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

K. Naga Vennela

121710302022

Under the Guidance of

Assistant Professor

GITAM

(DEEMED TO BE UNIVERSITY)

VISAKHAPATNAM HYDERABAD BENGALURU

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

June 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 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 K. Naga Vennela

Date: 121710302022

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled **"CHURN PREDICTION"** is being submitted by K. Naga Vennela (121710302022) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer Science and Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2020-21.

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

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. ,** ProVice Chancellor, GITAM Hyderabad and **Dr. ,** Principal, GITAM Hyderabad.

I would like to thank respected **Dr. ,** Head of the Department of Computer Science and Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present an 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.

K. Naga Vennela
121710302022

ABSTRACT

In recent days, telecom industry plays a major role in our daily life. The proliferation of telecommunication industry becomes very difficult for the service providers to survive in the market. To stabilize in this field, the service providers have to be aware of the features that make the customer to churn. The proposed predictive model identifies the traits that highly influence customer churn, with the help of machine learning techniques.

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

The Orange Telecom's Churn dataset has been analyzed to forecast the churn .To get a better understanding and work on a strategical approach for solution of the customers, I have adapted the view point of looking at different attributes that indulge in predicting the churn and for further deep understanding of the problem, I have used the techniques like KNN, Random Forest and Decision Tree. At last a comparative study has been made among the machine learning algorithm to identify the better algorithm of higher accuracy.

Table of Contents:

Chapter 1: MACHINE LEARNING

1.1 INTRODUCTION…………………………………………………………………..11
1.2 IMPORTANCE OF MACHINE LEARNING…………………………………...11
1.3 USES OF MACHINE LEARNING……………………………………………...12
1.4 TYPES OF LEARNING ALGORITHMS……………………………...................13
1.4.1 Supervised Learning…………………………………………………13
1.4.2 Unsupervised Learning………………………………………………14
1.4.3 Semi Supervised Learning……………………………………………15
1.5 RELATION BETWEEN DATA MINING, MACHINE LEARNING AND DEEP LEARNING ……………………………………………………………….15

Chapter 2: PYTHON……..…………………………………………………………16

2.1 INTRODUCTOIN TO PYTHON………………………………………………..16
2.2 HISTORY OF PYTHON…………………………………………………………...16
2.3 FEATURES OF PYTHON………………………………………………………....17
2.4 HOW TO SETUP PYTHON……………………………………….....................17
2.4.1 Installation (using python IDLE)……………………………………....17
2.4.2 Installation (using Anaconda)…………………………………………18
2.5 PYTHON VARIABLE TYPES……………………………………….....................20
2.5.1 Python Numbers……………………………………………………..20
2.5.2 Python Strings…………………………………………………………21
2.5.3 Python Lists……………………………………………………………21
2.5.4 Python Tuples…………………………………………………………..22
2.5.5 Python Dictionary………………………………………….............22
2.6 PYTHON FUNCTION……………………………………………………………...23
2.6.1 Defining a Function……………………………………………………23
2.6.2 Calling a Function……………………………………………………...23
2.7 PYTHON USING OOP's CONCEPTS………………………………………………23
2.7.1 Class……………………………………………………………………….23
2.7.2 __init__method in class……………………………………….........24

Chapter 3: CASE STUDY..…………………………………...........................25
3.1 PROBLEM STATEMENT……………………………………………………………...25
3.2 DATA SET……………………………………………………………………………...25
3.3 OBJECTIVE OF THE CASE STUDY………………………………....................26

Chapter 4: MODEL BUILDING…..27
4.1 PREPROCESSING OF THE DATA……………………………….................27
4.1.1 Getting the Data Set…………………………………………........27
4.1.2 Importing the Libraries…………………………………….......27
4.1.3 Importing the dataset……………………………………...........28
4.1.4 Visualizing the columns……………………………………..........29
4.1.5 Handling Missing values……………………………………..........33
4.1.6 Categorical Data……………………………………………...........35
4.2 TRAINING THE MODEL………………………………………….................36
4.2.1 Splitting the data…………………………………………….........36
4.2.2 Feature scaling……………………………………………..........38
4.2.3 Classification Activities (Models)…………………………….........39
4.2.3.1 Model-1 (K-Nearest Neighbors)...……………………….......39
4.2.3.2 Model-2 (Decision Tree Classifier)………………….........45
4.2.3.3 Model-3 (Random-Forest Classifier)…………………......50
4.3 EVALUATING THE CASE STUDY…………………………………..................60

CONCLUSION…………………………………………………………......61

REFERENCES…………………………………………………………......61

LIST OF FIGURES

Fig-1: **The Process Flow**……………………………………………………….12

Fig-2: **Supervised Learning**…………………………………………………...13

Fig-3: **Unsupervised Learning**……………………………………….........14

Fig-4: **Semi-Supervised Learning**…………………………………………….15

Fig-5: **Python Download**……………………………………………………18

Fig-6: **Anaconda Download**…………………………………………………...19

Fig-7: **Jupyter Notebook**……………………………………………………19

Fig-8: **Defining a class**………………………………………………………24

Fig-9: **Importing Libraries**…………………………………………………28

Fig-10: **Reading the Dataset**…………………………………………………..29

Fig-11: **Output Column Visualization**……………………………………….30

Fig-12: **Countplot of Account length column**………………………………..30

Fig-13: **Histogram of Customer Service Calls**……………………………….31

Fig-14: **Kdeplot of all columns**………………………………………………..32

Fig-15: **Facetgrid of International plan and Churn columns**…………………33

Fig-16: **Null Values**…………………………………………………………….34

Fig-17: **Visualization of Null Values**……………………………………….…35

Fig-18: **Transforming Categorical columns using LabelEncoder**…………….36

Fig-19: **Splitting the data into train and test**…………………………………...38

Fig-20: **Scaling of data**……………………………………………………..…39

Fig-21: **Accuracy scores for KNN**…………………………………………....40

Fig-22: **Plotting k-values and scores**…………………………………………41

Fig-23: **KNN Model Building and Prediction on Train & Test data**…............42

Fig-24: **Confusion Matrix for Train and Test data (KNN)**…………………....43

Fig-25: **Classification report for Train and Test data (KNN)**…………………44

Fig-26: **AUC_ROC curve for KNN Classifier**……………………….……45

Fig-27: **DTC Model Building and Prediction on Train and Test data**………..46

Fig-28: **Confusion Matrix for Train and Test data (DTC)**……………………47

Fig-29: **Classification report for Train and Test data (DTC)**…………………48

Fig-30: **Visualization of Decision Tree**……………………………………….49

Fig-31: **KFold CV on DTC**……………………………………………………49

Fig-32: **Accuracy scores for RFC**…………………………………………….50

Fig-33: **Plotting k-values and scores**………………………………………….51

Fig-34: **RFC Model Building and Prediction on Train & Test data**………….52

Fig-35: **Confusion Matrix for Train and Test data (RFC)**…………………….53

Fig-36: **Classification report for Train and Test data (RFC)**…………………54

Fig-37: **AUC_ROC curve for Random-Forest Classifier**……………………...55

Fig-38: **Applying Hyper-parameter Tuning for RFC**…………………………56

Fig-39: **Prediction on Train and Test data**……………………………………..57

Fig-40: **Classification Report on Train & Test data**…………………………..58

Fig-41: **AUC_ROC curve for RFC (GridSearchCV)** …………………………...59

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 technique. The process flow depicted here represents how machine learning works:

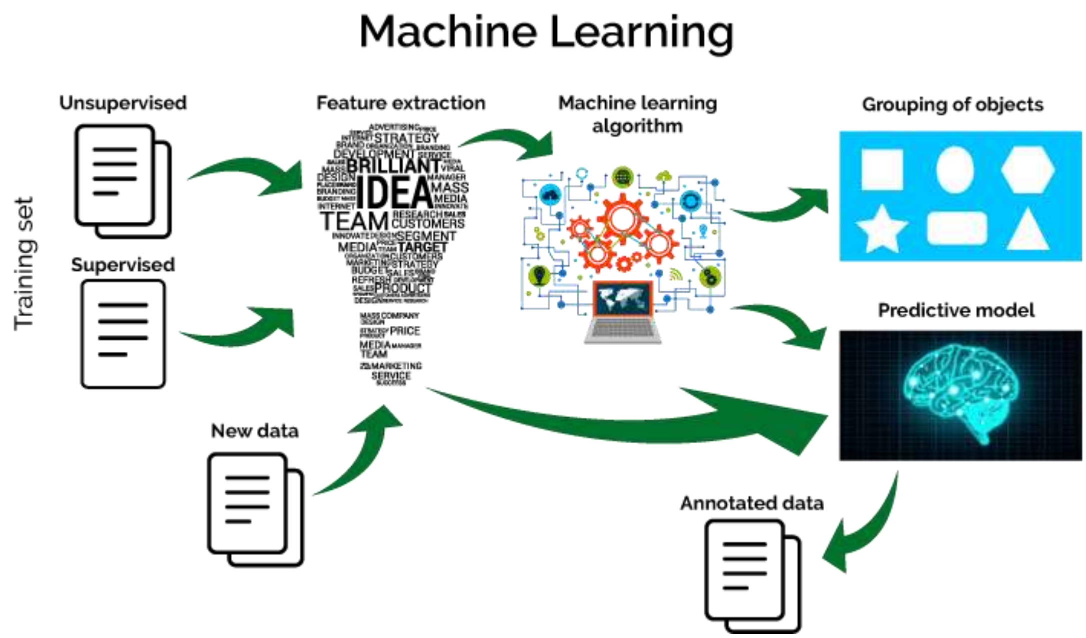


Figure 1: The Process Flow

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.

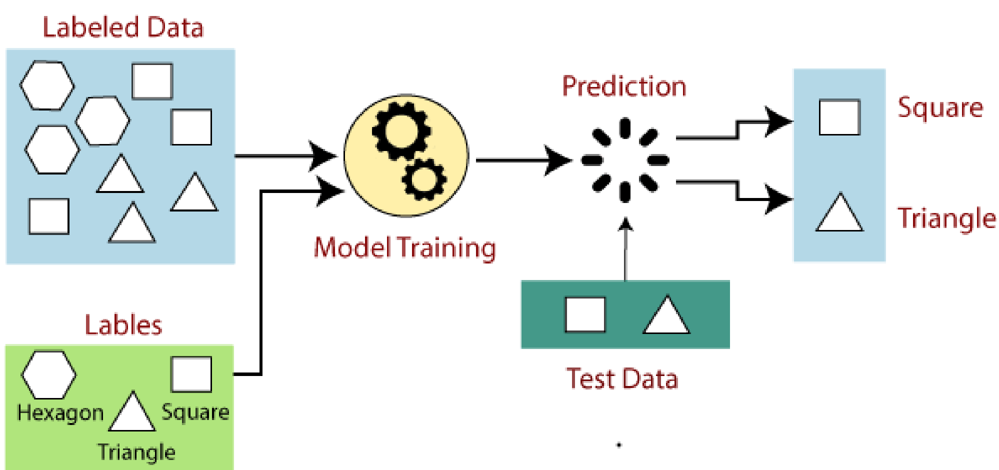


Fig 2: Supervised learning

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.

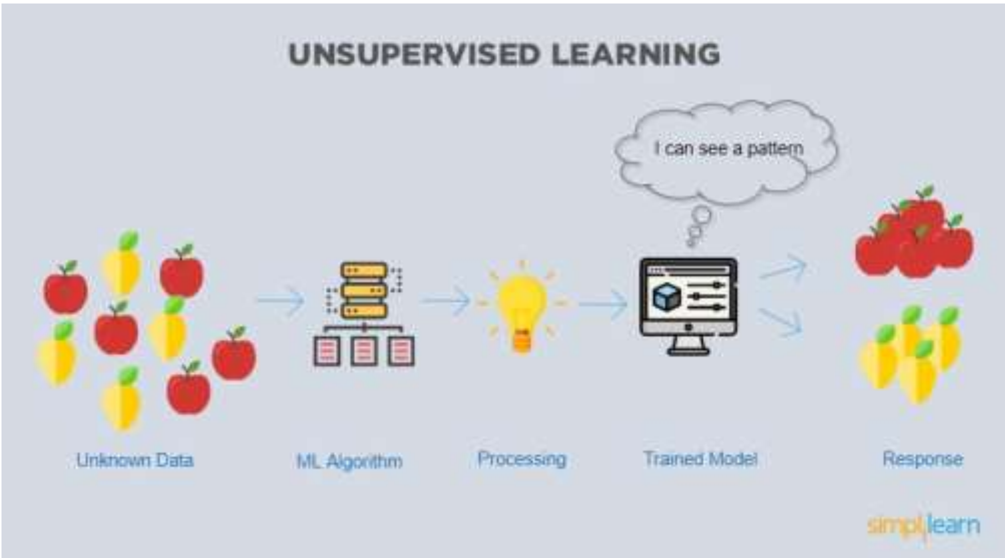


Figure 3: Unsupervised Learning

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.

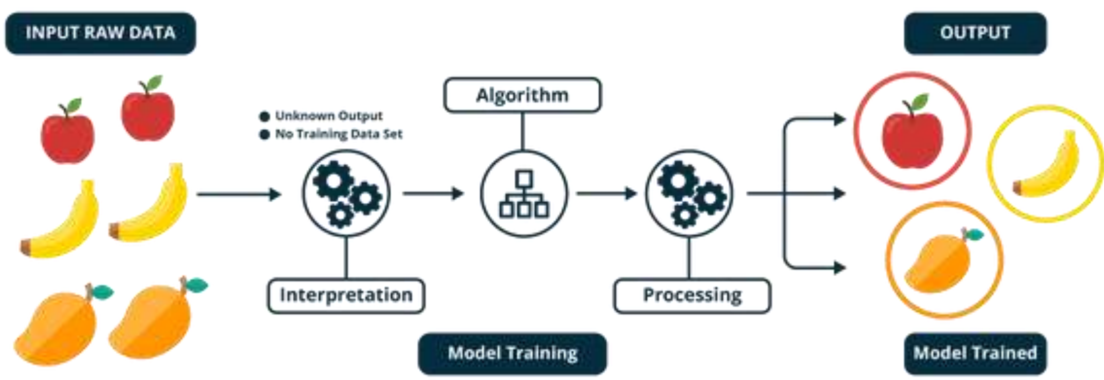


Figure 4: Semi Supervised Learning

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.

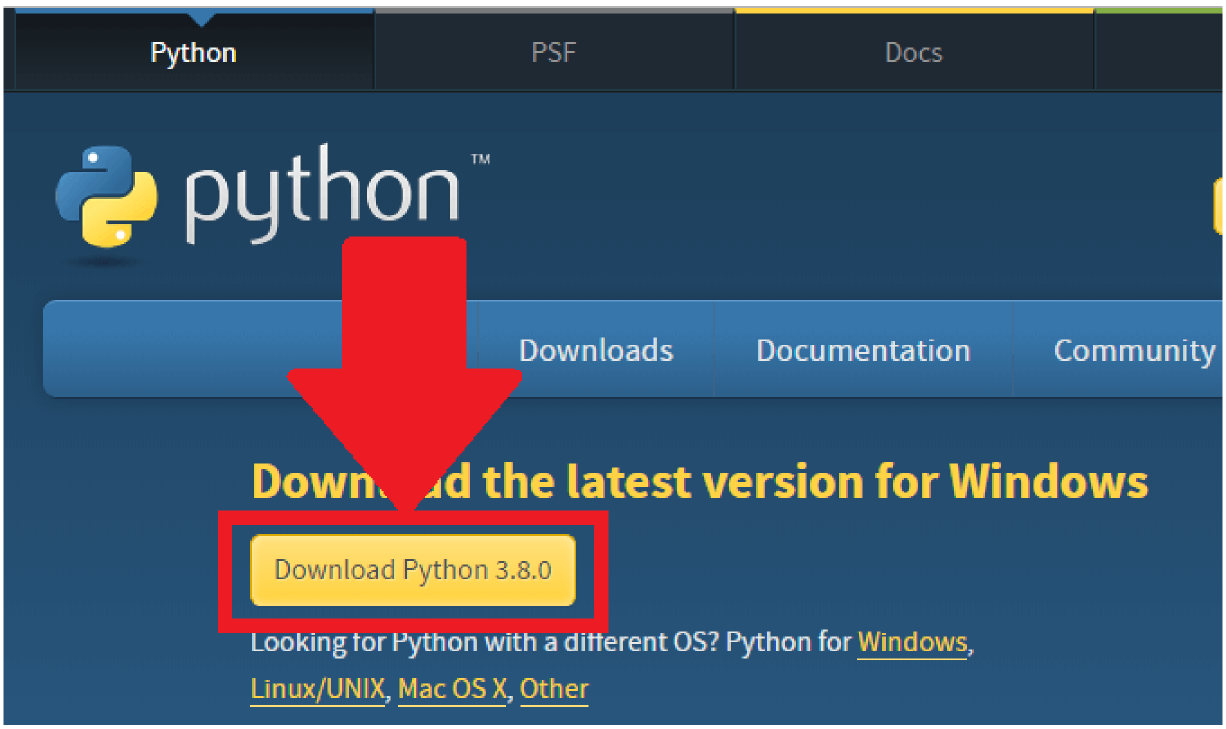
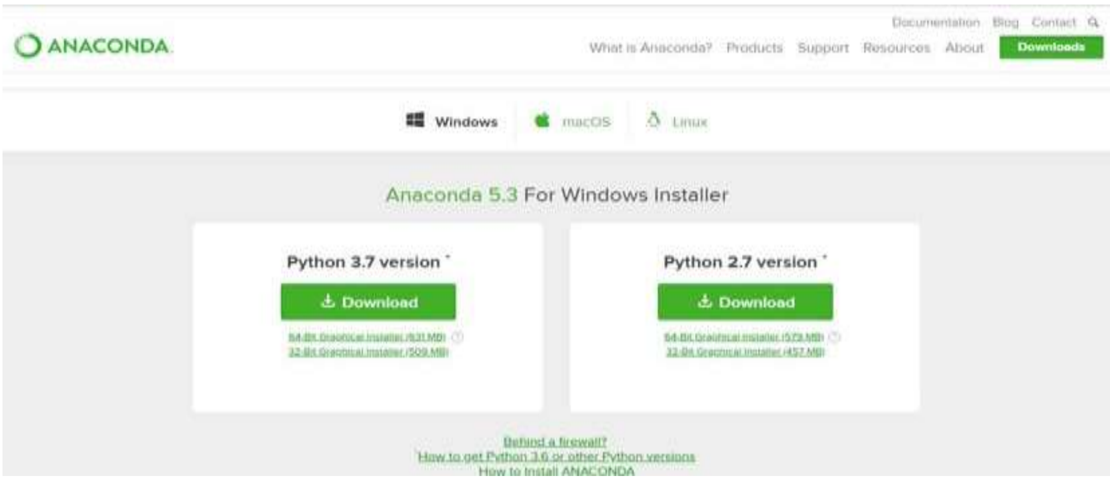


Figure 5: Python download

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)

Figure 6: Anaconda download

Figure 7: 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
```

Figure 8: 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:

Churn Prediction: Customer churn is a big problem for service providers because losing customers results in losing revenue and could indicate service deficiencies. There are many reasons why customers decide to leave services. With data analytics and machine learning, we can identify the important factors of churning, create a retention plan, and predict which customers are likely to churn.

3.2 DATA SET:

Each row represents a customer. Each columns represents customer's attributes.

The dataset have the following attributes or features.

1. **State:** Name of the state (string)
2. **Account length:** Length of account (integer)
3. **Area code:** Area code (integer)
4. **International plan:** [Yes/No] (string)
5. **Voice mail plan:** [Yes/No] (string)

6. **Number vmail messages:** Number of voice mail messages (integer)
7. **Total day minutes:** Number of minutes in a day (double)
8. **Total day calls:** Number of calls in a day (integer)
9. **Total day charge:** Total charge per day (double)
10. **Total eve minutes:** Number of minutes in the evening (double)
11. **Total eve calls:** Number of calls in the evening (integer)
12. **Total eve charge:** Total charge in the evening (double)
13. **Total night minutes:** Number of minutes in the night (double)
14. **Total night calls:** Number of calls in the night (integer)
15. **Total night charge:** Total charge in the night (double)
16. **Total intl minutes:** Total minutes for international call (double)
17. **Total intl calls:** Number of international calls (integer)
18. **Total intl charge:** Total charge for the international calls (double)
19. **Customer service calls:** Number of Customer Service calls (integer)
20. **Churn:** Customer has cancelled or not [True/False] (string)

3.3 OBJECTIVE OF THE CASE STUDY:

"Predict behavior to retain customers. You can analyze all relevant customer data and develop focused customer retention programs". This database was created to detect customers who are likely to cancel a subscription to a service. The database consists of cleaned customer activity data (features), along with a churn label specifying whether a customer canceled the subscription, will be used to develop predictive models.

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

Dataset:

- The Train and Test datasets which are considered in this notebook file are taken from Kaggle.
- The obtained dataset has been randomly partitioned into 80/20 ratio, where 80% is selected for generating the training data and 20% for the test data.
- Kaggle link

4.1.2 IMPORTING THE LIBRARIES:

● Numpy package can be used to perform mathematical operations like 'mean'.

● Pandas package can be used to process dataframes.

● Seaborn package can be used to visualize data in the form of various effective graph and plots.

● Sklearn is the main package which is used for machine learning.

● LabelEncoder is used to encode the non-numeric data into numericals so that Machine learning model can be built.

● Train_test_split module is used to split the data into training and testing sets.

- LinearRegression module is used to fit a LinearRegression model.

- Sklearn.metrics can be used to calculate statistical results like mean squared error, root mean squared error, etc.

IMPORTING LIBRARIES

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
from sklearn import tree
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
from sklearn.metrics import confusion_matrix, classification_report

from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

Figure 9: 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.

READING THE DATA:

```
df = pd.read_csv("C:\\Users\\HP\\Desktop\\AIML\\AI\\churn-dataset.csv")
df1 = pd.read_csv('C:\\Users\\HP\\Desktop\\AIML\\AI\\churn-dataset1.csv')
```

```
df.head(3)
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls	Total intl charge
0	LA	117	408	No	No	0	184.5	97	31.37	351.6	80	29.89	215.8	90	9.71	8.7	4	2.35
1	IN	65	415	No	No	0	129.1	137	21.95	228.5	83	19.42	208.8	111	9.40	12.7	6	3.43
2	NY	161	415	No	No	0	332.9	67	56.59	317.8	97	27.01	160.6	128	7.23	5.4	9	1.46

```
df1.head(3)
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls	Total intl charge
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29

```
d = pd.merge_ordered(df1,df)
d.head(3)
```

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Total night charge	Total intl minutes	Total intl calls	Total intl charge
0	AK	1	408	No	No	0	175.2	74	29.78	151.7	79	12.89	230.5	109	10.37	5.3	3	1.43
1	AK	36	408	No	Yes	30	146.3	128	24.87	162.5	80	13.81	129.3	109	5.82	14.5	6	3.92
2	AK	41	415	No	No	0	159.3	66	27.08	125.9	75	10.70	261.9	76	11.79	11.1	5	3.00

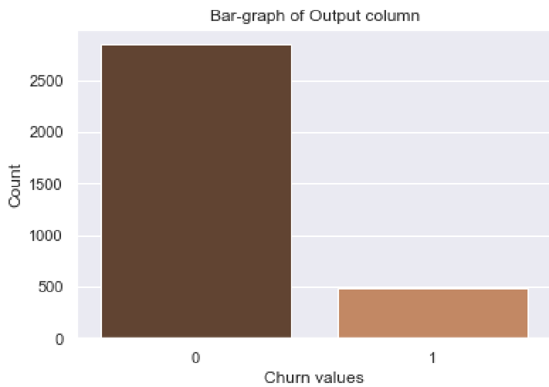
Figure 10: Reading the dataset

4.1.4 VISUALIZING THE COLUMNS:

Univariate Visualizations

```
## Visualization of output column
```

```
y = d["Churn"].value_counts()
sns.barplot(y.index, y.values,palette="copper")
plt.xlabel("Churn values")
plt.ylabel("Count")
plt.title("Bar-graph of Output column")
plt.show()
print("0 --> Not cancelled :",d['Churn'].value_counts()[0],"\n1 --> Cancelled :",d['Churn'].value_counts()[1])
```



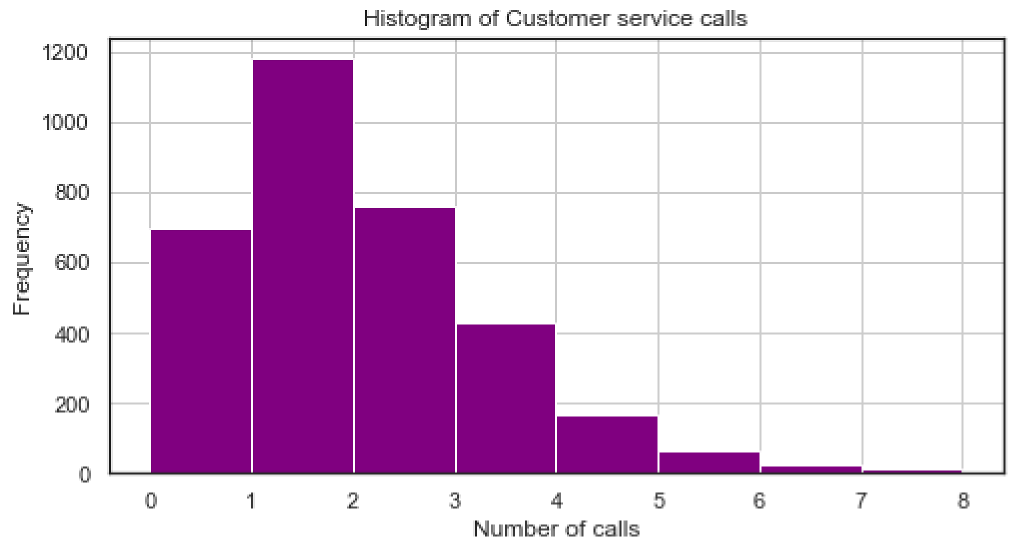

```
0 --> Not cancelled : 2850
1 --> Cancelled : 483
```

Figure 11: Output column visualization

```
plt.figure(figsize=(15,7))
sns.countplot(x=d['Account length'].head(50),data=d,color='red')
plt.xlabel("Account length")
plt.ylabel("Frequency")
plt.title("Count plot of Account lengths")
plt.show()
```

Figure 12: Countplot of Account length column

```
plt.figure(figsize=(8,4))
plt.hist(d['Customer service calls'],color='purple',bins=[0,1,2,3,4,5,6,7,8])
plt.title("Histogram of Customer service calls")
plt.xlabel("Number of calls")
plt.ylabel("Frequency")
plt.grid()
plt.show()
df['Customer service calls'].value_counts()
```




```
]: 1    236
   2    151
   0    142
   3     81
   4     33
   5     17
   6      5
   8      1
   7      1
```

Figure 13: Histogram of Customer service calls

Bi-variate Visualizations

```
plt.subplots(4,5,figsize=(15,15))
for i in range(2,20):
    plt.subplot(4,5,i)
    plt.title(d.columns[i-1])
    sns.kdeplot(d.loc[d['Churn'] == 0, d.columns[i-1]], color= 'yellow', label='False',shade='True')
    sns.kdeplot(d.loc[d['Churn'] == 1, d.columns[i-1]], color= 'blue', label='True')
```

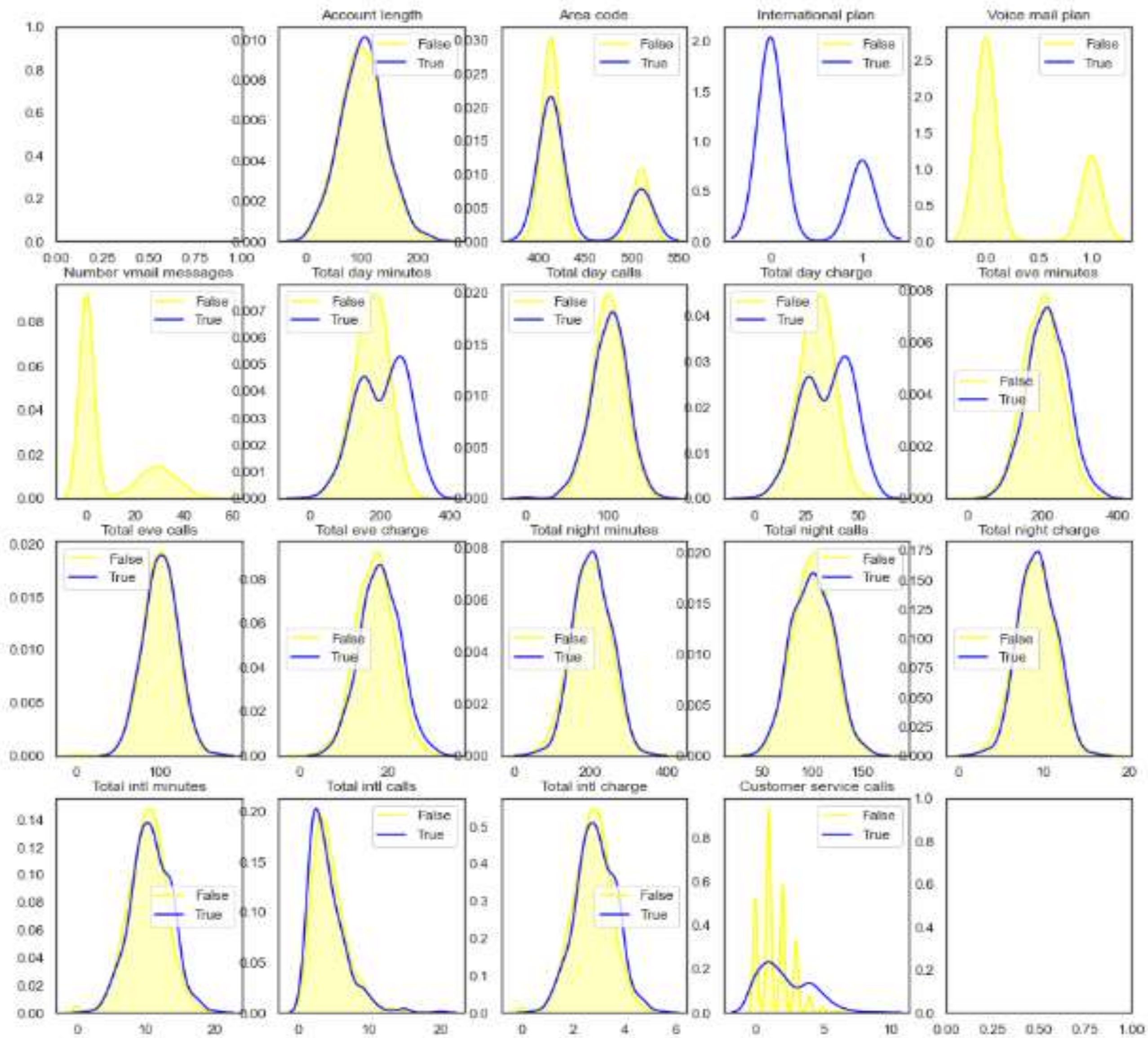


Figure 14: Kdeplot of all columns

```
plt.figure(figsize=(15,10))
g = sns.FacetGrid(d,row='International plan',col='Churn',size=4)
g =g.map(plt.hist,"Customer service calls",color='g')
```

```
<Figure size 1080x720 with 0 Axes>
```

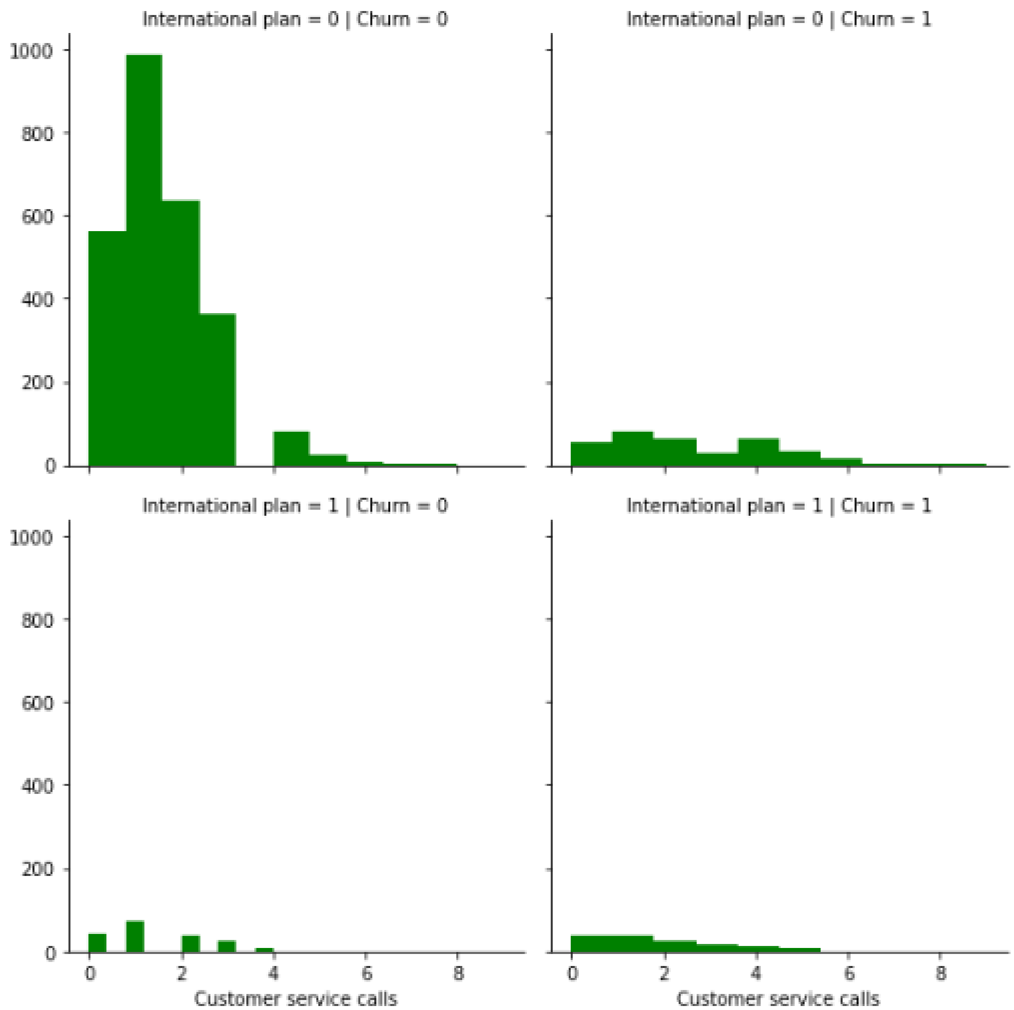


Figure 15: Facetgrid for International plan and Churn columns

4.1.5 HANDLING MISSING VALUES:

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

(a) dropna() : which drops all the rows and columns which are having the missing values.

(b) fillna() :

(c) interpolate()

(d) mean imputation and median imputation

```
null=d.isnull().sum()
print("There are",null.sum(),"missing values in dataset.")
null
```

```
There are 0 missing values in dataset.

State                     0
Account length            0
Area code                 0
International plan        0
Voice mail plan           0
Number vmail messages     0
Total day minutes         0
Total day calls           0
Total day charge          0
Total eve minutes         0
Total eve calls           0
Total eve charge          0
Total night minutes       0
Total night calls         0
Total night charge        0
Total intl minutes        0
Total intl calls          0
Total intl charge         0
Customer service calls    0
Churn                     0
dtype: int64
```

Figure 16: Null values

Visualizing the null values by using Heatmap

```
plt.figure(figsize=(10,8))
sns.heatmap(d.isnull(),cmap='autumn')
```

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x1f349eec320>
```

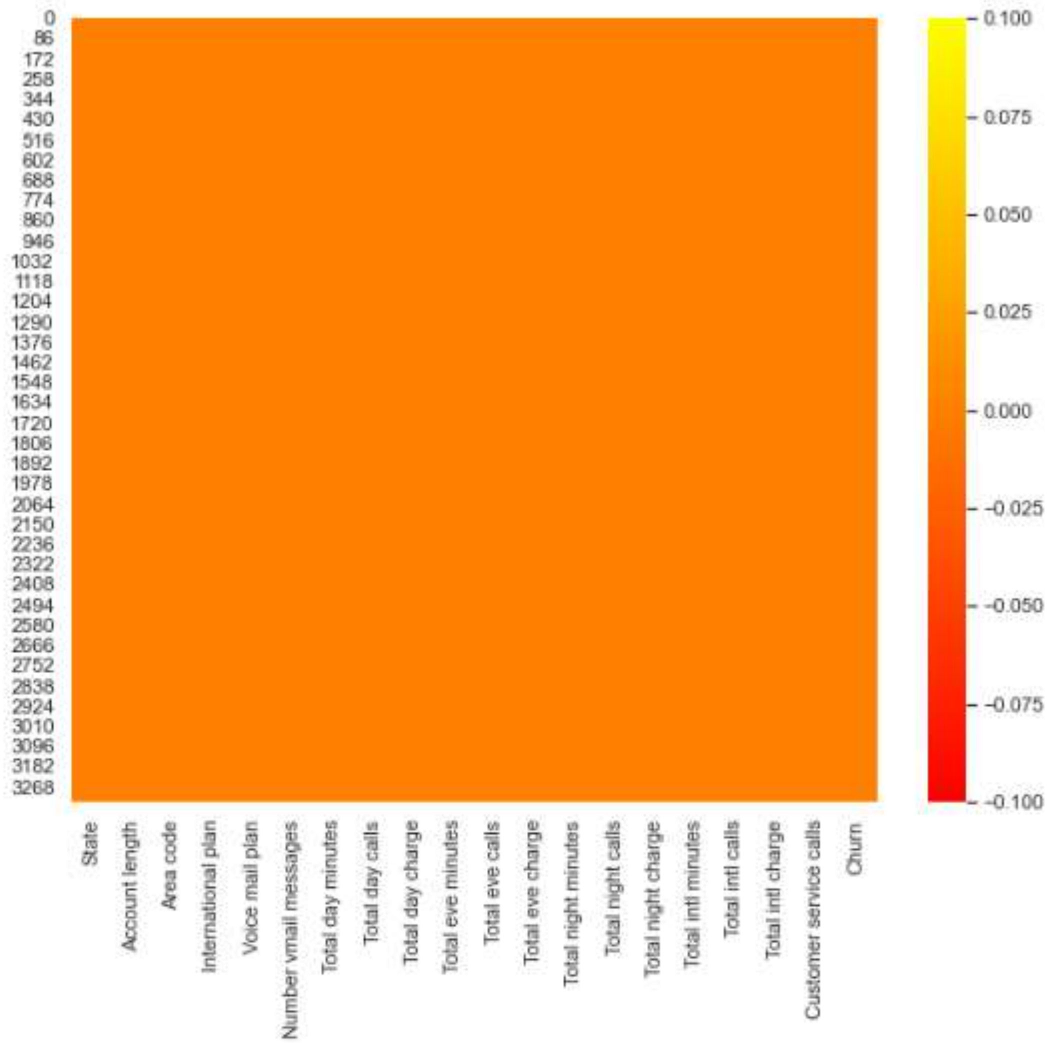


Figure 17: Visualizations of Null values

4.1.6 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 color
- **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
- Categorical data can be handled by using dummy variables, which are also called as indicator variables.
- Handling categorical data using dummies:

In pandas library we have a method called get_dummies () which creates dummy variables for those categorical data in the form of 0's and 1's.

Once these dummies got created we have to concat this dummy set to our dataframe or we can add that dummy set to the dataframe.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
d['Voice mail plan'] = le.fit_transform(d['Voice mail plan'])
d['International plan'] = le.fit_transform(d['International plan'])
d['Churn'] = le.fit_transform(d['Churn'])
```

Figure 18: Transforming the Categorical data using LabelEncoder

4.2 TRAINING THE MODEL:

4.2.1 Splitting the datasets:

- 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)

```
# 'X' -> i/p col ........... 'y' -> o/p col
X=d.drop(['Churn','State'],axis=1)
y=d.Churn
```

```
# Splitting the data into Training set and Testing set
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=0.2,random_state=1)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2666, 18)
(667, 18)
(2666,)
(667,)
```

Figure 19: Splitting the data into training and testing

4.2.2 FEATURE SCALING

Scaling a dataset usually produces better dataset and more accurate predictions.

First we check the range (the min and the max) for each of the datasets.

Let's try using the .describe() method and lets exclude the activity column which is the last Col

- Import Standard Scalar method which is available in preprocessing package from Scikit learn library.

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

```
# Scaling of training data
```

```
import warnings
warnings.filterwarnings("ignore")
scaled_X_train = pd.DataFrame(scaler.fit_transform(X_train),columns=X_train.columns)
scaled_X_train.head()
```

	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Tota nigh charg
0	1.394174	-0.682930	-0.330827	-0.626232	-0.598973	1.097372	-1.647677	1.096881	-0.954298	-0.109104	-0.953197	0.953346	-1.452839	0.95547
1	-0.358227	-0.682930	-0.330827	-0.626232	-0.598973	1.042197	1.158819	1.041705	0.412778	-0.661135	0.412504	-0.543701	-0.990058	-0.54307
2	-0.358227	-0.682930	-0.330827	-0.626232	-0.598973	0.096852	0.912635	0.097225	-0.182371	-0.861873	-0.181379	-0.015681	-0.218757	-0.01572
3	-0.458364	-0.516575	-0.330827	-0.626232	-0.598973	-0.390535	0.174083	-0.390703	0.014048	-0.109104	0.015041	0.557825	-1.401419	0.55996
4	-0.057816	-0.682930	-0.330827	-0.626232	-0.598973	0.181454	0.075610	0.181611	-0.854124	0.342558	-0.853831	1.398308	-0.424437	1.39932

```
# Scaling of test data
```

```
import warnings
warnings.filterwarnings("ignore")
scaled_X_test = pd.DataFrame(scaler.transform(X_test),columns=X_test.columns)
scaled_X_test.head()
```

	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	Total eve charge	Total night minutes	Total night calls	Tota nigh charg
0	-1.309531	1.741111	-0.330827	-0.626232	-0.598973	-1.124373	-1.549204	-1.124217	1.408623	-0.410211	1.408472	0.269095	2.712187	0.26992
1	1.494311	-0.516575	3.022728	-0.626232	-0.598973	-0.449389	0.666451	-0.449124	-0.418074	-1.865566	-0.417084	-0.419112	0.758224	-0.42002
2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Figure 20: Scaling of the data

4.2.3 CLASSIFICATION ACTIVITIES(MODELS)

To begin, I'll use various machine learning algorithms available inside the sklearn package that I have already imported.

For each algorithm, I'll calculate the accuracy of prediction and identify the most accurate algorithm.

For now, I will keep the default values of parameters as defined in sklearn for each classifier. I am using three algorithms for my train and test data.

MODEL 1:

K-Nearest Neighbors

The k-nearest neighbors algorithm (k-NN) is a non-parametric used for classification and regression.

In both cases, the input consists of the k closest training examples in the feature space. In k-NN classification,

Checking for Optimum k-value and building the model with these k-values

```
# Checking for optimum K value
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
scores=[]
for k in range(1, 20):
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(scaled_X_train, y_train)
    pred_test = knn_model.predict(scaled_X_test)
    scores.append(accuracy_score(y_test, pred_test))
scores
```

```
]: [0.8755622188905547,
    0.8920539730134932,
    0.904047976011994,
    0.9025487256371814,
    0.9115442278860569,
    0.9070464767616192,
    0.904047976011994,
    0.8995502248875562,
    0.9085457271364318,
    0.8980509745127436,
    0.9010494752623688,
    0.8935532233883059,
    0.896551724137931,
    0.8980509745127436,
    0.8980509745127436,
    0.8980509745127436,
    0.8980509745127436,
    0.8980509745127436,
    0.8980509745127436]
```

Figure 21: Accuracy scores

```python
# Plot of K values and Scores
```

```python
plt.plot(range(1,20), scores, marker='o', markerfacecolor='r', linestyle='--')
plt.xlabel("k")
plt.ylabel("scores")
plt.title("K vs scores")
plt.grid()
```

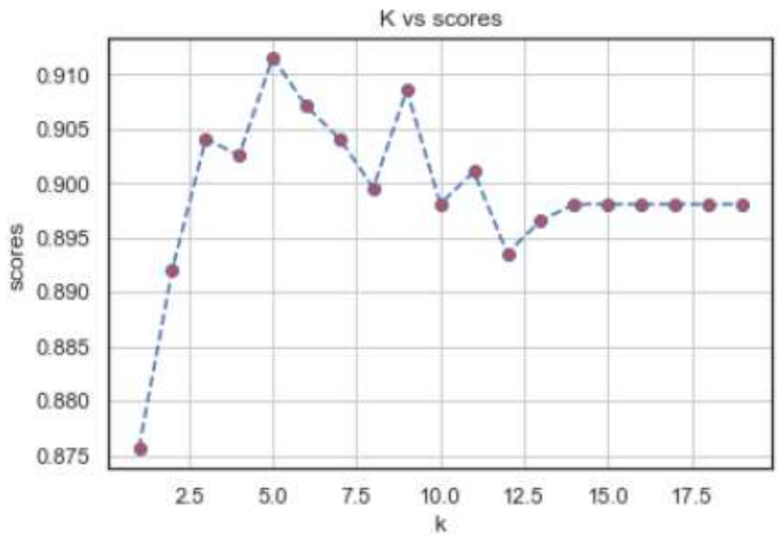


Figure 22: Plotting k values and scores

```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(scaled_X_train, y_train)
```

```
]: KNeighborsClassifier(metric='euclidean')
```

```python
# Prediction on  training data
final_train_pred = knn.predict(scaled_X_train)
print("Prediction on  training data\n",final_train_pred)

# Prediction on  testing data
final_test_pred = knn.predict(scaled_X_test)
print("\nPrediction on  testing data\n",final_test_pred)
```

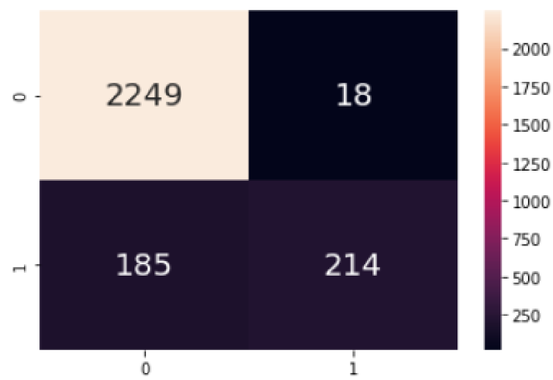
```
Prediction on  training data
 [0 0 0 ... 0 0 0]

Prediction on  testing data
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0]
```

Figure 23: Model building and predicting on train and test data

```python
# Confusion matrix for training data
```

```python
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(y_train, final_train_pred), annot=True,fmt='d', annot_kws={'size':20})
```

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x1f34bc14198>
```




```python
# Confusion matrix for testing data
```

```python
sns.heatmap(confusion_matrix(y_test, final_test_pred), annot=True, fmt='d',annot_kws = {'size':20})
```

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x1f34bc9b320>
```

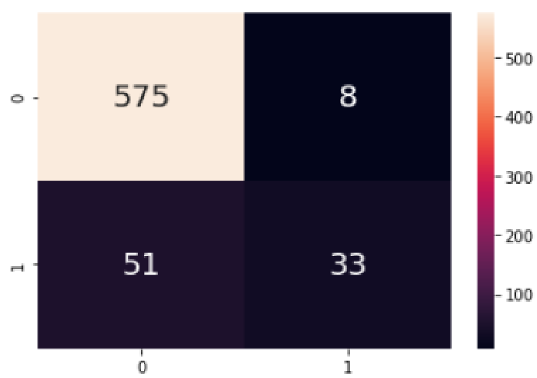


Figure 24: Confusion matrices for training and test data

```
from sklearn.metrics import classification_report
print("Classification Report for Training Data\n",classification_report(y_train, final_train_pred))
print("\nClassification Report for Test Data\n",classification_report(y_test, final_test_pred))
```

```
Classification Report for Training Data
               precision    recall  f1-score   support

           0       0.92      0.99      0.96      2267
           1       0.92      0.54      0.68       399

    accuracy                           0.92      2666
   macro avg       0.92      0.76      0.82      2666
weighted avg       0.92      0.92      0.92      2666


Classification Report for Test Data
               precision    recall  f1-score   support

           0       0.92      0.99      0.95       583
           1       0.80      0.39      0.53        84

    accuracy                           0.91       667
   macro avg       0.86      0.69      0.74       667
weighted avg       0.90      0.91      0.90       667
```

Figure 25: Classification report for train and test data

- **Using K-Nearest Neighbors Classifier the accuracy for train data is 92% and for test data accuracy is 91%. (As the output column is imbalanced, we consider the 'f1-score' accuracy values)**

AUC: Area Under the ROC Curve

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

ROC curve

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

```
from sklearn.metrics import roc_auc_score,roc_curve
k_prob=knn.predict_proba(scaled_X_test)[:,1]
fpr,tpr,threshold=roc_curve(y_test,k_prob,pos_label=1)

plt.plot(fpr,tpr)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
roc_auc_score(y_test,k_prob)
```

```
0.834681042228212
```

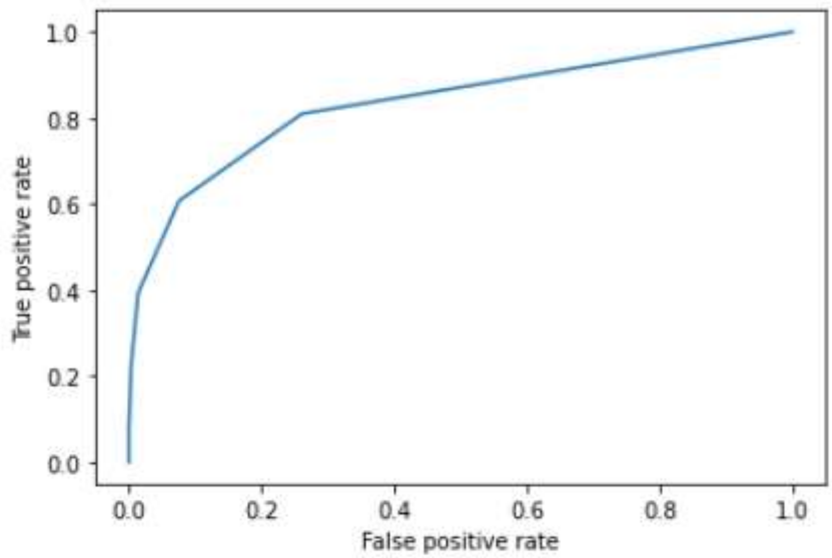


Figure 26: Plotting for AUC_ROC curve for KNN classifier

MODEL 2:

Decision Tree Classifier:

- A decision tree is a flowchart-like tree structure where an internal node represents feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome.

- The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions tree in recursively manner call recursive partitioning.

- This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.

```
from sklearn.tree import DecisionTreeClassifier
#initailization of object
dtree = DecisionTreeClassifier()
# Applying the classifier to the dataset
dtree.fit(scaled_X_train, y_train)
```

```
DecisionTreeClassifier()
```

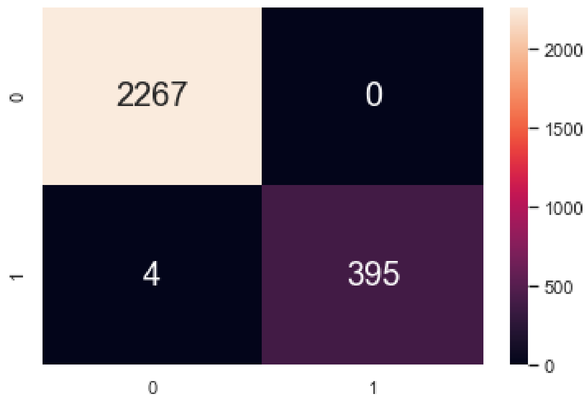
```
# Predict on training data
y1_train_pred = dtree.predict(scaled_X_train)
print("Prediction on training data\n",y1_train_pred)

y1_test_pred =  dtree.predict(scaled_X_test)
print("\nPrediction on testing data\n",y1_test_pred)
```

```
Prediction on training data
 [0 0 0 ... 0 0 0]

Prediction on testing data
 [0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 1
 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0
 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1
 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0
 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0
 0]
```

Figure 27: Model building and predicting on train and test data

```
# FOR TRAINING DATA
sns.heatmap(confusion_matrix(y_train, y_train_pred), annot=True,fmt='d', annot_kws={'size':20})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f34bd867f0>
```




```
# FOR TEST DATA
sns.heatmap(confusion_matrix(y_test, y_test_pred), annot=True,fmt='d', annot_kws={'size':20})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f349ea9d30>
```

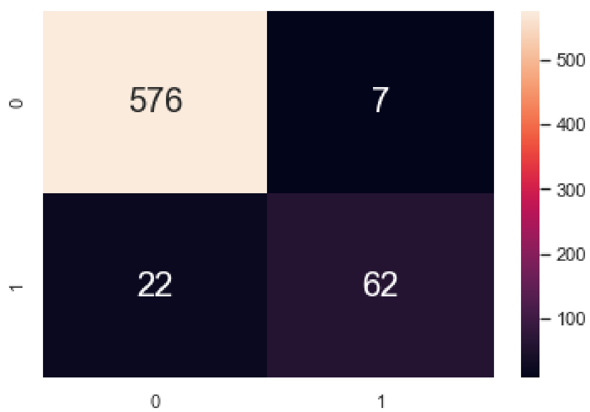


Figure 28: Confusion Matrix for training and testing data

```
# Classification Report on training data
from sklearn.metrics import classification_report, confusion_matrix
print("Classification Report on training data\n",classification_report(y_train, y1_train_pred))


y1_test_pred =  dtree.predict(scaled_X_test)
print("\nClassification Report on testing data\n",classification_report(y_test, y_test_pred))
```

```
Classification Report on training data
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2267
           1       1.00      1.00      1.00       399

    accuracy                           1.00      2666
   macro avg       1.00      1.00      1.00      2666
weighted avg       1.00      1.00      1.00      2666


Classification Report on testing data
               precision    recall  f1-score   support

           0       0.96      0.99      0.98       583
           1       0.90      0.74      0.81        84

    accuracy                           0.96       667
   macro avg       0.93      0.86      0.89       667
weighted avg       0.96      0.96      0.95       667
```

Figure 29: Classification report for training and testing data

- **Using Decision Tree Classifier the accuracy for train data is 100% and for test data accuracy is 96%. (As the output column is imbalanced, we consider the 'f1-score' accuracy values).**

```python
# Visualization of the Decision Tree
import sklearn
from sklearn import tree
sklearn.tree.plot_tree(dtree)
plt.show()
```

Figure 30: Visualization of the Decision Tree

- **KFOLD CV**

```python
from sklearn.model_selection import cross_val_score
cross_val_score(dtree, scaled_X_train, y_train, cv =5)
```

```
array([0.91011236, 0.90619137, 0.91744841, 0.90994371, 0.93058161])
```

Figure 31: KFold CV on Decision Tree

MODEL 3:

Random Forest Classification

- Random forest is a type of supervised machine learning algorithm based on ensemble learning. It is a type of learning where you join different types of algorithms or same algorithm multiple times to form a more powerful prediction model.

- The random forest algorithm can be used for both regression and classification tasks.

```
# Checking for optimum K value
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
score=[]
for k in range(1, 20):
    rf_model = RandomForestClassifier(n_estimators=k)
    rf_model.fit(scaled_X_train, y_train)
    pred_test = rf_model.predict(scaled_X_test)
    score.append(accuracy_score(y_test, pred_test))
score
```

```
]: [0.8785607196401799,
    0.9400299850074962,
    0.9295352323838081,
    0.9475262368815592,
    0.9475262368815592,
    0.9415292353823088,
    0.9490254872563718,
    0.9565217391304348,
    0.9535232383808095,
    0.9565217391304348,
    0.952023988005997,
    0.952023988005997,
    0.9535232383808095,
    0.9490254872563718,
    0.9580209895052474,
    0.9565217391304348,
    0.967016491754123,
    0.952023988005997,
    0.9610194902548725]
```

Figure 32: Accuracy scores

```
plt.plot(range(1,20), score, marker='o', markerfacecolor='r', linestyle='-.')
plt.xlabel("k values")
plt.ylabel("score")
plt.grid()
```

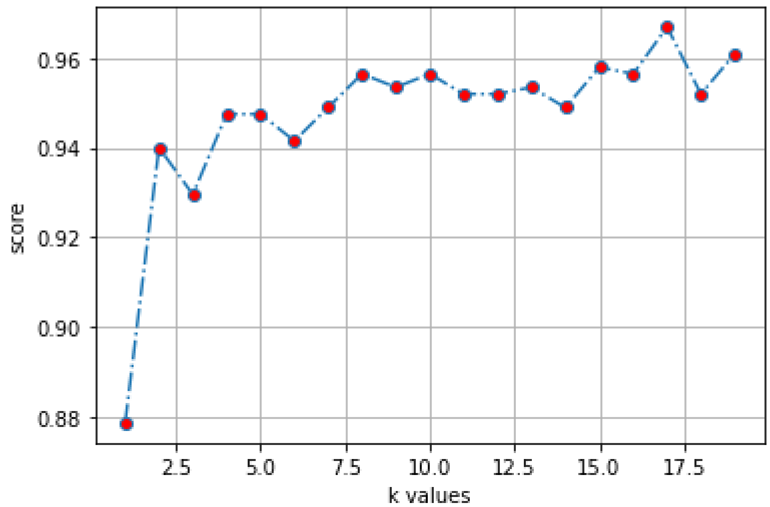


Figure 33: Plotting the k-values and scores

Building the model and predicting the training and testing models

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=17)
rfc.fit(scaled_X_train,y_train)
```

```
]: RandomForestClassifier(n_estimators=17)
```

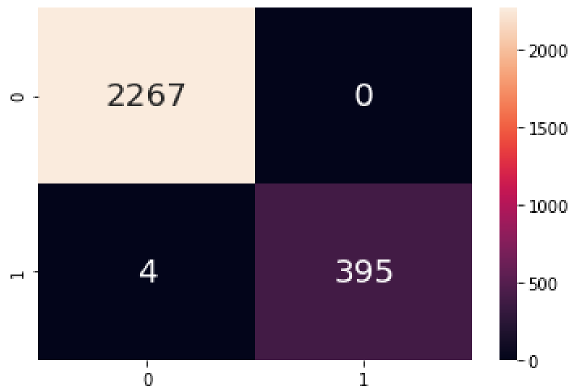
```
# Prediction on  training data
y_train_pred = rfc.predict(scaled_X_train)
print("Prediction on training data\n",y_train_pred)

# Prediction on  testing data
y_test_pred = rfc.predict(scaled_X_test)
print("\nPrediction on testing data\n",y_test_pred)
```

```
Prediction on training data
 [0 0 0 ... 0 0 0]

Prediction on testing data
 [0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0
 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
 0]
```

Figure 34: Building the model and predicting the training and testing models

```python
# Confusion matrix for training data
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(y_train, y_train_pred), annot=True,fmt='d', annot_kws={'size':20})
```

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x1f34bf294a8>
```




```python
# Confusion matrix for testing data
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(y_test, y_test_pred), annot=True,fmt='d', annot_kws={'size':20})
```

```
: <matplotlib.axes._subplots.AxesSubplot at 0x1f34c2c91d0>
```

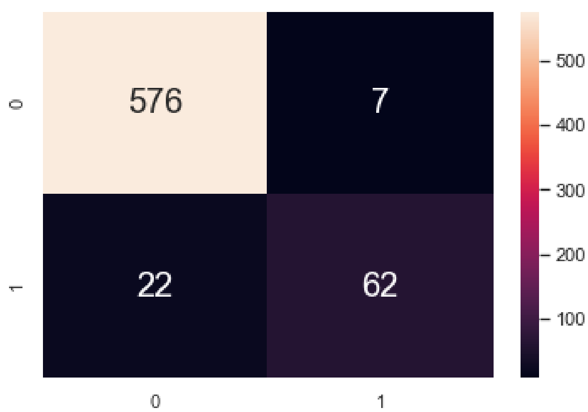


Figure 35: Confusion matrix for training and testing data

```python
from sklearn.metrics import classification_report
print("Classification Report for Training Data\n",classification_report(y_train, y_train_pred))
print("\nClassification Report for Test Data\n",classification_report(y_test, y_test_pred))
```

```
Classification Report for Training Data
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2267
           1       1.00      0.99      0.99       399

    accuracy                           1.00      2666
   macro avg       1.00      0.99      1.00      2666
weighted avg       1.00      1.00      1.00      2666


Classification Report for Test Data
               precision    recall  f1-score   support

           0       0.96      0.99      0.98       583
           1       0.90      0.74      0.81        84

    accuracy                           0.96       667
   macro avg       0.93      0.86      0.89       667
weighted avg       0.96      0.96      0.95       667
```

Figure 36: Classification report for training and testing data

```
from sklearn.metrics import roc_auc_score,roc_curve
fig = plt.figure()
fig.patch.set_facecolor('pink')
k_prob=rfc.predict_proba(scaled_X_test)[:,1]
fpr,tpr,threshold=roc_curve(y_test,k_prob,pos_label=1)

plt.plot(fpr,tpr)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
roc_auc_score(y_test,k_prob)
```

```
0.8923262272318877
```

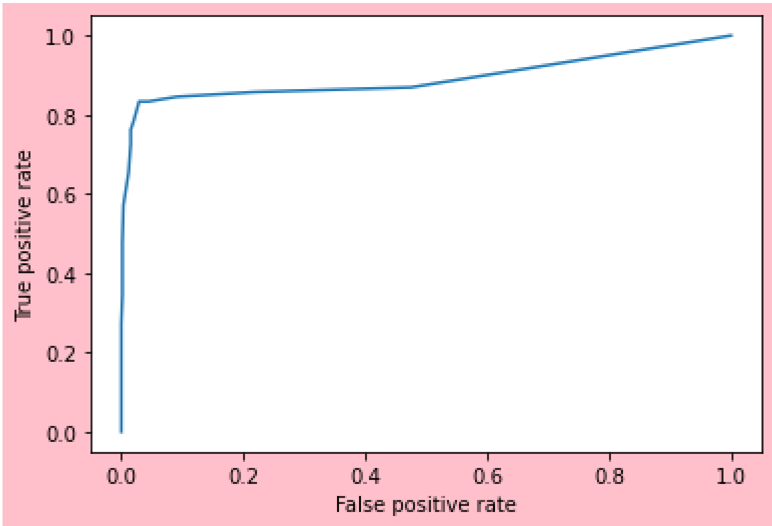


Figure 37: AUC_ROC curve for random forest classifier

To get the best result we use hyper parameters based on (Random Forest classifier) called Grid search CV.

<u>GridSearchCV</u>

- Grid-searching is the process of scanning the data to configure optimal parameters for a given model. Depending on the type of model utilized, certain parameters are necessary. Grid-searching does NOT only apply to one model type.

- It is important to note that Grid-searching can be extremely computationally expensive and may take your machine quite a long time to run. Grid-Search will build a model on each parameter combination possible.

- It iterates through every parameter combination and stores a model for each combination. Without further ado, lets jump into some examples and implementation

```
param_grid = {
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
```

```
from sklearn.model_selection import GridSearchCV
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(scaled_X_train, y_train)

print(CV_rfc.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt'}
```

```
rfc1=RandomForestClassifier(random_state=42, max_features='sqrt', n_estimators= 14, max_depth=8, criterion='gini')
```

```
rfc1.fit(scaled_X_train, y_train)
```

```
]: RandomForestClassifier(max_depth=8, max_features='sqrt', n_estimators=14,
                          random_state=42)
```

Figure 38: Applying the hyper parameter to get the optimum parameters and fitting the model

```python
train_pred_3=rfc1.predict(scaled_X_train)
print("Prediction on training data\n",train_pred_3)

test_pred_3=rfc1.predict(scaled_X_test)
print("\nPrediction on testing data\n",test_pred_3)
```

```
Prediction on training data
 [0 0 0 ... 0 0 0]

Prediction on testing data
 [0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0
 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
 0]
```

Figure 39: Prediction of training and testing data

```
print("Classification report on training data:\n",classification_report(y_train,train_pred_3))

print("\nClassification report on testing data:\n",classification_report(y_test,test_pred_3))
```

```
Classification report on training data:
               precision    recall  f1-score   support

           0       0.97      1.00      0.99      2267
           1       1.00      0.84      0.91       399

    accuracy                           0.98      2666
   macro avg       0.98      0.92      0.95      2666
weighted avg       0.98      0.98      0.97      2666


Classification report on testing data:
               precision    recall  f1-score   support

           0       0.96      0.98      0.97       583
           1       0.87      0.73      0.79        84

    accuracy                           0.95       667
   macro avg       0.92      0.86      0.88       667
weighted avg       0.95      0.95      0.95       667
```

- **As the output column is imbalanced, we consider f1-score accuracy**
    - Train : 98%
    - Test : 95%

Figure 40: Classification report on training and testing data

```python
from sklearn.metrics import roc_auc_score,roc_curve
fig = plt.figure()
fig.patch.set_facecolor('pink')
k1_prob=rfc1.predict_proba(scaled_X_test)[:,1]
fpr,tpr,threshold=roc_curve(y_test,k_prob,pos_label=1)

plt.plot(fpr,tpr)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
roc_auc_score(y_test,k_prob)
```

```
0.8923262272318877
```

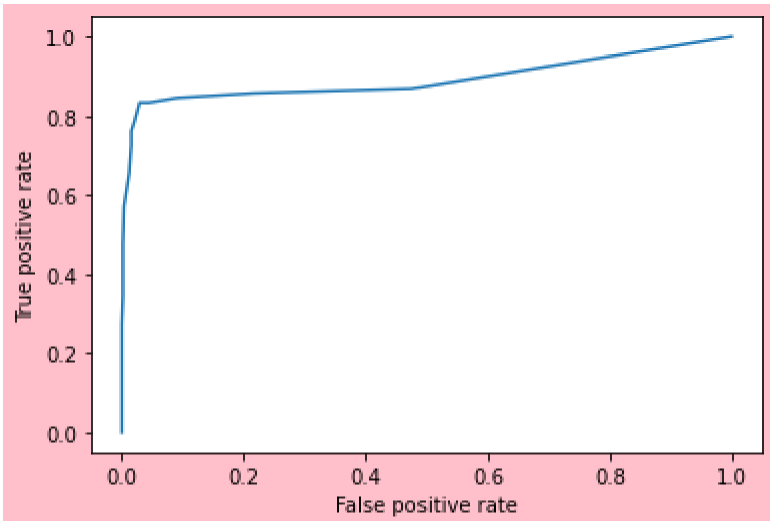


Figure 41: AUC_ROC curve and score

Using Grid Search CV, the accuracy rate is constant 89% (i.e... no change in accuracy rate)

EVALUATION

From above algorithms, we can see the following accuracies and auc_roc scores :

1. K-Nearest-Neighbors Classifier

- Training accuracy = 92%
- Testing accuracy = 91%
- AUC_ROC score = 83%

2. Random-forest Classifier

- Training accuracy = 99%
- Testing accuracy = 95%
- AUC_ROC score = 89%

  - **GridSearchCV**

    - Training accuracy = 98%
    - Testing accuracy = 95%
    - AUC_ROC score = 89%

3. Decision tree Classifier

- Training accuracy = 100%
- Testing accuracy = 92%
- AUC_ROC score = 82%

- From the above observation, Random-Forest classifier is best model to predict the given problem statement i.e, this model gives more accurate values for the given problem statement.
- As we can see that the highest accuracy is 89% which is given by Random-Forest classifier.

CONCLUSION:

- In this particular project, I have explored the Churn Prediction dataset. I checked the statistical analysis which includes mean, median, standard deviation, datatypes of attributes and null values.
- Then, visualization of each column and visualize the output column with all input columns using Matplotlib and Seaborn packages.
- Then, I applied numerous machine learning algorithms and found out that Random-Forest Classifier performed the best with highest accuracy in classifying customer's churn.

REFERENCES:

[1] https://www.kaggle.com/mnassrib/telecom-churn-datasets

[2] https://medium.com/code-heroku/introduction-to-exploratory-data-analysis-eda-c0257f888676

[3] https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

[4] https://matplotlib.org/index.html

[5] https://seaborn.pydata.org