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

NARRA SAI JEEVAN
221710305034

Under the Guidance of

Assistant Professor

GITAM
(DEEMED TO BE UNIVERSITY)
VISAKHAPATNAM • HYDERABAD • BENGALURU

Department Of Electronics and Communication Engineering
GITAM School of Technology
GITAM (Deemed to be University)
Hyderabad-502329
June 2020

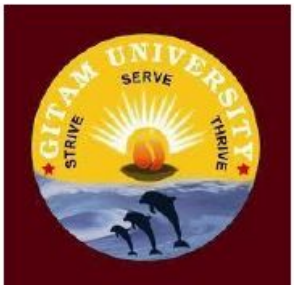
DECLARATION

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
Date:12-07-2020

Narra Sai Jeevan
221710305034



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 Narra Sai Jeevan(221710305034) 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 him at the Computer Science and Engineering Department , GITAM University Hyderabad Campus under my guidance and supervision.

Mr.
Assistant Professor
Department of CSE

Dr.S.PhaniKumar
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 Dr. **N. Siva Prasad**, Pro Vice Chancellor,GITAM Hyderabad and **Dr. N.Seetharamaiah**, Principal, GITAM Hyderabad.

I would like to thank respected **Dr. S.Phani Kumar**, 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 what we study for.

I would like to thank the respected faculty **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.

Narra Sai Jeevan
221710305034

ABSTRACT

Machine learning algorithms are used to predict the values from the data set by splitting the data set in to train and test and building Machine learning algorithms models of higher accuracy to predict the values is the primary task to be performed on Cereals data set My perception of understanding the given data set has been in the view of undertaking a client's requirement of overcoming the stagnant point of sales of the products being manufactured by clients.

To get a better understanding and work on a strategic approach for solution of the client, I have adapted the viewpoint of looking at ratings of the products and for further deep understanding of the problem, I have taken the stance of a consumer and reasoned out the various factors of choice of the products and they purchase , and my primary objective of this case study was to look up the factors which were dampening the sale of products and corelate them to ratings of products and draft out an outcome report to client regarding the various accepts of a product manufacturing , marketing and sale point determination

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 the 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:

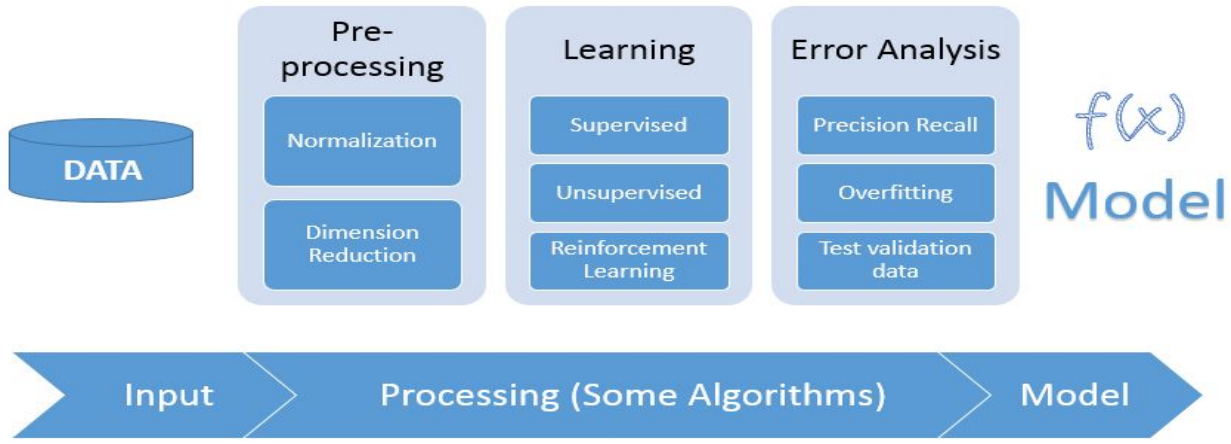


Figure 1 : The Process Flow

1.3 USES OF MACHINE LEARNING:

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

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data. By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

1.4 TYPES OF LEARNING ALGORITHMS:

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

1.4.1 Supervised Learning :

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

labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.
Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.
Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes are 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.

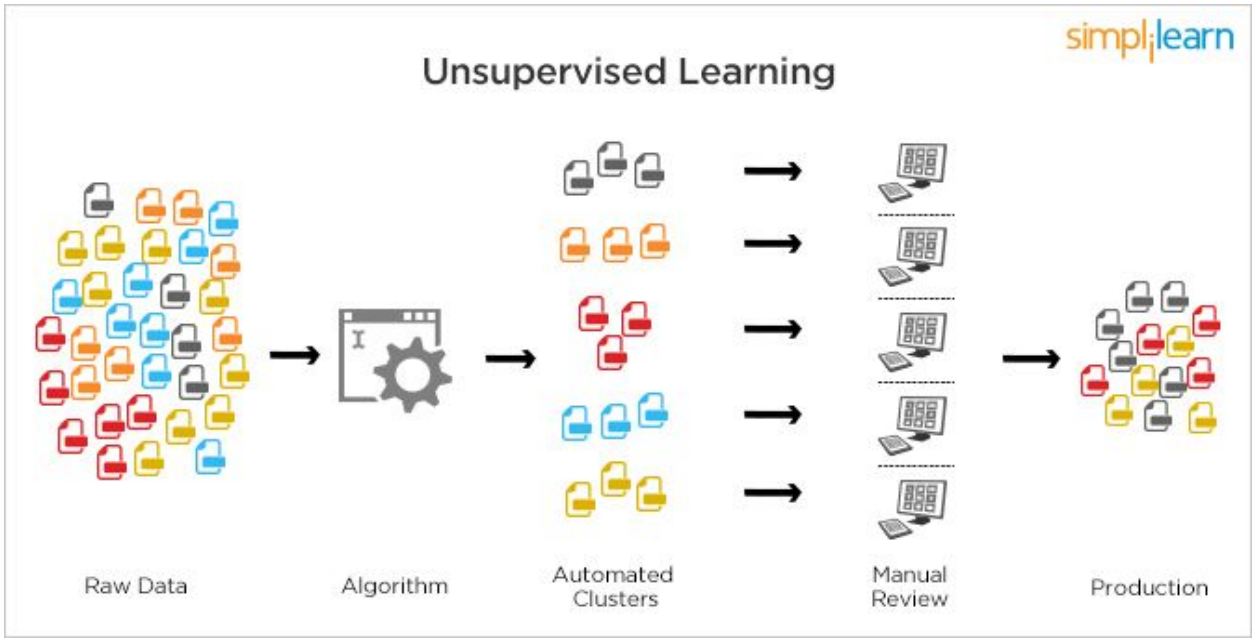


Figure 2 : Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically,

online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

1.4.3 Semi Supervised Learning:

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

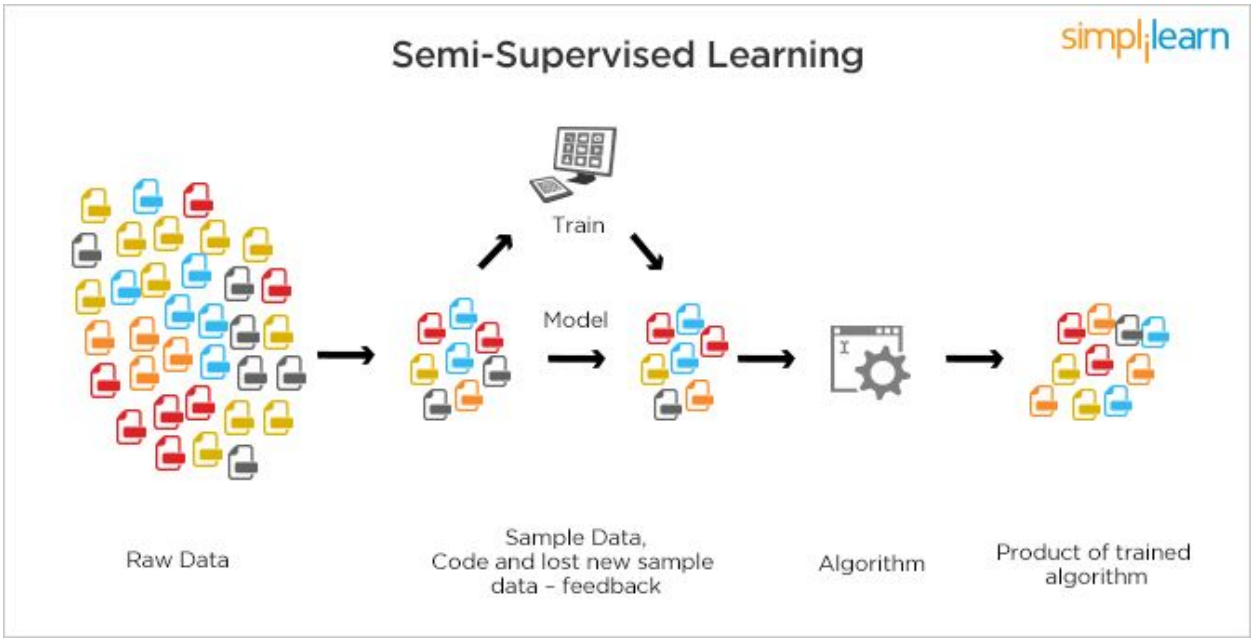


Figure 3 : Semi Supervised Learning

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

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovered previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.
Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2
PYTHON

Basic programming language used for machine learning is : PYTHON

2.1 INTRODUCTION TO PYTHON:

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

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
● 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.

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

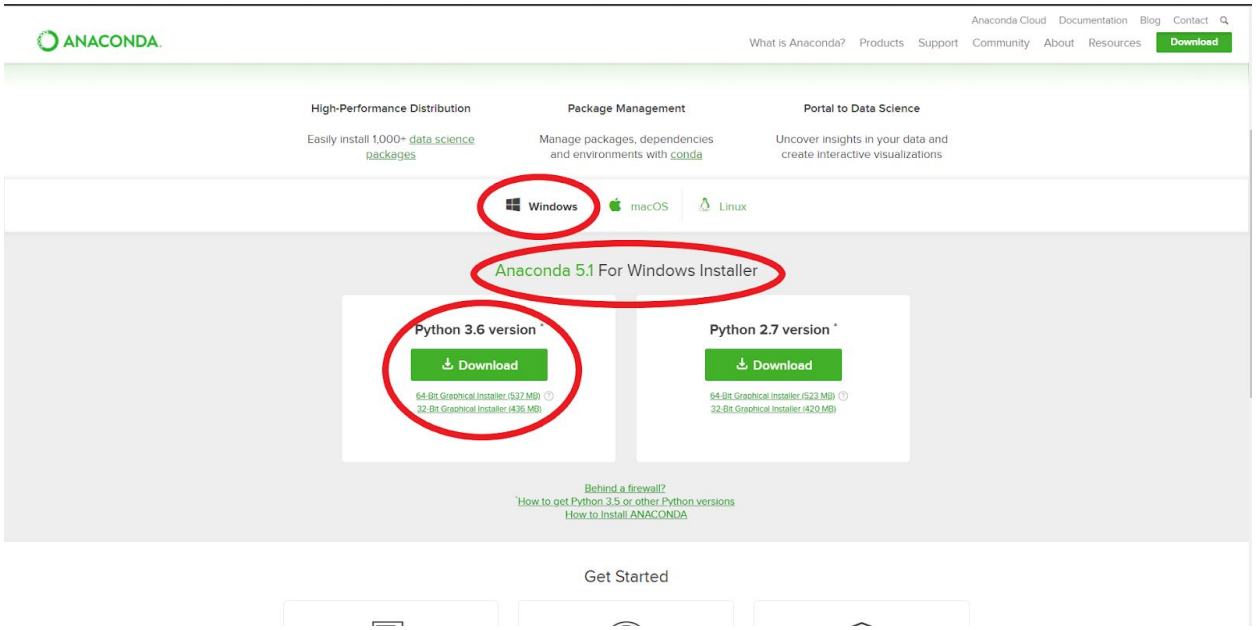


Figure 5 : Anaconda download

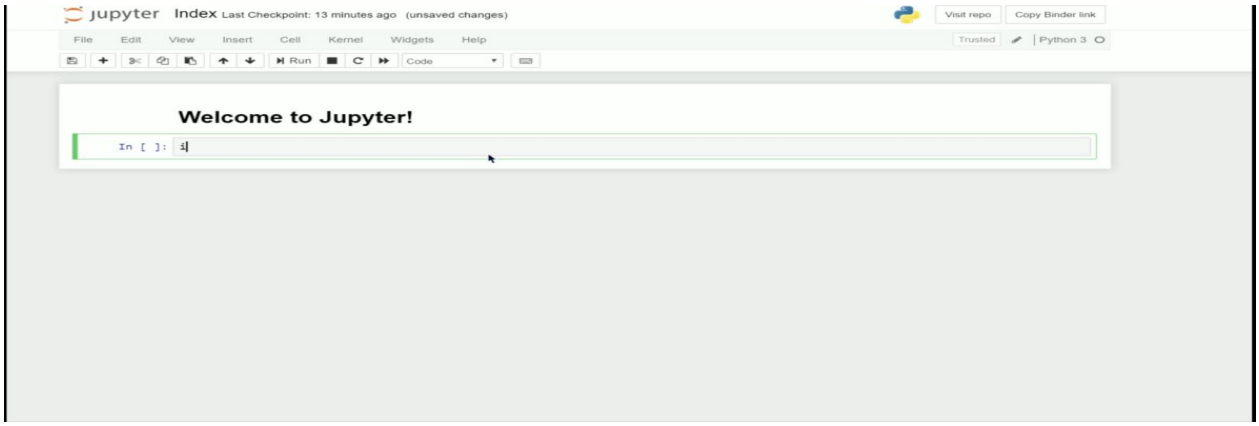


Figure 6 : Jupyter notebook

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 –
o Numbers
o Strings
o Lists
o Tuples
o 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 types.
● The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
● The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

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

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:
o We define a class in a very similar way how we define a function.
o 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 7 : 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:

To predict the Gender Recognition by Voice using Machine Learning algorithm called RANDOM FOREST CLASSIFIER

3.2 DATA SET:

The given data set consists of the following parameters:
A:meanfreq-mean frequency (in kHz)
B:sd-standard deviation of frequency
C:median-median frequency (in kHz)
D:Q25-first quantile (in kHz)
E:Q75-third quantile (in kHz)
F:IQR- interquartile range (in kHz)
G:skew-skewness (see note in specprop description)
H:kurt-kurtosis (see note in specprop description)
I:spent- spectral entropy
J:sfm-spectral flatness
K:centroid-frequency centroid (see specprop)
L:meanfun-mean fundamental frequency measured across acoustic signal
M:minfun-minimum fundamental frequency measured across acoustic signal
N:maxfun-maximum fundamental frequency measured across acoustic signal
O:meandom-mean of dominant frequency measured across acoustic signal
P:mindom-minimum of dominant frequency measured across acoustic signal
Q:maxdom-maximum of dominant frequency measured across acoustic signal
R:dfrange-range of dominant frequency measured across acoustic signal
S:mod index- modulation index

3.3 OBJECTIVE OF THE CASE STUDY:

To get a better understanding and chalking out a plan of action for solution of the client, we have adapted the viewpoint of looking at product categories and for further deep understanding of the problem, we have also considered gender age of the customer and reasoned out the various factors of choice of the products and they purchase , and

our primary objective of this case study was to look up the factors which were dampening the sale of products and corelate them to product categories and draft out an outcome report to the client regarding the various accepts of a product purchase.

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 the client.

4.1.2 IMPORTING THE LIBRARIES:

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

```
import numpy as np
import pandas as pd
```

```
import warnings
```

```
from sklearn import preprocessing
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
from sklearn import naive_bayes
from sklearn import tree
from sklearn import ensemble
from sklearn import svm
from sklearn import neural_network
```

Figure 8 : 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 has to be cleaned.

Reading Dataset

```
import warnings
warnings.filterwarnings('ignore')
voice=pd.read_csv('../AIML/voice.csv')
voice.head()
```

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	minfun	maxfun	meand
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

4.1.4 HANDLING MISSING VALUES:

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

```
: voice.isnull().sum()

: 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

: voice.isnull().values.any()

: False
```

Figure 10 : Missing values in dataset

CHAPTER 5
DATA PREPROCESSING/FEATURE ENGINEERING AND EDA

5.1 Statistical Analysis

Pandas DataFrame. describe() The describe() method is used for calculating some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame. It analyzes both numeric and object series and also the DataFrame column sets of mixed data types.

Analysing data

```
voice.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3168 entries, 0 to 3167
Data columns (total 21 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   meanfreq  3168 non-null   float64
 1   sd        3168 non-null   float64
 2   median    3168 non-null   float64
 3   Q25       3168 non-null   float64
 4   Q75       3168 non-null   float64
 5   IQR       3168 non-null   float64
 6   skew      3168 non-null   float64
 7   kurt      3168 non-null   float64
 8   sp.ent    3168 non-null   float64
 9   sfm       3168 non-null   float64
 10  mode      3168 non-null   float64
 11  centroid  3168 non-null   float64
 12  meanfun   3168 non-null   float64
 13  minfun    3168 non-null   float64
 14  maxfun    3168 non-null   float64
 15  meandom   3168 non-null   float64
 16  mindom    3168 non-null   float64
 17  maxdom    3168 non-null   float64
 18  dfrange   3168 non-null   float64
 19  modindx   3168 non-null   float64
 20  label     3168 non-null   object
dtypes: float64(20), object(1)
memory usage: 507.4+ KB
```

```
voice.describe()
```

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	mode	ce
count	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.(
mean	0.180907	0.057126	0.185621	0.140456	0.224765	0.084309	3.140168	36.568461	0.895127	0.408216	0.165282	0.1
std	0.029918	0.016652	0.036360	0.048680	0.023639	0.042783	4.240529	134.928661	0.044980	0.177521	0.077203	0.(
min	0.039363	0.018363	0.010975	0.000229	0.042946	0.014558	0.141735	2.068455	0.738651	0.036876	0.000000	0.(
25%	0.163662	0.041954	0.169593	0.111087	0.208747	0.042560	1.649569	5.669547	0.861811	0.258041	0.118016	0.1
50%	0.184838	0.059155	0.190032	0.140286	0.225684	0.094280	2.197101	8.318463	0.901767	0.396335	0.186599	0.1
75%	0.199146	0.067020	0.210618	0.175939	0.243660	0.114175	2.931694	13.648905	0.928713	0.533676	0.221104	0.1
max	0.251124	0.115273	0.261224	0.247347	0.273469	0.252225	34.725453	1309.612887	0.981997	0.842936	0.280000	0.2

Figure 11: Statistical Analysis

5.2 Generating Plots

Visualize the data between Target and the Features
Data Correlation: Is a way to understand the relationship between multiple variables and attributes in your dataset. Using Correlation, you can get some insights such as: One or multiple attributes depend on another attribute or a cause for another attribute.

Visualization

```
import matplotlib.pyplot as plt
plt.subplots(4,5,figsize=(15,15))
for i in range(1,21):
    plt.subplot(10,2,i)
    plt.title(voice.columns[i-1])
    plt.hist(voice.loc[voice['label'] == 0, voice.columns[i-1]], color= 'brown', label='Female')
    plt.hist(voice.loc[voice['label'] == 1, voice.columns[i-1]], color= 'purple', label='Male')
```

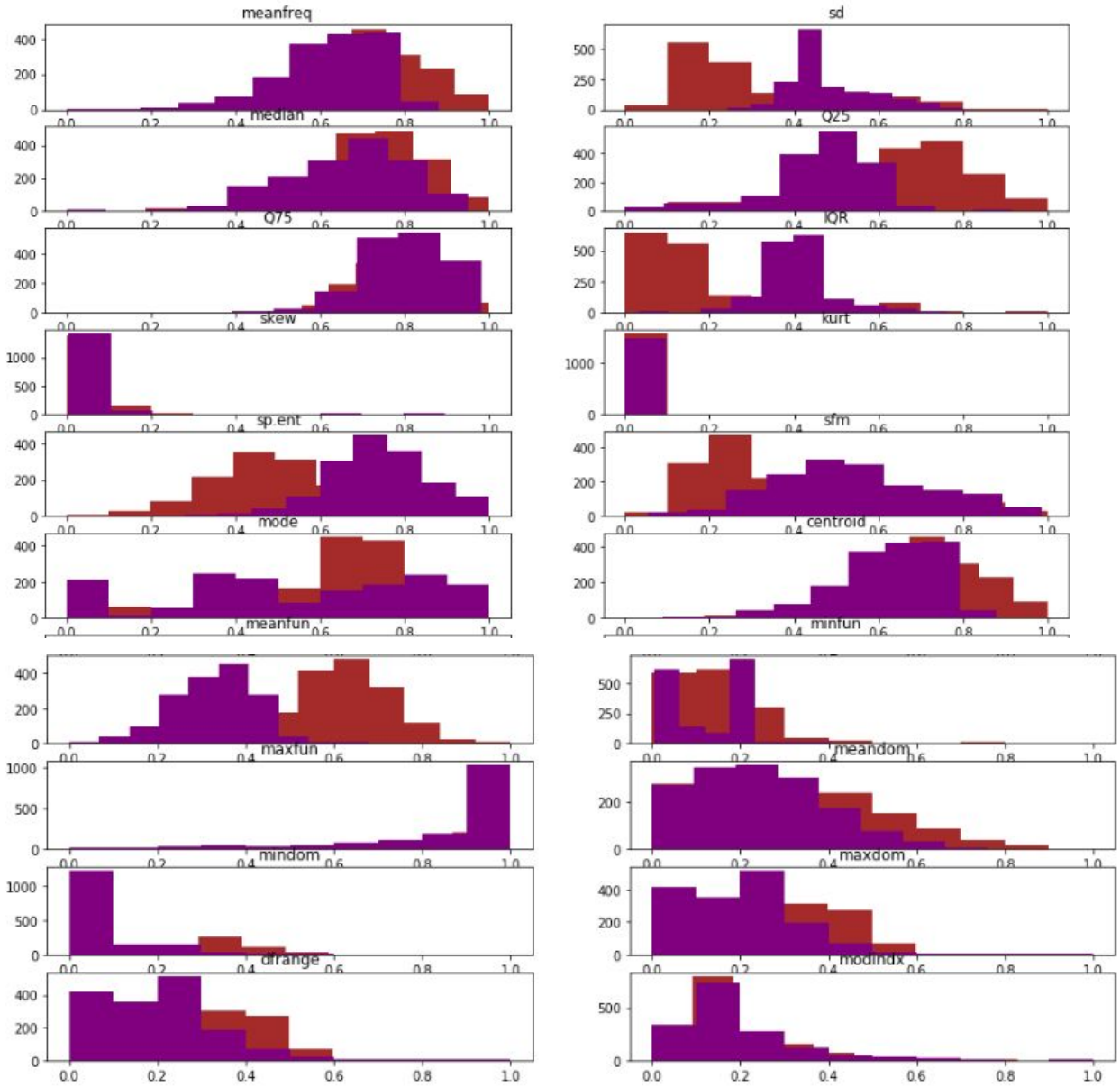


Figure 12: Visualization

```
correlation = voice.corr()
correlation
```

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	minfun	ma
meanfreq	1.000000	-0.739039	0.925445	0.911416	0.740997	-0.627605	-0.322327	-0.316036	-0.601203	-0.784332	...	1.000000	0.460844	0.383937	0.27
sd	-0.739039	1.000000	-0.562603	-0.846931	-0.161076	0.874660	0.314597	0.346241	0.716620	0.838086	...	-0.739039	-0.466281	-0.345609	-0.12
median	0.925445	-0.562603	1.000000	0.774922	0.731849	-0.477352	-0.257407	-0.243382	-0.502005	-0.661690	...	0.925445	0.414909	0.337602	0.25
Q25	0.911416	-0.846931	0.774922	1.000000	0.477140	-0.874189	-0.319475	-0.350182	-0.648126	-0.766875	...	0.911416	0.545035	0.320994	0.19
Q75	0.740997	-0.161076	0.731849	0.477140	1.000000	0.009636	-0.206339	-0.148881	-0.174905	-0.378198	...	0.740997	0.155091	0.258002	0.28
IQR	-0.627605	0.874660	-0.477352	-0.874189	0.009636	1.000000	0.249497	0.316185	0.640813	0.663601	...	-0.627605	-0.534462	-0.222680	-0.06
skew	-0.322327	0.314597	-0.257407	-0.319475	-0.206339	0.249497	1.000000	0.977020	-0.195459	0.079694	...	-0.322327	-0.167668	-0.216954	-0.08
kurt	-0.316036	0.346241	-0.243382	-0.350182	-0.148881	0.316185	0.977020	1.000000	-0.127644	0.109884	...	-0.316036	-0.194560	-0.203201	-0.04
sp.ent	-0.601203	0.716620	-0.502005	-0.648126	-0.174905	0.640813	-0.195459	-0.127644	1.000000	0.866411	...	-0.601203	-0.513194	-0.305826	-0.12
sfm	-0.784332	0.838086	-0.661690	-0.766875	-0.378198	0.663601	0.079694	0.109884	0.866411	1.000000	...	-0.784332	-0.421066	-0.362100	-0.19
mode	0.687715	-0.529150	0.677433	0.591277	0.486857	-0.403764	-0.434859	-0.406722	-0.325298	-0.485913	...	0.687715	0.324771	0.385467	0.17
centroid	1.000000	-0.739039	0.925445	0.911416	0.740997	-0.627605	-0.322327	-0.316036	-0.601203	-0.784332	...	1.000000	0.460844	0.383937	0.27
meanfun	0.460844	-0.466281	0.414909	0.545035	0.155091	-0.534462	-0.167668	-0.194560	-0.513194	-0.421066	...	0.460844	1.000000	0.339387	0.31
minfun	0.383937	-0.345609	0.337602	0.320994	0.258002	-0.222680	-0.216954	-0.203201	-0.305826	-0.362100	...	0.383937	0.339387	1.000000	0.21
maxfun	0.274004	-0.129662	0.251328	0.199841	0.285584	-0.069588	-0.080861	-0.045667	-0.120738	-0.192369	...	0.274004	0.311950	0.213987	1.00
meandom	0.536666	-0.482726	0.455943	0.467403	0.359181	-0.333362	-0.336848	-0.303234	-0.293562	-0.428442	...	0.536666	0.270840	0.375979	0.33
mindom	0.229261	-0.357667	0.191169	0.302255	-0.023750	-0.357037	-0.061608	-0.103313	-0.294869	-0.289593	...	0.229261	0.162163	0.082015	-0.24
maxdom	0.519528	-0.482278	0.438919	0.459683	0.335114	-0.337877	-0.305651	-0.274500	-0.324253	-0.436649	...	0.519528	0.277982	0.317860	0.35
dfrange	0.515570	-0.475999	0.435621	0.454394	0.335648	-0.331563	-0.304640	-0.272729	-0.319054	-0.431580	...	0.515570	0.275154	0.316486	0.35
modindx	-0.216979	0.122660	-0.213298	-0.141377	-0.216475	0.041252	-0.169325	-0.205539	0.198074	0.211477	...	-0.216979	-0.054858	0.002042	-0.36
label	-0.337415	0.479539	-0.283919	-0.511455	0.066906	0.618916	0.036627	0.087195	0.490552	0.357499	...	-0.337415	-0.833921	-0.136692	-0.16

21 rows × 21 columns

Figure 13:Correlation

Confusion Matrix:

A correlation matrix is a table showing correlation coefficients between sets of variables. Each random variable (Xi) in the table is correlated with each of the other values in the table (Xj). This allows you to see which pairs have the highest correlation.

(or)

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analysis.

```
#correlation matrix
import seaborn as sns
f, ax = plt.subplots(figsize = (8,8))
sns.heatmap(correlation, square = True)
plt.show()
```

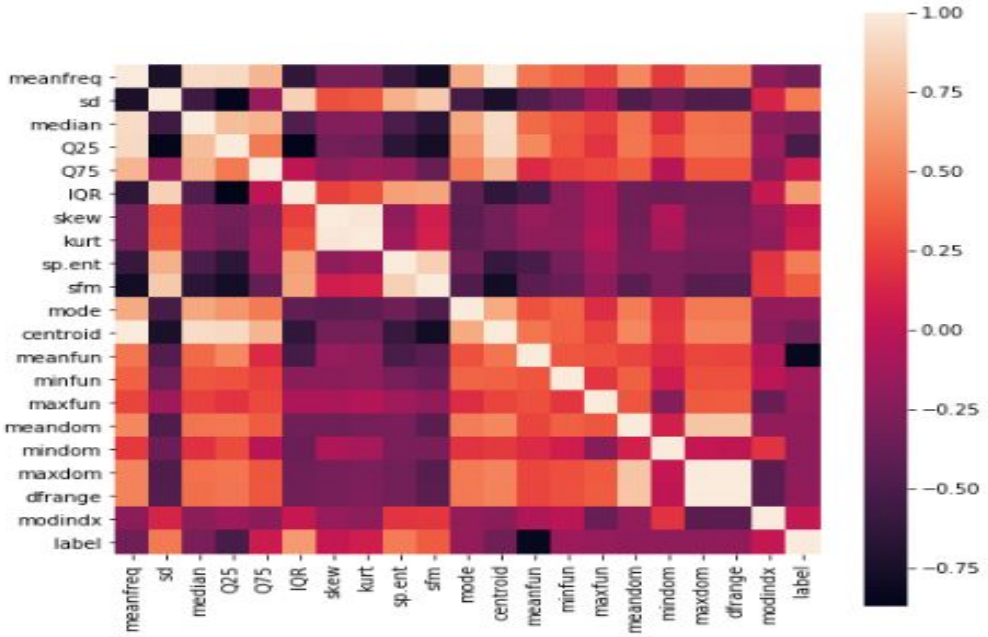


Figure 14: Correlation Matrix

Colour Map

```
#correlation
import matplotlib.pyplot as plt
import seaborn as sns
colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(voice.iloc[:,:-1].astype(float).corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap=colormap, linecolor='white', annot=True)
plt.show()
```

The idea behind choosing a good colormap is to find a good representation in 3D colorspace for your data set. The best colormap for any given data set depends on many things including:

- Whether representing form or metric data.
- Your knowledge of the data set.
- If there is an intuitive color scheme for the parameter you are plotting
- If there is a standard in the field the audience may be expecting.

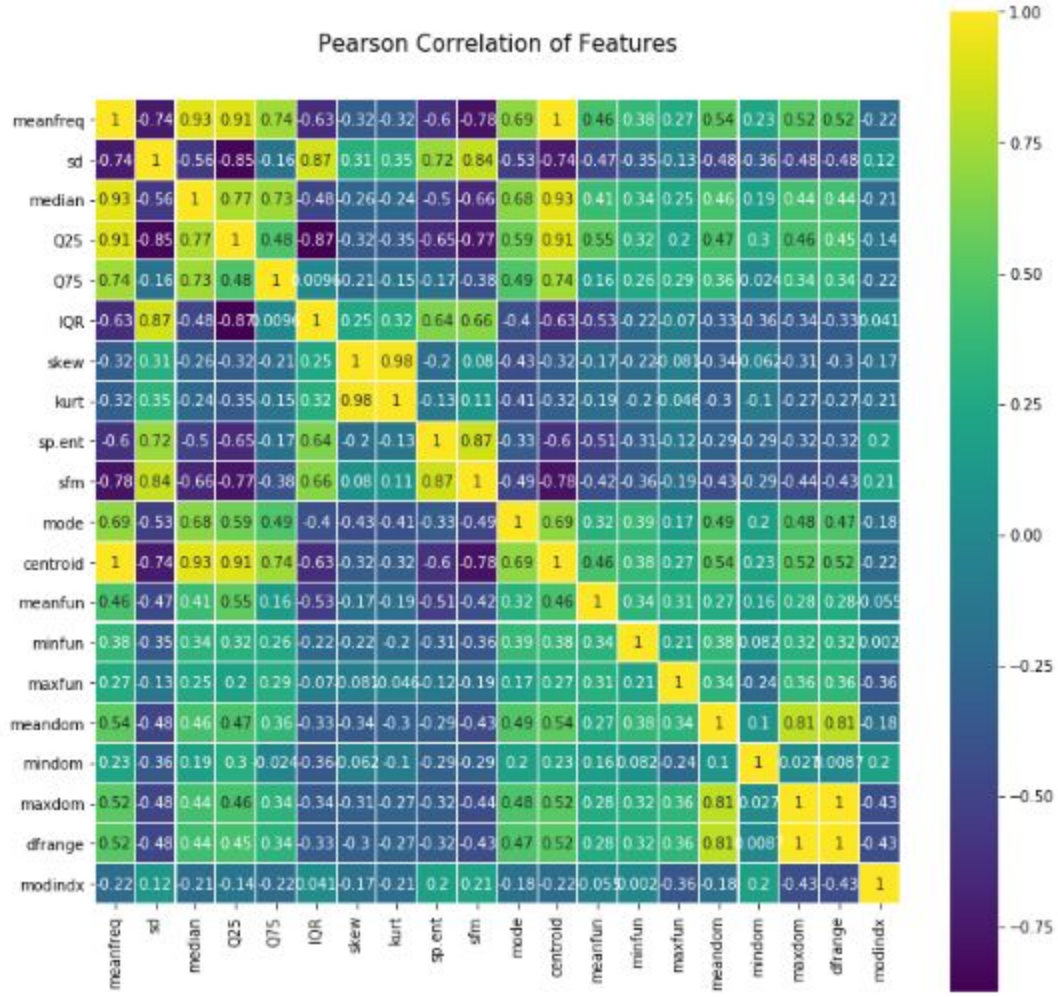


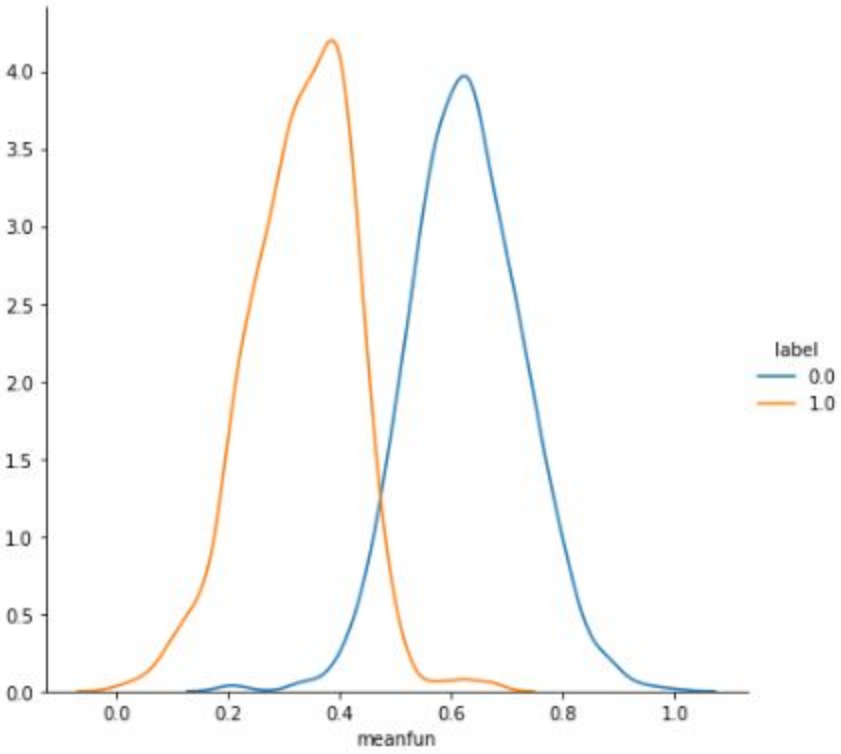
Figure 15:colour map

Plots used:

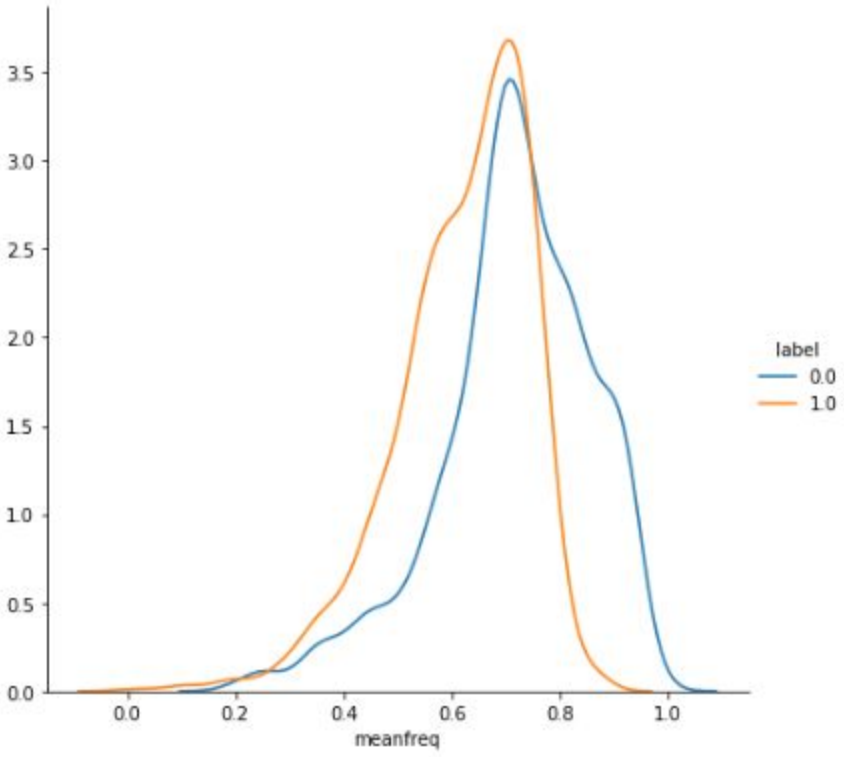
seaborn.FacetGrid

```
sns.FacetGrid(voice, hue="label", size=6) \
   .map(sns.kdeplot, "meanfun") \
   .add_legend()
plt.show()
```

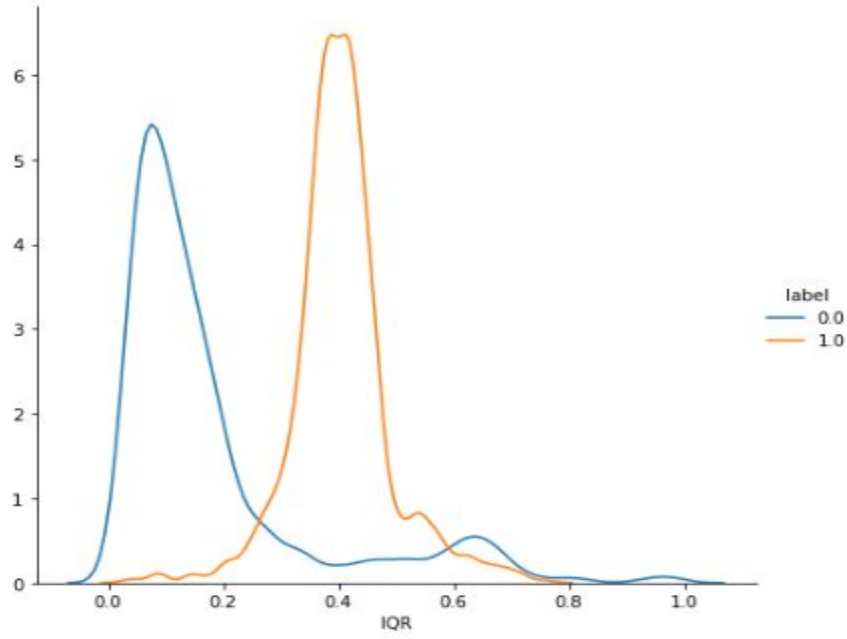
This class maps a dataset into multiple axes arrayed in a grid of rows and columns that correspond to levels of variables in the dataset. The plots it produces are often called “lattice”, “trellis”, or “small-multiple” graphics




```
sns.FacetGrid(voice, hue="label", size=6) \
   .map(sns.kdeplot, "meanfreq") \
   .add_legend()
plt.show()
```




```
sns.FacetGrid(voice, hue="label", size=6) \
   .map(sns.kdeplot, "IQR") \
   .add_legend()
plt.show()
```

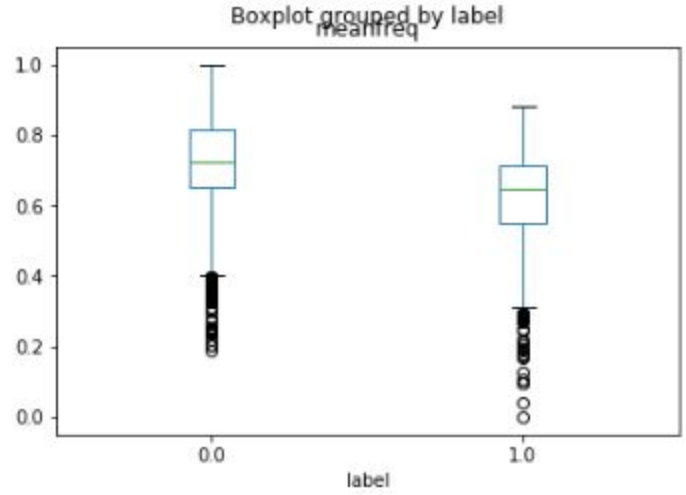


Seaborn.boxplot

A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range.

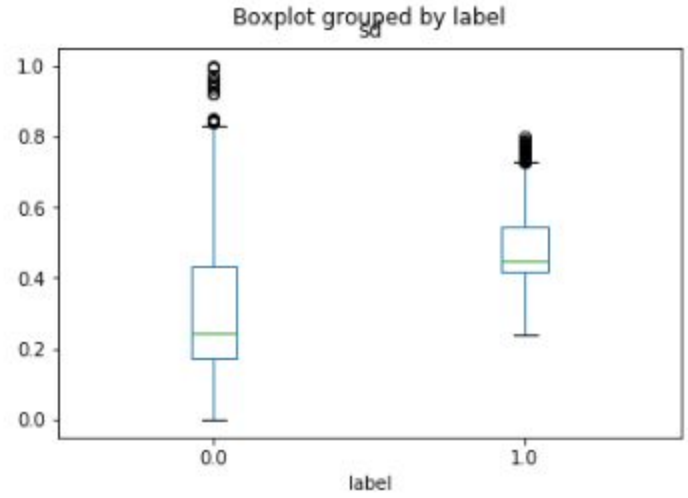
```
#Box plot see comparision in labels by other features
voice.boxplot(column= 'meanfreq', by='label', grid=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x115cf730>
```



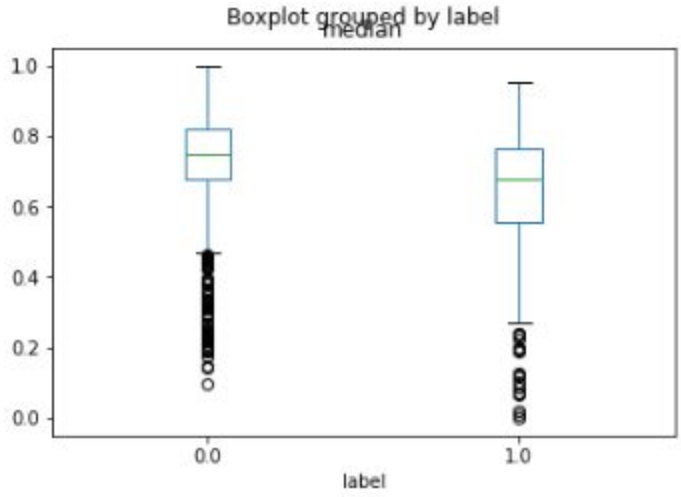

```
voice.boxplot(column= 'sd', by='label', grid=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1124c5f0>
```




```
voice.boxplot(column= 'median', by='label', grid=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11390b30>
```

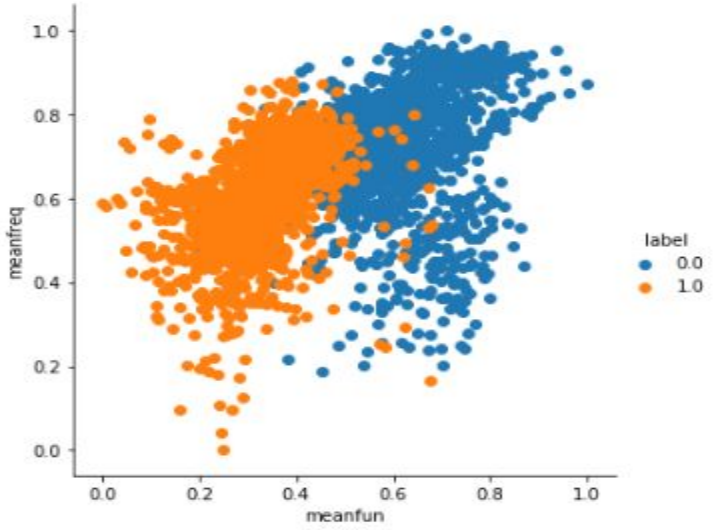


Scattered Plot

The relationship between x and y can be shown for different subsets of the data using the hue, size, and style parameters. These parameters control what visual semantics are used to identify the different subsets. It is possible to show up to three dimensions independently by using all

three semantic types, but this style of plot can be hard to interpret and is often ineffective. Using redundant semantics (i.e. both hue and style for the same variable) can be helpful for making graphics more accessible.

```
# scatter plot between features
sns.FacetGrid(voice, hue="label", size=5)\
   .map(plt.scatter, "meanfun", "meanfreq")\
   .add_legend()
plt.show()
```



CHAPTER 6

SPLITTING OF DATA

6.1 Splitting Of 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)

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from sklearn import neighbors
from sklearn import naive_bayes
from sklearn import tree
from sklearn import ensemble
from sklearn import svm
from sklearn import neural_network
```

```
train, test = train_test_split(voice, test_size=0.3)
```

```
train.head()
```

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	minfun	maxfun	meando
299	0.755369	0.352244	0.692252	0.615472	0.933540	0.384016	0.062737	0.005185	0.592708	0.065582	...	0.755369	0.433686	0.030819	0.579677	0.3951
2708	0.685969	0.216274	0.652808	0.654034	0.682483	0.100410	0.071080	0.006312	0.492572	0.245144	...	0.685969	0.534420	0.193074	0.999749	0.8421
345	0.560430	0.436244	0.608192	0.418727	0.710672	0.372416	0.083982	0.010041	0.723521	0.565885	...	0.560430	0.265659	0.124743	0.361172	0.2164
2217	0.652718	0.211203	0.667337	0.613771	0.661974	0.122381	0.053359	0.003774	0.483164	0.306666	...	0.652718	0.501065	0.196724	0.981526	0.3285
2290	0.850309	0.199665	0.860393	0.790003	0.872995	0.143819	0.033127	0.001162	0.506444	0.218295	...	0.850309	0.680739	0.193795	0.954963	0.6488

5 rows × 21 columns

```
x_train = train.iloc[:, :-1]
y_train = train["label"]
x_test = test.iloc[:, :-1]
y_test = test["label"]
```

```
x_train3 = train[["meanfun","IQR","Q25"]]
y_train3 = train["label"]
x_test3 = test[["meanfun","IQR","Q25"]]
y_test3 = test["label"]
```

```
from sklearn.metrics import accuracy_score

def classify(model,x_train,y_train,x_test,y_test):
    from sklearn.metrics import classification_report
    target_names = ['female', 'male']
    model.fit(x_train,y_train)
    y_pred=model.predict(x_test)
    print(classification_report(y_test, y_pred, target_names=target_names, digits=4))
```

CHAPTER 7

MODEL BUILDING AND EVALUATION

7.1 Brief about the algorithms used

7.1.1 Random Forest Model

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.
The first algorithm for random decision forests was created by Tin Kam Ho using the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

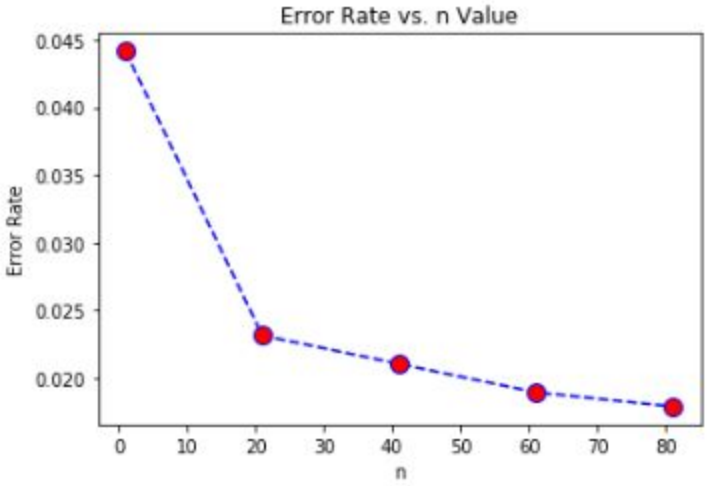
Random Forest Classifier

```
def rf_error(n,x_train,y_train,x_test,y_test):
    error_rate = []
    e=range(1,n,20)
    for i in e:
        model = ensemble.RandomForestClassifier(n_estimators = i)
        model.fit(x_train, y_train)
        y_pred = model.predict(x_test)
        error_rate.append(np.mean(y_pred != y_test))
    nloc = error_rate.index(min(error_rate))
    print("Lowest error is %s occurs at n=%s." % (error_rate[nloc], e[nloc]))

    plt.plot(e, error_rate, color='blue', linestyle='dashed', marker='o',
             markerfacecolor='red', markersize=10)
    plt.title('Error Rate vs. n Value')
    plt.xlabel('n')
    plt.ylabel('Error Rate')
    plt.show()
    return e[nloc]
```

```
: e=rf_error(100,x_train,y_train,x_test,y_test,)
```

```
Lowest error is 0.017875920084121977 occurs at n=81.
```

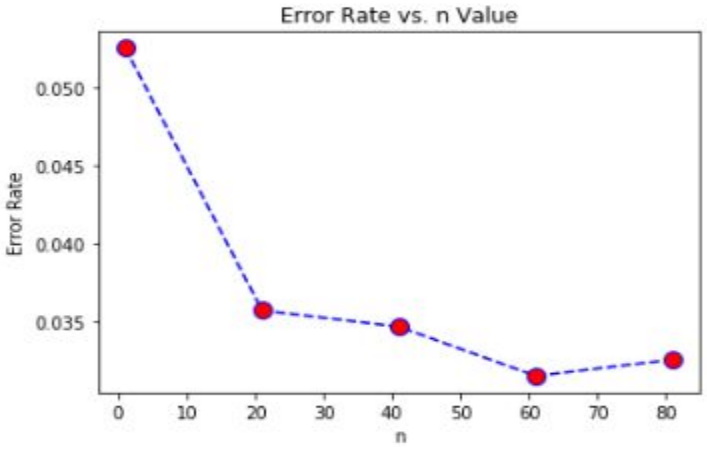



```
model=ensemble.RandomForestClassifier(n_estimators = e)
classify(model,x_train,y_train,x_test,y_test)
```

	precision	recall	f1-score	support
female	0.9702	0.9828	0.9764	464
male	0.9834	0.9713	0.9773	487
accuracy			0.9769	951
macro avg	0.9768	0.9770	0.9769	951
weighted avg	0.9769	0.9769	0.9769	951

```
e=rf_error(100,x_train3,y_train3,x_test3,y_test3)
```

```
Lowest error is 0.031545741324921134 occurs at n=61.
```




```
: model=ensemble.RandomForestClassifier(n_estimators = e)
  classify(model,x_train3,y_train3,x_test3,y_test3)
```

```
              precision    recall  f1-score   support

      female     0.9637    0.9720    0.9678       464
        male     0.9731    0.9651    0.9691       487

    accuracy                         0.9685       951
   macro avg     0.9684    0.9685    0.9684       951
weighted avg     0.9685    0.9685    0.9685       951
```

Hyperparameter Tuning

Hyperparameter Tuning

```
from sklearn.ensemble import RandomForestClassifier
#making the instance
model=RandomForestClassifier()
#hyper parameters set
params = {'criterion':['gini','entropy'],
          'n_estimators':[10,15,20,25,30],
          'min_samples_leaf':[1,2,3],
          'min_samples_split':[3,4,5,6,7],
          'random_state':[123],
          'n_jobs':[-1]}
#Making models with hyper parameters sets
model1 = GridSearchCV(model, param_grid=params, n_jobs=-1)
#Learning
model1.fit(x_train,y_train)
#The best hyper parameters set
print("Best Hyper Parameters:\n",model1.best_params_)
#Prediction
prediction=model1.predict(x_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("Accuracy:",metrics.accuracy_score(prediction,y_test))
```

```
Best Hyper Parameters:
 {'criterion': 'entropy', 'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 30, 'n_jobs': -1, 'random_state': 123}
Accuracy: 0.9810725552050473
```

7.1.2 KNN Model

The k -nearest neighbors algorithm (k -NN) is a non-parametric method proposed by Thomas Cover used for classification and regression.In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k -NN is used for classification or regression.
In k-NN classification , the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

Scaling

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
voice["label"] = le.fit_transform(voice["label"])
le.classes_
```

```
array(['female', 'male'], dtype=object)
```

```
voice[:]=preprocessing.MinMaxScaler().fit_transform(voice)
voice.head()
```

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	minfun	maxfun	meandom
0	0.096419	0.473409	0.084125	0.060063	0.204956	0.254828	0.367853	0.208279	0.635798	0.564526	...	0.096419	0.157706	0.030501	0.981526	0.000000
1	0.125828	0.505075	0.116900	0.077635	0.215683	0.246961	0.644279	0.483766	0.630964	0.591578	...	0.125828	0.287642	0.031140	0.834600	0.000407
2	0.179222	0.675536	0.102873	0.034284	0.385912	0.457148	0.885255	0.782275	0.442738	0.548382	...	0.179222	0.236945	0.030264	0.954963	0.000060
3	0.528261	0.554611	0.587559	0.389906	0.715802	0.407358	0.031549	0.001613	0.923261	0.856457	...	0.528261	0.183442	0.041287	0.834600	0.065659
4	0.452195	0.627209	0.454272	0.317627	0.707515	0.474474	0.027742	0.001732	0.958736	0.926348	...	0.452195	0.279190	0.036829	0.929285	0.238994

5 rows × 21 columns

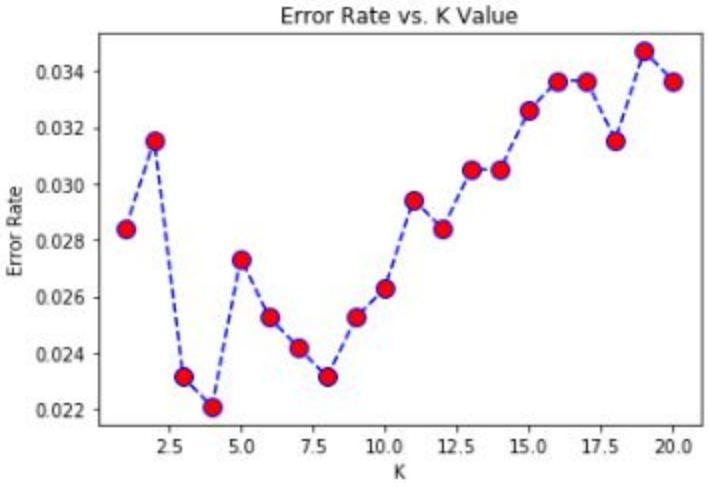
K-Nearest Neighbours

```
def knn_error(k,x_train,y_train,x_test,y_test):
    error_rate = []
    K=range(1,k)
    for i in K:
        knn = neighbors.KNeighborsClassifier(n_neighbors = i)
        knn.fit(x_train, y_train)
        y_pred = knn.predict(x_test)
        error_rate.append(np.mean(y_pred != y_test))
    kloc = error_rate.index(min(error_rate))
    print("Lowest error is %s occurs at k=%s." % (error_rate[kloc], K[kloc]))

    plt.plot(K, error_rate, color='blue', linestyle='dashed', marker='o',
             markerfacecolor='red', markersize=10)
    plt.title('Error Rate vs. K Value')
    plt.xlabel('K')
    plt.ylabel('Error Rate')
    plt.show()
    return K[kloc]
```

```
k=knn_error(21,x_train,y_train,x_test,y_test)
```

```
Lowest error is 0.022082018927444796 occurs at k=4.
```




```
model = neighbors.KNeighborsClassifier(n_neighbors = k)
classify(model,x_train,y_train,x_test,y_test)
```

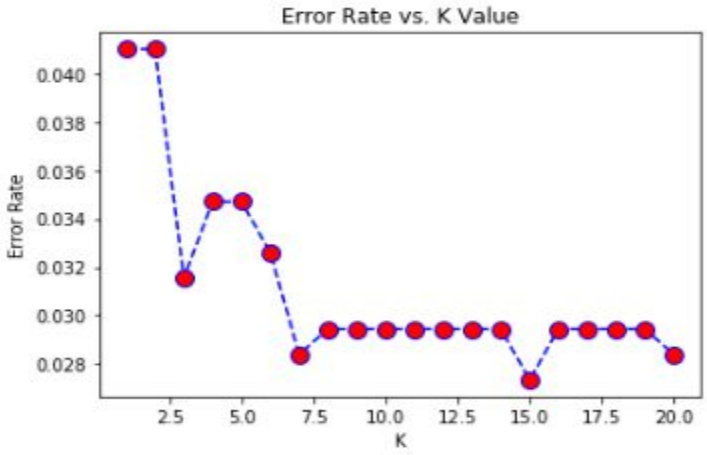
```
              precision    recall  f1-score   support

      female     0.9723    0.9828    0.9775       464
        male     0.9834    0.9733    0.9783       487

    accuracy                         0.9779       951
   macro avg     0.9778    0.9780    0.9779       951
weighted avg     0.9780    0.9779    0.9779       951
```

```
k=knn_error(21,x_train3,y_train3,x_test3,y_test3)
```

```
Lowest error is 0.027339642481598318 occurs at k=15.
```




```
model = neighbors.KNeighborsClassifier(n_neighbors = k)
classify(model,x_train,y_train,x_test,y_test)
```

```
              precision    recall  f1-score   support

      female     0.9779    0.9547    0.9662       464
        male     0.9578    0.9795    0.9685       487

    accuracy                         0.9674       951
   macro avg     0.9679    0.9671    0.9674       951
weighted avg     0.9676    0.9674    0.9674       951
```

Hyperparameter Tuning

hyperparameter Tuning

```
model = KNeighborsClassifier(n_jobs=-1)
#Hyper Parameters Set
params = {'n_neighbors':[5,6,7,8,9,10],
          'leaf_size':[1,2,3,5],
          'weights':['uniform', 'distance'],
          'algorithm':['auto', 'ball_tree','kd_tree','brute'],
          'n_jobs':[-1]}
#Making models with hyper parameters sets
model1 = GridSearchCV(model, param_grid=params, n_jobs=1)
#Learning
model1.fit(x_train,y_train)
#The best hyper parameters set
print("Best Hyper Parameters:\n",model1.best_params_)
#Prediction
prediction=model1.predict(x_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("Accuracy:",accuracy_score(y_test,prediction))
```

```
Best Hyper Parameters:
 {'algorithm': 'auto', 'leaf_size': 1, 'n_jobs': -1, 'n_neighbors': 6, 'weights': 'distance'}
Accuracy: 0.7097791798107256
```

7.1.3 Decision Tree Model

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

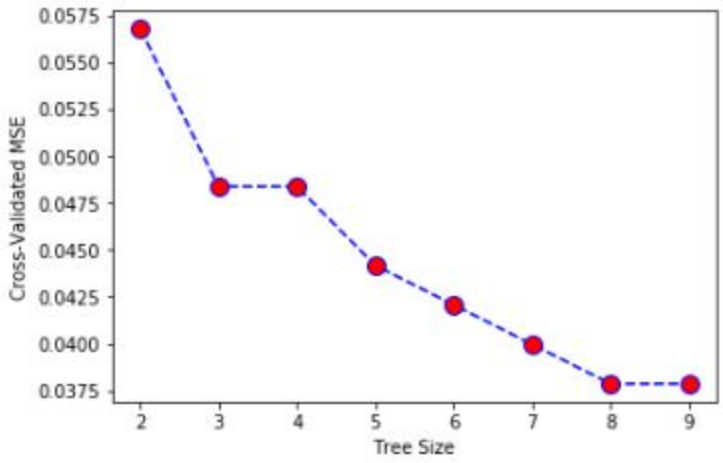
A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

Decision Tree

```
: def dt_error(n,x_train,y_train,x_test,y_test):
      nodes = range(2, n)
      error_rate = []
      for k in nodes:
          model = tree.DecisionTreeClassifier(max_leaf_nodes=k)
          model.fit(x_train, y_train)
          y_pred = model.predict(x_test)
          error_rate.append(np.mean(y_pred != y_test))
      kloc = error_rate.index(min(error_rate))
      print("Lowest error is %s occurs at n=%s." % (error_rate[kloc], nodes[kloc]))
      plt.plot(nodes, error_rate, color='blue', linestyle='dashed', marker='o',
               markerfacecolor='red', markersize=10)
      plt.xlabel('Tree Size')
      plt.ylabel('Cross-Validated MSE')
      plt.show()
      return nodes[kloc]
```

```
n=dt_error(10,x_train,y_train,x_test,y_test)
```

```
Lowest error is 0.037854889589905360 occurs at n=8.
```

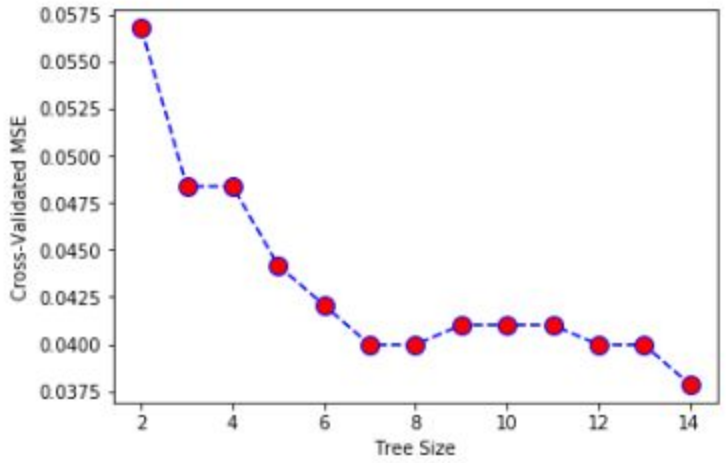


```
pruned_tree = tree.DecisionTreeClassifier(criterion = 'gini', max_leaf_nodes = n)
classify(pruned_tree,x_train,y_train,x_test,y_test)
```

	precision	recall	f1-score	support
female	0.9693	0.9526	0.9609	464
male	0.9556	0.9713	0.9633	487
accuracy			0.9621	951
macro avg	0.9624	0.9619	0.9621	951
weighted avg	0.9623	0.9621	0.9621	951

```
n=dt_error(15,x_train3,y_train3,x_test3,y_test3)
```

```
Lowest error is 0.03785488958990536 occurs at n=14.
```




```
pruned_tree = tree.DecisionTreeClassifier(criterion = 'gini', max_leaf_nodes = n)
classify(pruned_tree,x_train3,y_train3,x_test3,y_test3)
```

	precision	recall	f1-score	support
female	0.9496	0.9741	0.9617	464
male	0.9747	0.9507	0.9626	487
accuracy			0.9621	951
macro avg	0.9622	0.9624	0.9621	951
weighted avg	0.9625	0.9621	0.9622	951

Hyperparameter Tuning

hyperparameter Tuning

```
from sklearn.tree import DecisionTreeClassifier
#making the instance
model= DecisionTreeClassifier(random_state=1234)
#Hyper Parameters Set
params = {'max_features': ['auto', 'sqrt', 'log2'],
          'min_samples_split': [2,3,4,5,6,7,8,9,10,11,12,13,14,15],
          'min_samples_leaf':[1,2,3,4,5,6,7,8,9,10,11],
          'random_state':[123]}
#Making models with hyper parameters sets
model1 = GridSearchCV(model, param_grid=params, n_jobs=-1)
#Learning
model1.fit(x_train,y_train)
#The best hyper parameters set
print("Best Hyper Parameters:",model1.best_params_)
#Prediction
prediction=model1.predict(x_test)
#importing the metrics module
from sklearn import metrics
#evaluation(Accuracy)
print("Accuracy:",metrics.accuracy_score(prediction,y_test))
```

```
Best Hyper Parameters: {'max_features': 'auto', 'min_samples_leaf': 3, 'min_samples_split': 7, 'random_state': 123}
Accuracy: 0.9726603575184016
```

7.2 Evaluation of Models

Here we have used 3 models i.e

- Decision Tree
- KNN Model
- Random Forest Model

Hyperparameter Tuning

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm.A hyperparameter is a parameter whose value is set before the learning process begins.In sklearn, hyperparameters are passed in as arguments to the constructor of the model classes.Here we used grid search.

Grid Search

Grid search is a traditional way to perform hyperparameter optimization. It works by searching exhaustively through a specified subset of hyperparameters.Using sklearn's `GridSearchCV`, we first define our grid of parameters to search over and then run the grid search.

Metrics used:

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Metrics used

Accuracy:

Accuracy is the quintessential classification metric. It is pretty easy to understand. And easily suited for binary as well as a multiclass classification problem.

Accuracy = (TP+TN)/(TP+FP+FN+TN)
Accuracy is the proportion of true results among the total number of cases examined.
As observed from the models,the Random Forest Model has performed well with more accuracy.

When to use?
Accuracy is a valid choice of evaluation for classification problems which are well balanced and not skewed or No class imbalance.

Precision:

Precision = TP / TP + FP
When to use?
Precision is a valid choice of evaluation metric when we want to be very sure of our prediction. For example: If we are building a system to predict if we should decrease the credit limit on a particular account, we want to be very sure about our prediction or it may result in customer dissatisfaction.

Recall:

Another very useful measure is recall.

Recall = TP/(TP+FN)

When to use?

Recall is a valid choice of evaluation metric when we want to capture as many positives as possible. For example: If we are building a system to predict if a person has cancer or not, we want to capture the disease even if we are not very sure.

F1 Score:

The F1 score is a number between 0 and 1 and is the harmonic mean of precision and recall.

$$F1=2 * (Precision*Recall)/(Precision+Recall)$$

When to use?

We want to have a model with both good precision and recall.

Simply stated the F1 score sort of maintains a balance between the precision and recall for your classifier. If your precision is low, the F1 is low and if the recall is low again your F1 score is low.

CONCLUSION :

Recognizing the gender of human voice has been considered one of the challenging tasks because of its importance in various applications. The contributions are threefold including (i) studying the extracted features by examining the correlation between each other, (ii) building classification models using different ML techniques from distinct families, and (iii) evaluating the natural feature selection techniques in finding the optimal subset of relevant features on classification performance.

For designing the model for Gender Recognition By Voice , I have applied the KNN model,Random Forest and Decision Tree models. Among these models Random Forest Model has performed well and the gender recognition by voice has been done with Random Forest Model.

REFERENCES

https://www.kaggle.com/donghaoqiao/gender-recognition-by-voice-with-python

https://en.wikipedia.org/wiki/Machine_learning

https://www.google.com

https://scikit-learn.org/