MACHINE LEARNING

(Gender Recognition By Voice)

Summer Internship Report Submitted in partial fulfillment of the

requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Raghavendra Yerramsetty

221710301066

Under the Guidance of Assistant

Professor

Department Of Computer Science Engineering

GITAM School of Technology

GITAM (Deemed to be University)

Hyderabad-502329

July 2020

DECLARATION

I submit this industrial training work entitled **“Gender Recognition by Voice**” 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 Y Raghavendra

Date: 221710301066

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled **“GENDER RECOGNITION BY VOICE”** is being submitted by Raghavendra Yerramsetty (221710301066) 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 2020-2021

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

Assistant Professor
Department of CSE

Professor and HOD
Department of CSE

ACKNOWLEDGEMENT

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

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

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

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

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

Y Raghavendra

221710301066

ABSTRACT

The speech entailed in human voice comprises essentially paralinguistic information used in many voice-

recognition applications. Gender voice is considered one of the pivotal parts to be detected from a given voice

task that involves certain complications. In order to distinguish gender from a voice signal, a set of techniques

have been employed to determine relevant features to be utilized for building a model from a training set. This

model is useful for determining the gender (i.e., male or female) from a voice signal.

The contributions are three-fold including

(i) providing analysis information about well-known voice signal features using a prominent dataset.

(ii) studying various machine learning models of different theoretical families to classify the voice gender.

(iii) using three prominent feature selection algorithms to find promisingly optimal features for improving classification models. The experimental results show the importance of sub-features over others, which are vital for enhancing the efficiency of classification models' performance. Experimentation reveals that the best recall value is equal to 99.97%; the best recall value is 99.7% .

Machine learning algorithms are used to predict the values from the data set by splitting the data set to train and test and building Machine learning algorithms models of higher accuracy to predict the values.

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

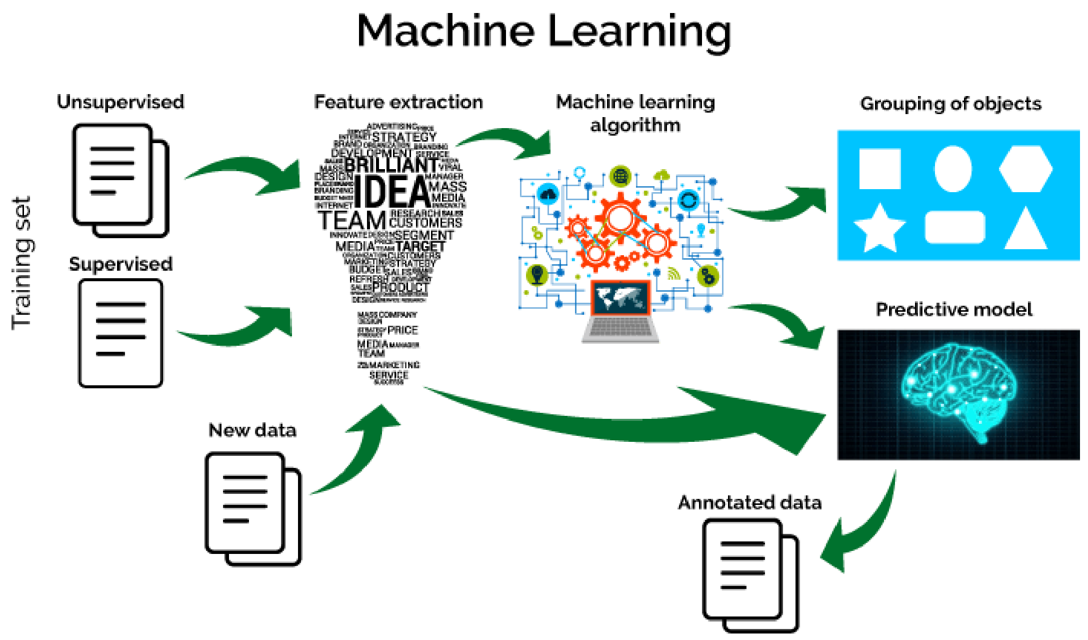


Figure 1.2.1: The Process Flow

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.

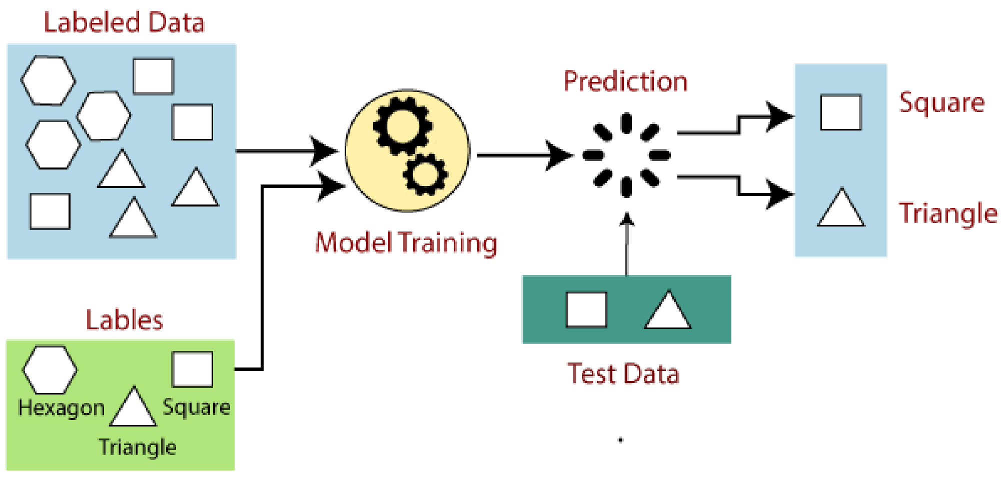


Figure 1.4.1.1 Supervised learning

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.

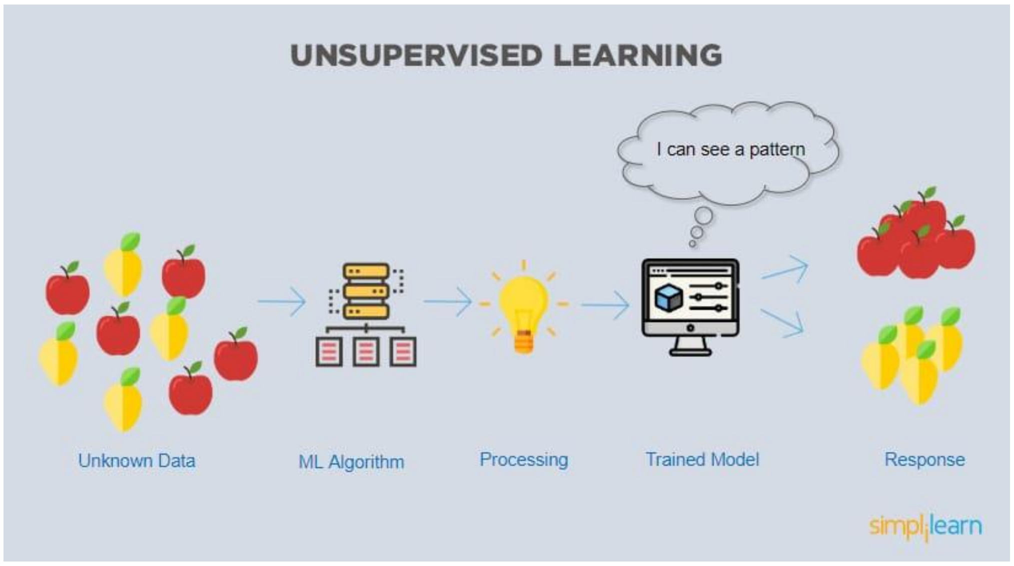


Figure 1.4.2.1 : Unsupervised Learning

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.

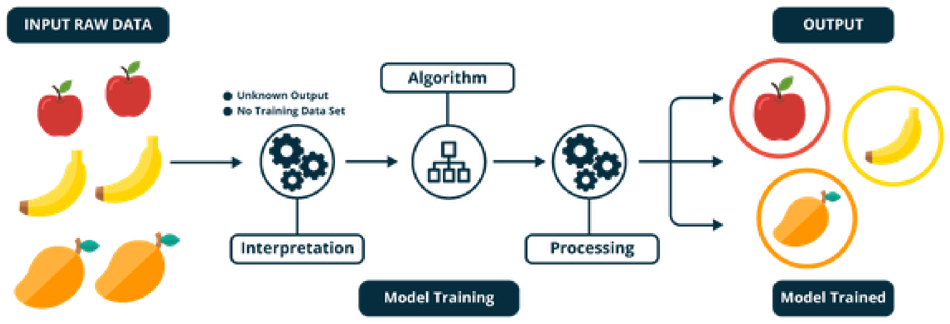


Figure 1.4.3.1 Semi Supervised Learning

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:

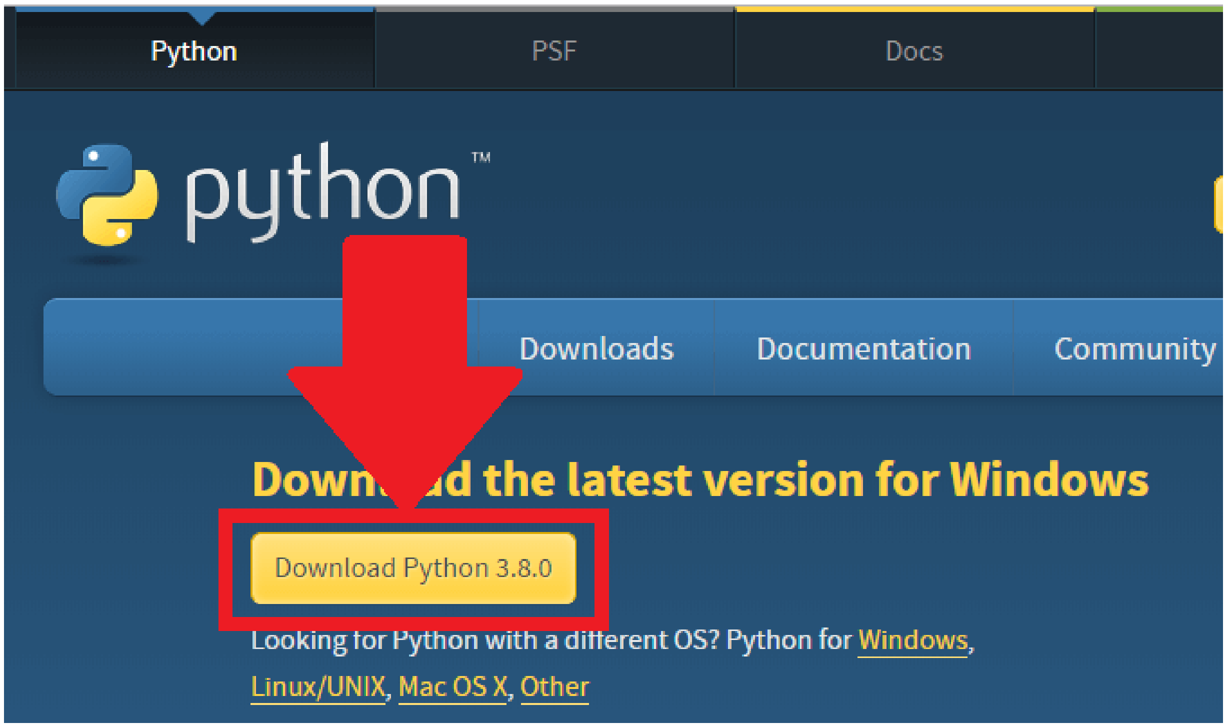
- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad 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.



Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Figure 2.4.1.1 Python download

2.4.2 Installation (using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.

- In WINDOWS:
- In windows
  - Step 1: Open Anaconda.com/downloads in web browser.
  - Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
  - Step 3: select installation type (all users)
  - Step 4: Select path (i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
  - Step 5: Open jupyter notebook (it opens in default browser)

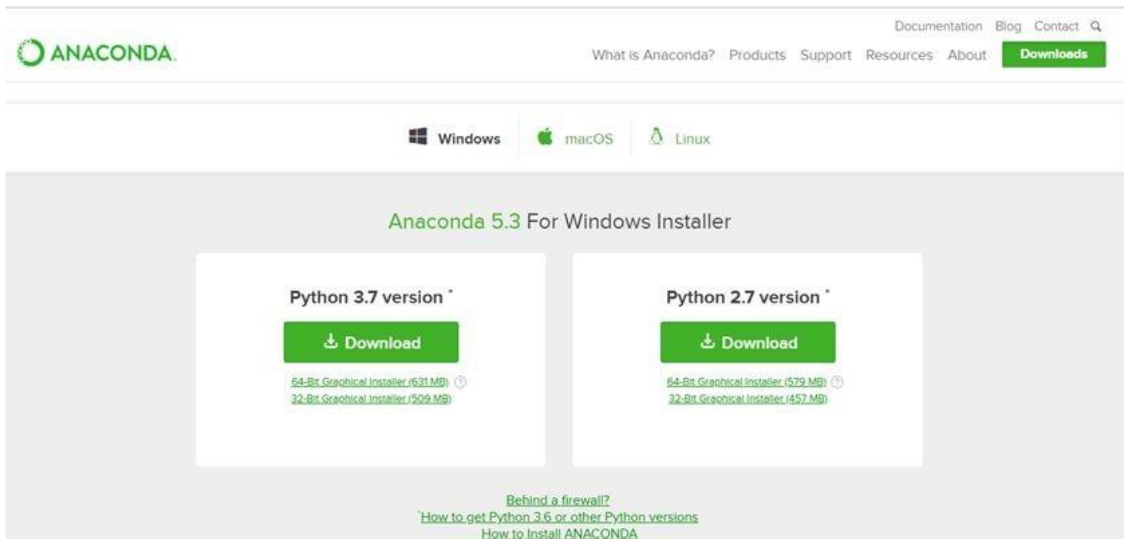

ANACONDA.
Documentation Blog Contact
What is Anaconda? Products Support Resources About Downloads
Windows macOS Linux
Anaconda 5.3 For Windows Installer
Python 3.7 version
Download
64-Bit Graphical Installer (631 MB)
32-Bit Graphical Installer (509 MB)
Python 2.7 version
Download
64-Bit Graphical Installer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

Figure 2.4.2.1 Anaconda download

Figure 2.4.2.2 Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

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

  - Tuples
  - Dictionary

2.5.1 Python Numbers:

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

2.5.2 Python Strings:

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

2.5.3 Python Lists:

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

([]).

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

2.5.4 Python Tuples:

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

2.5.5 Python Dictionary:

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

2.6 PYTHON FUNCTION:

2.6.1 Defining a Function:

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

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

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

2.6.2 Calling a Function:

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

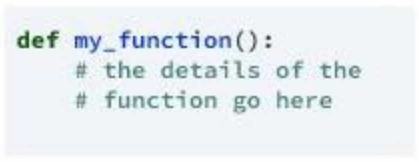
2.7 PYTHON USING OOP's CONCEPTS:

2.7.1 Class

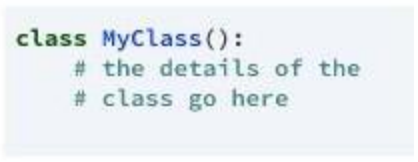
- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

- Defining a Class:
  - We define a class in a very similar way how we define a function.
  - Just like a function, we use parentheses and a colon after the class name (i.e. () :) when we define a class. Similarly, the body of our class is

indented like a functions body is.



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



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

Figure 2.7.1.1 Defining a Class

2.7.2 __init__method in Class:

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

CHAPTER 3

CASE STUDY

3.1 PROBLEM STATEMENT:

Gender Recognition by Voice: This database was created to identify a voice as male or female, based upon acoustic properties of the voice and speech. The dataset consists of 3,168 recorded voice samples, collected from male and female speakers

3.2 DATA SET:

The given data set consists of the following parameters:

Meanfreq	: mean frequency (in kHz)
Sd	: standard deviation of frequency
median	: median frequency (in kHz)
Q25	: first quantile (in kHz)
Q75	: third quantile (in kHz)
IQR	: Interquartile range (in kHz)
skew	: skewness (see note in spec prop)
Kurt	: kurtosis (see note in spec propdescription)
spent	: spectral entropy
sfm	: spectral flatness
mode	: mode frequency

centroid	: frequency centroid (see spec prop)
peakf	: peak frequency
meanfun	: average of fundamental frequency
minfun	: minimum fundamental frequency
maxfun	: maximum fundamental frequency
meandsom	: average of dominant frequency
mindom	: minimum of dominant frequency
maxdom	: maximum of dominant frequency
dfrange	: range of dominant frequency
modindx	: modulation index.
label	: male or female

3.3 OBJECTIVE OF THE CASE STUDY:

This work aims to build a classifier for recognizing the gender of a given human voice. The gender voices, whether they are male or female, are different from each other by the signal energy and tune. Therefore, it is necessary to construct a classifier model to differentiate the male human voice from female voice because it is important in many applications as discussed before.

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 two sets, where 70% of volunteers was selected for generating the training data and 30% the test data.

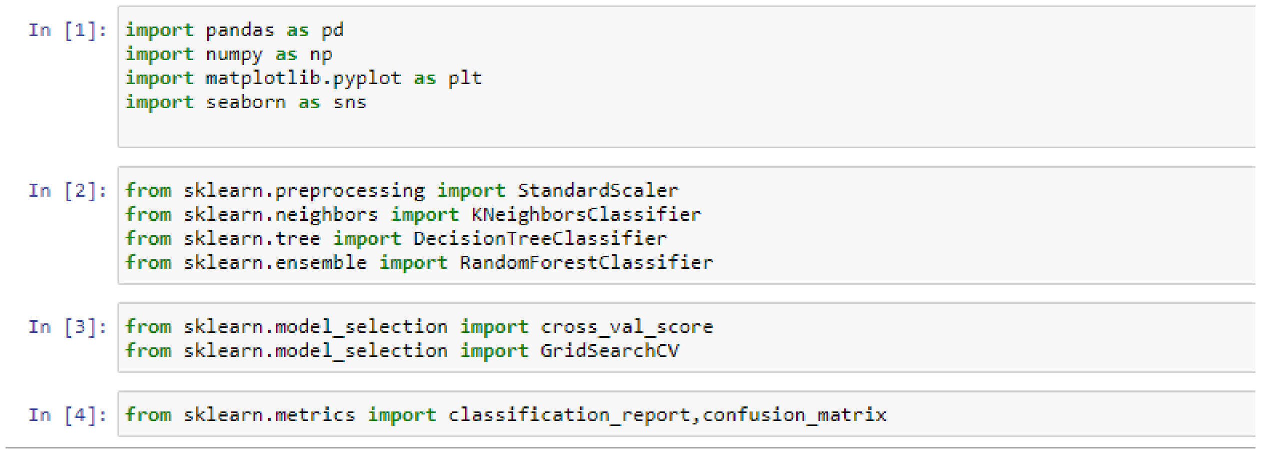
https://www.kaggle.com/primaryobjects/voicegender?select=voice.csv

4.1.2 IMPORTING THE LIBRARIES:

- NumPy package can be used to perform mathematical operations like 'mean'.
- pandas’ package can be used to process data frames.
- seaborn package can be used to visualize data in the form of various effective graphs and plots.
- sklearn is the main package which is used for machine learning.
- Label Encoder is used to encode the non-numeric data into numerical so that machine learning model can be built.
- train_test_split module is used to split the data into training and testing sets.
- Linear Regression module is used to fit a Linear Regression model.

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

IMPORTING LIBRARIES



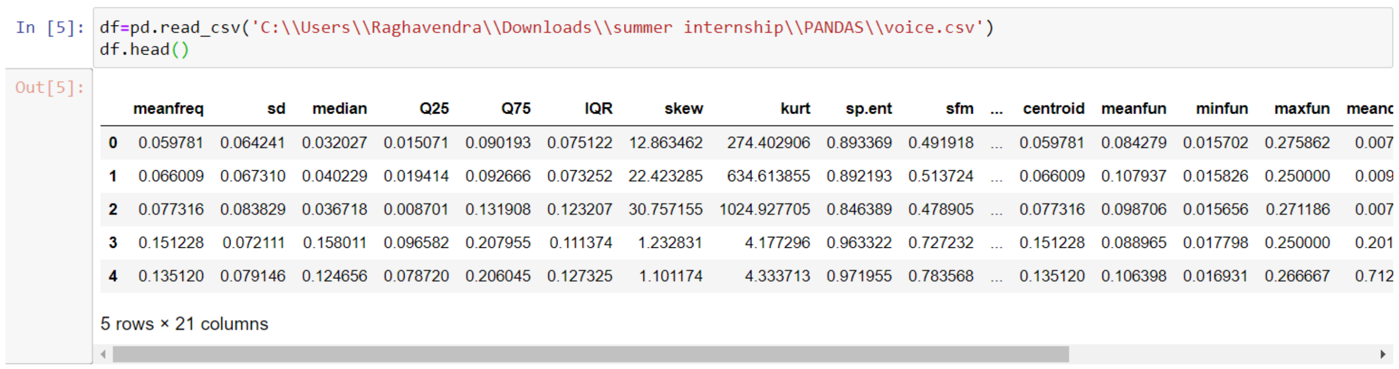
```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns

In [2]: from sklearn.preprocessing import StandardScaler
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier

In [3]: from sklearn.model_selection import cross_val_score
        from sklearn.model_selection import GridSearchCV

In [4]: from sklearn.metrics import classification_report,confusion_matrix
```

Figure 4.1.2.1 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 Data Frame 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 data frame. Any missing value or NaN value have to be cleaned.

READING THE DATA:



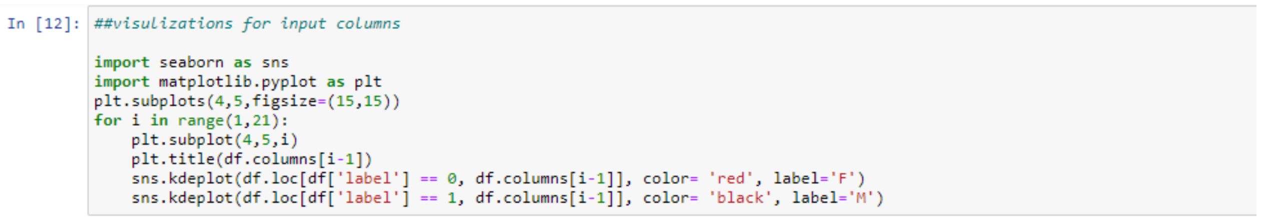
```
In [5]: df=pd.read_csv('C:\\Users\\Raghavendra\\Downloads\\summer internship\\PANDAS\\voice.csv')
        df.head()

Out[5]:
   meanfreq        sd    median       Q25       Q75       IQR       skew         kurt    sp.ent       sfm  ...  centroid   meanfun    minfun    maxfun  meanc
0  0.059781  0.064241  0.032027  0.015071  0.090193  0.075122  12.863462   274.402906  0.893369  0.491918  ...  0.059781  0.084279  0.015702  0.275862  0.007
1  0.066009  0.067310  0.040229  0.019414  0.092666  0.073252  22.423285   634.613855  0.892193  0.513724  ...  0.066009  0.107937  0.015826  0.250000  0.009
2  0.077316  0.083829  0.036718  0.008701  0.131908  0.123207  30.757155  1024.927705  0.846389  0.478905  ...  0.077316  0.098706  0.015656  0.271186  0.007
3  0.151228  0.072111  0.158011  0.096582  0.207955  0.111374   1.232831     4.177296  0.963322  0.727232  ...  0.151228  0.088965  0.017798  0.250000  0.201
4  0.135120  0.079146  0.124656  0.078720  0.206045  0.127325   1.101174     4.333713  0.971955  0.783568  ...  0.135120  0.106398  0.016931  0.266667  0.712

5 rows × 21 columns
```

Figure 4.1.3.1 Reading the dataset

4.1.4 VISUALIZING THE COLUMNS:

KDE Plot described as Kernel Density Estimate is used for visualizing the Probability Density of a continuous variable. It depicts the probability density at different values in a continuous variable. We can also plot a single graph for multiple samples which helps in more efficient data visualization**.**

KDE PLOT FOR INPUT COLUMNS:



```
In [12]: ##visulizations for input columns

         import seaborn as sns
         import matplotlib.pyplot as plt
         plt.subplots(4,5,figsize=(15,15))
         for i in range(1,21):
             plt.subplot(4,5,i)
             plt.title(df.columns[i-1])
             sns.kdeplot(df.loc[df['label'] == 0, df.columns[i-1]], color= 'red', label='F')
             sns.kdeplot(df.loc[df['label'] == 1, df.columns[i-1]], color= 'black', label='M')
```

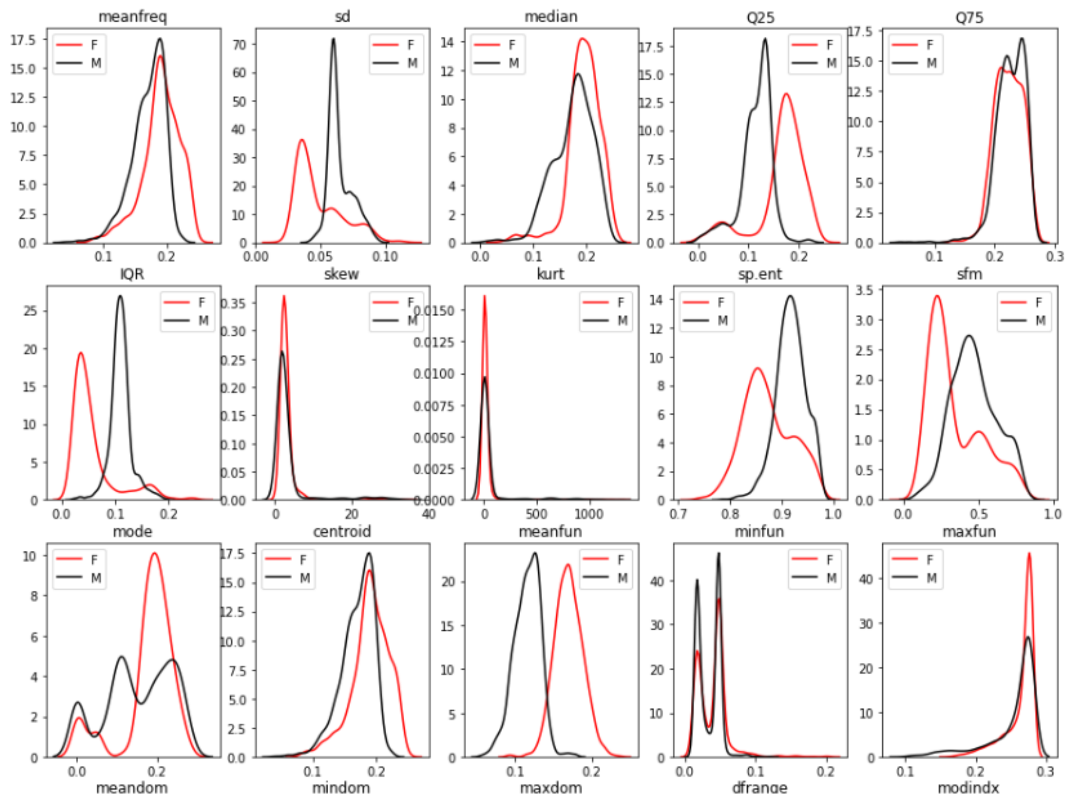


Figure 4.1.4.1 Visualizing the input columns

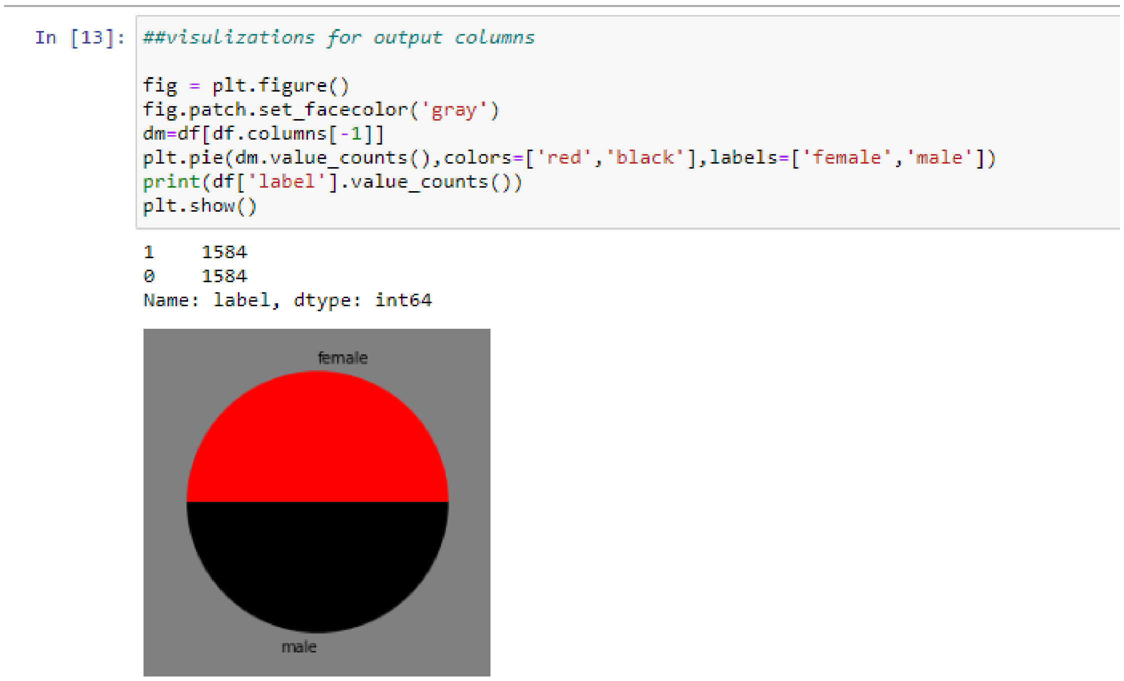
PIE PLOT

A Pie Chart can only display one series of data. Pie charts show the size of items (called wedge) in one data series, proportional to the sum of the items. The data points in a pie chart are shown as a percentage of the whole pie.

Matplotlib API has a pie() function that generates a pie diagram representing data in an array. The fractional area of each wedge is given by x/sum(x). If sum(x)< 1, then the values of x give the fractional area directly and the array will not be normalized. There resulting pie will have an empty wedge of size 1 - sum(x).

The pie chart looks best if the figure and axes are square, or the Axes aspect is equal.

PIE PLOT FOR OUTPUT COLUMNS



```
In [13]: ##visulizations for output columns

         fig = plt.figure()
         fig.patch.set_facecolor('gray')
         dm=df[df.columns[-1]]
         plt.pie(dm.value_counts(),colors=['red','black'],labels=['female','male'])
         print(df['label'].value_counts())
         plt.show()
```

```
1    1584
0    1584
Name: label, dtype: int64
```

Figure 4.1.4.2 Visualizing the Output columns

4.1.5 HANDLING MISSING VALUES:

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

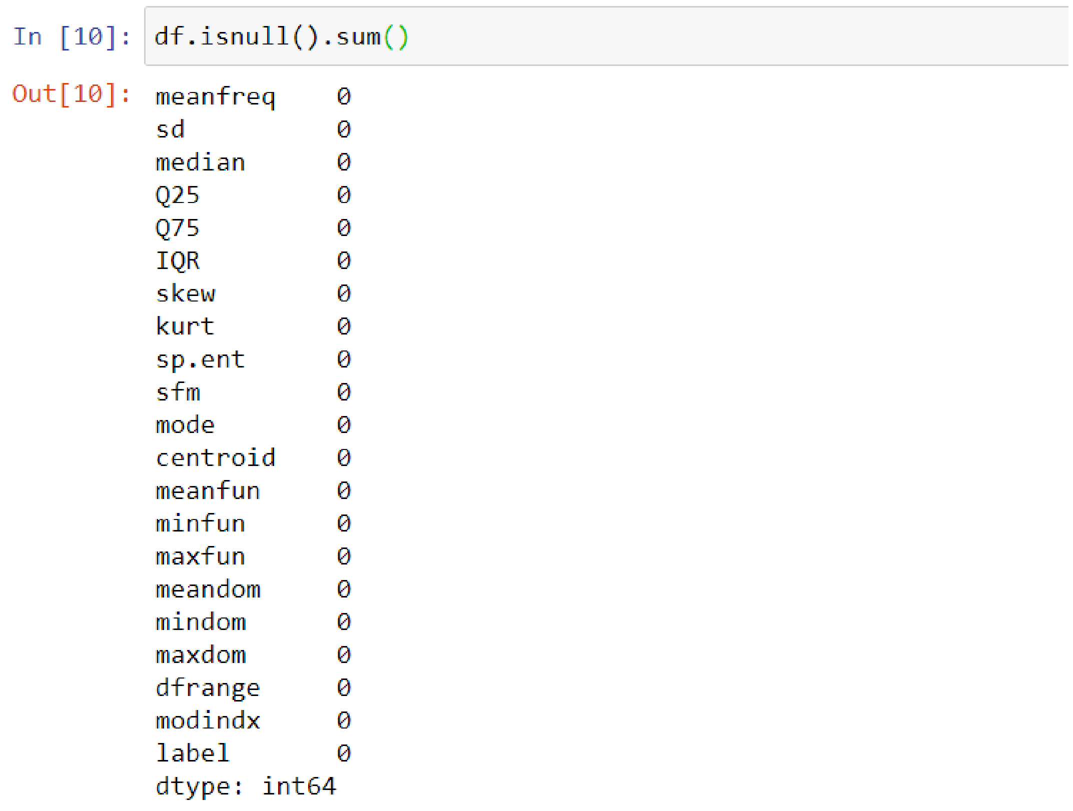
(a)dropna ()

(b)fillna ()

(c)interpolate ()

(d)mean imputation and median imputation

No null values in the Dataset



```
In [10]: df.isnull().sum()

Out[10]: meanfreq    0
         sd          0
         median      0
         Q25         0
         Q75         0
         IQR         0
         skew        0
         kurt        0
         sp.ent      0
         sfm         0
         mode        0
         centroid    0
         meanfun     0
         minfun      0
         maxfun      0
         meandom     0
         mindom      0
         maxdom      0
         dfrange     0
         modindx     0
         label       0
         dtype: int64
```

Figure 4.1.5.1 is null() values

Visualizing the null values by using Heatmap

```
In [16]: ## visualization for null values
         sns.heatmap(df.isnull(),cmap='autumn')

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x2008ff4f888>
```

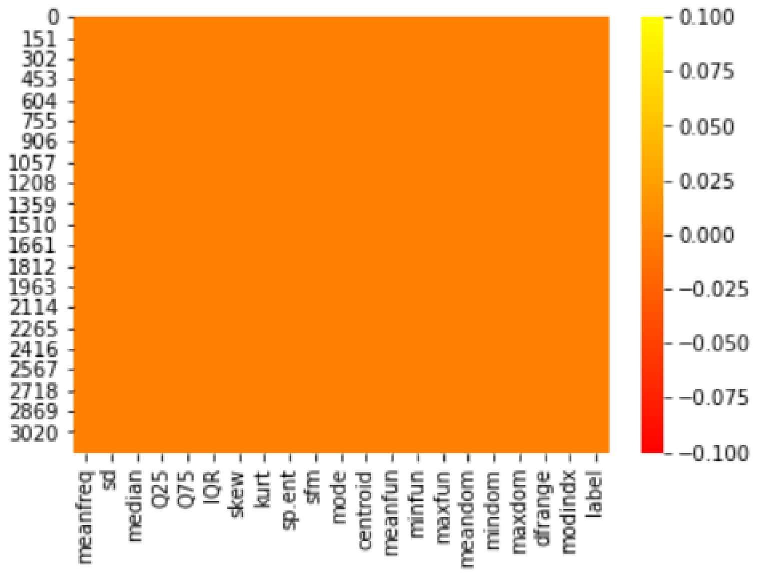


Figure 4.1.5.2 null values using is null ()

When we used dropna () the dataset didn’t change and displayed same number of columns, stating that there are no missing values in the data set.

So, when we tried to print null values it says

“Null values present in the training data: False”, that there are no nulvalues

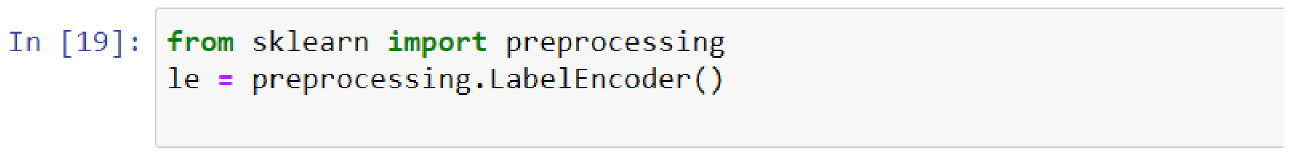
4.1.6 CATEGORICAL DATA:

- 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 panda's 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.

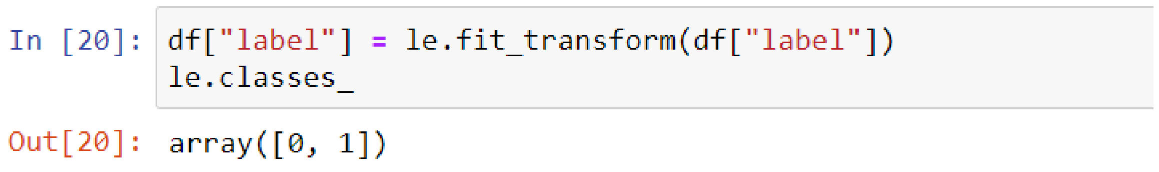
  Once these dummies got created, we have to concat this dummy set to our data frame or we can add that dummy set to the data frame.

Variables using Label Encoder:



```
In [19]: from sklearn import preprocessing
         le = preprocessing.LabelEncoder()
```

Figure 4.1.6.1 Importing the method



```
In [20]: df["label"] = le.fit_transform(df["label"])
         le.classes_

Out[20]: array([0, 1])
```

Figure 4.1.6.2 Transforming the data

4.2 TRAINING THE MODEL:

4.2.1: Splitting the Data

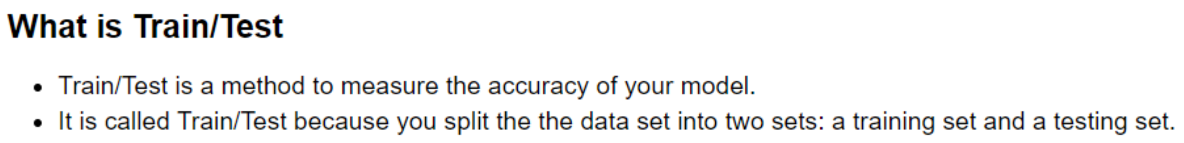
- Splitting the data : after the preprocessing is done then the data is split into train and test sets
- In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set'. An important 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)

Splitting the data into Train and Test

We can see that the dataset consists of accelerometer and gyroscope sensor values for each record.

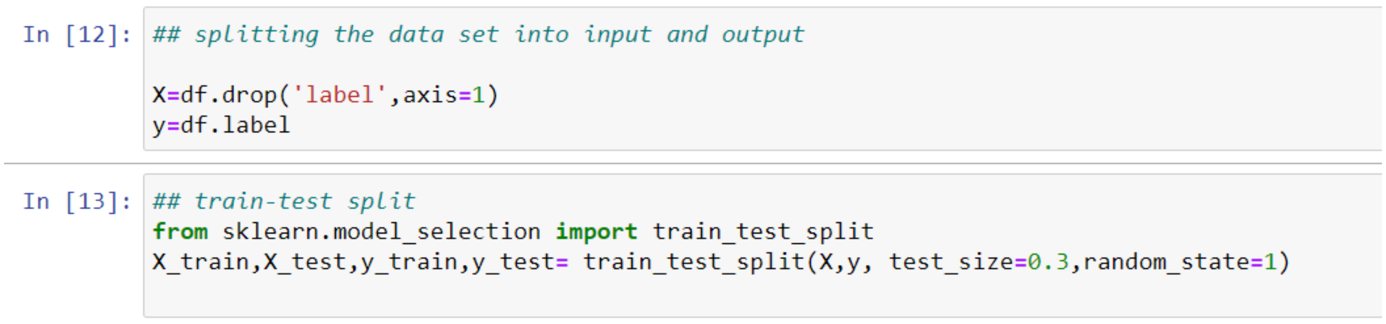
Further, the last two columns are subject which refers to subject number and Activity which defines the type of activity.

The Activity column acts as the label y and all the rest columns are features X.



What is Train/Test

- Train/Test is a method to measure the accuracy of your model.
- It is called Train/Test because you split the the data set into two sets: a training set and a testing set.

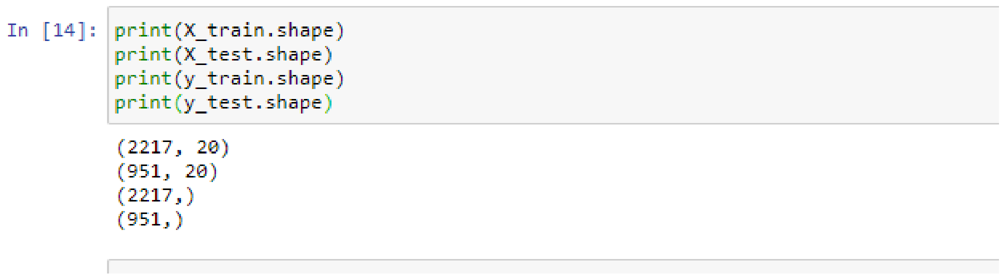


```
In [12]: ## splitting the data set into input and output

         X=df.drop('label',axis=1)
         y=df.label
```

```
In [13]: ## train-test split
         from sklearn.model_selection import train_test_split
         X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=0.3,random_state=1)
```

Here we drop the label column as if we look at the dataset, all the values in the label columns are categorical data(Male and Female).Hence we drop the label column to get remaining columns as input.

Figure 4.2.1.1 importing train_test_split



```
In [14]: print(X_train.shape)
         print(X_test.shape)
         print(y_train.shape)
         print(y_test.shape)

         (2217, 20)
         (951, 20)
         (2217,)
         (951,)
```

Figure 4.2.1.2 Shapes of both train and test data

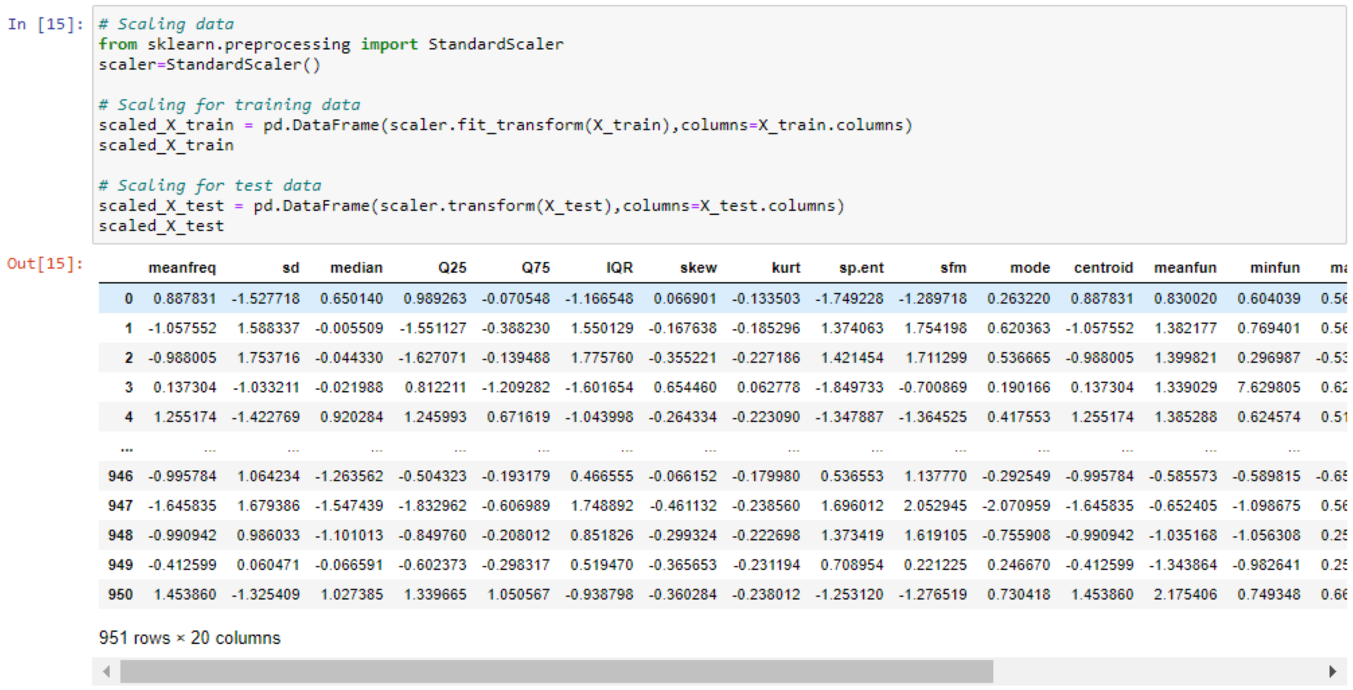
4.2.2 FEATURE SCALING

- **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

Import Standard Scalar method which is available in preprocessing package from scikit learn library



```
In [15]: # Scaling data
         from sklearn.preprocessing import StandardScaler
         scaler=StandardScaler()

         # Scaling for training data
         scaled_X_train = pd.DataFrame(scaler.fit_transform(X_train),columns=X_train.columns)
         scaled_X_train

         # Scaling for test data
         scaled_X_test = pd.DataFrame(scaler.transform(X_test),columns=X_test.columns)
         scaled_X_test
```

```
Out[15]:
      meanfreq        sd    median       Q25       Q75       IQR      skew      kurt    sp.ent       sfm      mode  centroid   meanfun    minfun   ma
  0   0.887831 -1.527718  0.650140  0.989263 -0.070548 -1.166548  0.066901 -0.133503 -1.749228 -1.289718  0.263220  0.887831  0.830020  0.604039  0.56
  1  -1.057552  1.588337 -0.005509 -1.551127 -0.388230  1.550129 -0.167638 -0.185296  1.374063  1.754198  0.620363 -1.057552  1.382177  0.769401  0.56
  2  -0.988005  1.753716 -0.044330 -1.627071 -0.139488  1.775760 -0.355221 -0.227186  1.421454  1.711299  0.536665 -0.988005  1.399821  0.296987 -0.53
  3   0.137304 -1.033211 -0.021988  0.812211 -1.209282 -1.601654  0.654460  0.062778 -1.849733 -0.700869  0.190166  0.137304  1.339029  7.629805  0.62
  4   1.255174 -1.422769  0.920284  1.245993  0.671619 -1.043998 -0.264334 -0.223090 -1.347887 -1.364525  0.417553  1.255174  1.385288  0.624574  0.51
...        ...       ...       ...       ...       ...       ...       ...       ...       ...       ...       ...       ...       ...       ...
946  -0.995784  1.064234 -1.263562 -0.504323 -0.193179  0.466555 -0.066152 -0.179980  0.536553  1.137770 -0.292549 -0.995784 -0.585573 -0.589815 -0.65
947  -1.645835  1.679386 -1.547439 -1.832962 -0.606989  1.748892 -0.461132 -0.238560  1.696012  2.052945 -2.070959 -1.645835 -0.652405 -1.098675  0.56
948  -0.990942  0.986033 -1.101013 -0.849760 -0.208012  0.851826 -0.299324 -0.222698  1.373419  1.619105 -0.755908 -0.990942 -1.035168 -1.056308  0.25
949  -0.412599  0.060471 -0.066591 -0.602373 -0.298317  0.519470 -0.365653 -0.231194  0.708954  0.221225  0.246670 -0.412599 -1.343864 -0.982641  0.25
950   1.453860 -1.325409  1.027385  1.339665  1.050567 -0.938798 -0.360284 -0.238012 -1.253120 -1.276519  0.730418  1.453860  2.175406  0.749348  0.66

951 rows × 20 columns
```

Figure 4.2.2.1 Scaling of the data

4.2.3 CLASSIFY ACTIVITIES

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.

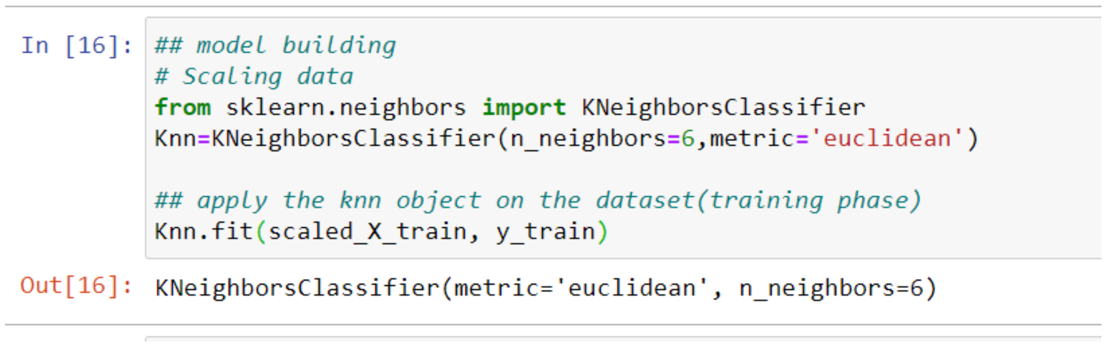
I am using three algorithms for my train and test data

4.2.3.1 K-Nearest Neighbors

K-nearest neighbours (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

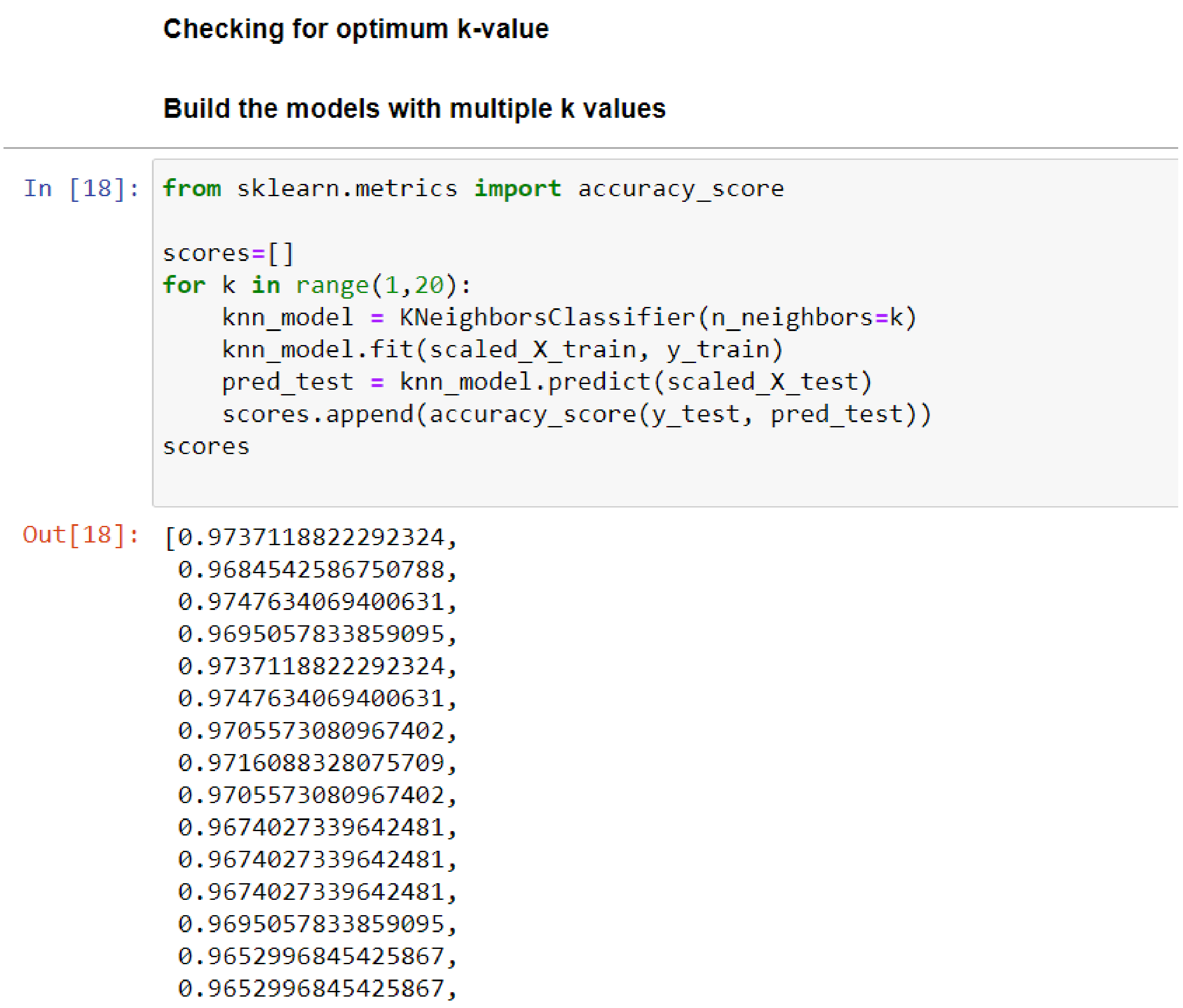
- **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.

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



```
In [16]: ## model building
         # Scaling data
         from sklearn.neighbors import KNeighborsClassifier
         Knn=KNeighborsClassifier(n_neighbors=6,metric='euclidean')

         ## apply the knn object on the dataset(training phase)
         Knn.fit(scaled_X_train, y_train)

Out[16]: KNeighborsClassifier(metric='euclidean', n_neighbors=6)
```

Figure 4.2.3.1.1 Fitting the scaled X_train and y_train data



Checking for optimum k-value

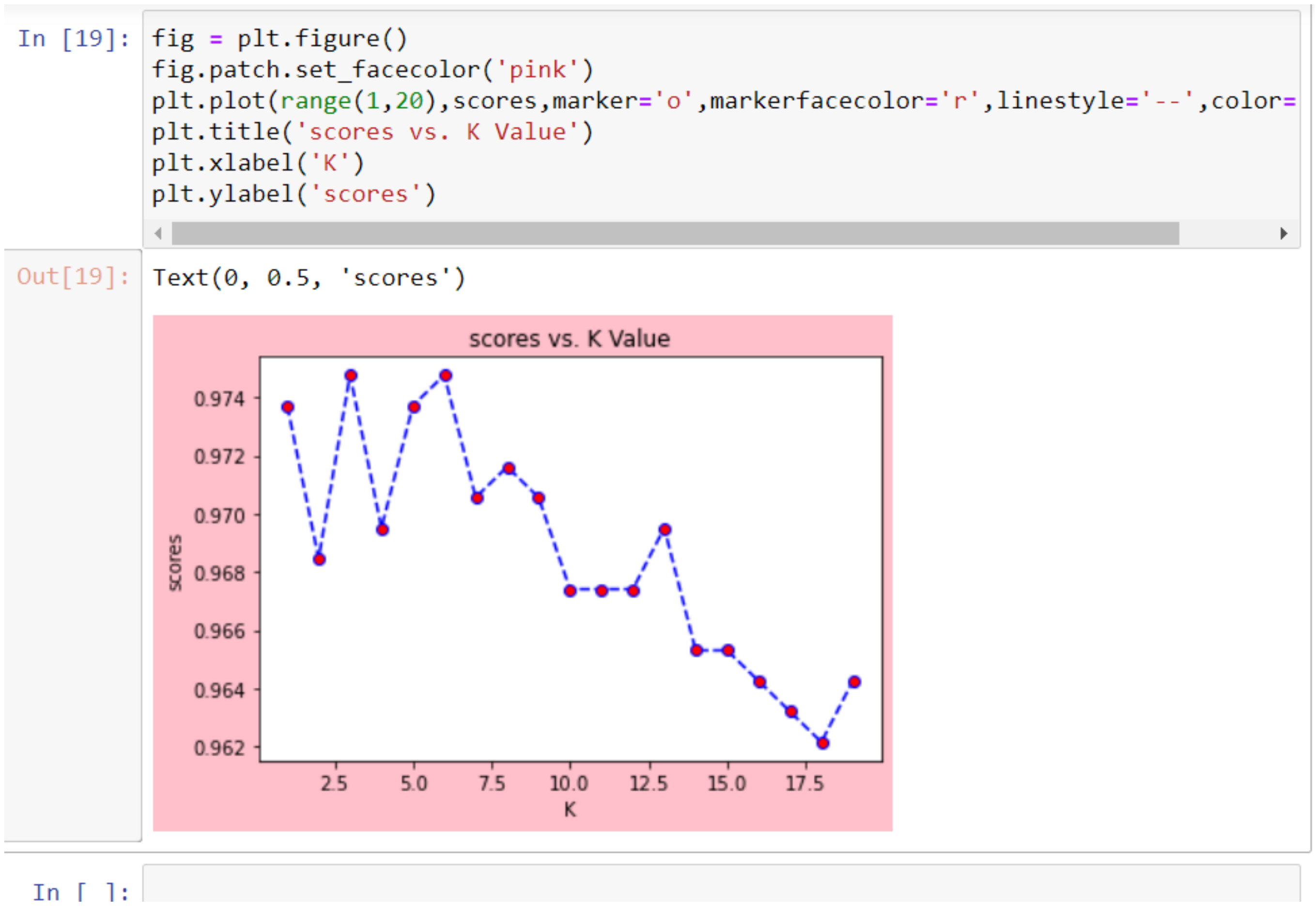
Build the models with multiple k values

```
In [18]: 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
```

```
Out[18]: [0.9737118822292324,
          0.9684542586750788,
          0.9747634069400631,
          0.9695057833859095,
          0.9737118822292324,
          0.9747634069400631,
          0.9705573080967402,
          0.9716088328075709,
          0.9705573080967402,
          0.9674027339642481,
          0.9674027339642481,
          0.9674027339642481,
          0.9695057833859095,
          0.9652996845425867,
          0.9652996845425867,
```

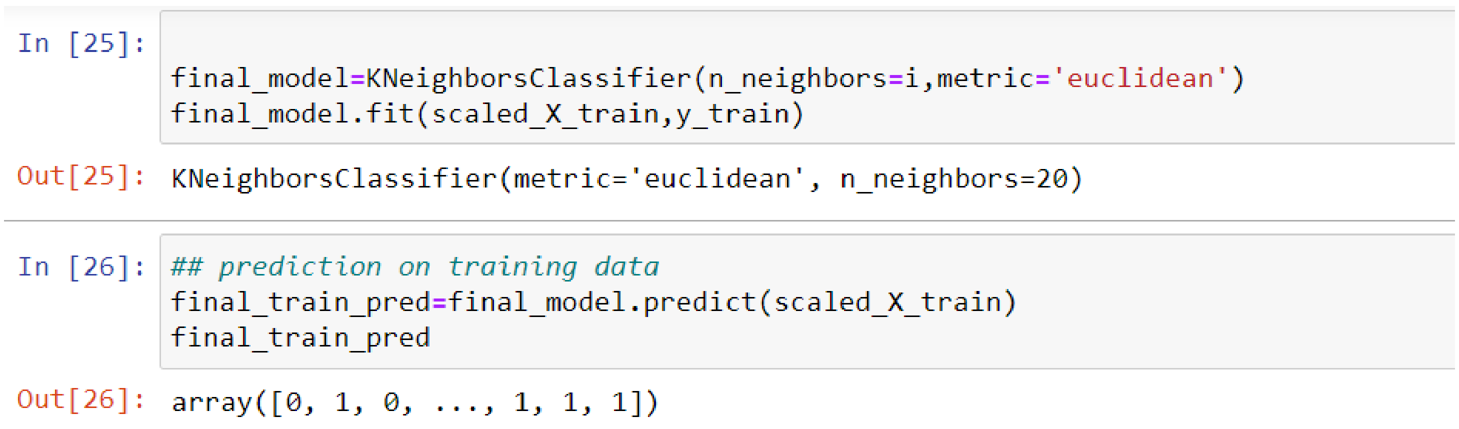
Figure 4.2.3.1.2 Accuracy scores

PLOTING:


```
In [19]: fig = plt.figure()
         fig.patch.set_facecolor('pink')
         plt.plot(range(1,20),scores,marker='o',markerfacecolor='r',linestyle='--',color=
         plt.title('scores vs. K Value')
         plt.xlabel('K')
         plt.ylabel('scores')

Out[19]: Text(0, 0.5, 'scores')
```

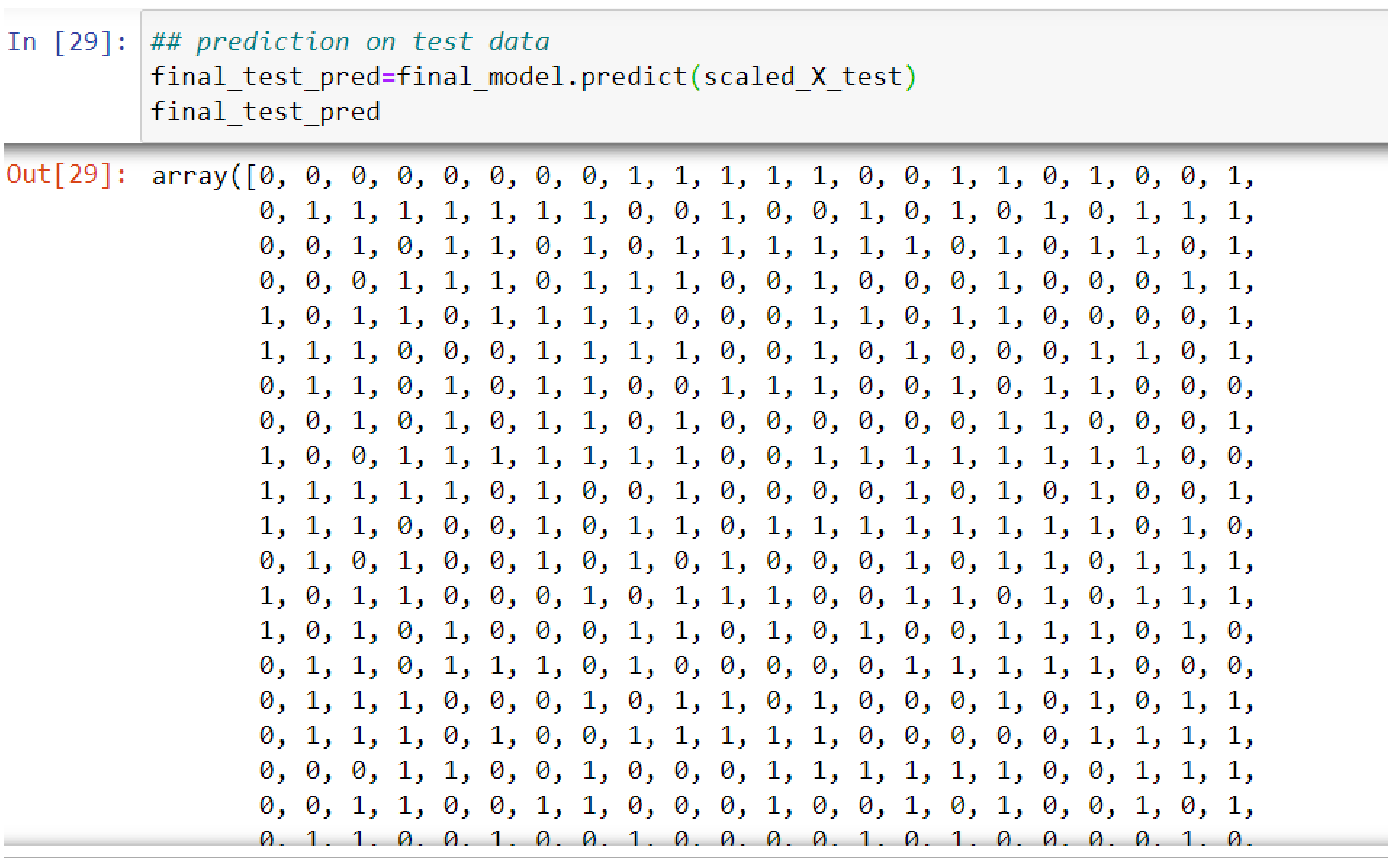
```
In [ ]:
```

Figure 4.2.3.1.3 Plotting the chart between scores and k values



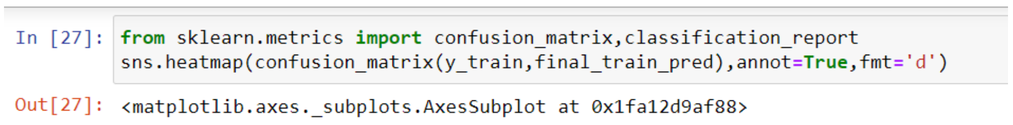
```
In [25]:
         final_model=KNeighborsClassifier(n_neighbors=i,metric='euclidean')
         final_model.fit(scaled_X_train,y_train)

Out[25]: KNeighborsClassifier(metric='euclidean', n_neighbors=20)

In [26]: ## prediction on training data
         final_train_pred=final_model.predict(scaled_X_train)
         final_train_pred

Out[26]: array([0, 1, 0, ..., 1, 1, 1])
```

Figure 4.2.3.1.4 **Prediction on train data**



```
In [29]: ## prediction on test data
         final_test_pred=final_model.predict(scaled_X_test)
         final_test_pred

Out[29]: array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
                0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
                0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1,
                0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1,
                1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
                1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1,
                0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
                0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
                1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1,
                1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
                0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1,
                1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
                1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0,
                0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0,
                0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
                0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1,
                0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
                0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1,
                0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0,
```

Figure 4.2.3.1.5 **Prediction on test data**



```
In [27]: from sklearn.metrics import confusion_matrix,classification_report
         sns.heatmap(confusion_matrix(y_train,final_train_pred),annot=True,fmt='d')

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1fa12d9af88>
```

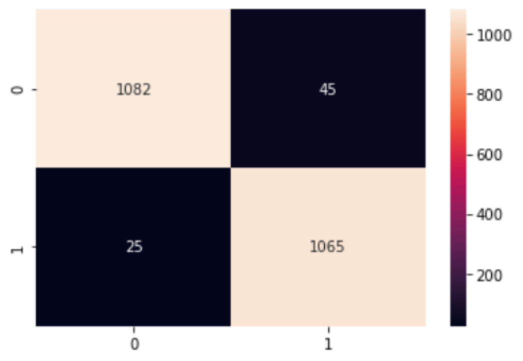
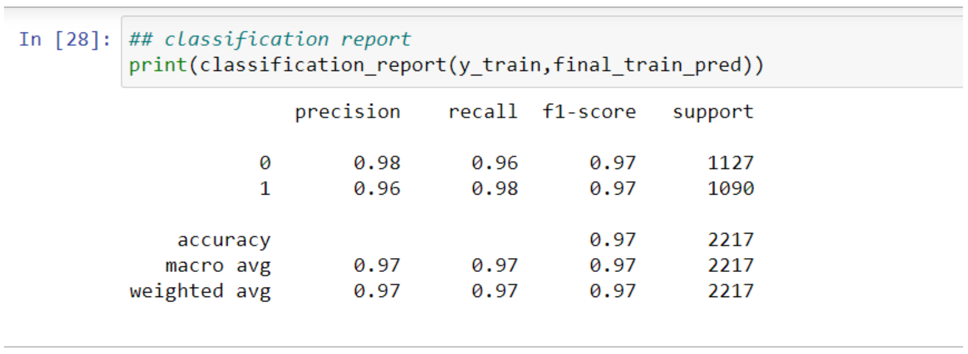


Figure 4.2.3.1.6 Visualization of Confusion Matrix



```
In [28]: ## classification report
         print(classification_report(y_train,final_train_pred))

              precision    recall  f1-score   support

           0       0.98      0.96      0.97      1127
           1       0.96      0.98      0.97      1090

    accuracy                           0.97      2217
   macro avg       0.97      0.97      0.97      2217
weighted avg       0.97      0.97      0.97      2217
```

Figure 4.2.3.1.7 Classification report on training data

Using K-Nearest Neighbors Classifier the accuracy for train data is 97%

PRECISION-RECALL:

Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced. In information retrieval, precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned.

The precision-recall curve shows the trade-off between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

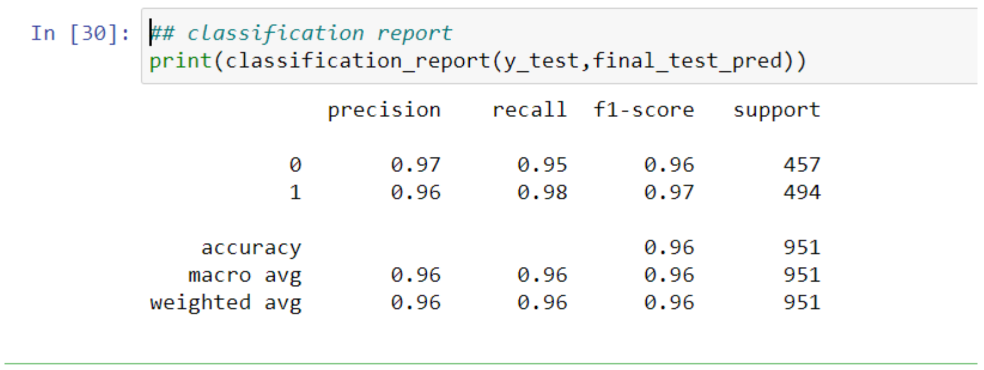
$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{True\ Positive}{Total\ Predicted\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{True\ Positive}{Total\ Actual\ Positive}$$

F1 score:

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

F1 = 2 * (precision * recall) / (precision + recall)



```
In [30]: ## classification report
         print(classification_report(y_test,final_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       457
           1       0.96      0.98      0.97       494

    accuracy                           0.96       951
   macro avg       0.96      0.96      0.96       951
weighted avg       0.96      0.96      0.96       951
```

Figure 4.2.3.1.8 Classification report on testing data

Using K-Nearest Neighbors Classifier the accuracy for test data is 96%

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

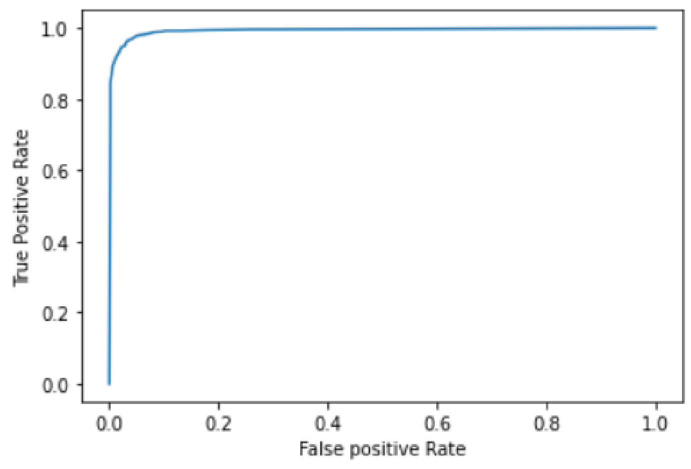
An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

```
In [98]: knn_model.predict_proba(X_test)

Out[98]: array([[0.57894737, 0.42105263],
                [0.        , 1.        ],
                [0.05263158, 0.94736842],
                ...,
                [0.52631579, 0.47368421],
                [0.42105263, 0.57894737],
                [0.52631579, 0.47368421]])

In [99]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         roc_auc = roc_auc_score(y_test,knn_model.predict(X_test))
         fpr,tpr,thresholds=roc_curve(y_test,knn_model.predict_proba(X_test) [:,1])
```

Figure 4.2.3.1.9 Importing AUC-ROC

```
In [69]: plt.figure()
         plt.plot(fpr,tpr,roc_auc)
         plt.xlabel('False positive Rate')
         plt.ylabel('True Positive Rate')
         plt.show()
```



```
In [70]: roc_auc

Out[70]: 0.9637022829755755
```

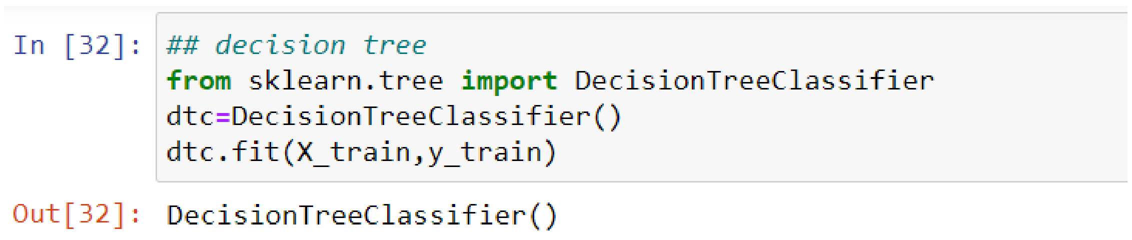
Figure 4.2.3.1.9.1 Plotting for AUC_ROC curve for Knn classifier

4.2.3.2 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.

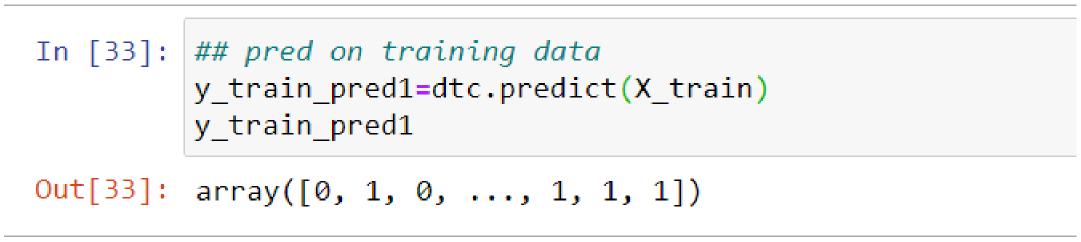
  The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions tree in recursively manner call recursive partitioning.

- 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.



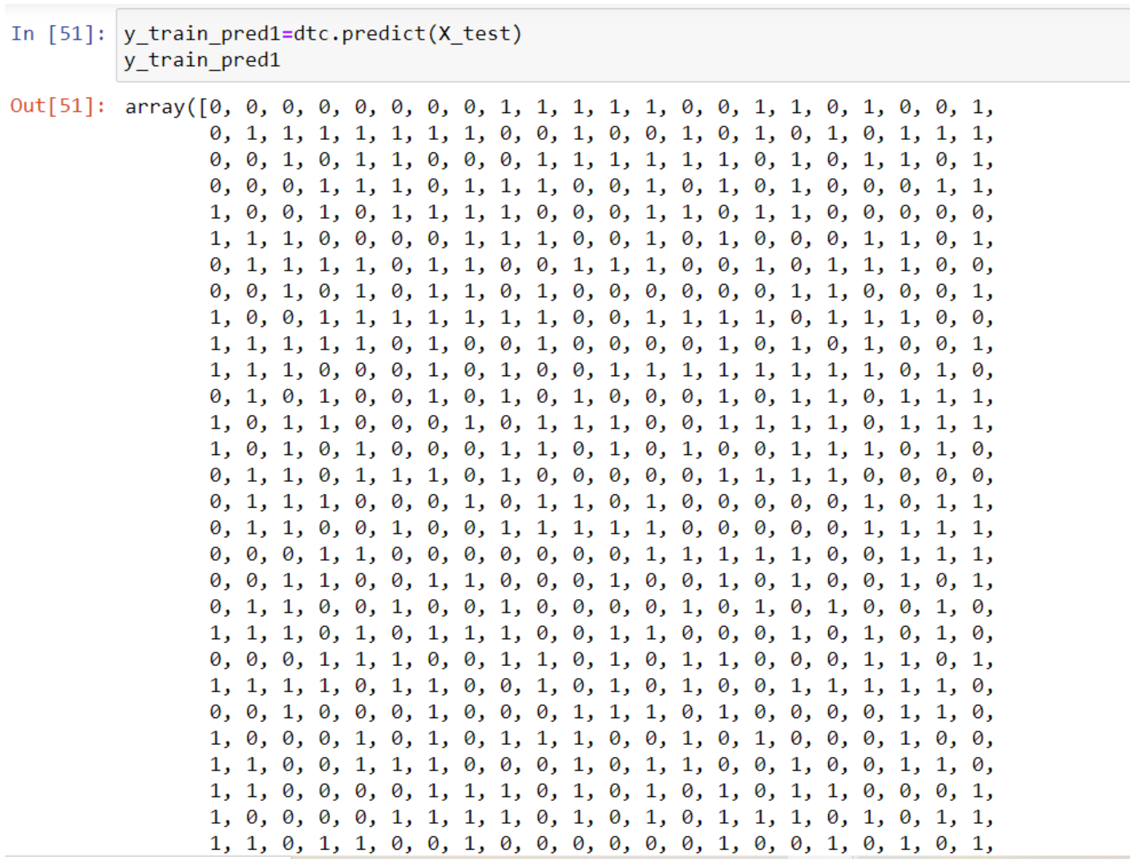
```
In [32]: ## decision tree
         from sklearn.tree import DecisionTreeClassifier
         dtc=DecisionTreeClassifier()
         dtc.fit(X_train,y_train)

Out[32]: DecisionTreeClassifier()
```

Figure 4.2.3.2.1 Importing the Decision Tree Classifier and fitting the train data



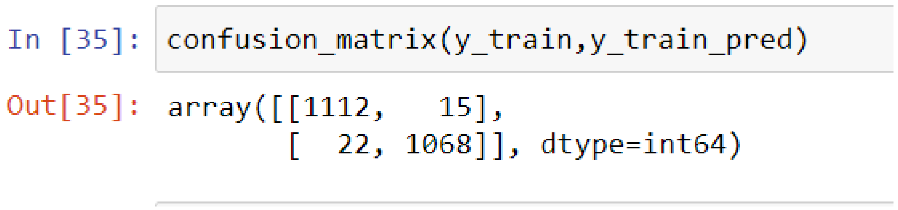
```
In [33]: ## pred on training data
         y_train_pred1=dtc.predict(X_train)
         y_train_pred1

Out[33]: array([0, 1, 0, ..., 1, 1, 1])
```

Figure 4.2.3.2.2 Prediction on training data



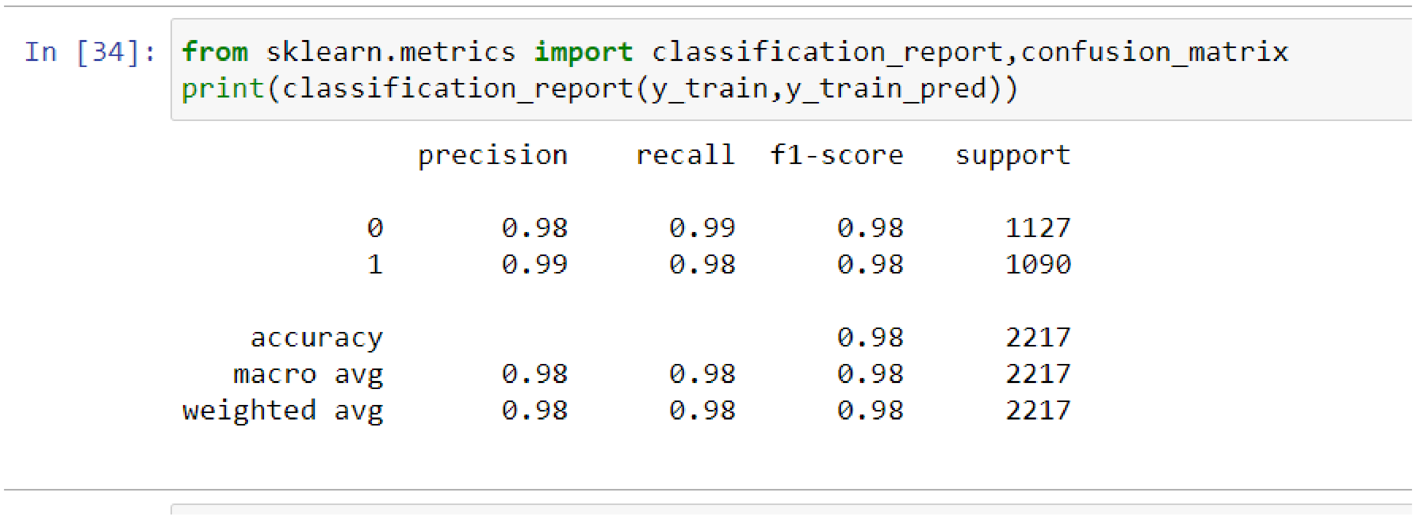
```
In [51]: y_train_pred1=dtc.predict(X_test)
         y_train_pred1

Out[51]: array([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
                0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
                0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1,
                0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1,
                1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
                1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1,
                0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
                0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
                1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0,
                1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1,
                1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
                0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1,
                1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
                1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0,
                0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
                0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1,
                0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1,
                0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
                0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1,
                0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
                1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
                0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1,
                1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0,
                0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0,
                1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
                1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0,
                1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1,
                1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
                1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
```

Figure 4.2.3.2.3 Prediction on Testing data



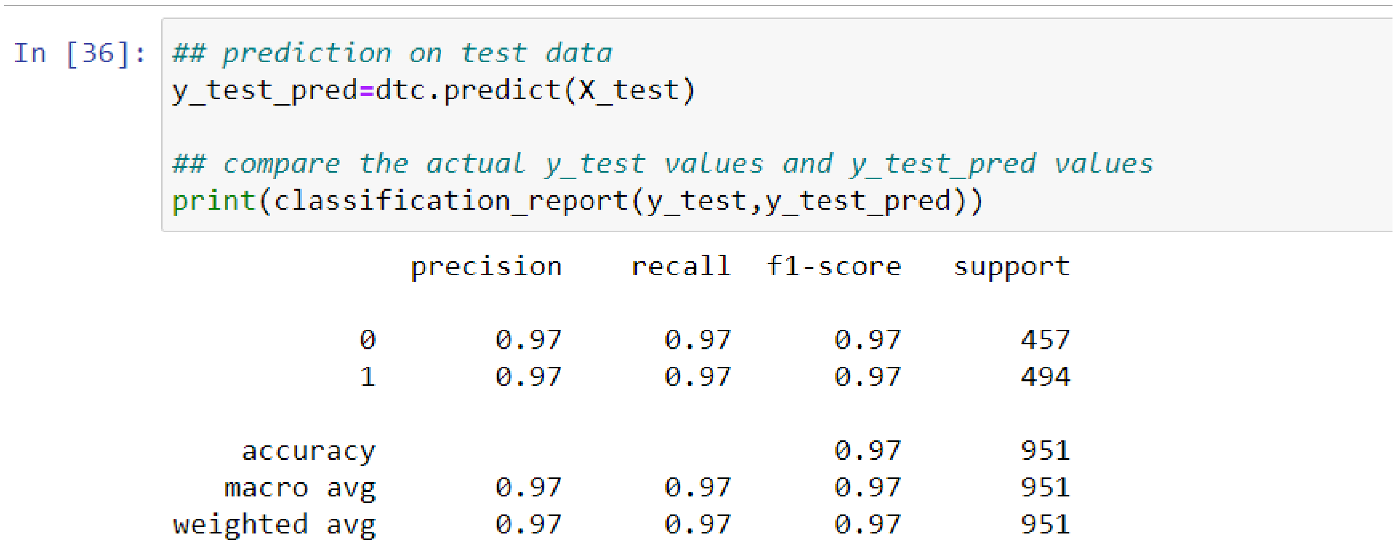
```
In [35]: confusion_matrix(y_train,y_train_pred)

Out[35]: array([[1112,   15],
                [  22, 1068]], dtype=int64)
```

Figure 4.2.3.2.4 Confusion matrix



```
In [34]: from sklearn.metrics import classification_report,confusion_matrix
         print(classification_report(y_train,y_train_pred))

                      precision    recall  f1-score   support

                   0       0.98      0.99      0.98      1127
                   1       0.99      0.98      0.98      1090

            accuracy                           0.98      2217
           macro avg       0.98      0.98      0.98      2217
        weighted avg       0.98      0.98      0.98      2217
```

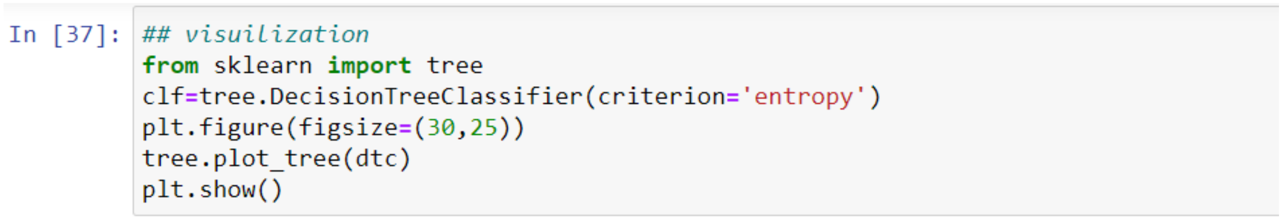
Figure 4.2.3.2.5 Classification report for Training data



```
In [36]: ## prediction on test data
         y_test_pred=dtc.predict(X_test)

         ## compare the actual y_test values and y_test_pred values
         print(classification_report(y_test,y_test_pred))

                      precision    recall  f1-score   support

                   0       0.97      0.97      0.97       457
                   1       0.97      0.97      0.97       494

            accuracy                           0.97       951
           macro avg       0.97      0.97      0.97       951
        weighted avg       0.97      0.97      0.97       951
```

Figure 4.2.3.2.6 Classification report for testing data

Visualization



```
In [37]: ## visuilization
         from sklearn import tree
         clf=tree.DecisionTreeClassifier(criterion='entropy')
         plt.figure(figsize=(30,25))
         tree.plot_tree(dtc)
         plt.show()
```

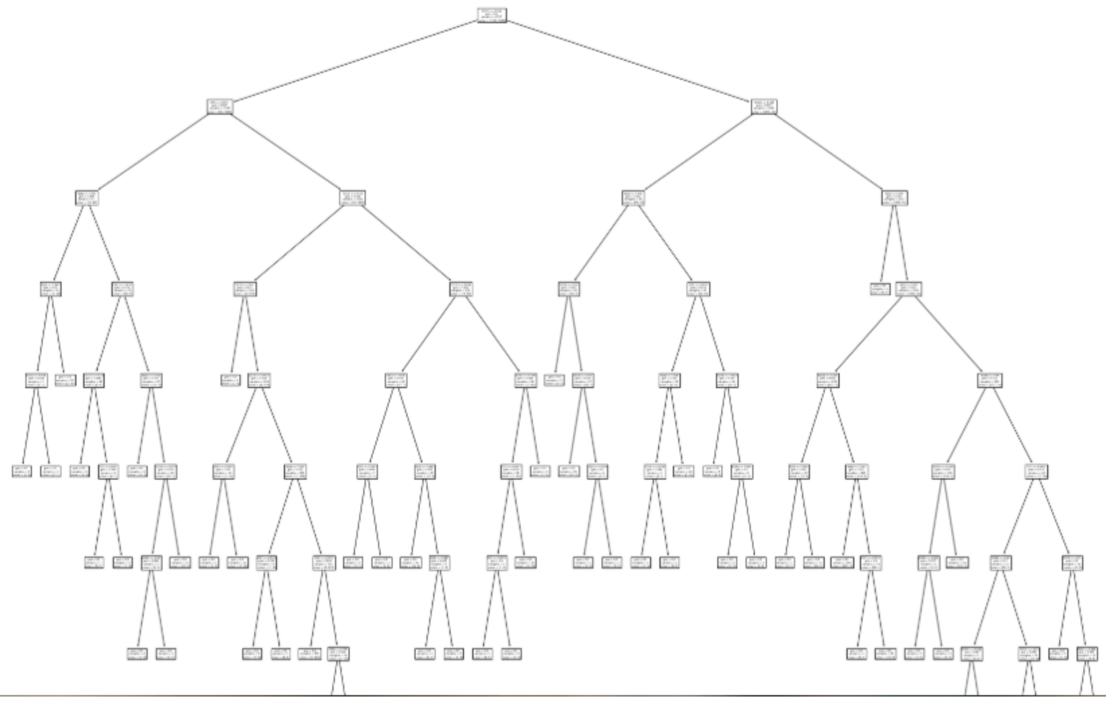
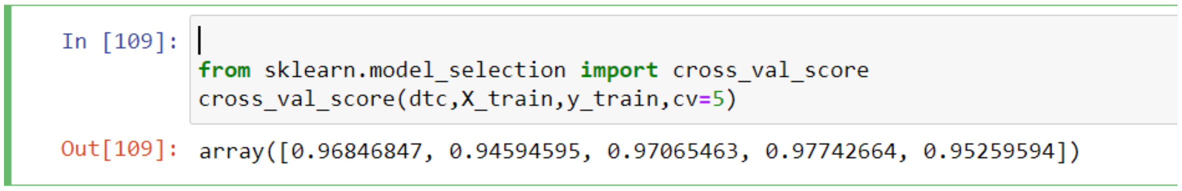


Figure 4.2.3.2.7 Visualizing the Decision Tree



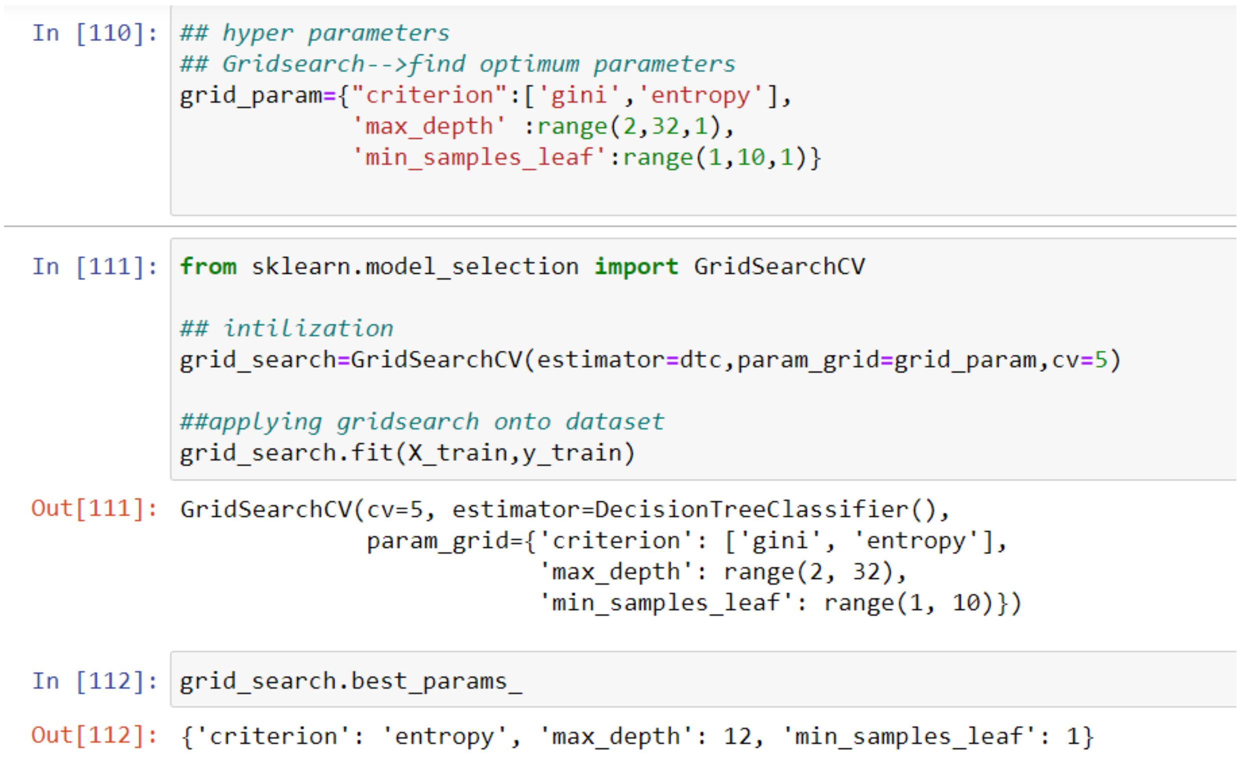
```
In [109]: from sklearn.model_selection import cross_val_score
          cross_val_score(dtc,X_train,y_train,cv=5)

Out[109]: array([0.96846847, 0.94594595, 0.97065463, 0.97742664, 0.95259594])
```

Figure 4.2.3.2.8 Cross validation score

To get the best result we use hyper parameters based on (decision tree classifier) called **Grid search CV**

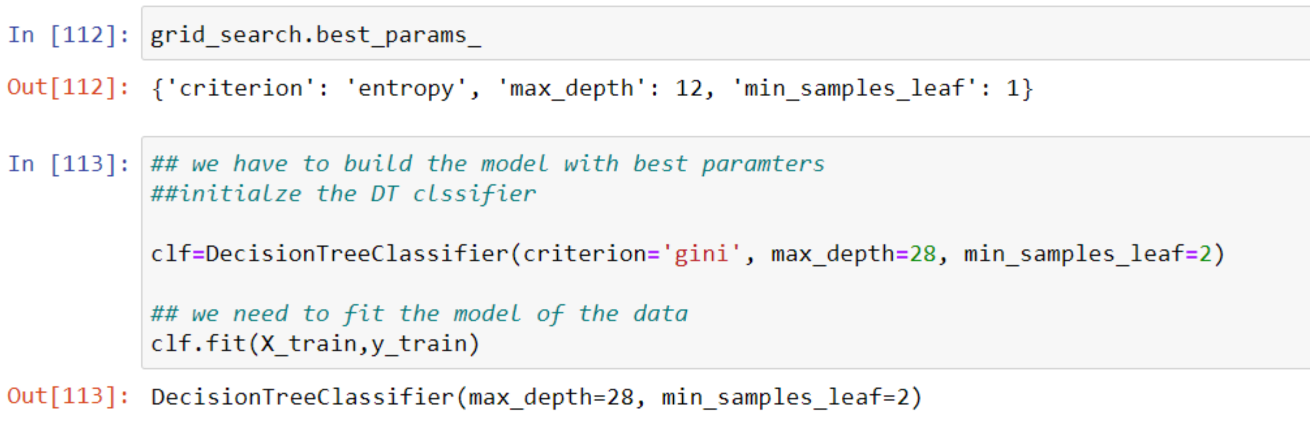
Grid Search CV

- 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



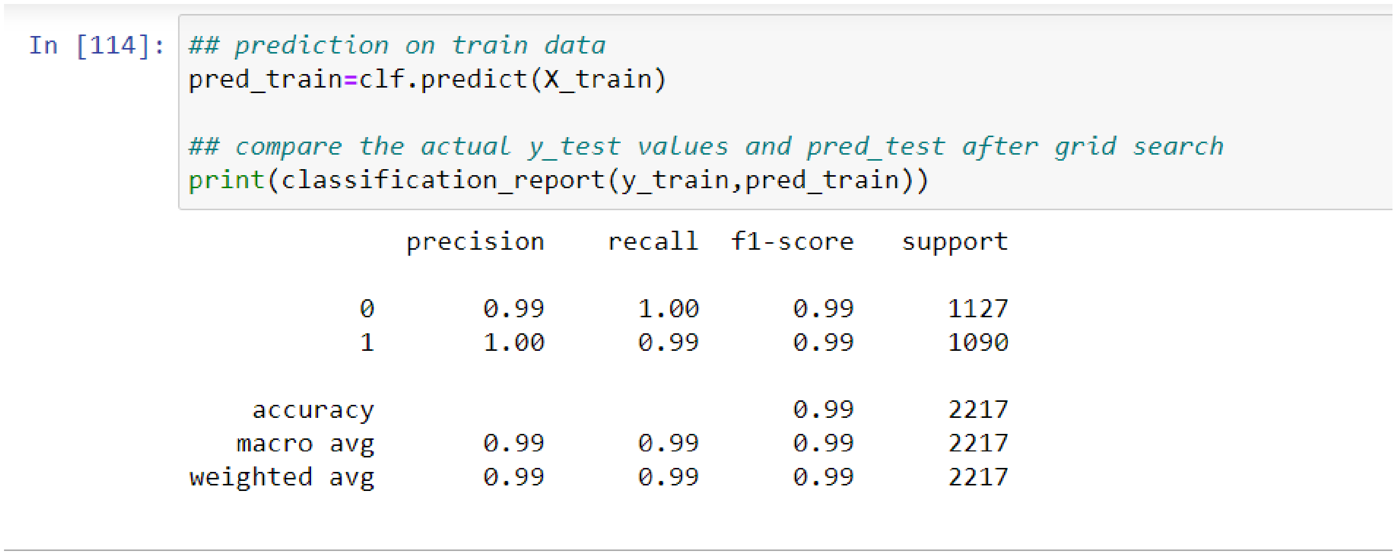
```
In [110]: ## hyper parameters
          ## Gridsearch-->find optimum parameters
          grid_param={"criterion":['gini','entropy'],
                      'max_depth' :range(2,32,1),
                      'min_samples_leaf':range(1,10,1)}

In [111]: from sklearn.model_selection import GridSearchCV

          ## intilization
          grid_search=GridSearchCV(estimator=dtc,param_grid=grid_param,cv=5)

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

Out[111]: GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
                       param_grid={'criterion': ['gini', 'entropy'],
                                   'max_depth': range(2, 32),
                                   'min_samples_leaf': range(1, 10)})

In [112]: grid_search.best_params_

Out[112]: {'criterion': 'entropy', 'max_depth': 12, 'min_samples_leaf': 1}
```

Figure 4.2.3.2.9 Applying the hyper parameter to get the optimum parameters



```
In [112]: grid_search.best_params_

Out[112]: {'criterion': 'entropy', 'max_depth': 12, 'min_samples_leaf': 1}

In [113]: ## we have to build the model with best paramters
          ##initialze the DT clssifier

          clf=DecisionTreeClassifier(criterion='gini', max_depth=28, min_samples_leaf=2)

          ## we need to fit the model of the data
          clf.fit(X_train,y_train)

Out[113]: DecisionTreeClassifier(max_depth=28, min_samples_leaf=2)
```

Figure 4.2.3.2.9.1 After applying the best parameters, fit the model

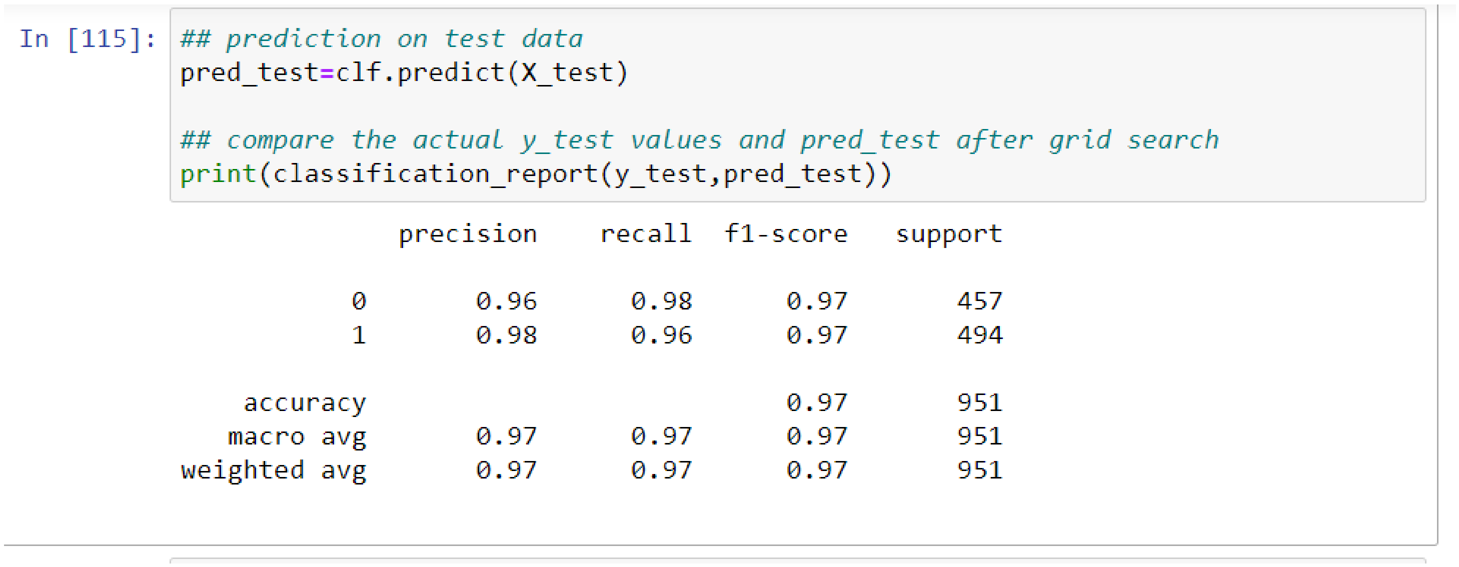


```
In [114]: ## prediction on train data
          pred_train=clf.predict(X_train)

          ## compare the actual y_test values and pred_test after grid search
          print(classification_report(y_train,pred_train))

                        precision    recall  f1-score   support

                     0       0.99      1.00      0.99      1127
                     1       1.00      0.99      0.99      1090

              accuracy                           0.99      2217
             macro avg       0.99      0.99      0.99      2217
          weighted avg       0.99      0.99      0.99      2217
```

Figure 4.2.3.2.9.2 Prediction on training data

- Predicting the train data and compare the actual y_train values of decision tree classifier and pred_train of grid search values

Using Grid search the accuracy for train data is 99%

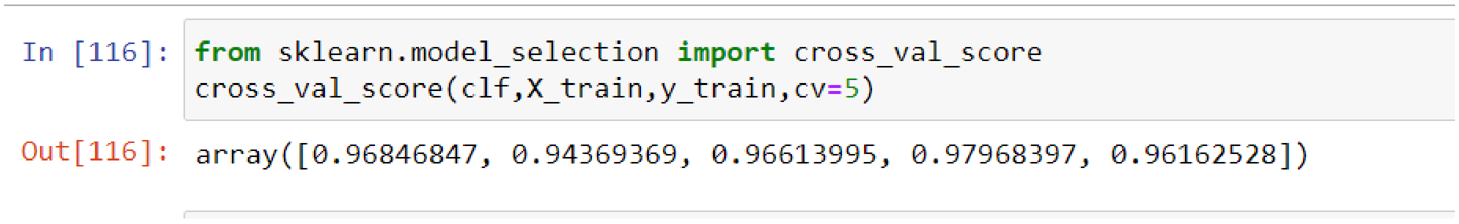


```
In [115]: ## prediction on test data
          pred_test=clf.predict(X_test)

          ## compare the actual y_test values and pred_test after grid search
          print(classification_report(y_test,pred_test))

                        precision    recall  f1-score   support

                     0       0.96      0.98      0.97       457
                     1       0.98      0.96      0.97       494

              accuracy                           0.97       951
             macro avg       0.97      0.97      0.97       951
          weighted avg       0.97      0.97      0.97       951
```

Figure 4.2.3.2.9.3 **Prediction on testing data**

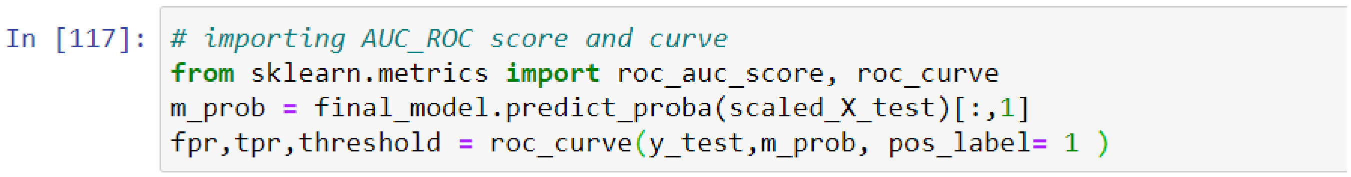
- Predicting the train data and compare the actual y_test values of decision tree classifier and pred_test of grid search values

Using Grid search the accuracy for test data is 97%



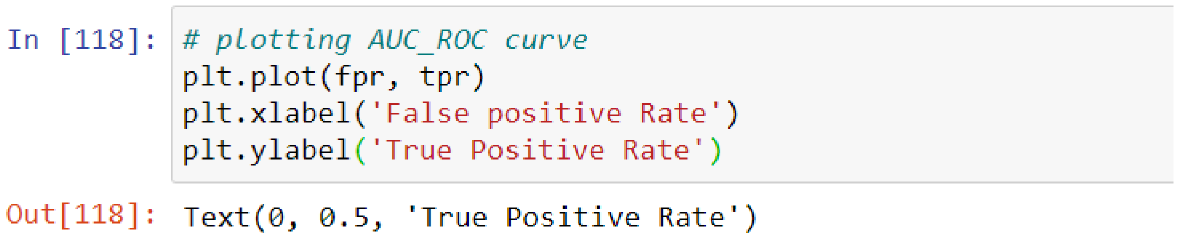
```
In [116]: from sklearn.model_selection import cross_val_score
          cross_val_score(clf,X_train,y_train,cv=5)

Out[116]: array([0.96846847, 0.94369369, 0.96613995, 0.97968397, 0.96162528])
```

Figure 4.2.3.2.9.4 **Cross validation score**



```
In [117]: # importing AUC_ROC score and curve
          from sklearn.metrics import roc_auc_score, roc_curve
          m_prob = final_model.predict_proba(scaled_X_test)[:,1]
          fpr,tpr,threshold = roc_curve(y_test,m_prob, pos_label= 1 )
```

Figure 4.2.3.2.9.5 Applying the AUC-ROC curve



```
In [118]: # plotting AUC_ROC curve
          plt.plot(fpr, tpr)
          plt.xlabel('False positive Rate')
          plt.ylabel('True Positive Rate')

Out[118]: Text(0, 0.5, 'True Positive Rate')
```

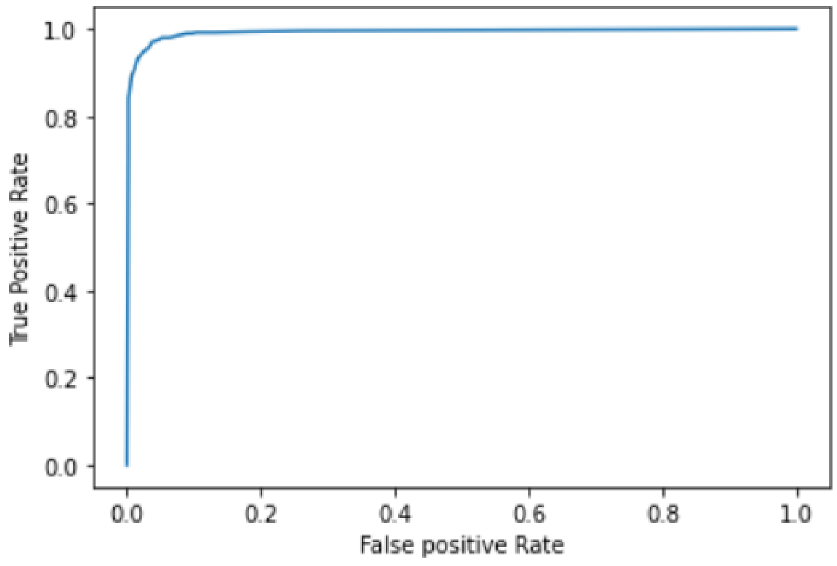
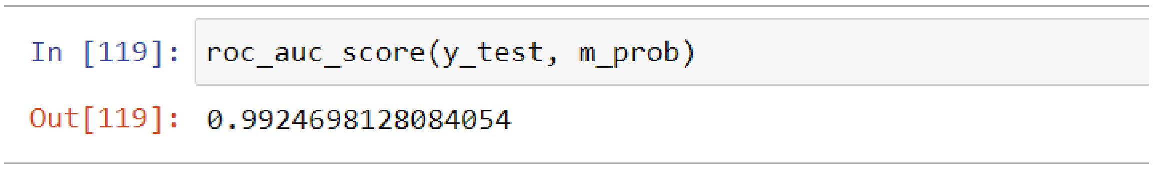


Figure 4.2.3.2.9.6 Plotting the AUC-ROC curve



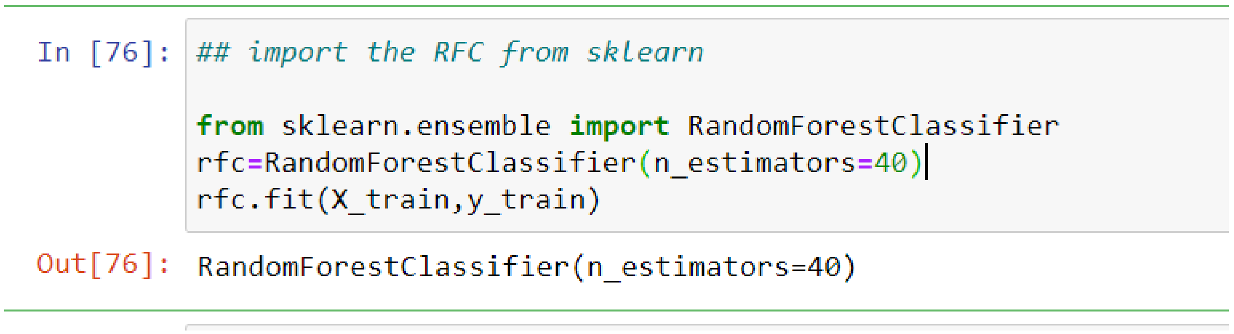
```
In [119]: roc_auc_score(y_test, m_prob)

Out[119]: 0.9924698128084054
```

Figure 4.2.3.2.9.7 AUC-ROC Score

Using Grid Search CV, we increased the accuracy rate to 99%

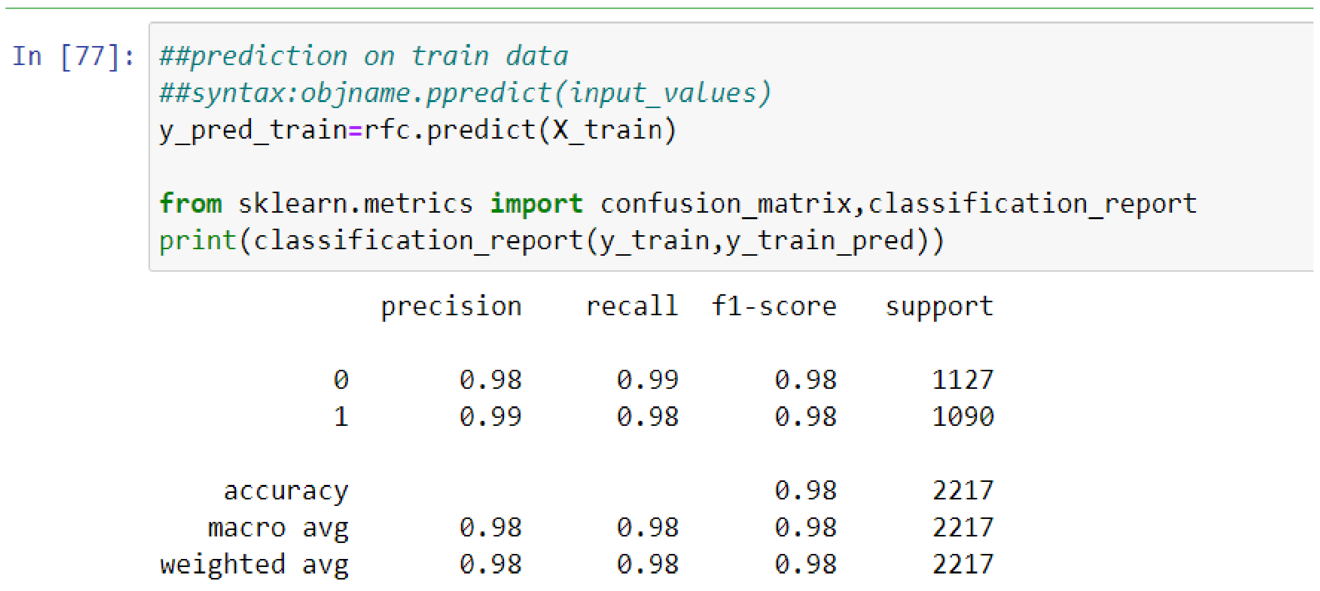
4.2.3.3 Random Forest Classification

- Random forest is a type of supervised machine learning algorithm based on ensemble learning. learning 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.
- Random forest is a supervised learning algorithm which is used for both classification as well as regression. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.



```
In [76]: ## import the RFC from sklearn

         from sklearn.ensemble import RandomForestClassifier
         rfc=RandomForestClassifier(n_estimators=40)
         rfc.fit(X_train,y_train)

Out[76]: RandomForestClassifier(n_estimators=40)
```

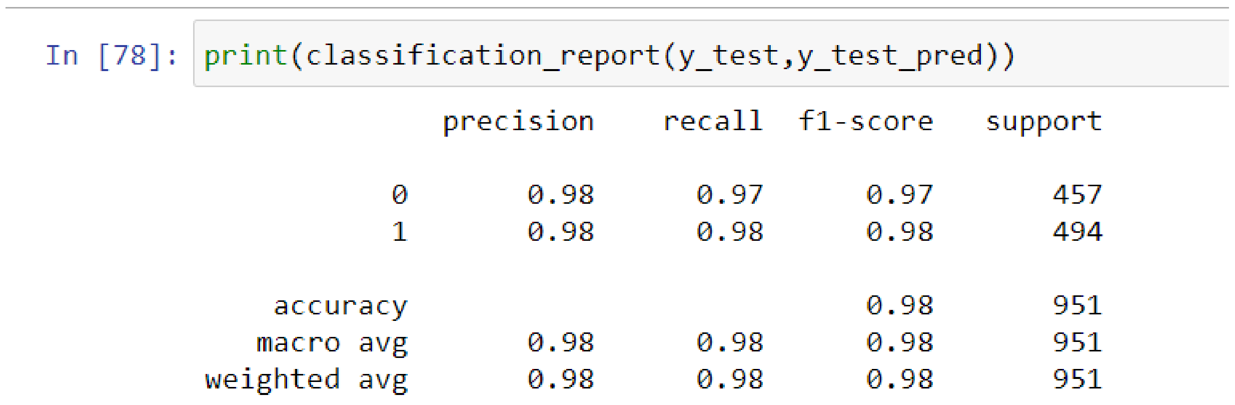
Figure 4.2.3.3.1 Importing RFC from sklearn



```
In [77]: ##prediction on train data
         ##syntax:objname.ppredict(input_values)
         y_pred_train=rfc.predict(X_train)

         from sklearn.metrics import confusion_matrix,classification_report
         print(classification_report(y_train,y_train_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      1127
           1       0.99      0.98      0.98      1090

    accuracy                           0.98      2217
   macro avg       0.98      0.98      0.98      2217
weighted avg       0.98      0.98      0.98      2217
```

Figure 4.2.3.3.2 Classification report on Training data



```
In [78]: print(classification_report(y_test,y_test_pred))
```

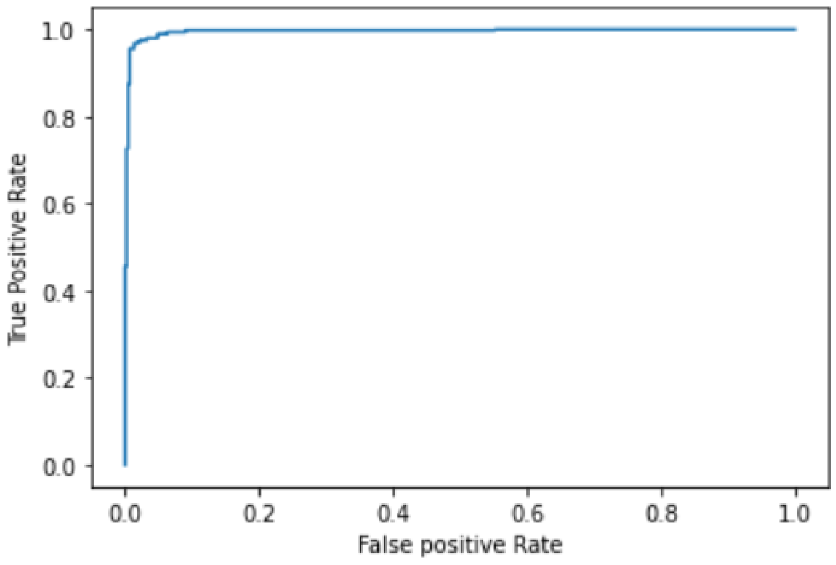
```
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       457
           1       0.98      0.98      0.98       494

    accuracy                           0.98       951
   macro avg       0.98      0.98      0.98       951
weighted avg       0.98      0.98      0.98       951
```

Figure 4.2.3.3.3 Classification report on Testing data

```
In [82]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve
         roc_auc = roc_auc_score(y_test,rfc.predict(X_test))
         fpr,tpr,thresholds=roc_curve(y_test,rfc.predict_proba(X_test) [:,1])
```

Figure 4.2.3.3.4 **Importing the AUC-ROC**

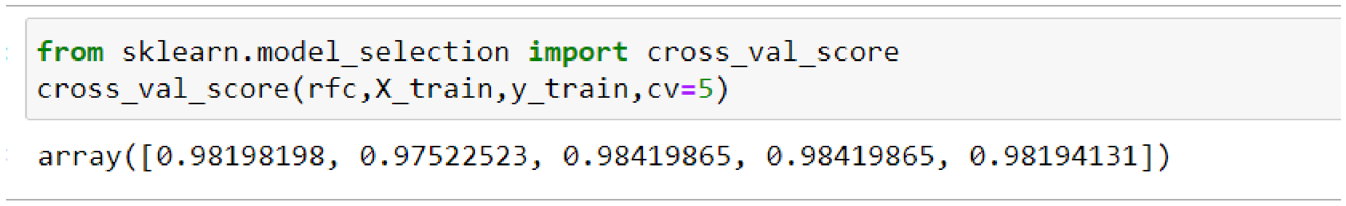
```
 : plt.figure()
   plt.plot(fpr,tpr,roc_auc)
   plt.xlabel('False positive Rate')
   plt.ylabel('True Positive Rate')
   plt.show()
```



```
roc_auc
```

```
0.982301845338814
```

Figure 4.2.3.3.5 **Plotting the AUC-ROC Curve & Value**



```
from sklearn.model_selection import cross_val_score
cross_val_score(rfc,X_train,y_train,cv=5)
```

```
array([0.98198198, 0.97522523, 0.98419865, 0.98419865, 0.98194131])
```

Figure 4.2.3.3.6 Cross Validation Score

4.3 EVALUATING THE CASE STUDY:

1.KNeighbors Classifier:

- Training accuracy =0.97
- Testing accuracy =0.96
- AUC_ROC -> =0.96

2.Decision Tree Classifier:

- Training accuracy =0.99
- Testing accuracy =0.97
- AUC_ROC -> =0.99

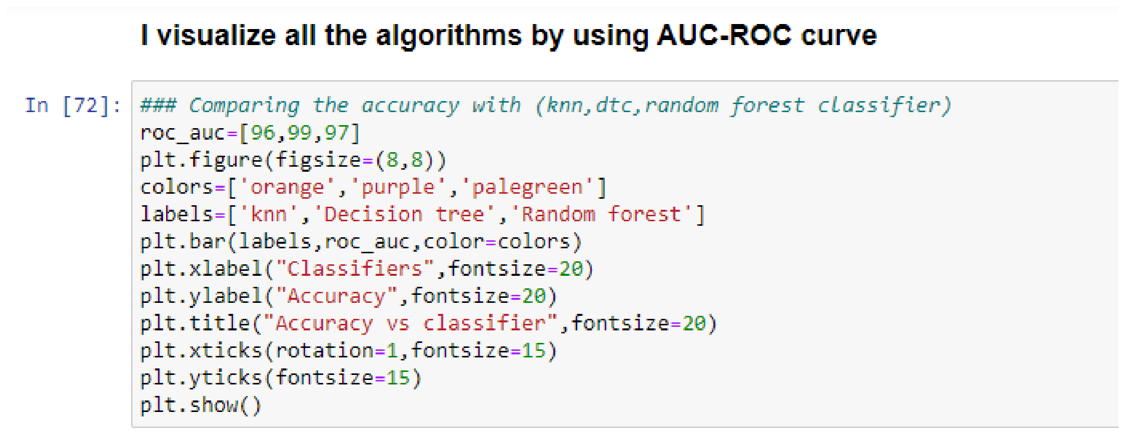
3.Random Forest Classifier:

- Training accuracy =0.98
- Testing accuracy =0.98
- AUC_ROC -> =0.97

From the above observation Decision tree classifier is best model to predict the given problem statement i.e, these are accuracy values for the given problem statement

We can see that the highest accuracy is 99.74% which is made by Decision Tree Classifier. Decision Tree Classifier is a powerful algorithm, and very popular in Data Science competition

I visualize all the algorithms by using AUC-ROC curve



I visualize all the algorithms by using AUC-ROC curve

```
In [72]: ### Comparing the accuracy with (knn,dtc,random forest classifier)
         roc_auc=[96,99,97]
         plt.figure(figsize=(8,8))
         colors=['orange','purple','palegreen']
         labels=['knn','Decision tree','Random forest']
         plt.bar(labels,roc_auc,color=colors)
         plt.xlabel("Classifiers",fontsize=20)
         plt.ylabel("Accuracy",fontsize=20)
         plt.title("Accuracy vs classifier",fontsize=20)
         plt.xticks(rotation=1,fontsize=15)
         plt.yticks(fontsize=15)
         plt.show()
```

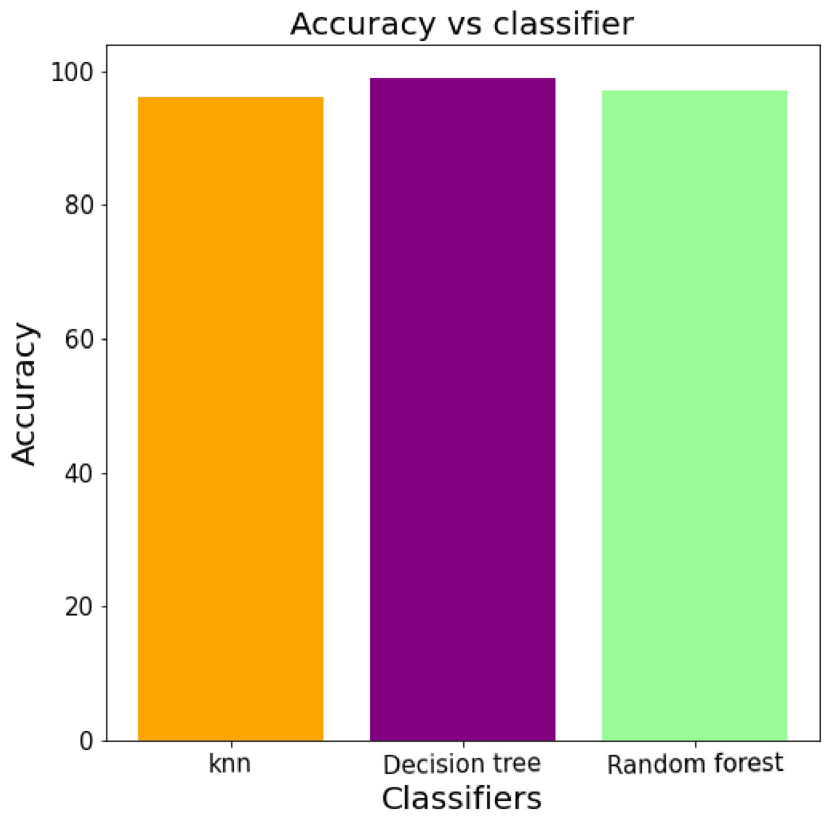
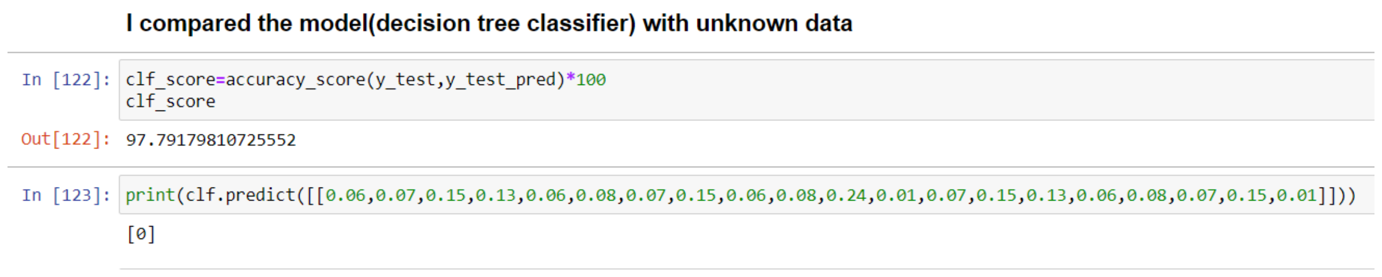


Figure 4.3.1 Visualization for all algorithms



I compared the model(decision tree classifier) with unknown data

```
In [122]: clf_score=accuracy_score(y_test,y_test_pred)*100
          clf_score

Out[122]: 97.79179810725552

In [123]: print(clf.predict([[0.06,0.07,0.15,0.13,0.06,0.08,0.07,0.15,0.06,0.08,0.24,0.01,0.07,0.15,0.13,0.06,0.08,0.07,0.15,0.01]]))
          [0]
```

Figure 4.3.1.2 Comparing the unknown data with Best accuracy model(DTC)

CONCLUSION:

In this particular project, I explored the activity recognition dataset. I visualized the data using matplotlib. Then, I applied numerous machine learning algorithms and found out that Decision Tree Classifier(Grid Search CV) performed the best in classifying different activities with accuracy of almost 99%.

REFERENCES:

https://www.kaggle.com/primaryobjects/voicegender?select=voice.csv

https://matplotlib.org/index.html

https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

GitHub Link:

https://github.com/raghavendra-1999/

Google sites:

https://sites.google.com/view/ai-and-ml-intern/home