracy score with respect to training data and testing data of all models it has been observed that Random Forest has better performance than Logistic Regression and KNN.



```
#visualizing the train accuracies of all the models
print("Accuracy Comparision for TRAIN data for all the models:\n\n")
plt.subplots(figsize=(5,5))
plt.bar(labels,train_accu,color=['green','red','black'])
plt.xlabel('Classifiers')
plt.ylabel('Accuracy')
plt.title('Train Accuracy of various algorithms')
```

```
Accuracy Comparision for TRAIN data for all the models:


Text(0.5, 1.0, 'Train Accuracy of various algorithms')
```

Figure 8 2:Visualizing the TrainAccuracies of all 3 models



```
[ ] #visualizing the test accuracies of all the models
    print("Accuracy Comparision for TEST data for all the models:\n\n")
    plt.subplots(figsize=(5,5))
    plt.bar(labels,test_accu,color=['green','red','black'])
    plt.xlabel('Classifiers')
    plt.ylabel('Accuracy')
    plt.title('Test Accuracy of various algorithms')
```

```
Accuracy Comparision for TEST data for all the models:


Text(0.5, 1.0, 'Test Accuracy of various algorithms')
```

Figure 8 3: Visualizing the Test Accuracies of all 3 models

CONCLUSION:

Hence this project is aimed to build a machine learning model that predicts the customer churn by use of other relevant features. To test and train the model, the sample data is divided into 80% for training and 30% for testing.Three algorithms were chosen for the prediction which are LogisticRegression,K-Nearest Neighbors classifier and RandomForestClassifier out of which RandomForestClassifier produced results with high accuracy.So,through this project I conclude that RandomForestClassifier works best in predicting the churn rate.

REFERENCES:

1.DATASET: https://www.kaggle.com/blastchar/telco-customer-churn

2. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

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

5.https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=random%20forest%20classifier#sklearn.ensemble.RandomForestClassifier