DATA SCIENCE

(Prediction Of Heart Disease Occurrence)

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

Bachelor of Technology

In

Computer Science Engineering

By

Sreeja Reddy Dasari

221710313054

Under the Guidance of

Assistant Professor

Department of Computer Science and Engineering

GITAM School of Technology

GITAM(Deemed to be University)

Hyderabad-502329

July 2020

DECLARATION

I submit this industrial training work entitled "**PREDICTION OF HEART DISEASE OCCURRENCE**" to GITAM(Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of "**Bachelor of Technology** " in " **Computer Science Engineering**". I declare that it was carried out independently by me under the guidance of ________, _________, 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: **Sreeja Reddy Dasari**

Hyderabad **221710313054**

//certificate

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. CH. Sanjay,** Principal, GITAM Hyderabad.

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

I would like to thank the respected faculties **Mr. M. Venkateswarlu** 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.

SREEJA REDDY DASARI
221710313054

ABSTRACT

In recent times, Heart disease is one of the biggest causes of morbidity and mortality among the population of the world. Prediction of cardiovascular disease is regarded as one of the most important subjects in the section of clinical data analysis. The amount of data in the healthcare industry is huge. Data mining turns the large collection of raw healthcare data into information that can help to make informed decisions and predictions

This makes heart disease a major concern to be dealt with. But it is difficult to identify heart disease because of several contributory risk factors such as diabetes, high blood pressure, high cholesterol, abnormal pulse rate, and many other factors. Due to such constraints, scientists have turned towards modern approaches like Data Mining and Machine Learning for predicting the disease.

Machine learning algorithms are used to predict the values from the dataset by splitting the dataset into train and test and building Machine learning algorithms models of higher accuracy to predict the values is the primary task to be performed on prediction of heart disease

Table of Contents

CHAPTER 5. DATA PREPROCESSING / FEATURE ENGINEERING & EDA

CHAPTER 6.FEATURES SELECTION

CHAPTER 7.MODEL BUILDING AND EVALUATION

Figure Index : Figure Name--page no

CHAPTER 1

DATA SCIENCE

INFORMATION ABOUT DATA SCIENCE

1.1 What is Data Science

Data science is a multidisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from data in various forms, both structured and unstructured, [1] [2] similar to data mining.

1.2 Need of Data Science

Industries need data to help them make careful decisions. Data Science churns raw data into meaningful insights. Therefore, industries need data science. A Data Scientist is a wizard who knows how to create magic using data. The model will know how to dig out meaningful information with whatever data he comes across. The company requires strong data-driven decisions. The Data Scientist is an expert in various underlying fields of Statistics and Computer Science.

Companies are applying big data and data science to everyday activities to bring value to consumers. Banking institutions are capitalizing on big data to enhance their fraud detection successes. Asset management firms are using big data to predict the likelihood of a security's price moving up or down at a stated time.

Companies such as Netflix mine big data to determine what products to deliver to its users. Netflix also uses algorithms to create personalized recommendations for users based on their viewing history. Data science is evolving at a rapid rate, and its applications will continue to change lives into the future.

1.3 Uses of Data Science

Fraud and Risk Detection

The earliest applications of data science were in Finance. Companies were fed up with bad debts and losses every year. However, they had a lot of data which used to get collected during the initial paperwork while sanctioning loans. They decided to bring in a data scientist in order to rescue them out of losses.

Over the years, banking companies learned to divide and conquer data via customer profiling, past expenditures, and other essential variables to analyze the probabilities of risk and default. Moreover, it also helped them to push their banking products based on the customer's purchasing power.

Healthcare

1.Medical Image Analysis:

Procedures such as detecting tumours, artery stenosis, organ delineation employ various different methods and frameworks like Map Reduce to find optimal parameters for tasks like lung texture classification. It applies machine learning methods, support vector machines (SVM), content-based medical image indexing, and wavelet analysis for solid texture classification.

2. assistance for patients and customer supporVirtualt:

The AI-powered mobile apps can provide basic healthcare support, usually as chatbots. You simply describe your symptoms, or ask questions, and then receive key information about your medical condition derived from a wide network linking symptoms to causes. Apps can remind you to take your medicine on time, and if necessary, assign an appointment with a doctor.

Targeted Advertising:

Digital ads have been able to get a lot higher CTR (Call-Through Rate) than traditional advertisements. They can be targeted based on a user’s past behavior.

This is the reason why you might see ads for Data Science Training Programs while I see ads for apparels in the same place at the same time.

CHAPTER 2

INFORMATION ABOUT MACHINE LEARNING

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).

2.1 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.

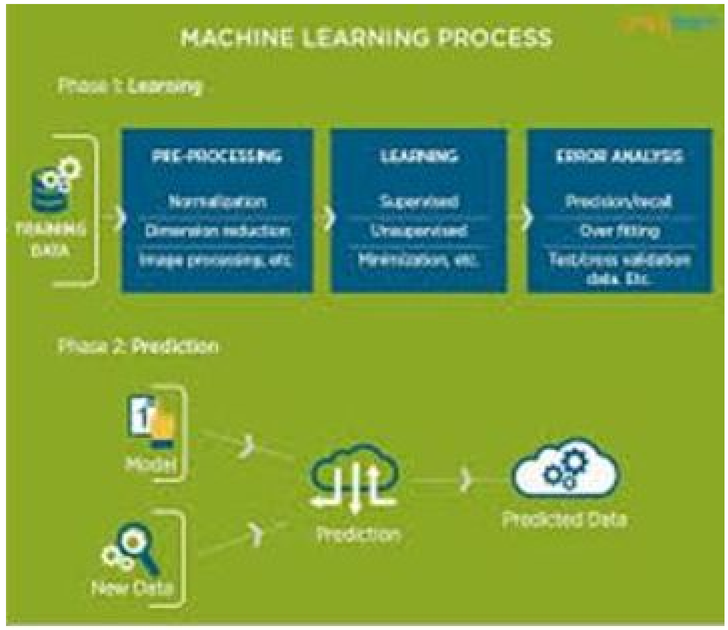


Figure 2.1.1 : The Process Flow

The process flow depicted here represents how machine learning works.

2.2 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.

2.3 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.

2.3.1 Supervised Learning

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning. Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan.Choosing between more than two classes is referred to as multiclass classification.

2.3.2 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.

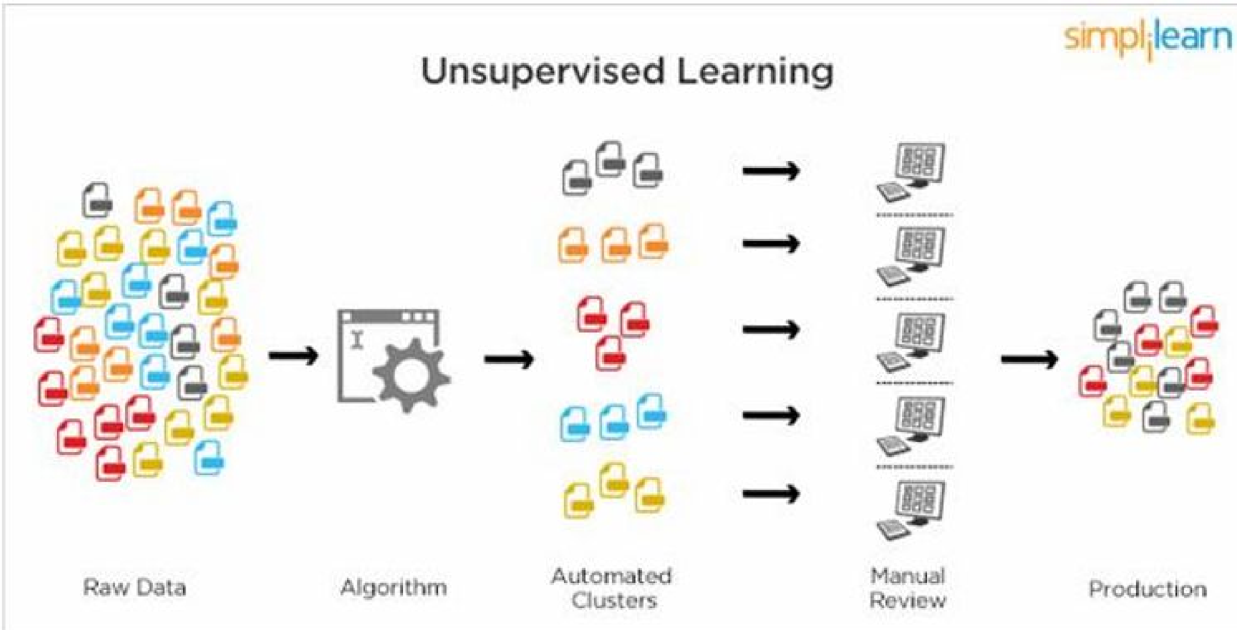


Figure2.3.2.1 : Unsupervised Learning.

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.

2.3.3 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.

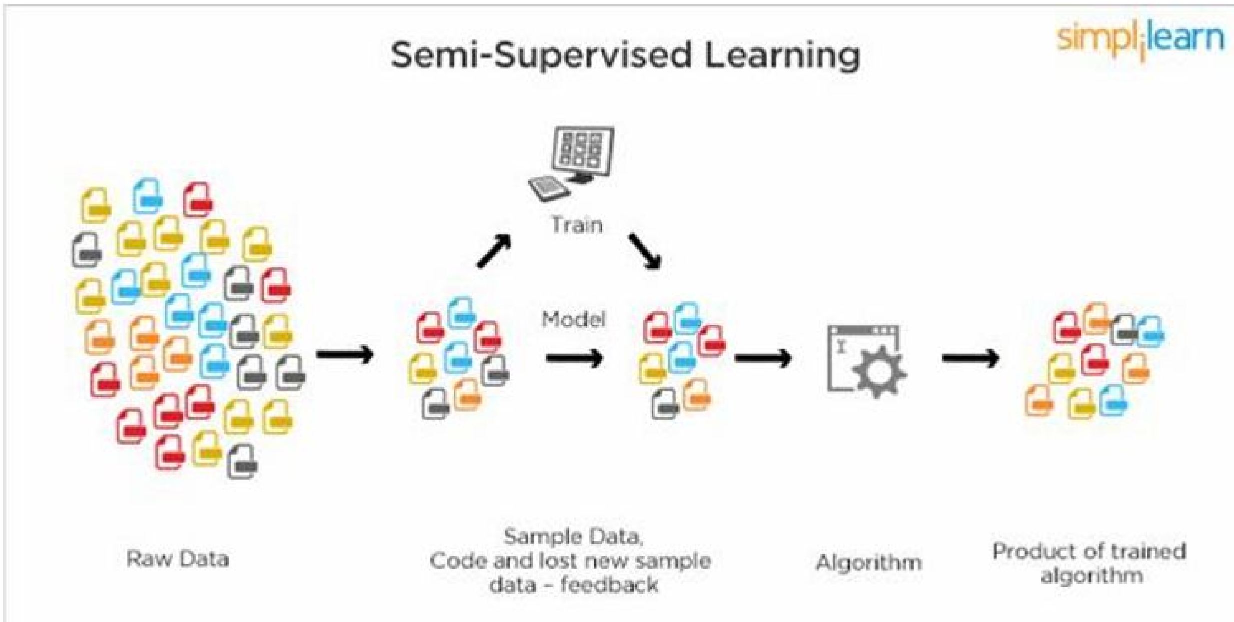


Figure 2.3.3.1 : Semi Supervised Learning

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

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

CHAPTER 3

INFORMATION ABOUT PYTHON

3.1 Introduction

Python is a high-level, interpreted, interactive and object-oriented scripting language.

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

3.2 HISTORY OF PYTHON:

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

3.3 FEATURES OF PYTHON:

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.

- **Easy-to-maintain:** Python's source code is fairly easy-to-maintaining.

- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

3.4 HOW TO SETUP PYTHON:

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

3.4.1 Installation(using python IDLE):

Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.

Download python from www.python.org

When the download is completed, double click the file and follow the instructions to install it.

When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python

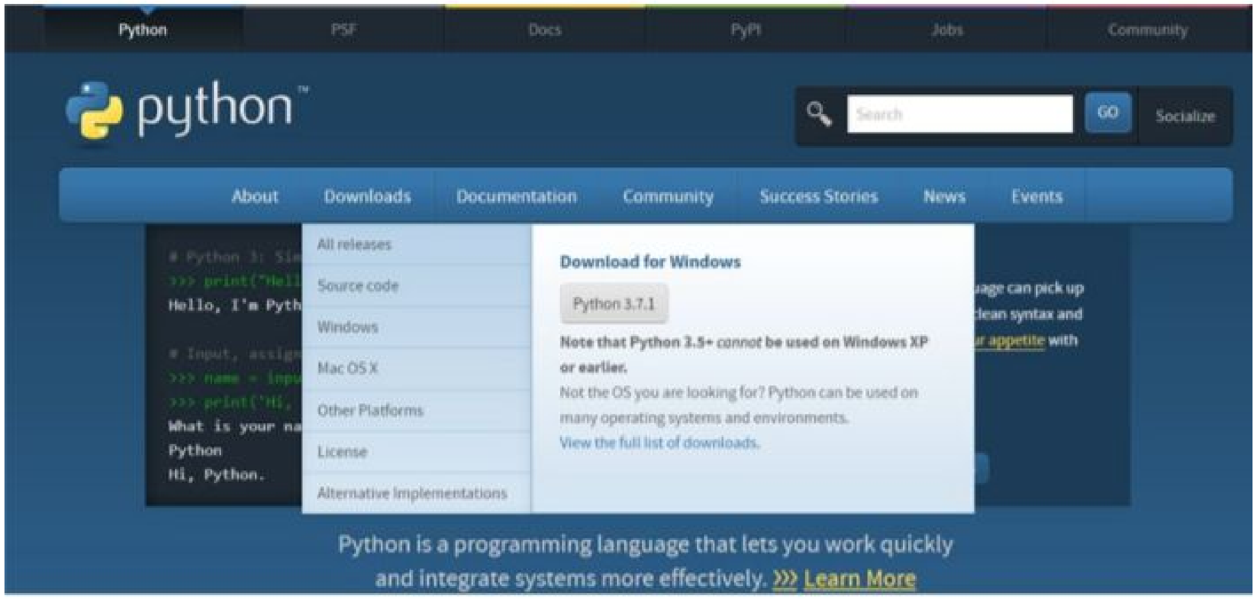


Figure 3.4.1.1 : Python download

3.4.2 Installation(using Anaconda):

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

- Step 5: Open jupyter notebook (it opens in default browser)

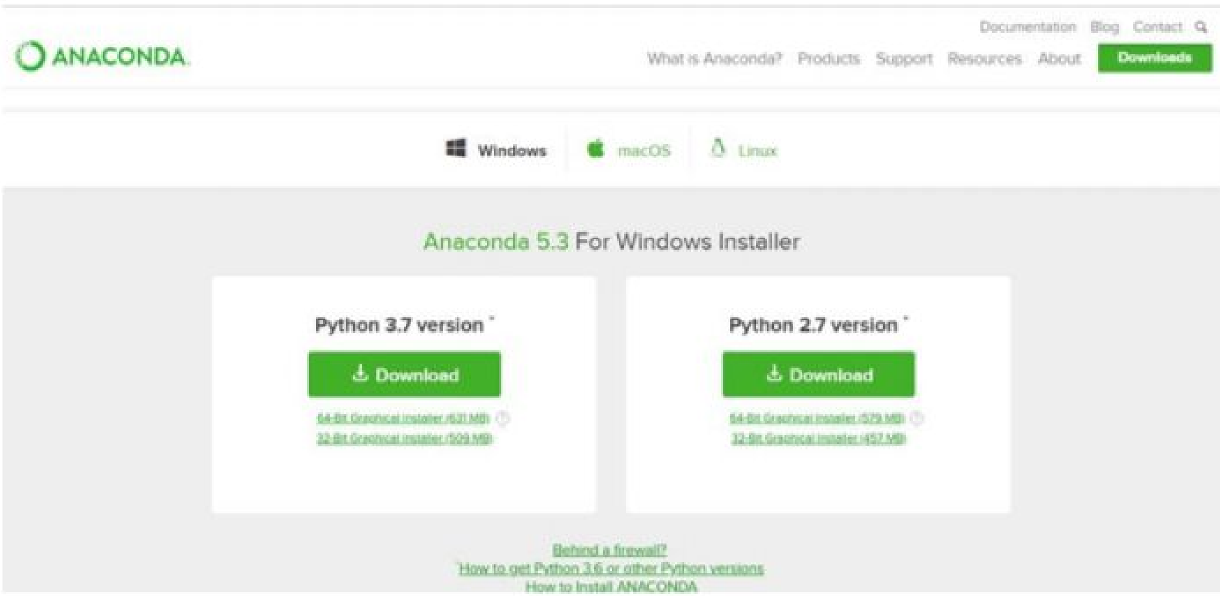


Figure 3.4.2.1 : Anaconda download

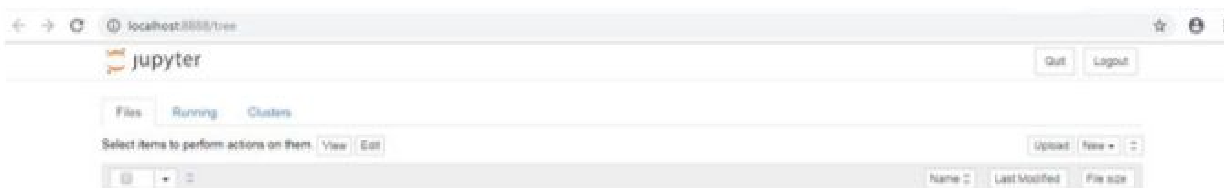


Figure3.4.2.2: Jupyter notebook

3.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.

- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
    - Numbers
    - Strings
    - Lists
    - Tuples
    - Dictionary

3.5.1 Python Numbers:

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

3.5.2 Python Strings:

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

- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

3.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data 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.

3.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add

elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

3.5.5 Python Dictionary:

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

3.6 PYTHON FUNCTION:

3.6.1 Defining a Function:

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

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

3.6.2 Calling a Function:

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

3.7 PYTHON USING OOPs CONCEPTS:

3.7.1 Class:

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

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

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

Figure 3.7.1.1 : Defining a Class

3.7.2 __init__ method in Class:

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

CHAPTER 4

PROJECT NAME(INFORMATION ABOUT THE PROJECT)

4.1 Packages Used:

```
#importing the  required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Figure 4.1.1 : Importing Libraries

4.2 Problem Statement :

The goal is to predict the binary class, heart_disease which represents whether or not a patient has heart disease.

4.3 Dataset Description :

There are 14 columns in the dataset, which are described below.

1. Age: displays the age of the individual.
2. Sex: displays the gender of the individual using the following format :

   1 = male

   0 = female
3. Chest-pain type: displays the type of chest-pain experienced by the individual using the following format :

   1 = typical angina

   2 = atypical angina
4. 3 = non — anginal pain

   4 = asymptotic
5. Resting Blood Pressure: displays the resting blood pressure value of an individual in mmHg (unit)
6. Serum Cholesterol: displays the serum cholesterol in mg/dl (unit)
7. Fasting Blood Sugar: compares the fasting blood sugar value of an individual with 120mg/dl.

   If fasting blood sugar > 120mg/dl then : 1 (true)

   else : 0 (false)
8. Resting ECG : displays resting electrocardiographic results

   0 = normal

   1 = having ST-T wave abnormality

   2 = left ventricular hypertrophy
9. Max heart rate achieved : displays the max heart rate achieved by an individual.
10. Exercise induced angina :

    1 = yes

    0 = no
11. ST depression induced by exercise relative to rest: displays the value which is an integer or float.

12. Peak exercise ST segment :
    1 = upsloping
    2 = flat
    3 = downsloping
13. Number of major vessels (0–3) colored by fluoroscopy : displays the value as integer or float.
14. Thal : displays the thalassemia :
    3 = normal
    6 = fixed defect
    7 = reversible defect
15. Diagnosis of heart disease : Displays whether the individual is suffering from heart disease or not :
    0 = absence
    1, 2, 3, 4 = present.

4.4 Objective of the Case Study:

To get better understanding to make predictions on whether a person is suffering from Heart Disease or not by considering the features of the data and provide the client with desired results.

CHAPTER 5

DATA PREPROCESSING/FEATURE ENGINEERING AND EDA

5.1 IMPORTING THE LIBRARIES:

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

```
#importing the  required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Figure 5.1.1 : Importing Libraries

5.2 READING 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 accessed using the dataframe. Any missing value or NaN value has to be cleaned.

```
#reading the data
data=pd.read_csv("/content/drive/My Drive/Summer Internship/datasets_216167_477177_heart.csv")
data
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

Figure 5.2.1 : Reading the dataset

5.3 HANDLING MISSING VALUES

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

1. dropna()
2. fillna()
3. interpolate()
4. mean imputation and median imputation .

1. dropna():

dropna() is a function which drops all the rows and columns which are having the missing values(i.e. NaN).

dropna() function has a parameter called how which works as follows:

2. fillna():

fillna() is a function which replaces all the missing values using different ways

fillna() also have parameters called method and axis.

3. interpolate():

● interpolate() is a function which comes up with a guess value based on the other values in the dataset and fills those guess values in the place of missing values .

4. mean and median imputation

● mean and median imputation can be performed by using fillna().

● mean imputation calculates the mean for the entire column and replaces the missing values in that column with the calculated mean.

● median imputation calculates the median for the entire column and replaces the missing values in that column with the calculated median.

Missing values can be checked using isna() or isnull() functions which returns the output in a boolean format.

Total number of missing values in each column can be calculated using isna().sum() or isnull().sum().

```
#to detect the missing values in the dataset
data.isnull()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1021	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1022	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1023	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1024	False	False	False	False	False	False	False	False	False	False	False	False	False	False

1025 rows × 14 columns

Figure 5.3.1: Checking missing values.

From above we can observe that we have used isnull() in order to check whether there are any missing values present in the given dataset .

```
[14] #calculating the number of null values
     data.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Figure 5.3.2 : Total number of missing values in each column.

From the above output we can observe that the given dataset does not contain any missing values**.**

5.4 CATEGORICAL DATA:

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

Categorical Variables are of two types: Nominal and Ordinal

- **Nominal:**

The categories do not have any numeric ordering in between them. They don't have any ordered relationship between each of them. Examples: Male or Female, any colour

- **Ordinal:**

The categories have a numerical ordering in between them. Example: Graduate is less than Post Graduate, Post Graduate is less than Ph.D. customer satisfaction survey, high low medium

- Categorical data can be handled by using dummy variables, which are also called as indicator variables.
- Handling categorical data using dummies: In pandas library we have a method called get_dummies() which creates dummy variables for those categorical data in the form of 0's and 1's. Once these dummies have been created we have to concat this dummy set to our dataframe or we can add that dummy set to the dataframe.

```
[6] #checking the dtypes
    data.dtypes

age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

Figure 5.4.1 : Description about the type of each feature in the dataset.(Categorical or Numerical).

From the above figure it looks like it returns a series with the datatype of each column.

5.5 Statistical Analysis

```
[17] #calculating the mean
     data.mean()
```

```
age          54.434146
sex           0.695610
cp            0.942439
trestbps    131.611707
chol        246.000000
fbs           0.149268
restecg       0.529756
thalach     149.114146
exang         0.336585
oldpeak       1.071512
slope         1.385366
ca            0.754146
thal          2.323902
target        0.513171
dtype: float64
```

Figure 5.5.1: Mean of the given data set

From the above figure it shows the mean of the given dataset,, we usually do this by dividing the sum of given numbers with the count of the number present.

```
[18] #calculating the median
     data.median()
```

```
age          56.0
sex           1.0
cp            1.0
trestbps    130.0
chol        240.0
fbs           0.0
restecg       1.0
thalach     152.0
exang         0.0
oldpeak       0.8
slope         1.0
ca            0.0
thal          2.0
target        1.0
dtype: float64
```

Figure 5.5.2: median of the given dataset

From the above figure the median of the given dataset is calculated as shown above.

5.6 Generating Plots

5.6.1 Visualize the data between Target and the Features

```
[66] # Visualizing the data using heatmap
     sns.heatmap(data)

<matplotlib.axes._subplots.AxesSubplot at 0x7f9a95fbf860>
```

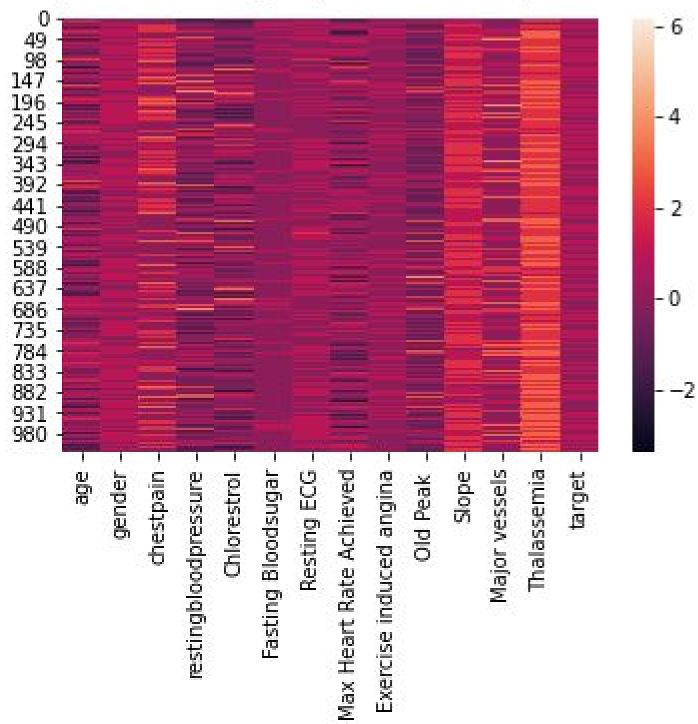


Figure 5.6.1.1: visualizing the data using heatmap

The above figure shows the data visualisation by using a heat map which displays numeric tabular data where the cells are coloured depending upon the contained value.

```
sns.countplot(x='chestpain', hue='gender',data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a9bc37d30>
```

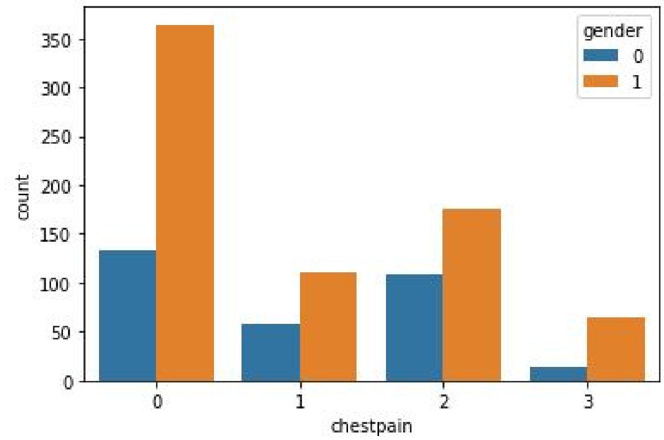


Figure 5.6.1.2: countplot graph

The above graph shows the chest pain count experienced with respect to male and female.

Here 1 means male and 0 denotes female. We observe females having heart disease are comparatively less when compared to males.We can observe that male are getting more chest pain when compared to that of females.

```
[36] sns.countplot(x='gender', data=data,  hue='target')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9a996ddba8>
```

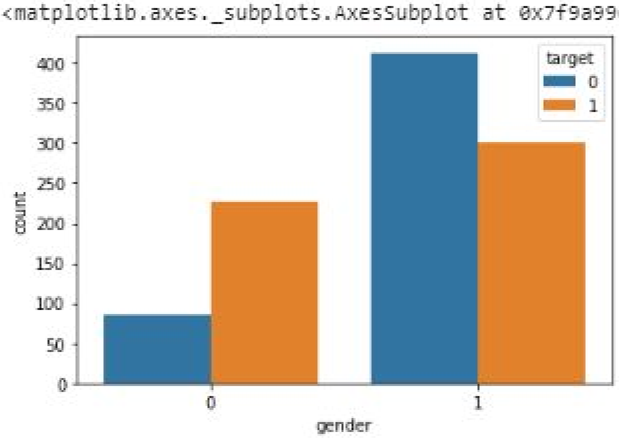


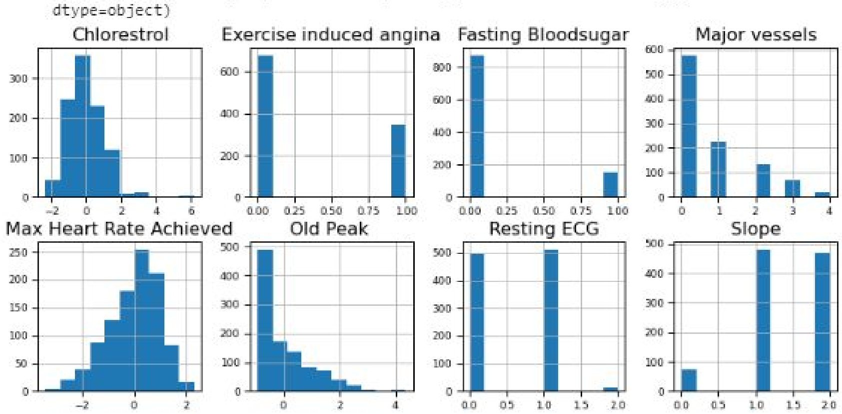
Figure 5.6.1.3: Bar graph based on gender and target

Here 1 means male and 0 denotes female. We observe females having heart disease are comparatively less when compared to males.

Males have low heart diseases as compared to females in the given dataset.

```
data.hist(figsize=(12,12))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe2ba4278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe2b040f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe2980fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe293d390>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe28ef710>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe28a1a90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe2854e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe2814198>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe2814208>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe27f78d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe27abc50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe275bfd0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe2719390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe274b710>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe26fca90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbbe26b2e10>]],
      dtype=object)
```



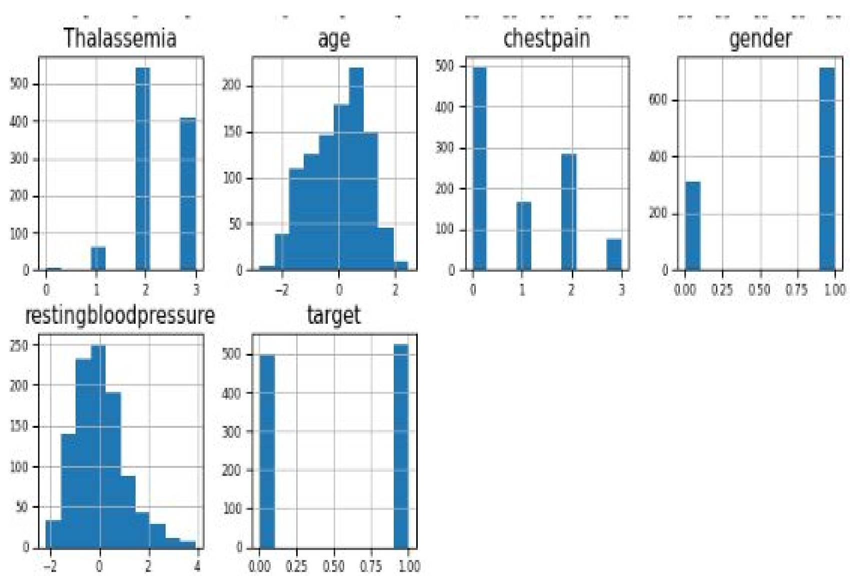


Figure 5.6.1.4: histogram constructed between all the columns present.

The above figure conveys the histogram plotted between the data which is present.

The histogram is constructed which gives an approximate representation of the distribution of numerical data which is present.

The above graph also shows the frequency distributions .

```
sns.countplot(x='chestpain' ,hue='target', data=data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbe2a496d8>
```

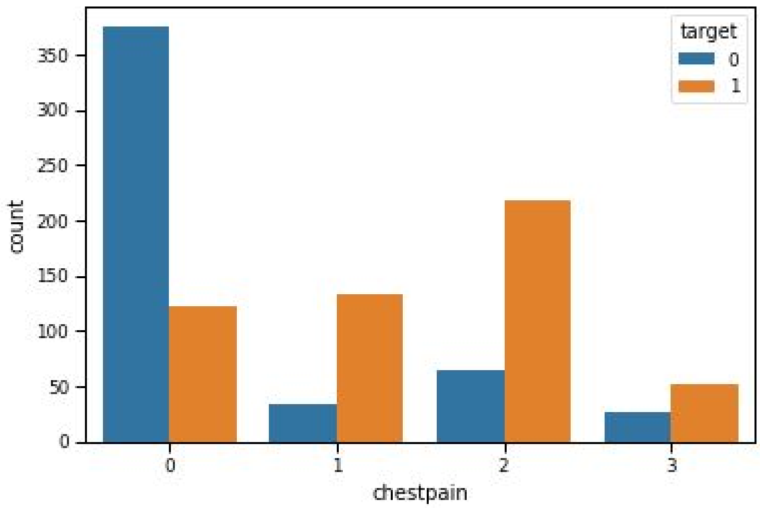


Figure 5.6.1.5: Bar graph plotted for the count of chest pain

The above plotted graph depicts the count of chest pain experienced by male and female.

Here 1 means male and 0 denotes female.

From the above plotted graph we can see that the count of chest pain for male is greater when compared to that of females.

CHAPTER 6

FEATURE SELECTION

6.1 Select relevant features for the analysis

```
[10] #checked whether there are balanced number of 0's  and 1's
     data['target'].value_counts()

     1    526
     0    499
     Name: target, dtype: int64
```

Figure 6.1.1: imbalanced data

As there is no large difference between the target values so there is no need to balance the data as it would not give much difference.

6.2 Train-Test-Split

Splitting the data : after the preprocessing is done then the data is split into train and test sets.

- In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set'. An important point to note is that during training the classifier only uses the training set . The test set must not be used during training the classifier. The 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) .

```
[295] #Splitting the dataset into training and test data.
      # 80% of the data will be in training data and 20% of the data will be in testing
      X = data.drop(['target'],axis=1)
      y = data.target
      from sklearn.model_selection import train_test_split
      X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

```
[296] print('X_train-', X_train.size)
      print('X_test-',X_test.size)
      print('y_train-', y_train.size)
      print('y_test-', y_test.size)
```

```
X_train- 10660
X_test- 2665
y_train- 820
y_test- 205
```

Figure 6.2.1:importing train_test_split and splitting the data

We have split the data into training data and test data and have printed the size of that train and test data.

6.3 Feature Scaling :

Feature Scaling--> when applied, this units and scaling will be removed To make the data unitless and scaleless, we have to apply Feature Scaling.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
StandardScaler = StandardScaler()
columns_to_scale = ['age','restingbloodpressure','Chlorestrol','Max Heart Rate Achieved','Old Peak ']
data[columns_to_scale] = StandardScaler.fit_transform(data[columns_to_scale])
data.head()
```

	age	gender	chestpain	restingbloodpressure	Chlorestrol	Fasting Bloodsugar	Resting ECG	Max Heart Rate Achieved	Exercise induced angina	Old Peak	Slope	v
0	-0.268437	1	0	-0.377636	-0.659332	0	1	0.821321	0	-0.060888	2	
1	-0.158157	1	0	0.479107	-0.833861	1	0	0.255968	1	1.727137	0	
2	1.716595	1	0	0.764688	-1.396233	0	1	-1.048692	1	1.301417	0	
3	0.724079	1	0	0.936037	-0.833861	0	1	0.516900	0	-0.912329	2	
4	0.834359	0	0	0.364875	0.930822	1	1	-1.874977	0	0.705408	1	

Figure 6.3.1: feature scaling data

I have used StandardScalar from sklearn to scale my dataset .

Scaling data is the process of increasing or decreasing the magnitude according to a fixed ratio , in simpler words you change the size but not the shape of the data . .

CHAPTER 7

MODEL BUILDING AND EVALUATION

7.1 Logistic regression

Logistic Regression is used when the dependent variable(target) is categorical.

In a binary logistic regression model, the dependent variable has two levels (categorical). Outputs with more than two values are modeled by multinomial logistic regression and, if the multiple categories are ordered, by ordinal logistic regression (for example the proportional odds ordinal logistic model). The logistic regression model itself simply models probability of output in terms of input and does not perform statistical classification (it is not a classifier), though it can be used to make a classifier, for instance by choosing a cutoff value and classifying inputs with probability greater than the cutoff as one class, below the cutoff as the other; this is a common way to make a binary classifier.

Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis and it predicts the probability

Example: Yes or No, get a disease or not, pass or fail, defective or non-defective, etc.,

Also called a classification algorithm, because we are classifying the data. It predicts the probability associated with each dependent variable category.

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression() # creating an object for Logistic Regression
## We have to apply this object(log_reg) to the training data
m1=log_reg.fit(X_train, y_train) # with help of fit method we are fitting the
                           ##Logistic Regression on training data
## objectName.fit(InputData, OutputData)
```

Figure 7.1.1 : Applying logistic regression on training data.

Instead of directly predicting on test data, let us see how well the model predicts the training data.

<u>Predicting on training data</u>

```
## Predicting on the training data
## Syntax: objectName.predict(TrainInput)
y_train_pred = log_reg.predict(X_train)
y_train ==y_train_pred  # comparing original data output and model predicted output
```

```
315      True
204      True
363      True
5        True
1017     True
         ...
835      True
192      True
629     False
559      True
684      True
Name: target, Length: 820, dtype: bool
```

Figure 7.1.2 : Predicting on train data and comparing the predicted value with the original one .

```
#classification report on training data
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_train,y_train_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.83      0.85       401
           1       0.84      0.90      0.87       419

    accuracy                           0.86       820
   macro avg       0.86      0.86      0.86       820
weighted avg       0.86      0.86      0.86       820
```

Figure 7.1.3: Overall performance of the logistic regression model based on training data.

The reported averages include macro average (averaging the unweighted mean per label), weighted average (averaging the support-weighted mean per label), and sample average (only for multilabel classification). Micro average (averaging the total true positives, false negatives and false positives) is only shown for multi-label or multi-class with a subset of classes, because it corresponds to accuracy otherwise.

Predicting on test data

```
[ ]  ## Predict the model on Test:
     y_test_pred = log_reg.predict(X_test)
     y_test==y_test_pred
```

```
807     True
27     False
77      True
406     True
886     True
       ...
877     True
320     True
362     True
452     True
500     True
Name: target, Length: 205, dtype: bool
```

Figure 7.1.4 : Predicting on test data and comparing the predicted value with the original test data.

```
#classification report on testing data
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,y_test_pred))
```

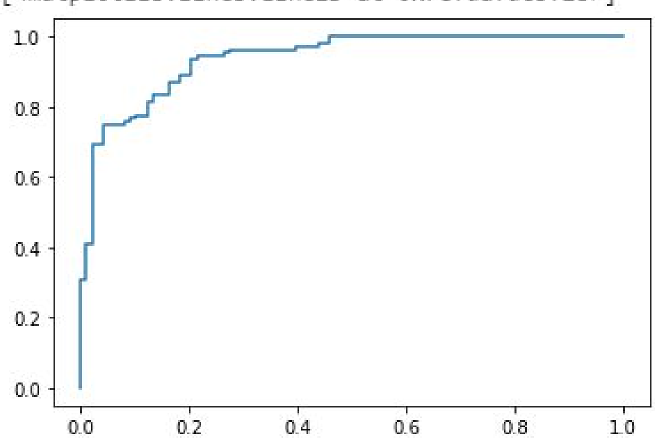
```
              precision    recall  f1-score   support

           0       0.92      0.79      0.85        98
           1       0.83      0.93      0.88       107

    accuracy                           0.86       205
   macro avg       0.87      0.86      0.86       205
weighted avg       0.87      0.86      0.86       205
```

Figure 7.1.5: Overall performance of the logistic regression model based on test data.

The reported averages include macro average (averaging the unweighted mean per label), weighted average (averaging the support-weighted mean per label), and sample average (only for multilabel classification). Micro average (averaging the total true positives, false negatives and false positives) is only shown for multi-label or multi-class with a subset of classes, because it corresponds to accuracy otherwise.

```
#To plot roc curve and find accuracy
from sklearn.metrics import roc_auc_score,roc_curve
prob=m1.predict_proba(X_test)[:,1]
fp1,tp1,threshold1=roc_curve(y_test,prob)
plt.plot(fp1,tp1)
```

```
[<matplotlib.lines.Line2D at 0x7efdafdc3f28>]
```




```
[120] roc_auc_score(y_test,prob)
```

```
0.9387755102040817
```

Figure 7.1.6:Measuring the accuracy of Logistic Regression model using the Area Under the Precision-Recall Curve (AUPRC).

ROC Curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.Precision-Recall curves summarize the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds.

7.2 Naive Bayes

Naive Bayes is the most straightforward and fast classification algorithm, which is suitable for a large chunk of data. Naive Bayes classifier is successfully used in various applications such as spam filtering, text classification, sentiment analysis, and recommender systems. It uses Bayes theorem of probability for prediction of unknown class.

Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- P(h): the probability of hypothesis h being true (regardless of the data). This is known as the prior probability of h.
- P(D): the probability of the data (regardless of the hypothesis). This is known as the prior probability.
- P(h|D): the probability of hypothesis h given the data D. This is known as posterior probability.
- P(D|h): the probability of data d given that the hypothesis h was true. This .

Gaussian Naive Bayes

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

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Figure 7.2.1 : Applying Naive Bayes algorithm

```
[50] ## Predicting on the training data
     ## Syntax: objectName.predict(TrainInput)
     y_train_pred = classifier.predict(X_train)
     y_train ==y_train_pred  # comparing original data output and model predicted output
```

```
880    False
358    False
772     True
682     True
848     True
       ...
905     True
767     True
72      True
908     True
235     True
Name: target, Length: 820, dtype: bool
```

Figure 7.2.2: predicting on training data

```
[68] #classification report on training data
     from sklearn.metrics import classification_report,confusion_matrix
     print(classification_report(y_train,y_train_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.81      0.83       390
           1       0.83      0.87      0.85       430

    accuracy                           0.84       820
   macro avg       0.84      0.84      0.84       820
weighted avg       0.84      0.84      0.84       820
```

Figure 7.2.3: Overall performance of training data.

The reported averages include macro average (averaging the unweighted mean per label), weighted average (averaging the support-weighted mean per label), and sample average (only for multilabel classification). Micro average (averaging the total true positives, false negatives and false positives) is only shown for multi-label or multi-class with a subset of classes, because it corresponds to accuracy otherwise.

▾ predicting on test data

```
[ ] ## Predict the model on Test:
    y_test_pred = classifier.predict(X_test)
    y_test==y_test_pred
```

```
49      True
525     True
119     True
629    False
186     True
       ...
860    False
521    False
790     True
340     True
447     True
Name: target, Length: 205, dtype: bool
```

Figure 7.2.4: predicting on test data

```
[ ] #classification report on testing data
    from sklearn.metrics import classification_report,confusion_matrix
    print(classification_report(y_test,y_test_pred))
```

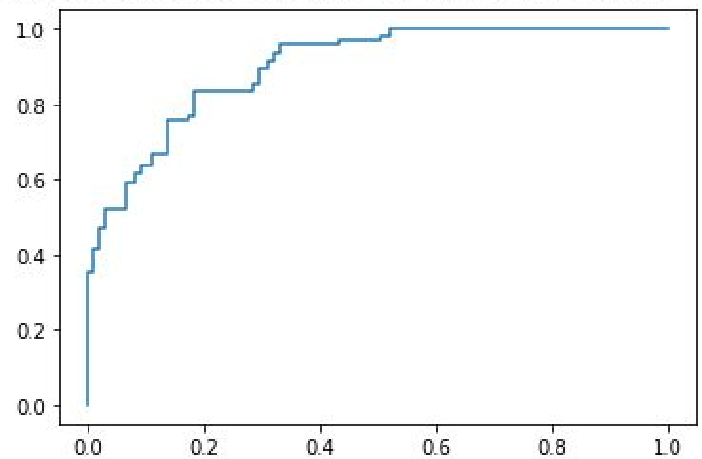
```
              precision    recall  f1-score   support

           0       0.86      0.71      0.77       109
           1       0.72      0.86      0.79        96

    accuracy                           0.78       205
   macro avg       0.79      0.79      0.78       205
weighted avg       0.79      0.78      0.78       205
```

Figure 7.2.5: Overall performance of test data

The reported averages include macro average (averaging the unweighted mean per label), weighted average (averaging the support-weighted mean per label), and sample average (only for multilabel classification). Micro average (averaging the total true positives, false negatives and false positives) is only shown for multi-label or multi-class with a subset of classes, because it corresponds to accuracy otherwise.

```
#To plot roc curve and find accuracy
from sklearn.metrics import roc_auc_score
prob1=m2.predict_proba(X_test)[:,1]
fp2,tp2,threshold2=roc_curve(y_test,prob1)
plt.plot(fp2,tp2)
```

```
[<matplotlib.lines.Line2D at 0x7efdafcc62b0>]
```




```
[130] roc_auc_score(y_test,prob1)
```

```
0.9002293577981652
```

Figure 7.2.6:Measuring the accuracy of naive bayes model using the Area Under the Precision-Recall Curve (AUPRC).

ROC Curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.Precision-Recall curves summarize the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds.

7.3 K Nearest Neighbour

A supervised machine learning algorithm is one that relies on labeled input data to learn a function that produces an appropriate output when given new unlabeled data.

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. However, it is more widely used in classification problems in the industry.It is used for classification and regression of known data where usually the target variable is known beforehand.

K nearest neighbors is an algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). It should also be noted that all three distance measures are only valid for continuous variables.

The 'k' stands for the number of nearest neighbors for the newly entered value.
The kNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. kNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood—calculating the distance between points on a graph.

This works based on minimum distance from the query instance to the training samples to determine the k-nearest neighbors. After we gather these k-nearest neighbors, we take the simple majority of these k nearest neighbors to be the prediction of the query instance.

KNeighoursClassifier

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 2)
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier(n_neighbors=5, metric='euclidean')## creating an object for KNeighborsClassifier
## We have to apply this object() to the training data
KNN.fit(X_train, y_train) # with help of fit method we are fitting the
                              ##KNeighborsClassifier on training data
## objectName.fit(InputData, OutputData)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

Figure 7.3.1 : Applying kNeighbours Classifier algorithm

```
## Predicting on the training data
## Syntax: objectName.predict(TrainInput)
y_train_pred = KNN.predict(X_train)
y_train ==y_train_pred  # comparing original data output and model predicted output
```

```
172     True
128     True
363     True
636     True
798     True
        ...
299    False
534    False
584     True
493     True
527     True
Name: target, Length: 820, dtype: bool
```

Figure 7.3.2: predicting on training data

```
## Predict the model on Test:
y_test_pred = KNN.predict(X_test)
y_test==y_test_pred
```

```
546    False
980    False
908    False
577    False
846     True
       ...
922     True
832    False
451     True
775     True
926    False
Name: target, Length: 205, dtype: bool
```

Figure 7.3.3: predicting on test data

```
#classification report on testing data
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,y_test_pred))
```

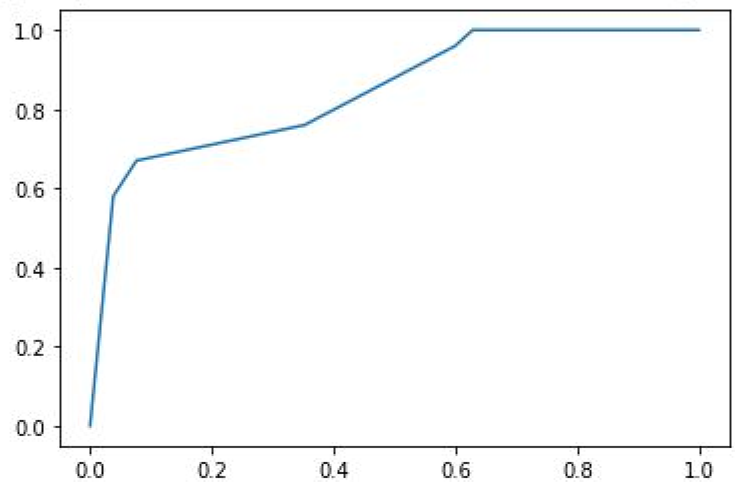
```
              precision    recall  f1-score   support

           0       0.74      0.65      0.69       105
           1       0.67      0.76      0.71       100

    accuracy                           0.70       205
   macro avg       0.71      0.70      0.70       205
weighted avg       0.71      0.70      0.70       205
```

Figure:7.3.4 classification report on testing data

The reported averages include macro average (averaging the unweighted mean per label), weighted average (averaging the support-weighted mean per label), and sample average (only for multilabel classification). Micro average (averaging the total true positives, false negatives and false positives) is only shown for multi-label or multi-class with a subset of classes, because it corresponds to accuracy otherwise.

```
#To plot roc curve and find accuracy
from sklearn.metrics import roc_auc_score,roc_curve
prob3=m3.predict_proba(X_test)[:,1]
fp3,tp3,threshold3=roc_curve(y_test,prob3)
plt.plot(fp3,tp3)
```

```
[<matplotlib.lines.Line2D at 0x7efdafb199b0>]
```




```
[138] roc_auc_score(y_test,prob3)
```

```
0.8447142857142858
```

Figure 7.3.5:Measuring the accuracy of K Neighbour Classifier model using the Area Under the Precision-Recall Curve (AUPRC).

ROC Curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.Precision-Recall curves summarize the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds.

The accuracy for the test set achieved by logistic regression is: 0.86

The accuracy for the test set achieved by naive bayes:0.78

The accuracy for the test set achieved by knn is:0.70

We see that the highest accuracy for the test set is achieved by Logistic Regression which is equal to 86%.

Visualising the best model among logistic regression, NaiveBayes, kNeighboursClassifier

```
models=['Logistic Regression','NaiveBayes','kNeighboursClassifier']
accuracy_scores=[0.86,0.78,0.70]
plt.bar(models,accuracy_scores,color=['lightblue','lightgrey','pink'])
plt.ylabel("accuracy scores")
plt.title("which model has high accuracy")
plt.show()
```

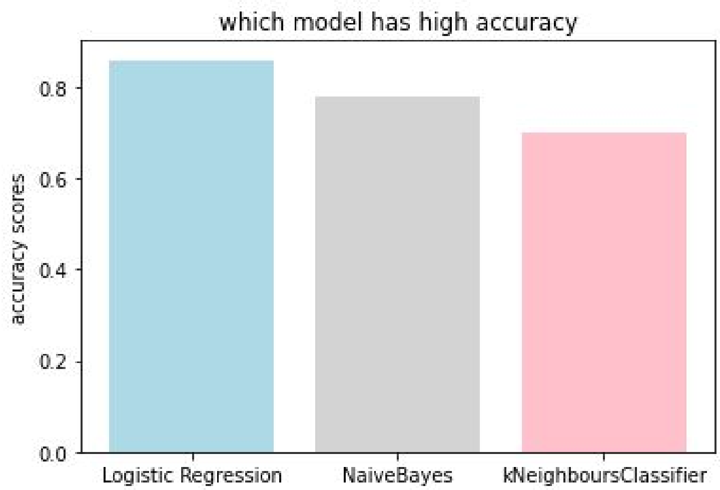


Figure 7.3.6 :Comparison of applied models

CONCLUSION

Heart Disease is one of the major concerns for society today.It is difficult to manually determine the odds of getting heart disease based on risk factors. However, machine learning techniques are useful to predict the output from existing data.

There are many possible improvements that could be explored to improve the scalability and accuracy of this prediction system. As we have developed a generalized system, in future we can use this system for the analysis of different data sets. The performance of the health's diagnosis can be improved significantly by handling numerous class labels in the prediction process, and it can be another positive direction of research.

REFERENCES

https://www.edureka.co/blog/data-science-applications/

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

https://www.edureka.co/blog/logistic-regression-in-python/#:~:text=two%20possible%20outcomes.-,Logistic%20Regression%20In%20Python,variable%20is%20categorical%20in%20nature.

https://www.kaggle.com/johnsmith88/heart-disease-dataset