DATA SCIENCE WITH PYTHON

(Classification Of Liver Diseases)

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

Bachelor of Technology
In
Computer Science Engineering

By
M.SHREYA REDDY
221710305031

GITAM
(DEEMED TO BE UNIVERSITY
(Estd. u/s 3 of the UGC Act, 1956)
VISAKHAPATNAM ✱ HYDERABAD ✱ BENGALURU

JUNE 2020

DECLARATION

I submit this training work entitled **“Classification Of Liver Diseases”** 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 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	M.SHREYA REDDY
Date : 13-07-2020	221710305031

GITAM (DEEMED TO BE UNIVERSITY)
Hyderabad-502329, India

CERTIFICATION

Table of Contents:

CHAPTER 1

DATA SCIENCE

1.1.1 What is Data Science :

Data science is an inter-disciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from many structural and unstructured data. Data science is related to data mining, deep learning and big data.

Data science is a "concept to unify statistics, data analysis, machine learning, domain knowledge and their related methods" in order to "understand and analyze actual phenomena" with data. It uses techniques and theories drawn from many fields within the context of mathematics, statistics, computer science, domain knowledge and information science. Turing award winner Jim Gray imagined data science as a "fourth paradigm" of science (empirical, theoretical, computational and now data-driven) and asserted that "everything about science is changing because of the impact of information technology" and the data deluge.

1.1.2 Need of Data Science :

The principal purpose of Data Science is to find patterns within data. It uses various statistical techniques to analyze and draw insights from the data. From data extraction, wrangling and pre-processing, a Data Scientist must scrutinize the data thoroughly. Then, he has the responsibility of making predictions from the data. The goal of a Data Scientist is to derive conclusions from the data. Through these conclusions, he is able to assist companies in making smarter business decisions.

Data creates magic. 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. A skilled Data Scientist will know how to dig out meaningful information with whatever data he comes across. He helps the company in the right direction. The company requires strong data-driven decisions at which he’s an expert. The Data Scientist is an expert in various underlying fields of Statistics and Computer Science. He uses his analytical aptitude to solve business problems.

Data Scientist is well versed with problem-solving and is assigned to find patterns in data. His goal is to recognize redundant samples and draw insights from it. Data Science requires a variety of tools to extract information from the data. A Data Scientist is responsible for collecting, storing and maintaining the structured and unstructured form of data.

While the role of Data Science focuses on the analysis and management of data, it is dependent on the area that the company is specialized in. This requires the Data Scientist to have domain knowledge of that particular industry.

1.1.3 Uses of Data Science :

Data scientists tackle questions about the future. They start with big data, characterized by the three V's: volume, variety and velocity. Then, they use it as fodder for algorithms and models. The most cutting-edge data scientists, working in machine learning and AI, make models that automatically self-improve, noting and learning from their mistakes.

Data scientists have changed almost every industry. In medicine, their algorithms help predict patient side effects. In sports, their models and metrics have redefined "athletic potential." Data science has even tackled traffic, with route-optimizing models that capture typical rush hours and weekend lulls.

APPLICATIONS OF DATA SCIENCE :

- Identifying and predicting disease
- Personalized healthcare recommendations
- Optimizing shipping routes in real-time
- Getting the most value out of soccer rosters
- Finding the next slew of world-class athletes
- Stamping out tax fraud
- Automating digital ad placement
- Algorithms that help you find love
- Predicting incarceration rates

CHAPTER 2

MACHINE LEARNING

2.1.1 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).

2.1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and “more items to consider” and “get yourself a little something” on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today’s data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that’s in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

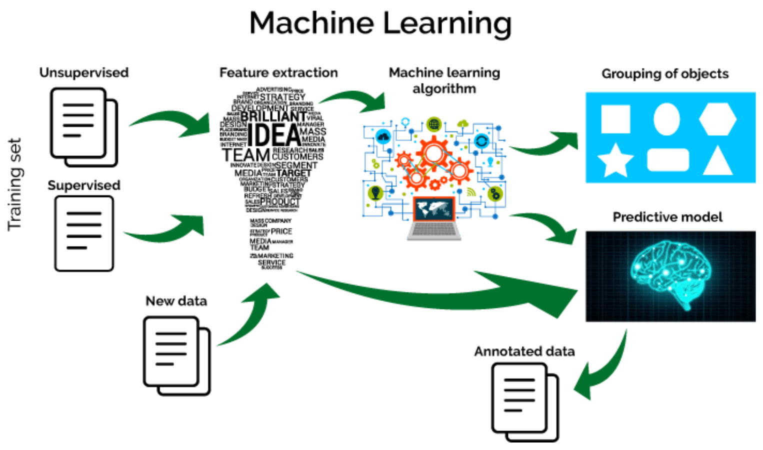


FIG 1 : MACHINE LEARNING

2.1.3 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.

2.1.4 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.

2.1.4 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.

2.1.4 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.

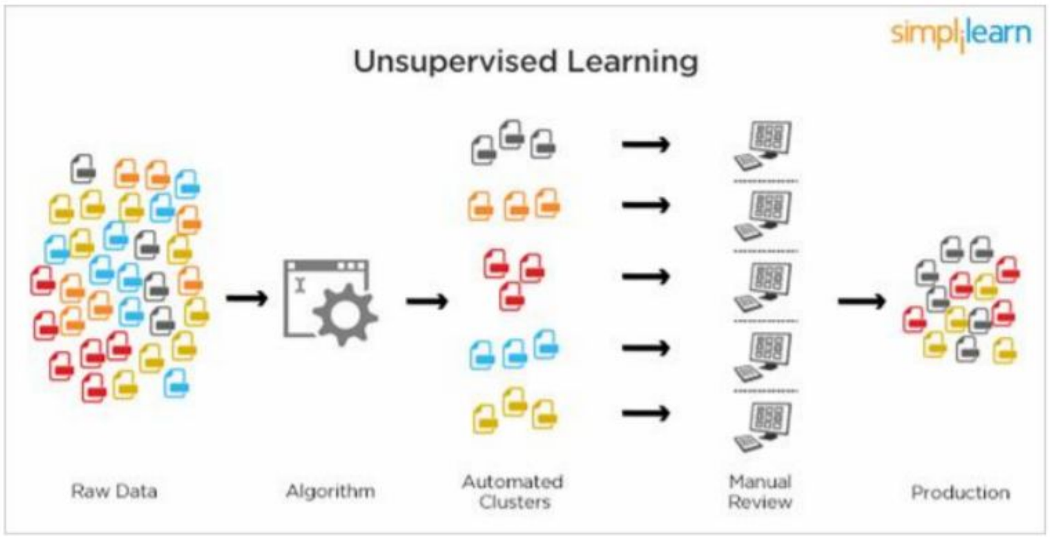


Figure 2 : Unsupervised Learning

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.

2.1.4 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.

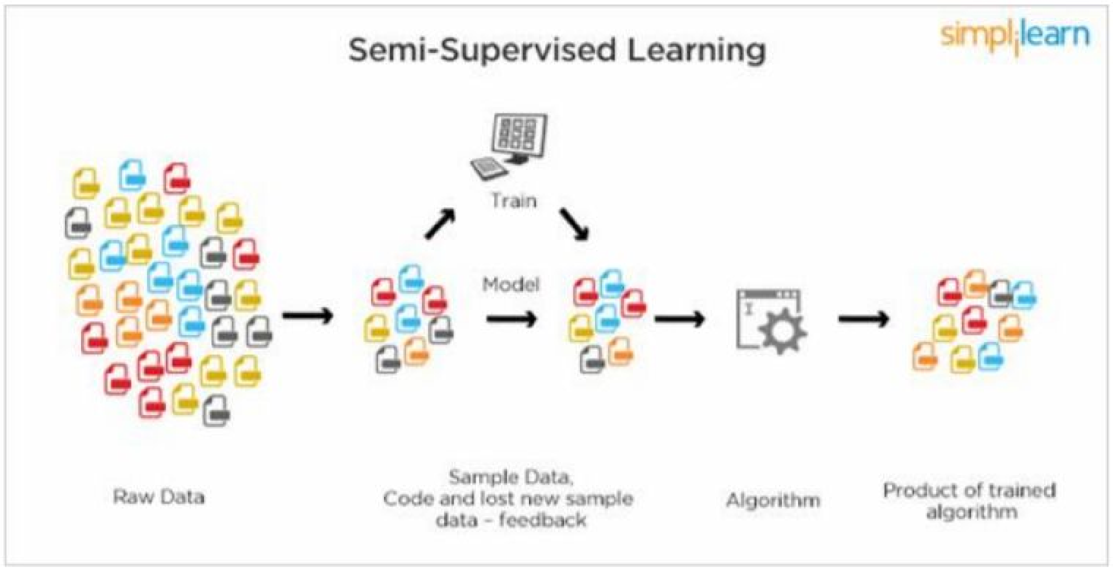

Semi-Supervised Learning
simplilearn
Train
Model
Raw Data
Sample Data, Code and lost new sample data - feedback
Algorithm
Product of trained algorithm

Figure 3 : Semi Supervised Learning

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

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 3

PYTHON

Basic programming language used for machine learning is : PYTHON

3.1 INTRODUCTION TO PYHTON:

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

3.2 HISTORY OF PYTHON:

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

3.3 FEATURES OF PYTHON:

Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
● Easy-to-read: Python code is more clearly defined and visible to the eyes.
● Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
● A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
● Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
● Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
● Databases: Python provides interfaces to all major commercial databases.

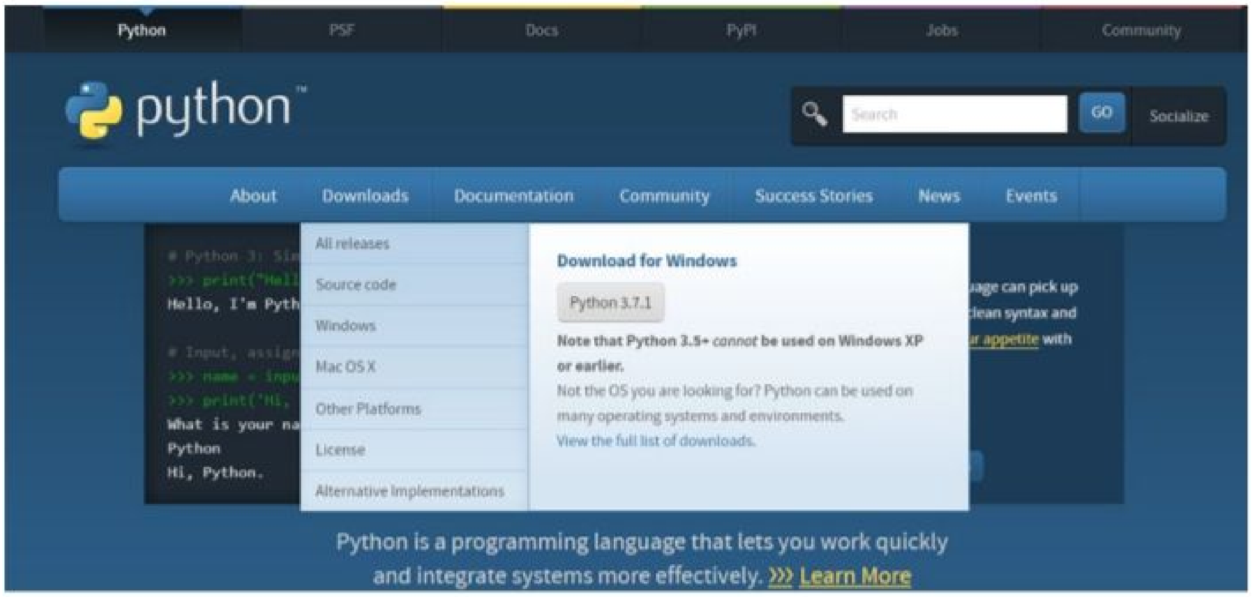
● GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

3.4 HOW TO SETUP PYTHON:

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

3.4.1 Installation(using python IDLE):

Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
● Download python from www.python.org .
● When the download is completed, double click the file and follow the instructions to install it.
● When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.



```
# Python 3: Sim
>>> print("Hell
Hello, I'm Pyth

# Input, assign
>>> name = inpu
>>> print('Hi,
What is your na
Python
Hi, Python.
```

Note that Python 3.5+ *cannot* be used on Windows XP or earlier.
Not the OS you are looking for? Python can be used on many operating systems and environments.
View the full list of downloads.

uage can pick up
clean syntax and
ur appetite with

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

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

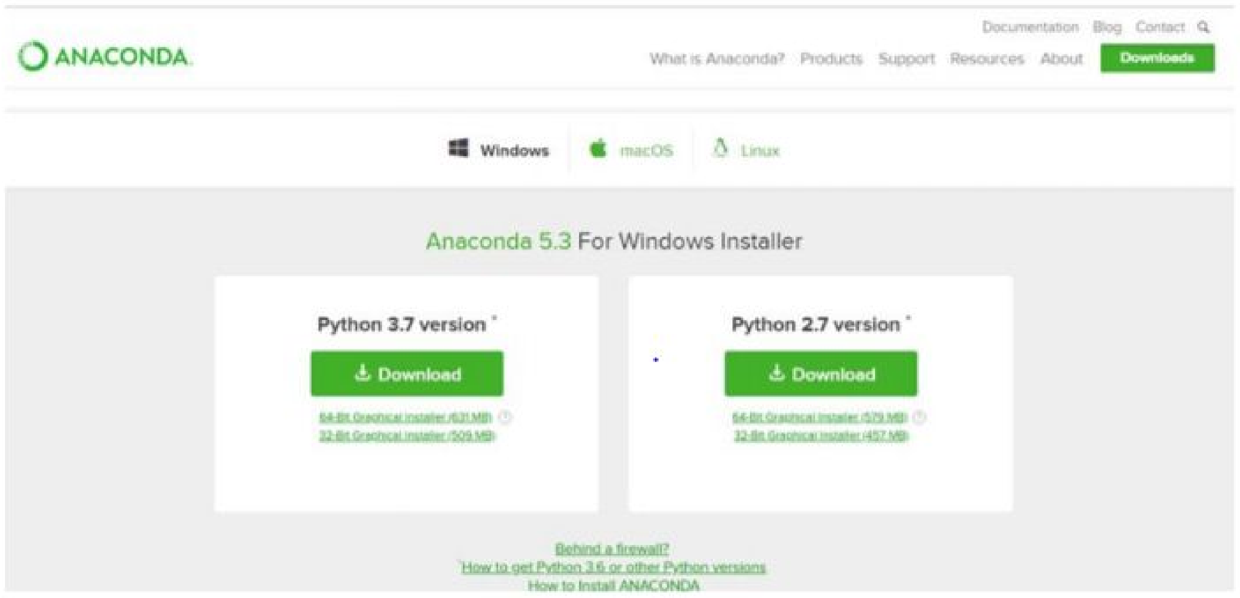

ANACONDA
Documentation Blog Contact
What is Anaconda? Products Support Resources About Downloads
Windows macOS Linux
Anaconda 5.3 For Windows Installer
Python 3.7 version
Download
64-Bit Graphical Installer (631 MB)
32-Bit Graphical Installer (509 MB)
Python 2.7 version
Download
64-Bit Graphical Installer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

Figure 5 : Anaconda download

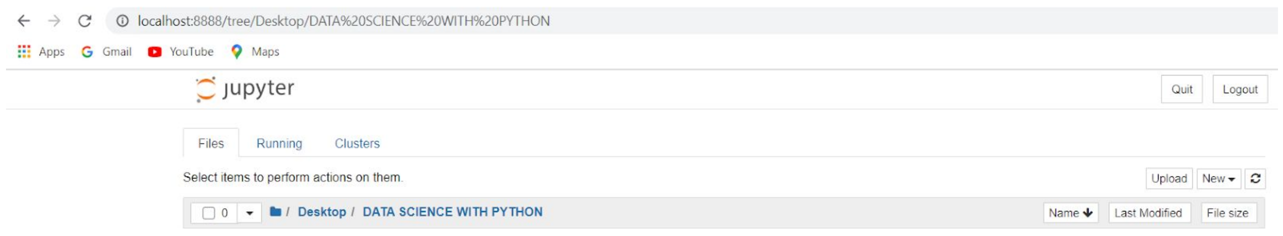

localhost:8888/tree/Desktop/DATA%20SCIENCE%20WITH%20PYTHON
Apps
Gmail
YouTube
Maps
jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
Desktop / DATA SCIENCE WITH PYTHON
Name
Last Modified
File size

Figure 6 : Jupyter notebook

3.5 PYTHON VARIABLE TYPES :

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

● Variables are nothing but reserved memory locations to store values.

● Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.

● Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.

● Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

● Python has five standard data types –

o Numbers
o Strings
o Lists
o Tuples
o Dictionary

3.5.1 Python Numbers:

● Number data types store numeric values. Number objects are created when you assign a value to them.

● Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

3.5.2 Python Strings:

● Strings in Python are identified as a contiguous set of characters represented in the quotation marks.

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

3.5.3 Python Lists:

● Lists are the most versatile of Python's compound data types.
● A list contains items separated by commas and enclosed within square brackets ([]).
● To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data 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 kind of hash table type. They work like associative arrays 12 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 OOP's CONCEPTS:

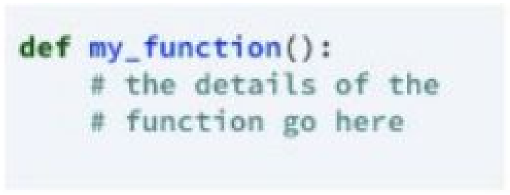
3.7.1 Class:

● 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.

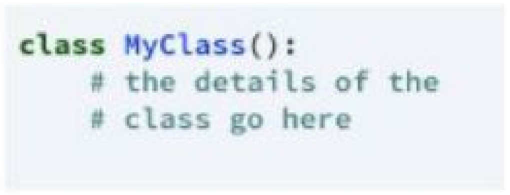
● Data member: A class variable or instance variable that holds data associated with a class and its objects.
● Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

● 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 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

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

CLASSIFICATION OF LIVER DISEASES

4.1 PROJECT REQUIREMENTS :

Context :

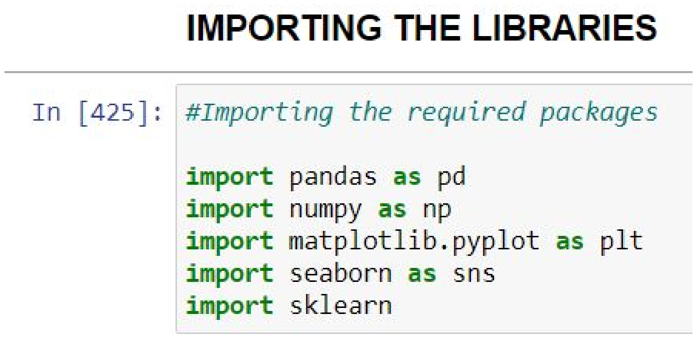
Patients with Liver disease have been continuously increasing because of excessive consumption of alcohol, inhale of harmful gases, intake of contaminated food, pickles and drugs. This dataset was used to evaluate prediction algorithms in an effort to reduce burden on doctors.

Content :

This data set contains 416 liver patient records and 167 non liver patient records collected from North East of Andhra Pradesh, India. The "Dataset" column is a class label used to divide groups into liver patient (liver disease) or not (no disease). This data set contains 441 male patient records and 142 female patient records.

Any patient whose age exceeded 89 is listed as being of age "90".

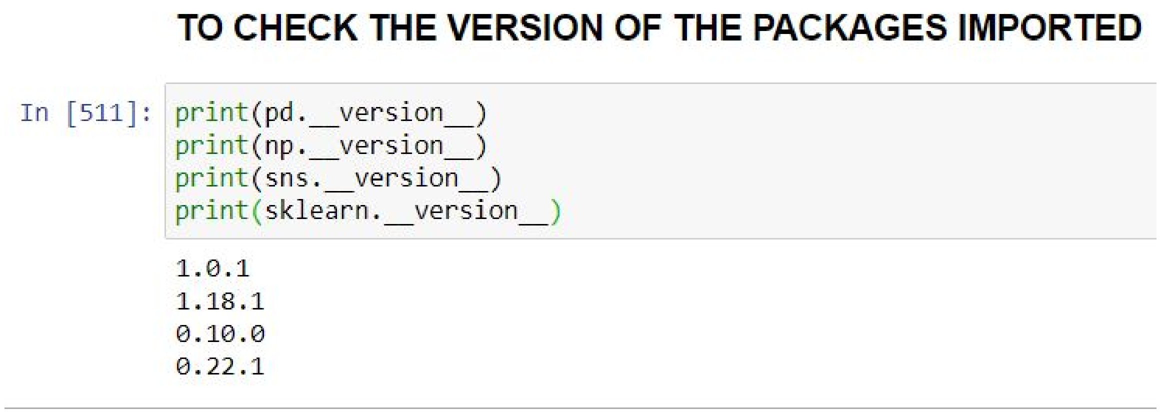
4.1.1 PACKAGES USED :


IMPORTING THE LIBRARIES

```
In [425]: #Importing the required packages

          import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          import seaborn as sns
          import sklearn
```

FIGURE 8 : IMPORTING THE LIBRARIES

4.1.2 VERSION OF THE PACKAGES IMPORTED :


TO CHECK THE VERSION OF THE PACKAGES IMPORTED

```
In [511]: print(pd.__version__)
          print(np.__version__)
          print(sns.__version__)
          print(sklearn.__version__)

          1.0.1
          1.18.1
          0.10.0
          0.22.1
```

FIGURE 9 : VERSION OF THE PACKAGES IMPORTED

4.1.3 ALGORITHMS USED :

Algorithms used on this dataset are :

1) LOGISTIC REGRESSION
2) K-NEAREST NEIGHBORS (KNN)
3) RANDOM FOREST

4.2 PROBLEM STATEMENT :

To predict whether a patient has liver disease or not based on certain features.

In this project, we are going to use the Indian Liver Patient Records dataset from kaggle.

4.3 Dataset Description :

About the Dataset :

This data set contains 416 liver patient records and 167 non liver patient records collected from North East of Andhra Pradesh, India. The "Dataset" column is a class label used to divide groups into liver patient (liver disease) or not (no disease). This data set contains 441 male patient records and 142 female patient records.

Any patient whose age exceeded 89 is listed as being of age "90".

DATA SET :

The given data set consists of the following parameters :

Columns:

- Age of the patient
- Gender of the patient
- Total Bilirubin
- Direct Bilirubin
- Alkaline Phosphotase
- Alamine Aminotransferase
- Aspartate Aminotransferase
- Total Protiens
- Albumin
- Albumin and Globulin Ratio
- Dataset : field used to split the data into two sets (patient with liver disease, or no disease)

4.4 Objective of the Case Study :

Based on chemical compounds(bilrubin,albumin,protiens,alkaline phosphatase) present in the human body and tests like SGOT , SGPT the outcome mentioned whether a person is patient i.e needs to be diagnosed or not.

To apply machine learning algorithms on the dataset to train,test the model and to check for various metrics.

CHAPTER 5

DATA PREPROCESSING/FEATURE ENGINEERING AND EDA

5.1 PREPROCESSING OF THE DATA:

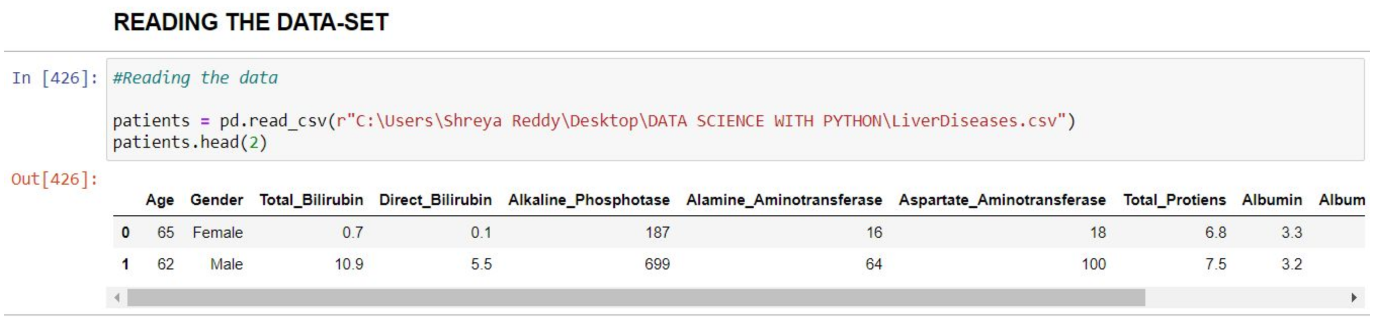
Preprocessing of the data actually involves the following steps:

5.1.1 GETTING THE DATASET:

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

5.1.2 IMPORTING THE DATA-SET :

Pandas in python provide an interesting method read_csv(). The read_csv function reads the entire dataset from a comma separated values file and we can assign it to a DataFrame to which all the operations can be performed. It helps us to access each and every row as well as columns and each and every value can be access using the dataframe. Any missing value or NaN value have to be cleaned.



READING THE DATA-SET

```
In [426]: #Reading the data

          patients = pd.read_csv(r"C:\Users\Shreya Reddy\Desktop\DATA SCIENCE WITH PYTHON\LiverDiseases.csv")
          patients.head(2)
```

Out[426]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Album
0	65	Female	0.7	0.1	187	16	18	6.8	3.3	
1	62	Male	10.9	5.5	699	64	100	7.5	3.2	

FIGURE 10 : READING THE DATASET

5.1.3 HANDLING MISSING VALUES :

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

(a) dropna()
(b) fillna()
(c) interpolate()
(d) mean imputation and median imputation

(a)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

● if how = 'all' is passed then it drops the rows where all the columns of the particular row are missing

● if how = 'any' is passed then it drops the rows where all the columns of the particular row are missing

We haven't dropped any of the rows and columns as of now in this dataset.

(b)fillna() :

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

fillna() also have parameters called method and axis

● if we use method = ’ffill’ where ffill is a method called forward fill, which carry forwards the previous row’s value

● if we use method = ‘bfill’ where bfill is a method called backward fill, which carry backward the next row’s value

● if we use method = ‘ffill’ , axis = ‘columns’ then it carry forwards the previous column’s value

● if we use method = ‘bfill’ , axis = ‘columns’ then it carry backward the next column’s value

(c)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.

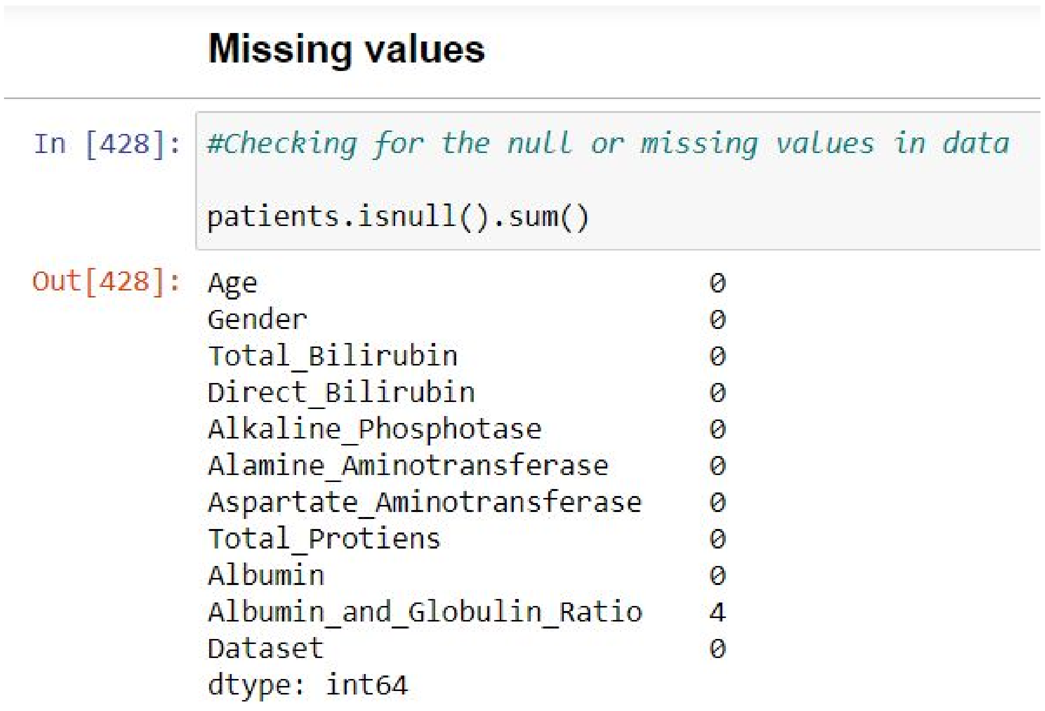
(d)mean and median imputation :

● 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.

TO CHECK FOR THE MISSING VALUES IN THE DATASET :

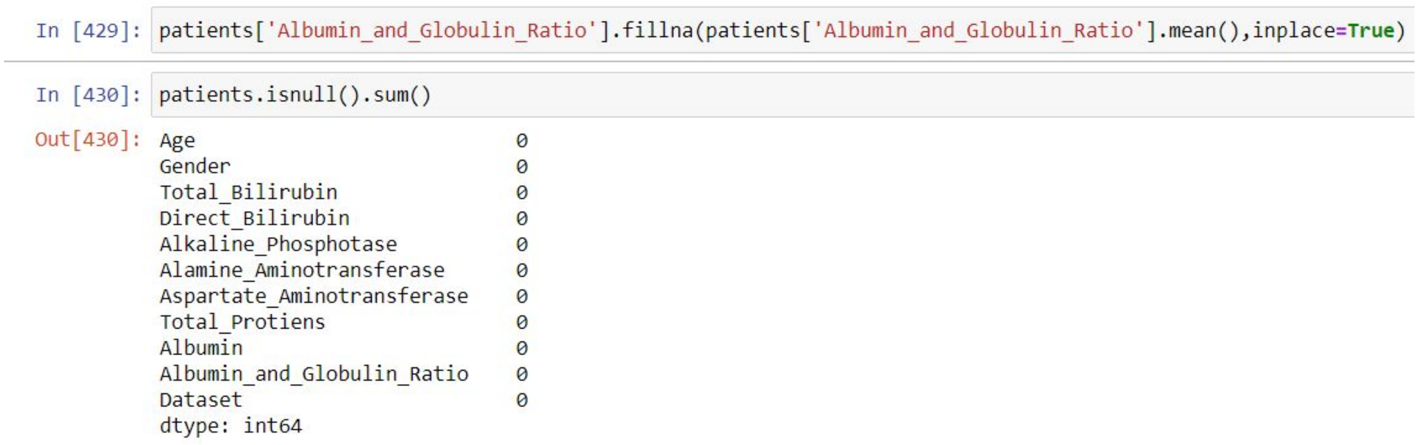


Missing values

```
In [428]: #Checking for the null or missing values in data

          patients.isnull().sum()

Out[428]: Age                            0
          Gender                         0
          Total_Bilirubin                0
          Direct_Bilirubin               0
          Alkaline_Phosphotase           0
          Alamine_Aminotransferase       0
          Aspartate_Aminotransferase     0
          Total_Protiens                 0
          Albumin                        0
          Albumin_and_Globulin_Ratio     4
          Dataset                        0
          dtype: int64
```

FIGURE 11 : MISSING VALUES

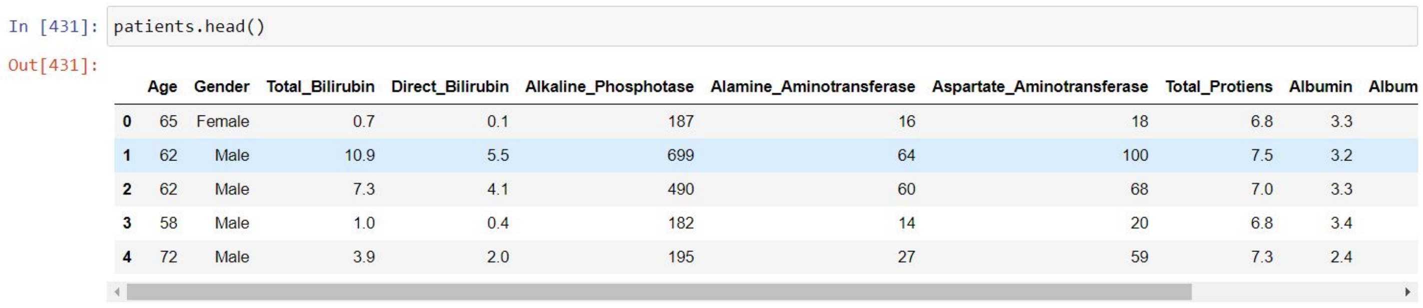
Since Albumin_and_Globulin_Ratio contains 4 null values,they either can be discarded or filled with suitable values. I am going to fill the values with mean of the column.



```
In [429]: patients['Albumin_and_Globulin_Ratio'].fillna(patients['Albumin_and_Globulin_Ratio'].mean(),inplace=True)

In [430]: patients.isnull().sum()

Out[430]: Age                           0
          Gender                        0
          Total_Bilirubin               0
          Direct_Bilirubin              0
          Alkaline_Phosphotase          0
          Alamine_Aminotransferase      0
          Aspartate_Aminotransferase    0
          Total_Protiens                0
          Albumin                       0
          Albumin_and_Globulin_Ratio    0
          Dataset                       0
          dtype: int64
```

FIGURE 12 : FILLING ALBUMIN_and_GLOBULIN_RATIO column WITH MEAN VALUES OF THAT COLUMN

Now as we can see in above figure there are no null values present as we filled the null values in the Albumin_and_Globulin_Ratio column with the mean of its column.



```
In [431]: patients.head()

Out[431]:
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Album
0	65	Female	0.7	0.1	187	16	18	6.8	3.3	
1	62	Male	10.9	5.5	699	64	100	7.5	3.2	
2	62	Male	7.3	4.1	490	60	68	7.0	3.3	
3	58	Male	1.0	0.4	182	14	20	6.8	3.4	
4	72	Male	3.9	2.0	195	27	59	7.3	2.4	

FIGURE 13 : DATASET

5.1.4 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

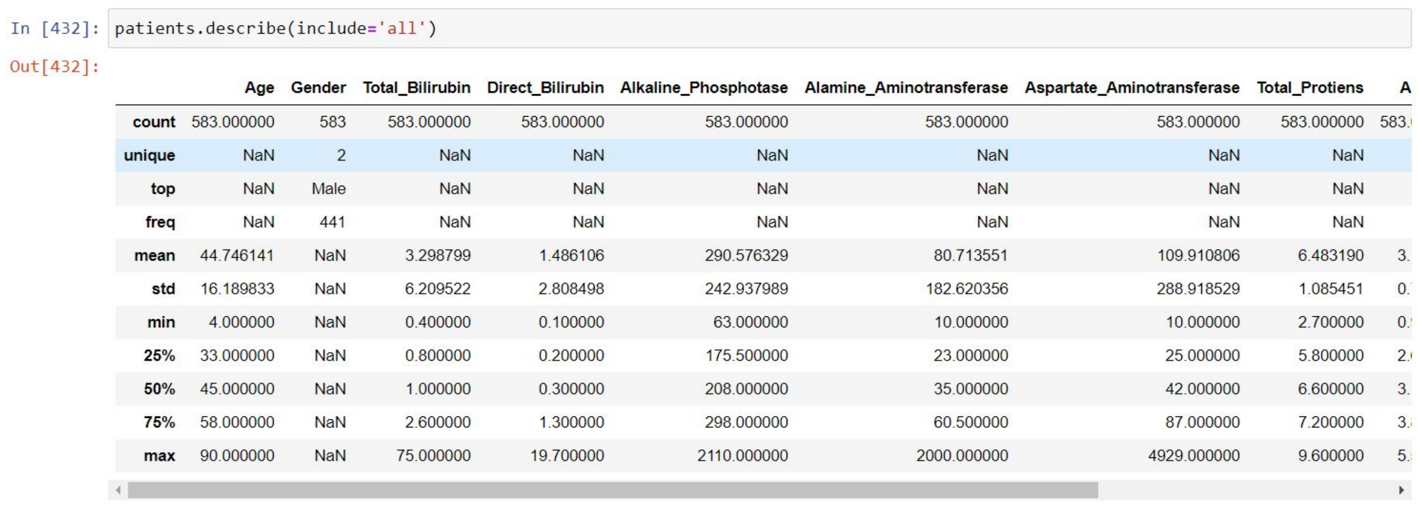
● Categorical data can be handled by using dummy variables, which are also called as indicator variables.

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

There is only one categorical column in this dataset which is the GENDER COLUMN and it is converted into the numerical format in the later steps of implementing the model.

5.1.5 STATISTICAL ANALYSIS :

- **The describe() function computes a summary of statistics pertaining to the DataFrame columns.**

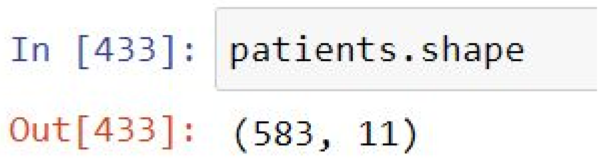


```
In [432]: patients.describe(include='all')
Out[432]:
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	A
count	583.000000	583	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.
unique	NaN	2	NaN	NaN	NaN	NaN	NaN	NaN	
top	NaN	Male	NaN	NaN	NaN	NaN	NaN	NaN	
freq	NaN	441	NaN	NaN	NaN	NaN	NaN	NaN	
mean	44.746141	NaN	3.298799	1.486106	290.576329	80.713551	109.910806	6.483190	3.
std	16.189833	NaN	6.209522	2.808498	242.937989	182.620356	288.918529	1.085451	0.
min	4.000000	NaN	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000	0.
25%	33.000000	NaN	0.800000	0.200000	175.500000	23.000000	25.000000	5.800000	2.
50%	45.000000	NaN	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000	3.
75%	58.000000	NaN	2.600000	1.300000	298.000000	60.500000	87.000000	7.200000	3.
max	90.000000	NaN	75.000000	19.700000	2110.000000	2000.000000	4929.000000	9.600000	5.

FIGURE 14 : DESCRIBE FUNCTION

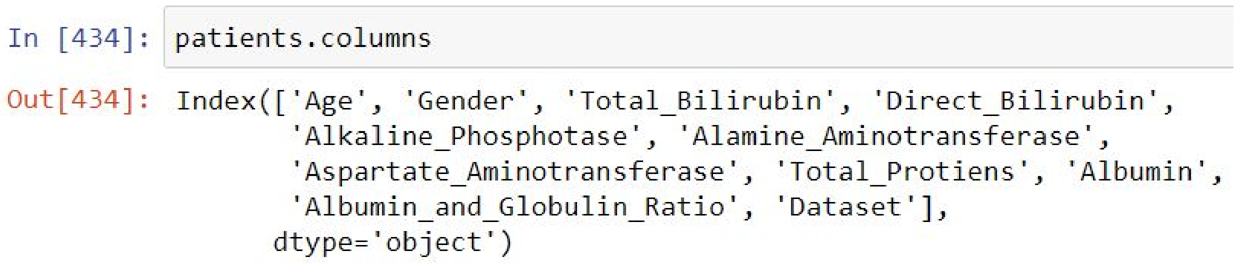
- **So there are 583 rows and 11 columns in our dataset.**



```
In [433]: patients.shape
Out[433]: (583, 11)
```

FIGURE 15 : SHAPE OF THE DATASET

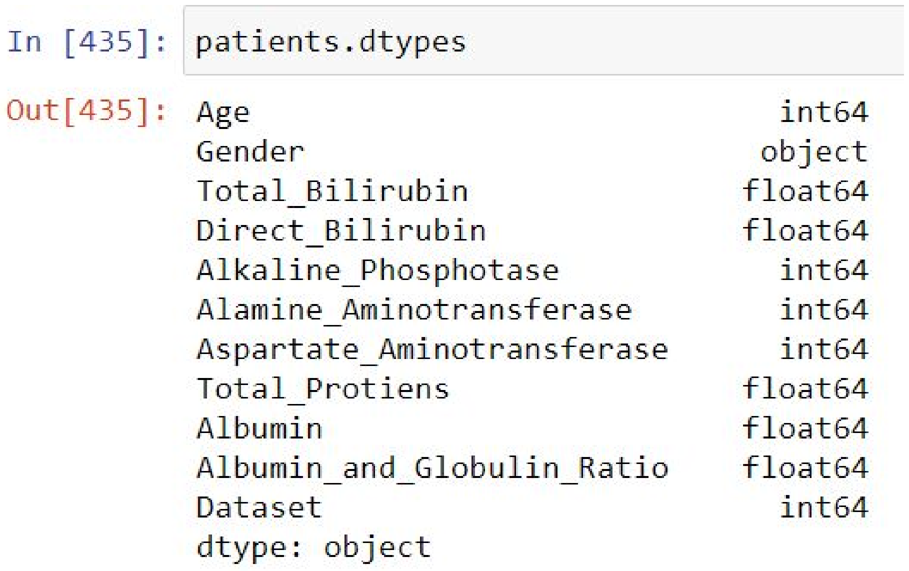
- **COLUMNS PRESENT IN THE DATASET :**



```
In [434]: patients.columns

Out[434]: Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
                 'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
                 'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
                 'Albumin_and_Globulin_Ratio', 'Dataset'],
                dtype='object')
```

FIGURE 16 : DISPLAY OF THE COLUMNS

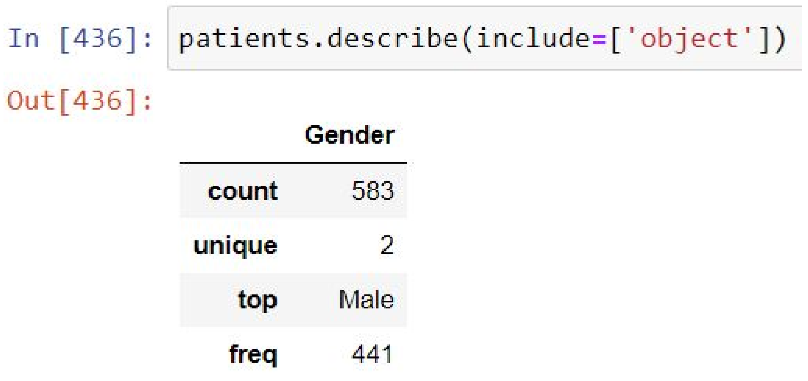
- **DATATYPES OF THE COLUMNS :**



```
In [435]: patients.dtypes

Out[435]: Age                             int64
          Gender                         object
          Total_Bilirubin               float64
          Direct_Bilirubin              float64
          Alkaline_Phosphotase            int64
          Alamine_Aminotransferase        int64
          Aspartate_Aminotransferase      int64
          Total_Protiens                float64
          Albumin                       float64
          Albumin_and_Globulin_Ratio    float64
          Dataset                         int64
          dtype: object
```

FIGURE 17 : DATATYPES OF THE COLUMNS

- **DESCRIBE FUNCTION USED ON THE OBJECT DATATYPE COLUMN (GENDER) :**



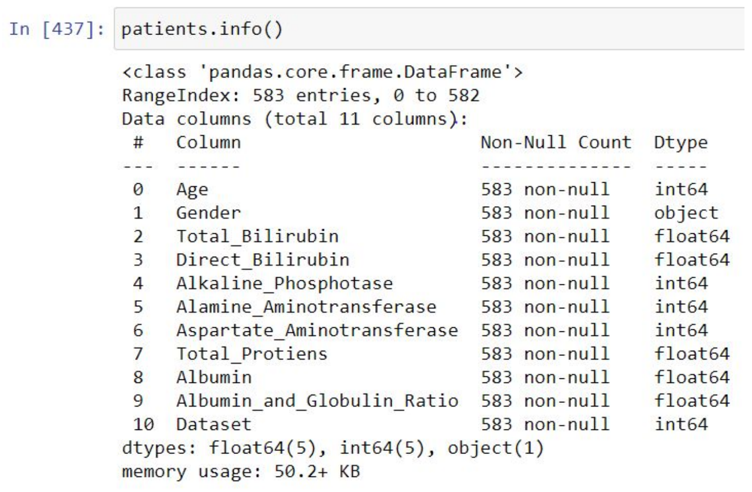
```
In [436]: patients.describe(include=['object'])
Out[436]:
```

	Gender
count	583
unique	2
top	Male
freq	441

FIGURE 18 : DESCRIBE FUNCTION ON OBJECT DATA

- **INFO FUNCTION :**

The info() function is used to print a concise summary of a DataFrame. This method prints information about a DataFrame including the index dtype and column dtypes, non-null values and memory usage.



```
In [437]: patients.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Age                         583 non-null    int64  
 1   Gender                      583 non-null    object 
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphotase        583 non-null    int64  
 5   Alamine_Aminotransferase    583 non-null    int64  
 6   Aspartate_Aminotransferase  583 non-null    int64  
 7   Total_Protiens              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  583 non-null    float64
 10  Dataset                     583 non-null    int64  
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

FIGURE 19 : INFO FUNCTION

Here are the observations from the dataset:

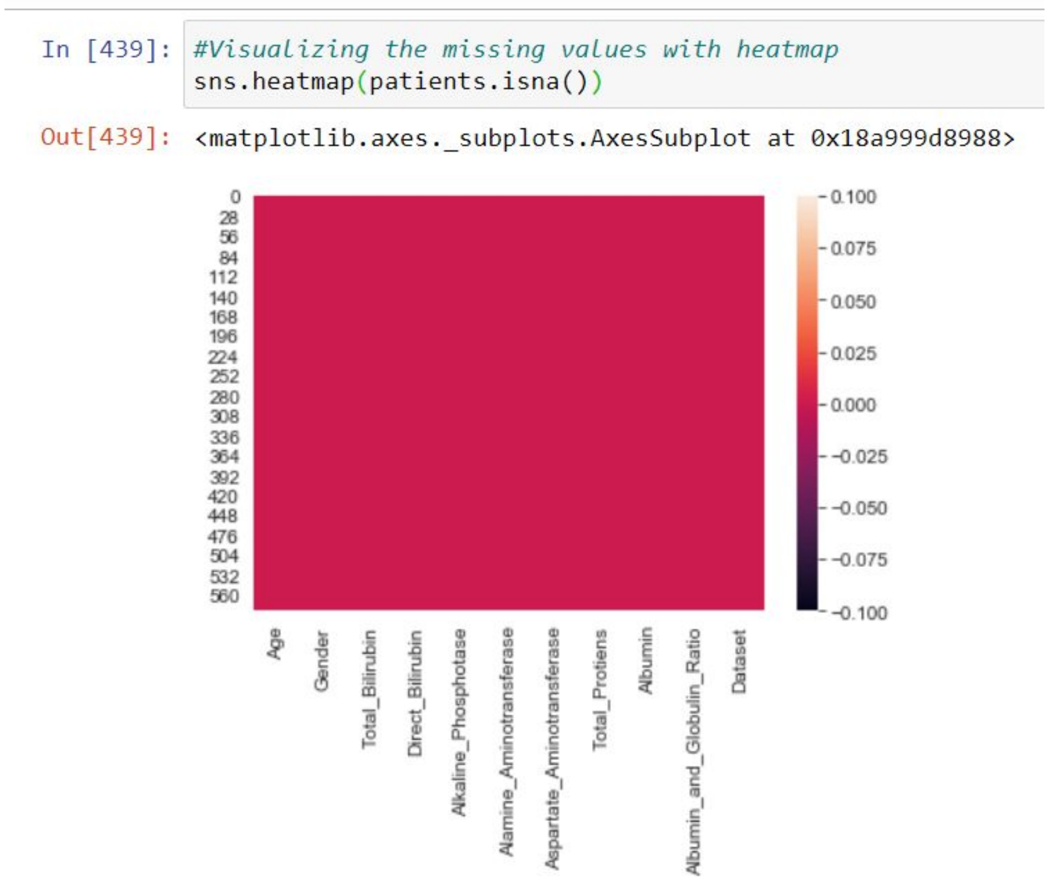
- Only gender is non-numeric veriable. All others are numeric.

- There are 10 features and 1 output - dataset column. Value 1 indicates that the patient has liver disease and 0 indicates the patient does not have liver disease.

CHAPTER 6

DATA VISUALIZATION

6.1 Visualizing the missing values with heatmap :

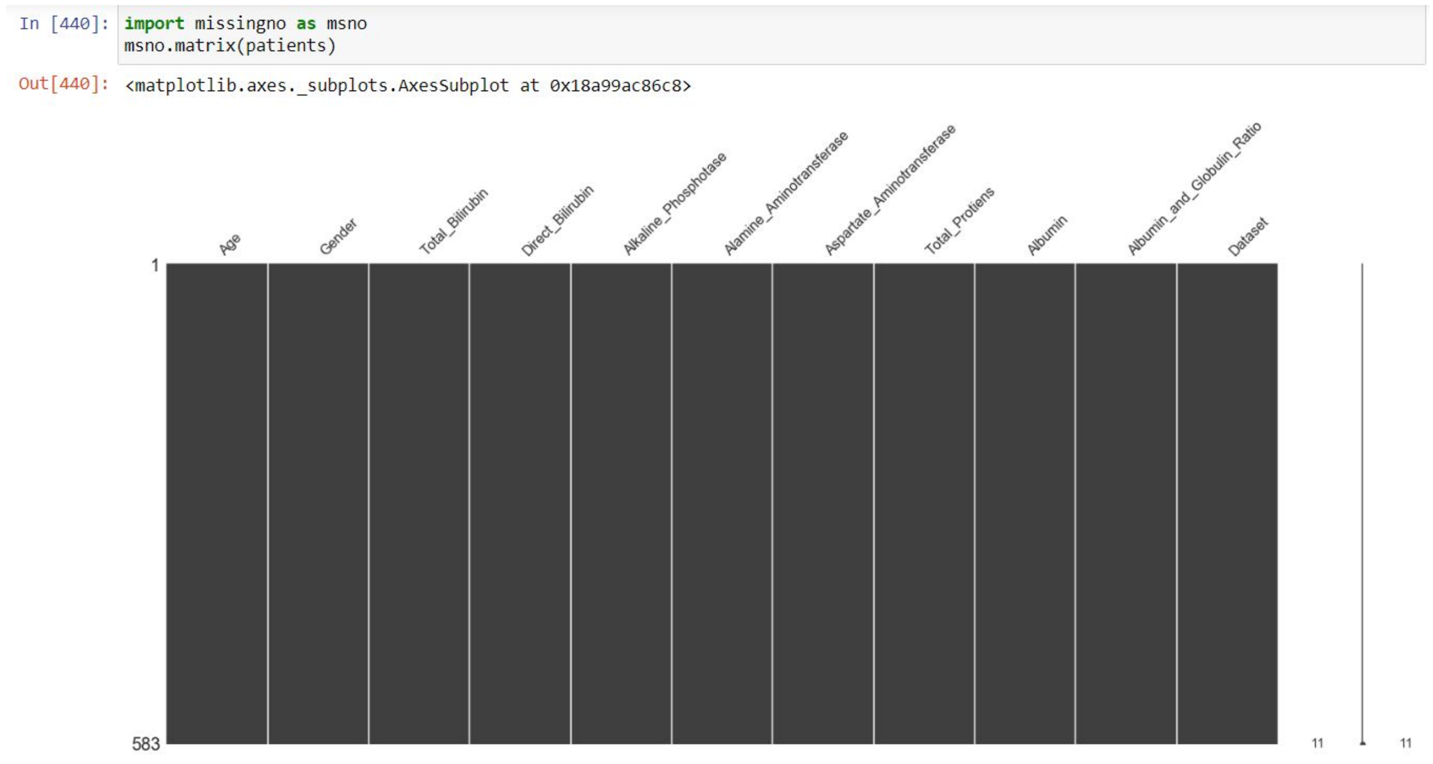


```
In [439]: #Visualizing the missing values with heatmap
          sns.heatmap(patients.isna())

Out[439]: <matplotlib.axes._subplots.AxesSubplot at 0x18a999d8988>
```

FIGURE 20

6.2 missingno :

It is specially used for checking(visualizing) the missing values.

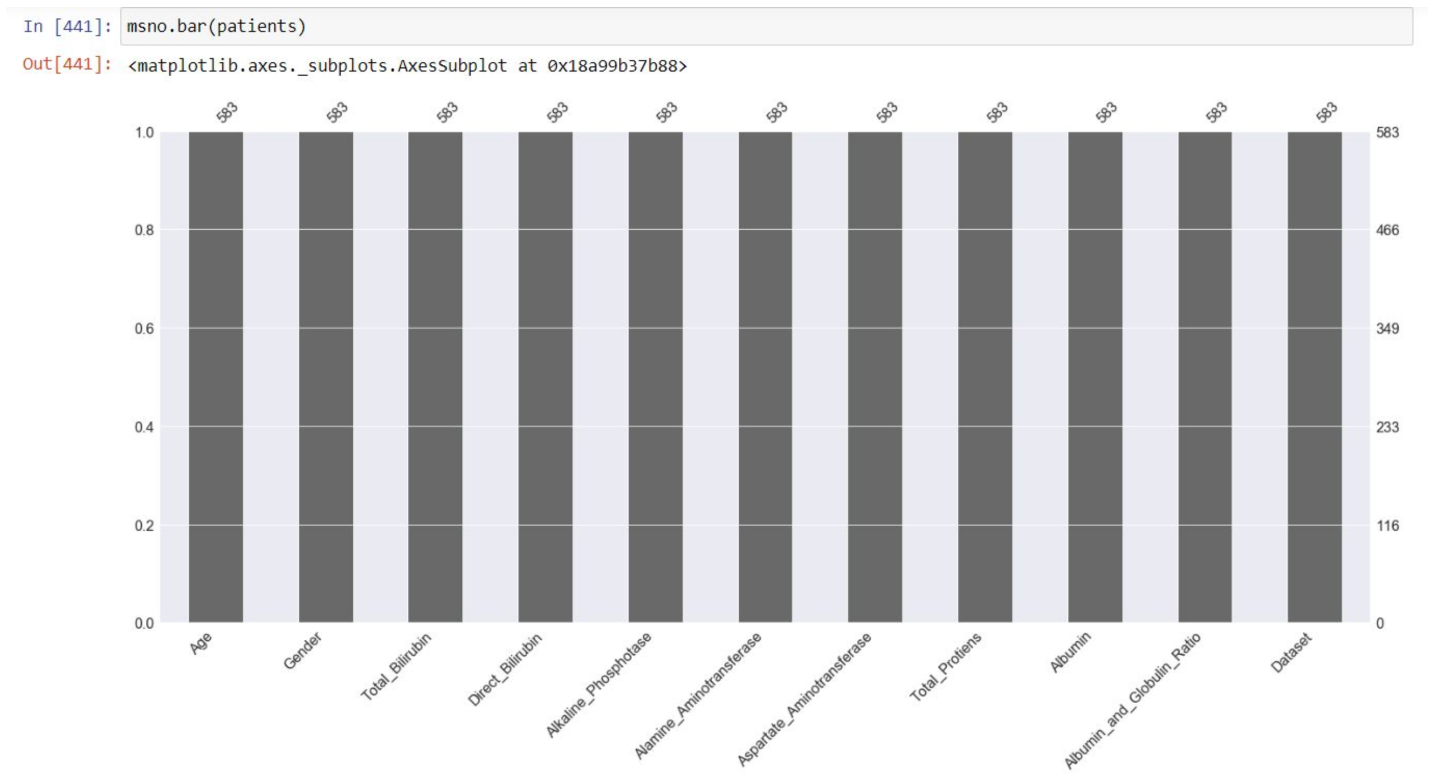


```
In [440]: import missingno as msno
          msno.matrix(patients)
Out[440]: <matplotlib.axes._subplots.AxesSubplot at 0x18a99ac86c8>
```

FIGURE 21

OBSERVATIONS : The above graph shows no null values because there are no null values in the dataset.

6.2.1 To plot the above same missingno graph in the form of bar graph :



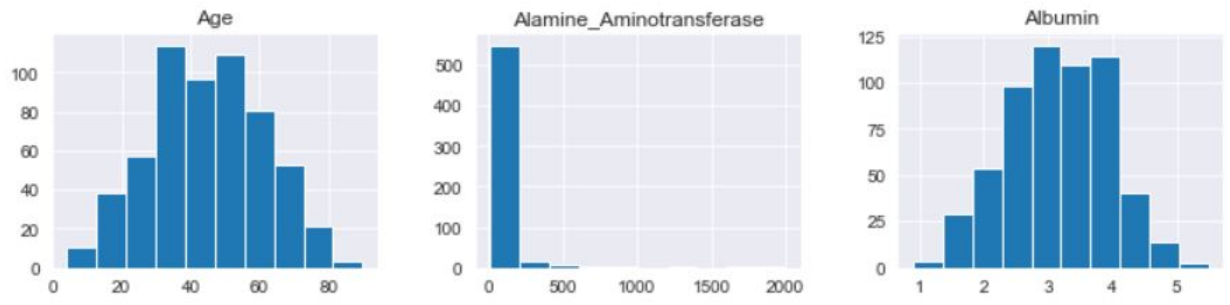
```
In [441]: msno.bar(patients)
Out[441]: <matplotlib.axes._subplots.AxesSubplot at 0x18a99b37b88>
```

FIGURE 22

6.3 To plot the histogram graphs for each and every column present in the dataset :

```
In [442]: patients.hist(figsize=(12,12))
```

```
Out[442]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000018A99B78EC8>,
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000018A9A4CA3C8>,
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000018A9A940C88>],
                 [<matplotlib.axes._subplots.AxesSubplot object at 0x0000018A9A978DC8>,
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000018A9A9B0F08>,
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000018A9A9ED048>],
                 [<matplotlib.axes._subplots.AxesSubplot object at 0x0000018A9AA26108>,
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000018A9AA5F208>,
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000018A9AA65DC8>],
                 [<matplotlib.axes._subplots.AxesSubplot object at 0x0000018A9AA9DF88>,
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000018A9AB07548>,
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000018A9AB44E88>]],
                dtype=object)
```



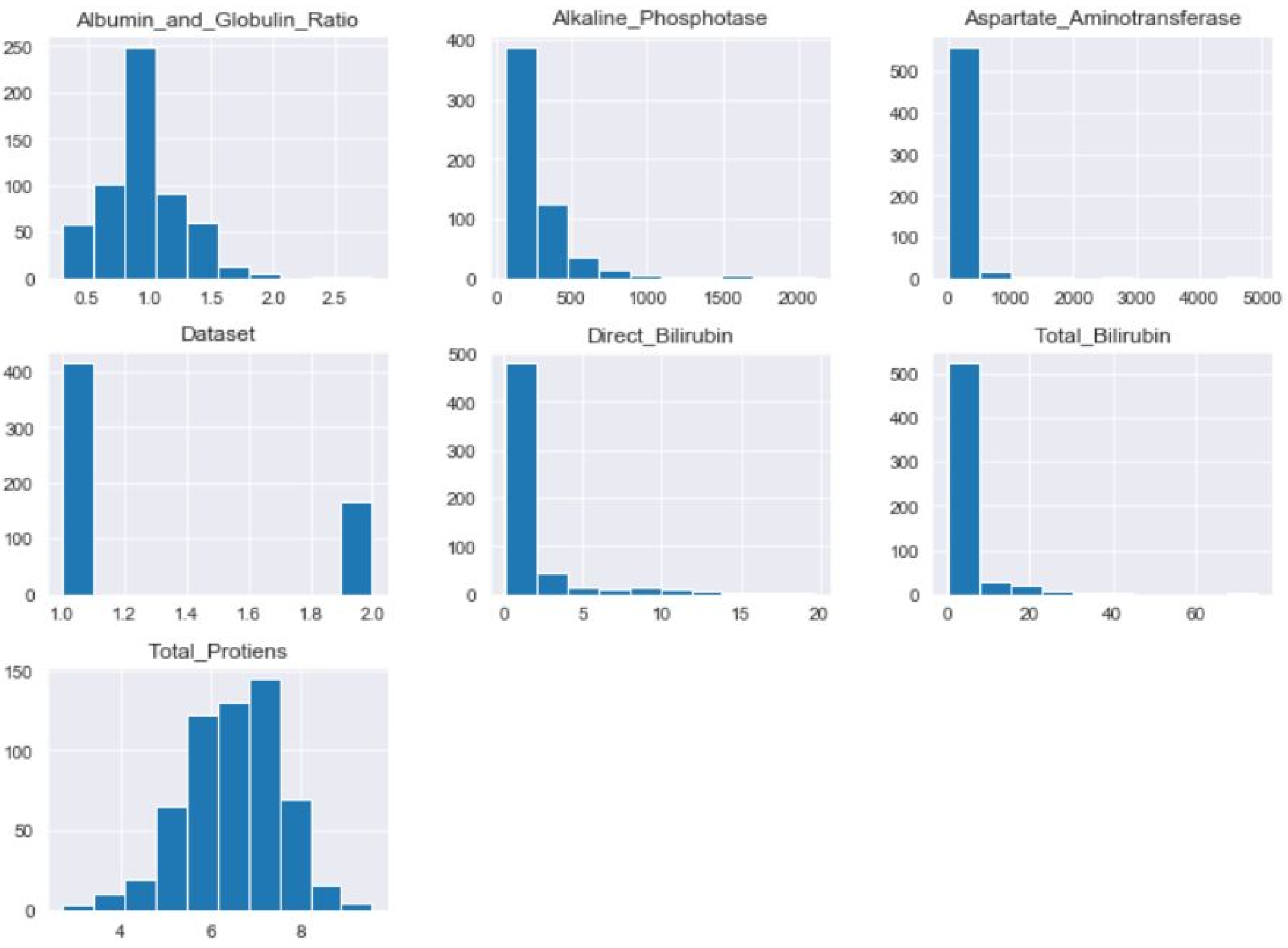

Albumin_and_Globulin_Ratio
Alkaline_Phosphotase
Aspartate_Aminotransferase
Dataset
Direct_Bilirubin
Total_Bilirubin
Total_Protiens

FIGURE 23

6.4 To check the countplot of our Dataset (output) column :

```
In [443]: sns.countplot(data=patients, x = 'Dataset', label='Count')

          LD, NLD = patients['Dataset'].value_counts()
          print('Number of patients diagnosed with liver disease: ',LD)
          print('Number of patients not diagnosed with liver disease: ',NLD)
```

```
Number of patients diagnosed with liver disease:  416
Number of patients not diagnosed with liver disease:  167
```

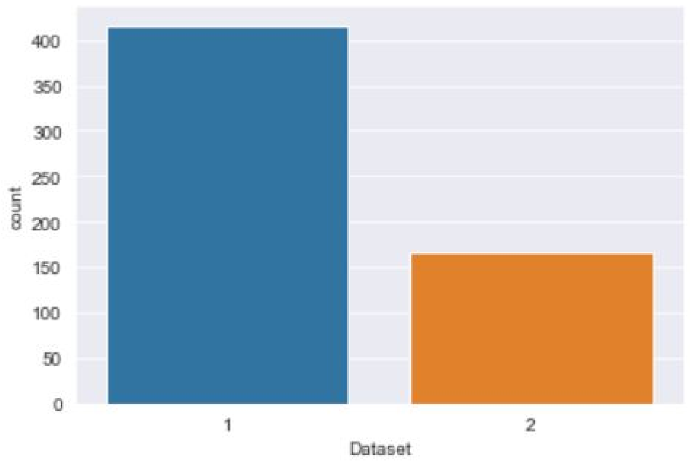


FIGURE 24

OBSERVATIONS : From the above graph we can see that there are 416 patients diagnosed with liver disease and 167 patients not diagnosed with liver disease.

6.4.1 To check the countplot of gender column :

```
In [444]: sns.countplot(data=patients, x = 'Gender', label='Count')

          M, F = patients['Gender'].value_counts()
          print('Number of patients that are male: ',M)
          print('Number of patients that are female: ',F)
```

```
Number of patients that are male:  441
Number of patients that are female:  142
```

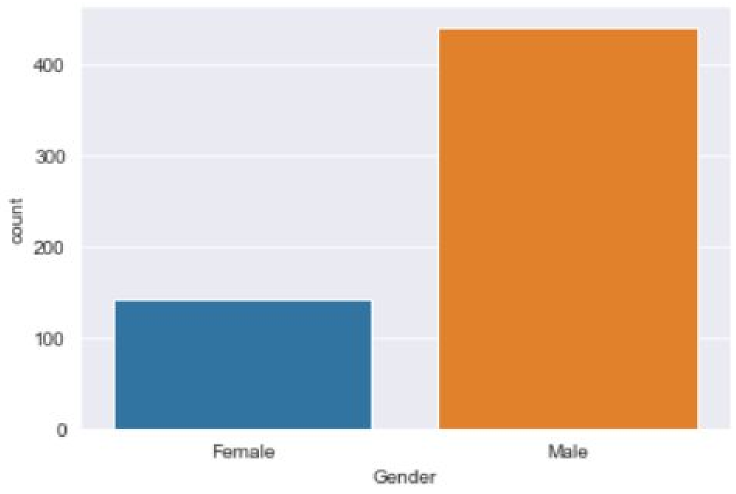


FIGURE 25

Observation :

- From the above graph, we can see that Number of males are more than the Number of females.

6.5 To check the age group of patients :

```
In [445]: sns.set_style('darkgrid')
          plt.figure(figsize=(25,10))
          patients['Age'].value_counts().plot.bar(color='blue')

Out[445]: <matplotlib.axes._subplots.AxesSubplot at 0x18a9ad24648>
```

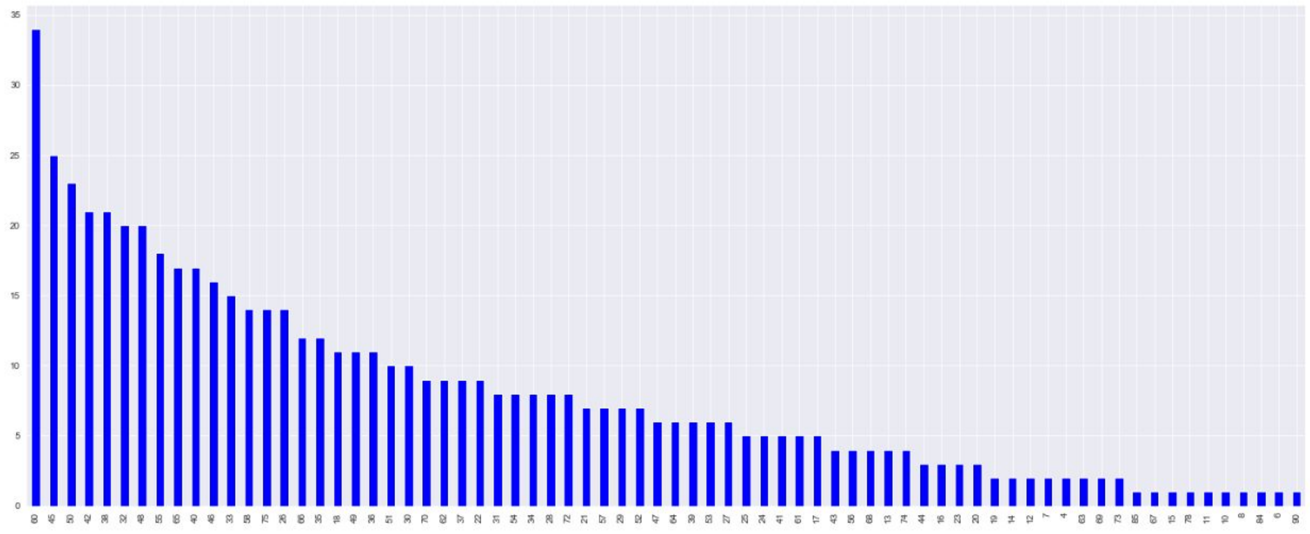


FIGURE 26

6.6 To view the pairplot of data based on Gender :

```
In [446]: import seaborn as sns
          sns.pairplot(patients,hue='Gender')
```

```
Out[446]: <seaborn.axisgrid.PairGrid at 0x18a9ad16d08>
```

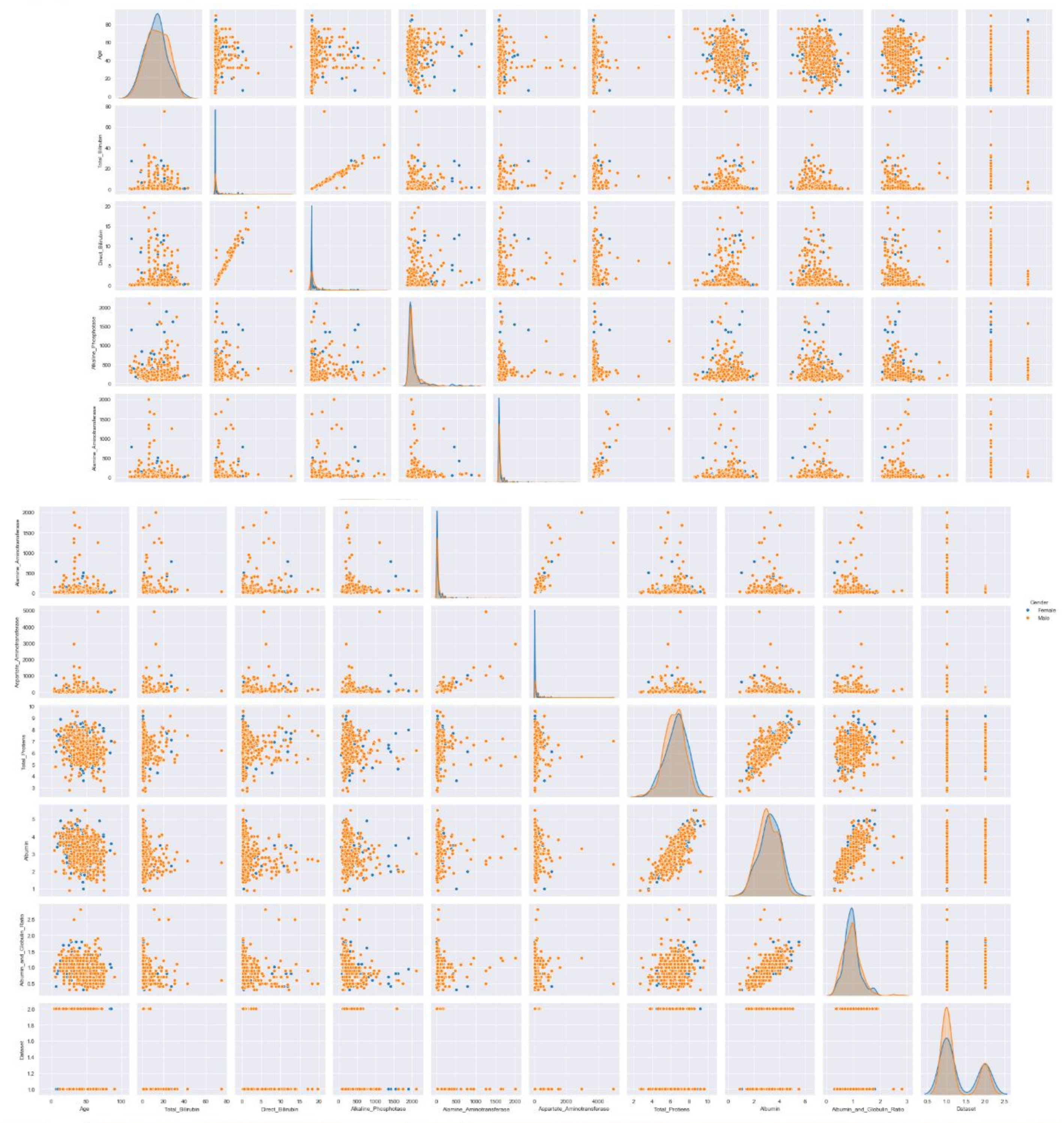


FIGURE 27

6.7 To view pairplot for the data :

```
In [447]: sns.pairplot(patients)
```

```
Out[447]: <seaborn.axisgrid.PairGrid at 0x18aa0449848>
```

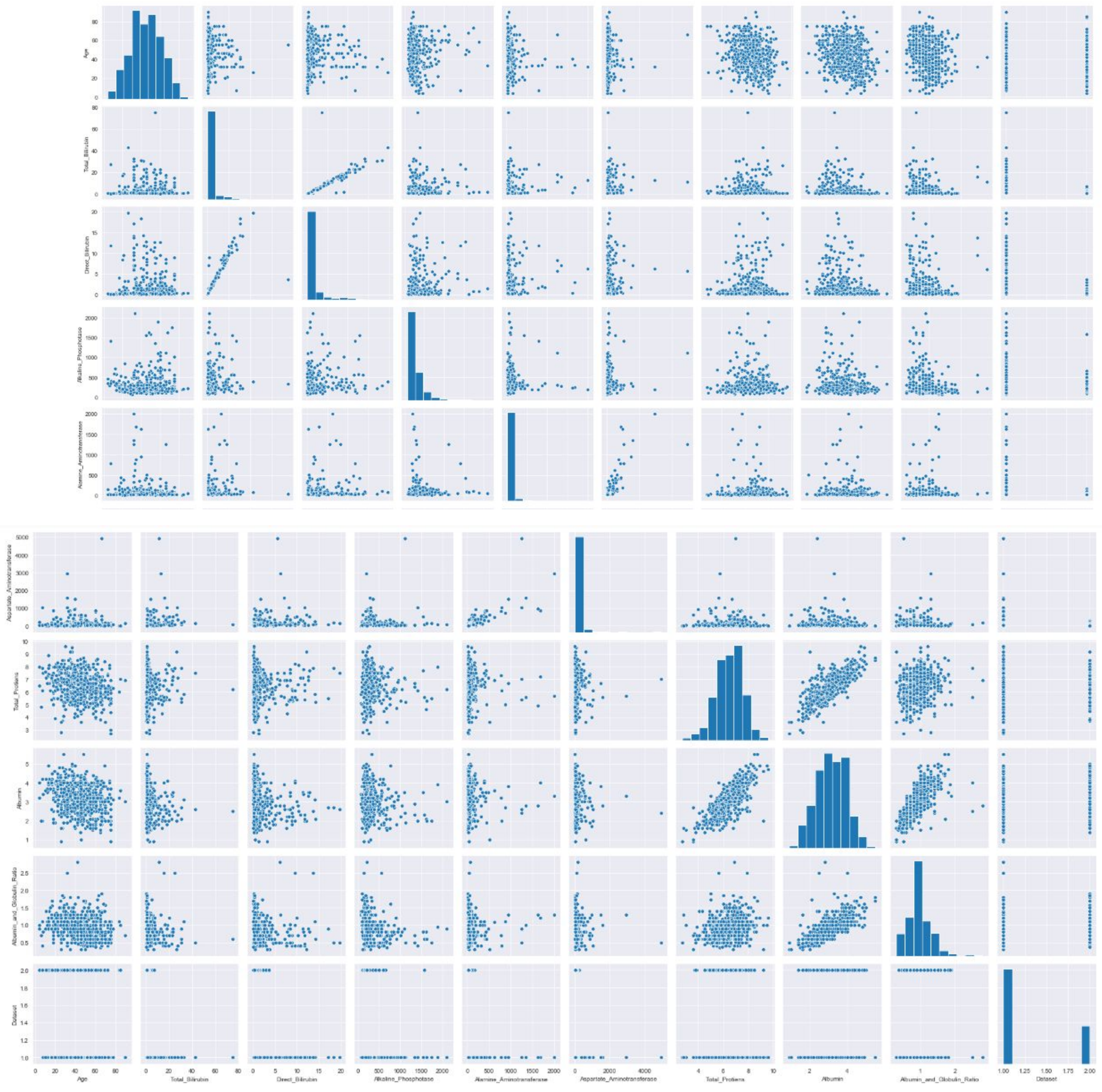


FIGURE 28

6.8 To compare the Gender based on the Protein Intake :

```
In [448]: plt.figure(figsize=(8,6))
          patients.groupby('Gender').sum()["Total_Protiens"].plot.bar(color='blue')

Out[448]: <matplotlib.axes._subplots.AxesSubplot at 0x18aa35d2888>
```

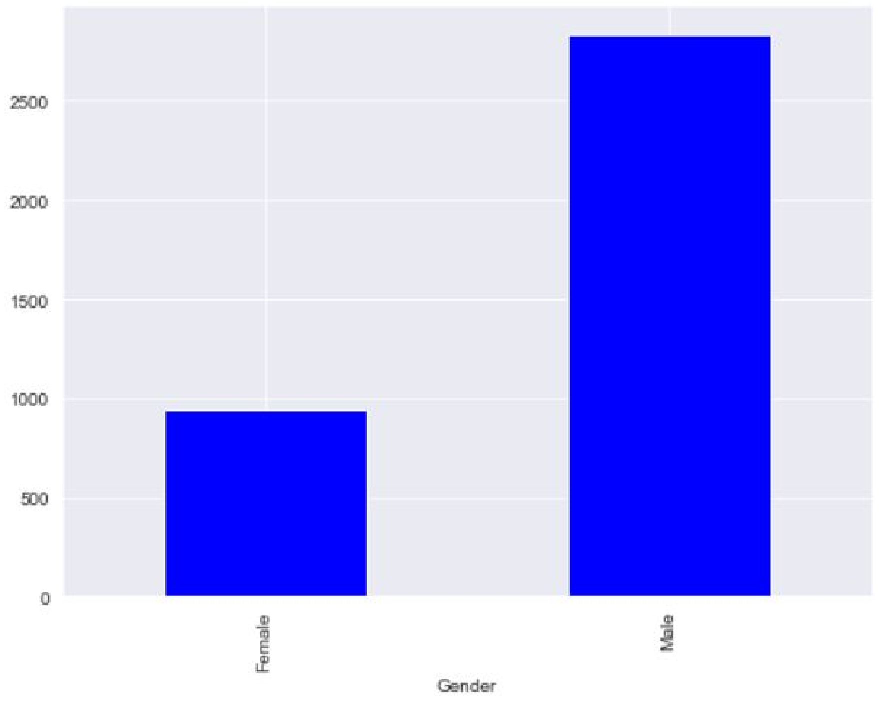


FIGURE 29

Observations :

- **Protein intake is higher in the case of male and comparitively less in females.**

6.8.1 To compare the Gender column based on Albumin Level :

```
In [449]: plt.figure(figsize=(8,6))
          patients.groupby('Gender').sum()['Albumin'].plot.bar(color='coral')

Out[449]: <matplotlib.axes._subplots.AxesSubplot at 0x18aa5295448>
```

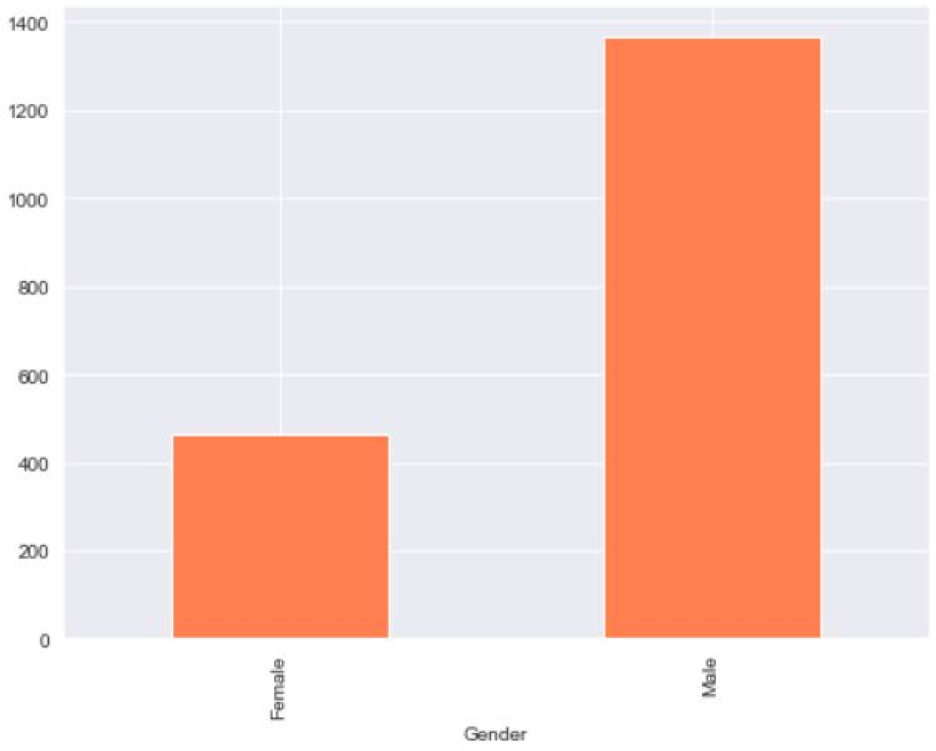


FIGURE 30

Observation :

- Albumin Level is higher in the case in the case of male compared to female.

6.8.2 To compare the Gender column based on the Bilirubin content :

```
In [450]: plt.figure(figsize=(8,6))
          patients.groupby('Gender').sum()['Total_Bilirubin'].plot.bar(color='pink')

Out[450]: <matplotlib.axes._subplots.AxesSubplot at 0x18aa5fea6c8>
```

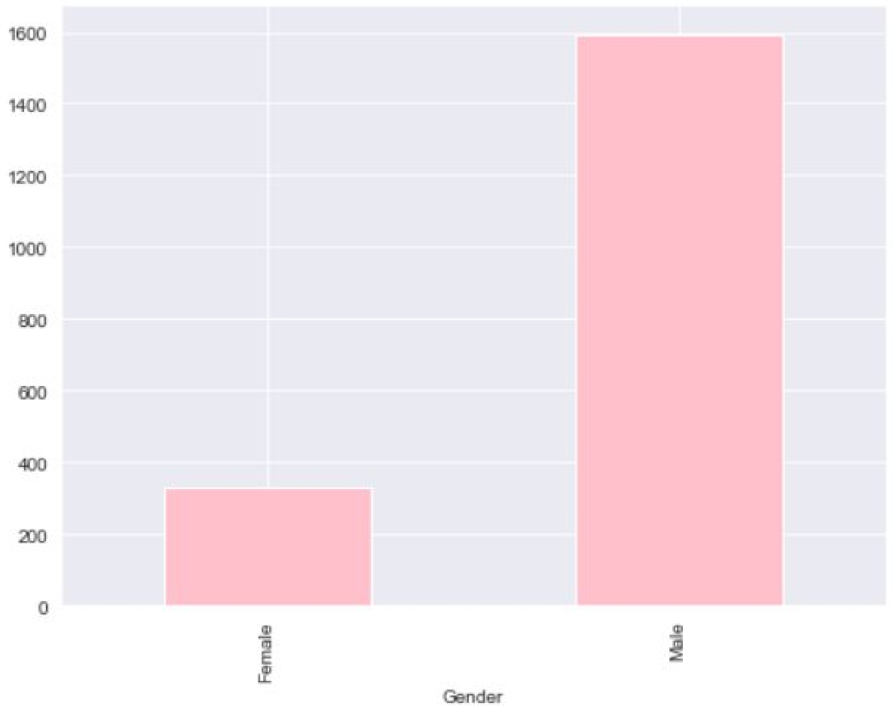


FIGURE 31

Observations :

- It is clearly seen that males have more bilirubin content compared to females.
- Higher the Bilirubin content, higher the case is prone to Liver disease.

6.9 To compare the albumin and albumin and globulin ratio by a scatterplot :

```
In [451]: f, ax = plt.subplots(figsize=(8, 6))
          sns.scatterplot(x="Albumin", y="Albumin_and_Globulin_Ratio",color='green',data=patients);
          plt.show()
```

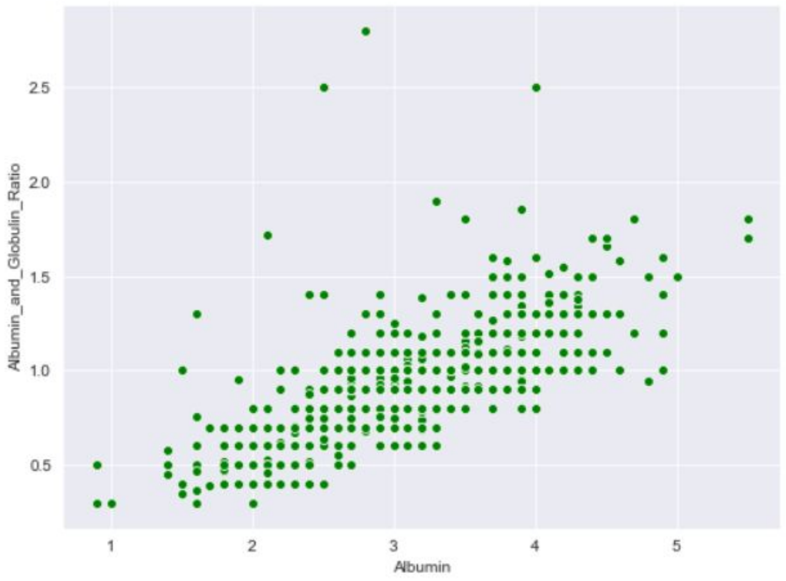


FIGURE 32

CHAPTER 7

CORRELATION

7.1 To check the correlation between the features using a heatmap :

```
In [452]: corr=patients.corr()
```

```
In [453]: plt.figure(figsize=(20,10))
          sns.heatmap(corr,cmap="icefire",annot=True)
```

```
Out[453]: <matplotlib.axes._subplots.AxesSubplot at 0x18aa5fea908>
```

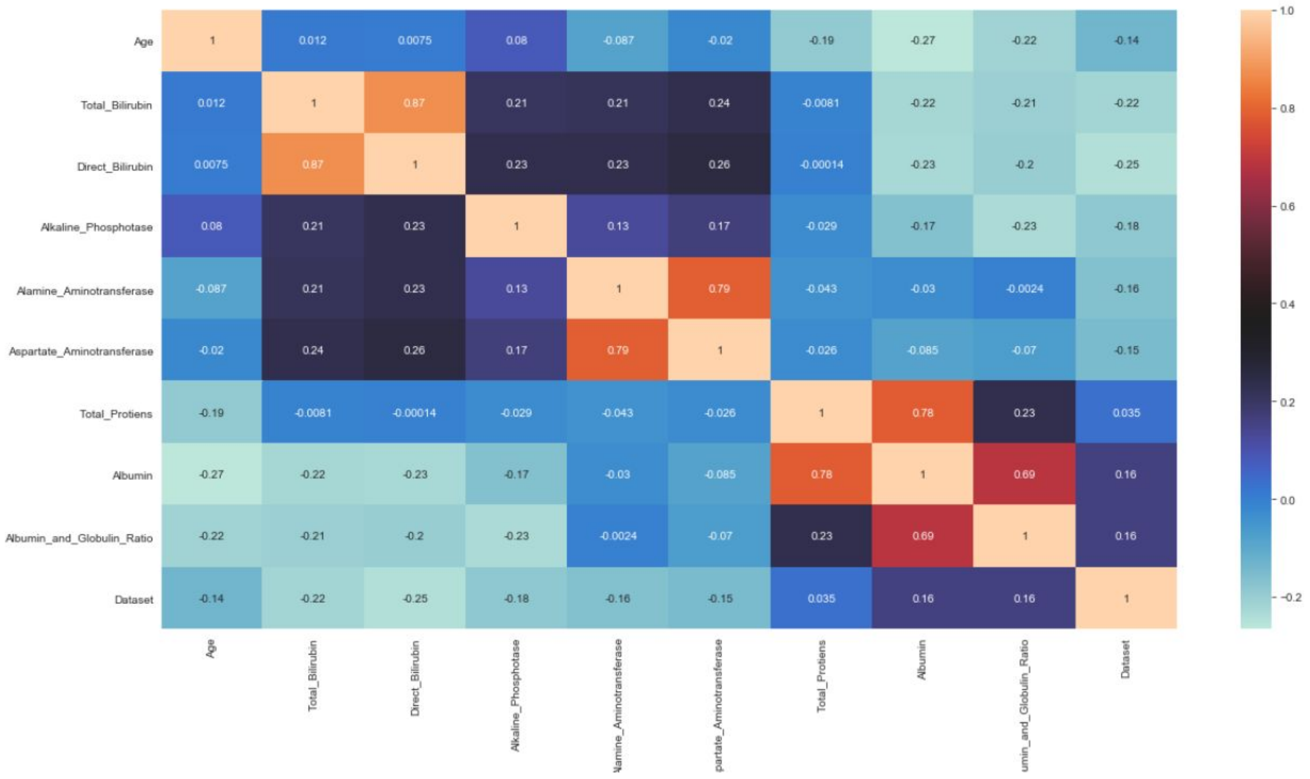


FIGURE 33

OBSERVATIONS :

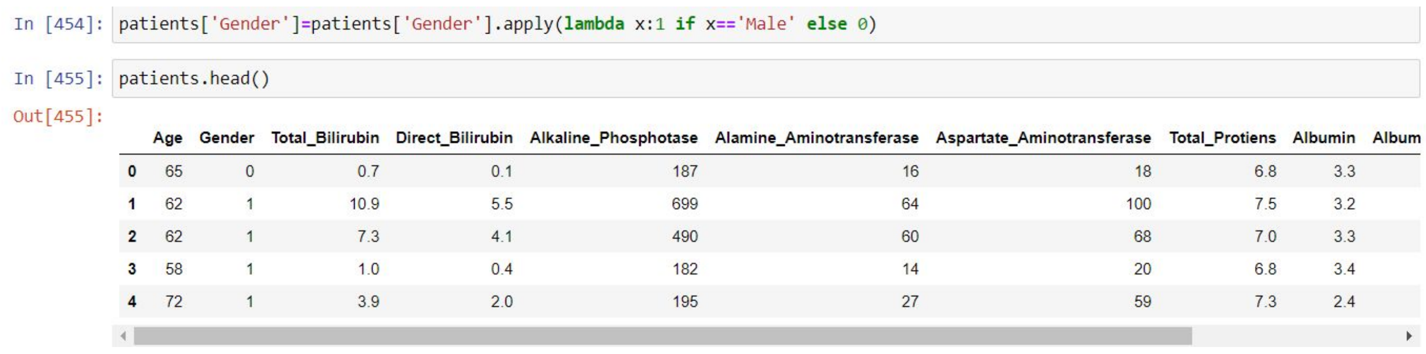
- The above correlation also indicates the following correlation :
- Total_Protiens & Albumin
- Alamine_Aminotransferase & Aspartate_Aminotransferase
- Direct_Bilirubin & Total_Bilirubin

- There is some correlation between Albumin_and_Globulin_Ratio and Albumin. But its not as high as Total_Protiens & Albumin

CHAPTER 8

FEATURE SELECTION,MODEL BUILDING AND EVALUATION

8.1 To convert the categorical column(Gender) into numerical format :



```
In [454]: patients['Gender']=patients['Gender'].apply(lambda x:1 if x=='Male' else 0)

In [455]: patients.head()
Out[455]:
```

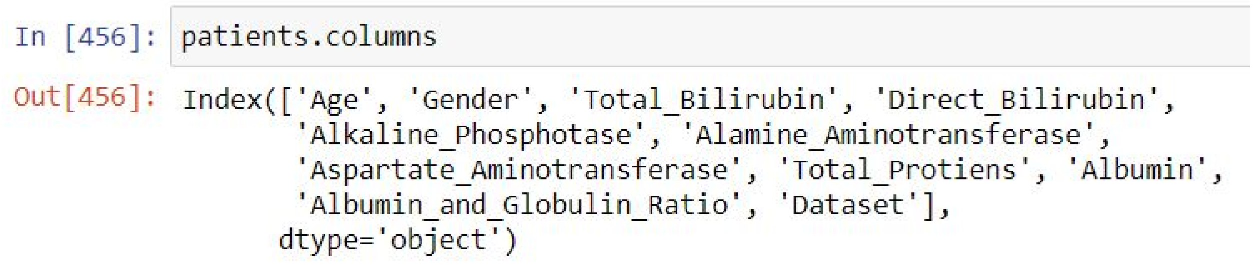
	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Album
0	65	0	0.7	0.1	187	16	18	6.8	3.3	
1	62	1	10.9	5.5	699	64	100	7.5	3.2	
2	62	1	7.3	4.1	490	60	68	7.0	3.3	
3	58	1	1.0	0.4	182	14	20	6.8	3.4	
4	72	1	3.9	2.0	195	27	59	7.3	2.4	

FIGURE 34

The Gender column is now converted into numerical format as shown in the above figure.

8.2 Building the model :

- **Inorder to build a successful model we have to train and test the model.**

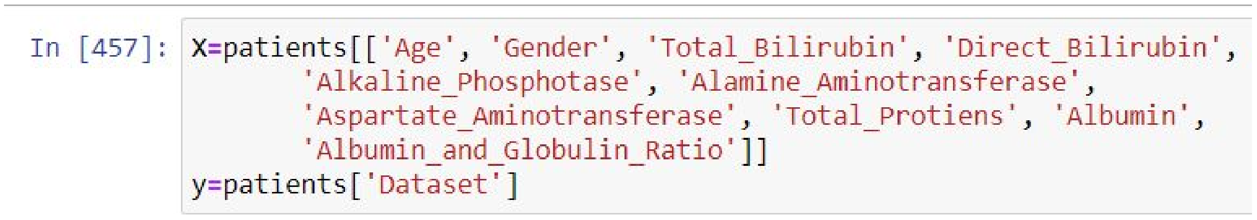


```
In [456]: patients.columns

Out[456]: Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
                 'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
                 'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
                 'Albumin_and_Globulin_Ratio', 'Dataset'],
                dtype='object')
```

FIGURE 35

8.3 - Define X and y.

- **X is our features and y is our target.**

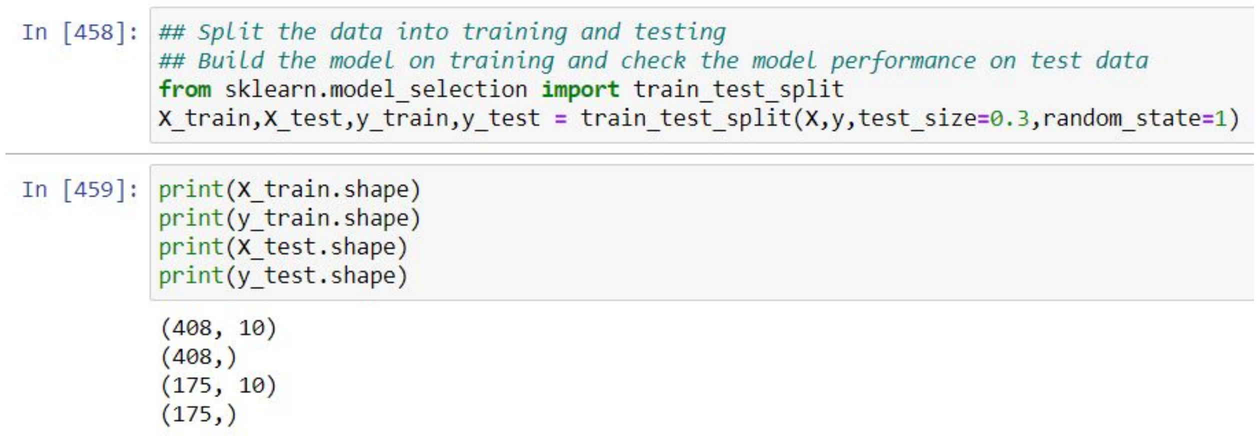


```
In [457]: X=patients[['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
                 'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
                 'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
                 'Albumin_and_Globulin_Ratio']]
          y=patients['Dataset']
```

FIGURE 36

8.4 - Split the data into training and testing

- Build the model on training and check the model performance on test data.



```
In [458]: ## Split the data into training and testing
          ## Build the model on training and check the model performance on test data
          from sklearn.model_selection import train_test_split
          X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=1)
```

```
In [459]: print(X_train.shape)
          print(y_train.shape)
          print(X_test.shape)
          print(y_test.shape)

          (408, 10)
          (408,)
          (175, 10)
          (175,)
```

Figure 37

CHAPTER 9

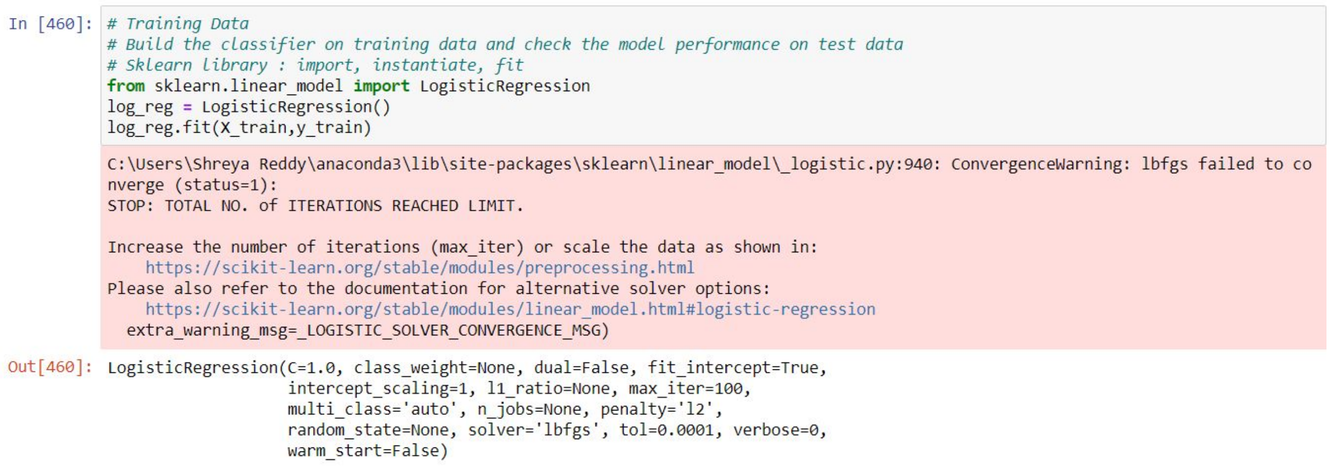
LOGISTIC REGRESSION

9.1 LOGISTIC REGRESSION :

Logistic regression is a machine learning algorithm used for classification problems. The term logistic is derived from the cost function (logistic function) which is a type of sigmoid function known for its characteristic S-shaped curve. A logistic regression model predicts probability values which are mapped to two (binary classification) or more (multiclass classification) classes.

9.2 - Training Data

- **Build the classifier on training data and check the model performance on test data**
- **Sklearn library : import, instantiate, fit**

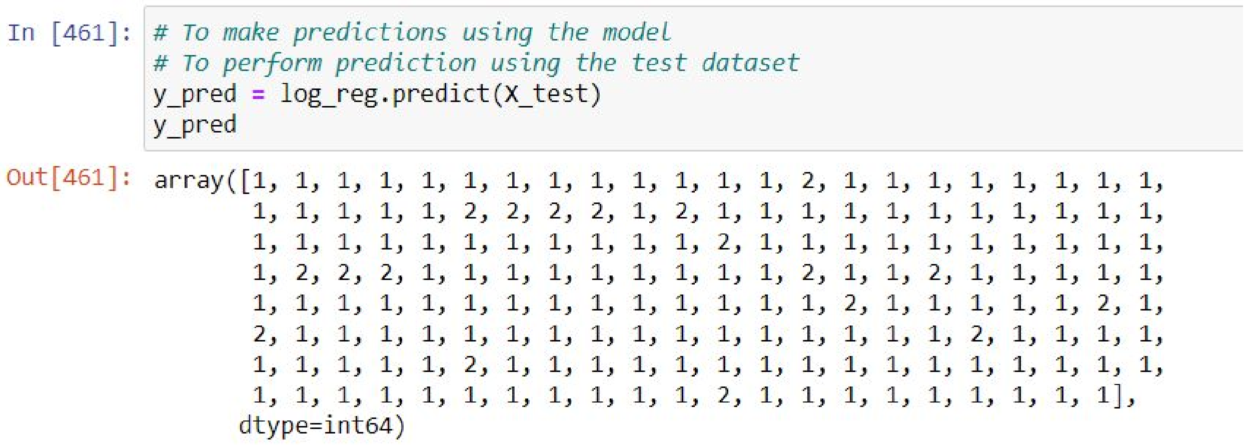


```
In [460]: # Training Data
          # Build the classifier on training data and check the model performance on test data
          # Sklearn library : import, instantiate, fit
          from sklearn.linear_model import LogisticRegression
          log_reg = LogisticRegression()
          log_reg.fit(X_train,y_train)

          C:\Users\Shreya Reddy\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning: lbfgs failed to co
          nverge (status=1):
          STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

          Increase the number of iterations (max_iter) or scale the data as shown in:
              https://scikit-learn.org/stable/modules/preprocessing.html
          Please also refer to the documentation for alternative solver options:
              https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
            extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

Out[460]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                             intercept_scaling=1, l1_ratio=None, max_iter=100,
                             multi_class='auto', n_jobs=None, penalty='l2',
                             random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                             warm_start=False)
```

FIGURE 38

9.3 - To make predictions using the model

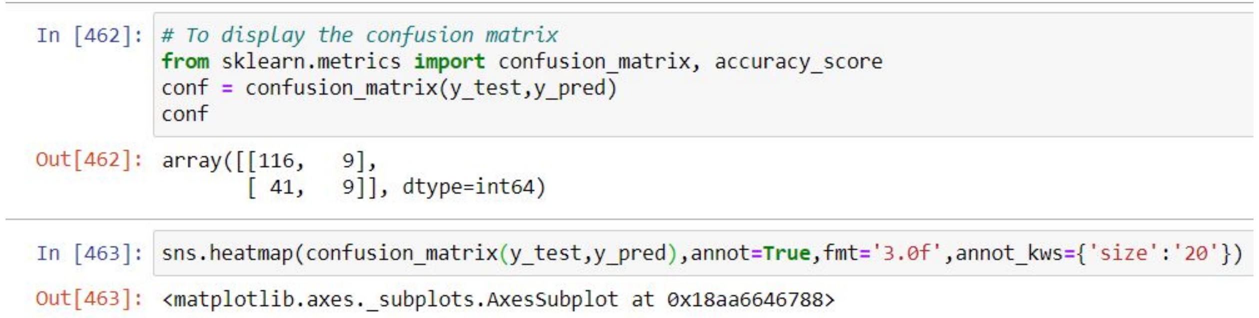
- **To perform prediction using the test dataset**



```
In [461]: # To make predictions using the model
          # To perform prediction using the test dataset
          y_pred = log_reg.predict(X_test)
          y_pred

Out[461]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1,
                 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                dtype=int64)
```

FIGURE 39

9.4 - To display the confusion matrix :



```
In [462]: # To display the confusion matrix
          from sklearn.metrics import confusion_matrix, accuracy_score
          conf = confusion_matrix(y_test,y_pred)
          conf

Out[462]: array([[116,   9],
                 [ 41,   9]], dtype=int64)

In [463]: sns.heatmap(confusion_matrix(y_test,y_pred),annot=True,fmt='3.0f',annot_kws={'size':'20'})

Out[463]: <matplotlib.axes._subplots.AxesSubplot at 0x18aa6646788>
```

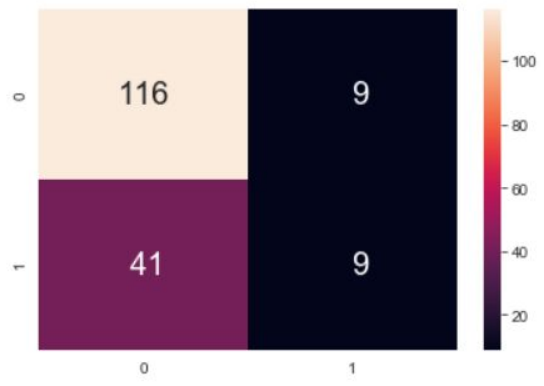
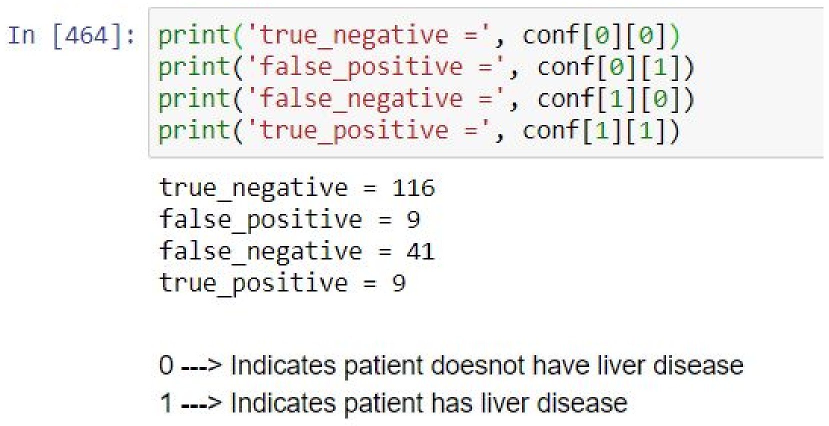


FIGURE 40

9.5 - To print the true negative,false positive,false negative and true positive values from the confusion matrix seen above.



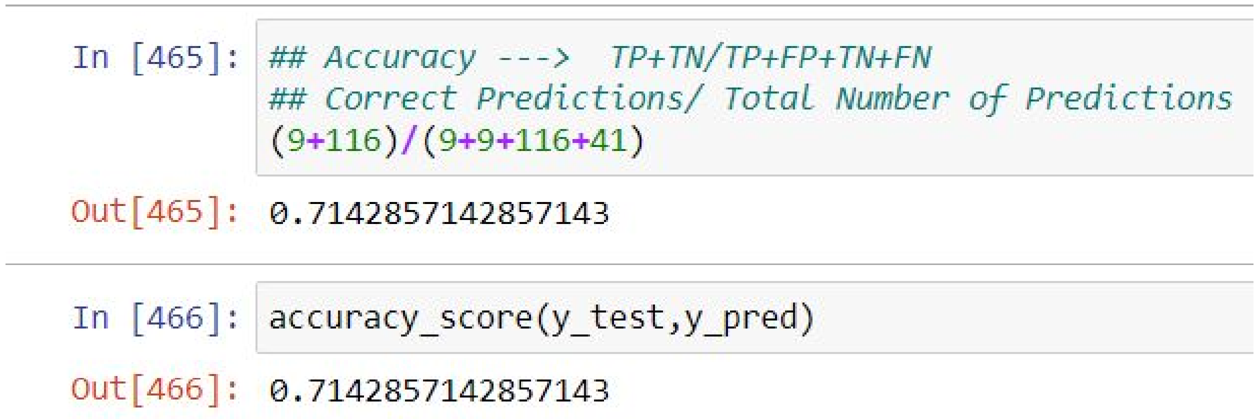
```
In [464]: print('true_negative =', conf[0][0])
          print('false_positive =', conf[0][1])
          print('false_negative =', conf[1][0])
          print('true_positive =', conf[1][1])

          true_negative = 116
          false_positive = 9
          false_negative = 41
          true_positive = 9
```

0 ---> Indicates patient doesnot have liver disease
1 ---> Indicates patient has liver disease

FIGURE 41

9.6 - Accuracy ---> TP+TN/TP+FP+TN+FN

- **Correct Predictions/ Total Number of Predictions**



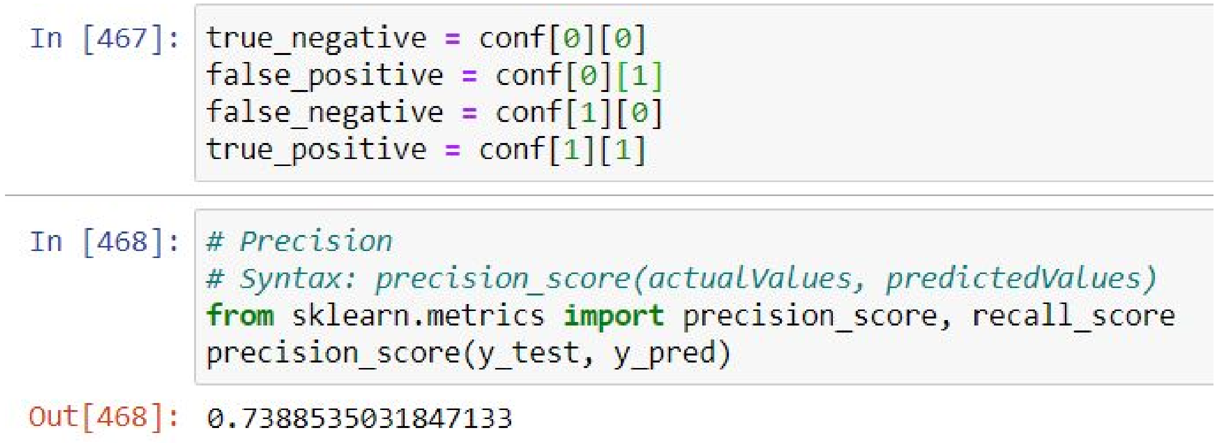
```
In [465]: ## Accuracy --->  TP+TN/TP+FP+TN+FN
          ## Correct Predictions/ Total Number of Predictions
          (9+116)/(9+9+116+41)

Out[465]: 0.7142857142857143

In [466]: accuracy_score(y_test,y_pred)

Out[466]: 0.7142857142857143
```

Figure 42

Accuracy is 71.42%

9.7 - Precision

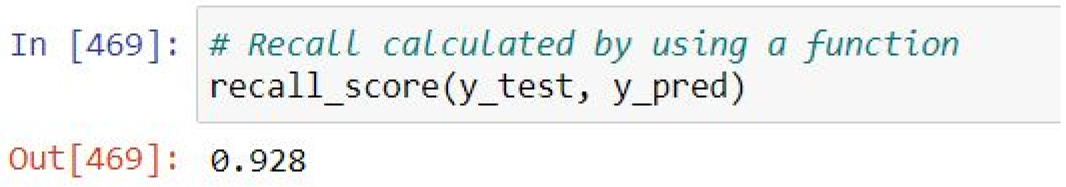
- **Syntax: precision_score(actualValues, predictedValues)**



```
In [467]: true_negative = conf[0][0]
          false_positive = conf[0][1]
          false_negative = conf[1][0]
          true_positive = conf[1][1]

In [468]: # Precision
          # Syntax: precision_score(actualValues, predictedValues)
          from sklearn.metrics import precision_score, recall_score
          precision_score(y_test, y_pred)

Out[468]: 0.7388535031847133
```

Figure 43

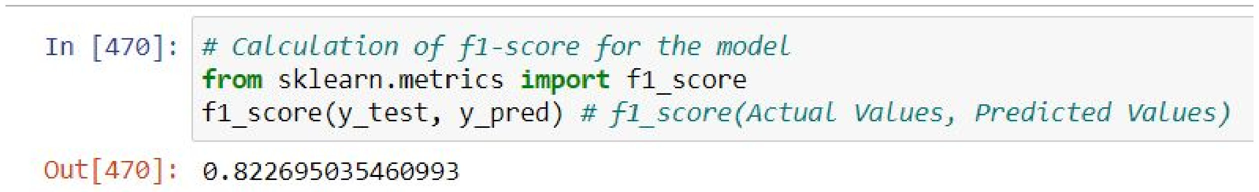
9.8 - Recall calculated by using a function :



```
In [469]: # Recall calculated by using a function
          recall_score(y_test, y_pred)

Out[469]: 0.928
```

Figure 44

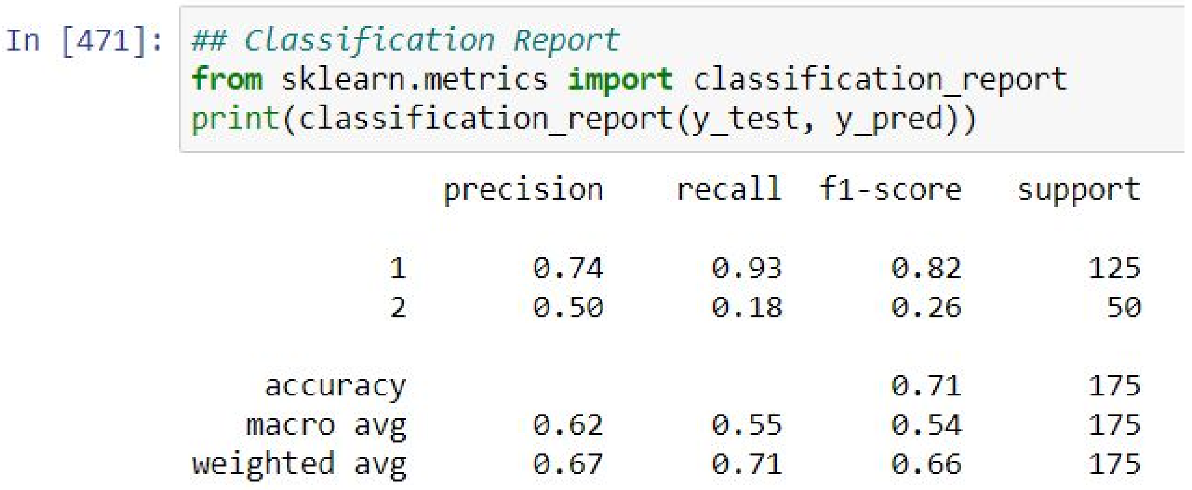
9.8.1 - Calculation of f1-score for the model :



```
In [470]: # Calculation of f1-score for the model
          from sklearn.metrics import f1_score
          f1_score(y_test, y_pred) # f1_score(Actual Values, Predicted Values)

Out[470]: 0.822695035460993
```

Figure 45

9.9 - Classification Report :



```
In [471]: ## Classification Report
          from sklearn.metrics import classification_report
          print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

           1       0.74      0.93      0.82       125
           2       0.50      0.18      0.26        50

    accuracy                           0.71       175
   macro avg       0.62      0.55      0.54       175
weighted avg       0.67      0.71      0.66       175
```

Figure 46

CHAPTER 10

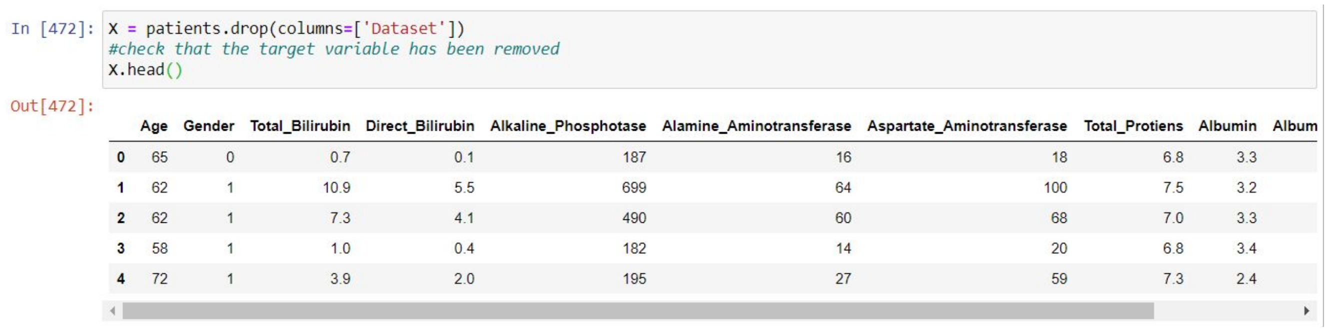
K-NEAREST NEIGHBORS(KNN)

K-Nearest Neighbors (KNN) Classifier :

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

- Lazy learning algorithm − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
- Non-parametric learning algorithm − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

10.1 - Check that the target variable has been removed :

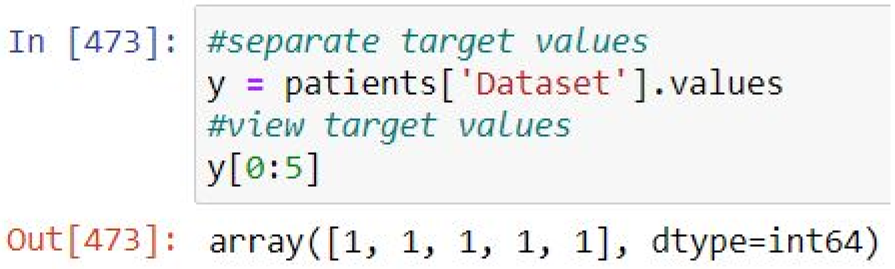


```
In [472]: X = patients.drop(columns=['Dataset'])
          #check that the target variable has been removed
          X.head()
Out[472]:
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Album
0	65	0	0.7	0.1	187	16	18	6.8	3.3	
1	62	1	10.9	5.5	699	64	100	7.5	3.2	
2	62	1	7.3	4.1	490	60	68	7.0	3.3	
3	58	1	1.0	0.4	182	14	20	6.8	3.4	
4	72	1	3.9	2.0	195	27	59	7.3	2.4	

Figure 47

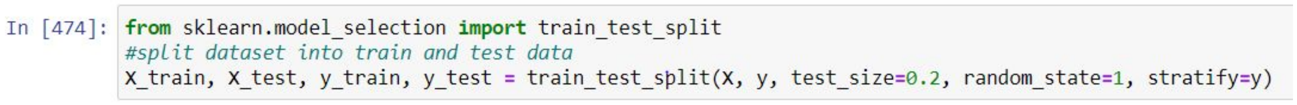
10.2 - Separate target values :



```
In [473]: #separate target values
          y = patients['Dataset'].values
          #view target values
          y[0:5]

Out[473]: array([1, 1, 1, 1, 1], dtype=int64)
```

Figure 48

10.3 - Splitting the data :

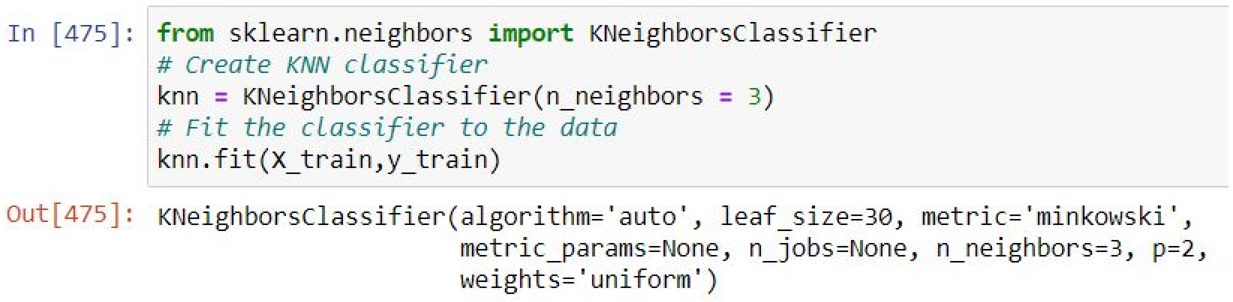


```
In [474]: from sklearn.model_selection import train_test_split
          #split dataset into train and test data
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1, stratify=y)
```

Figure 49

10.4 - Create KNN classifier :

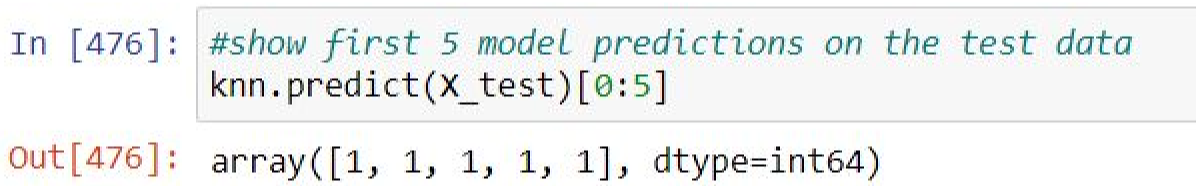
- Fit the classifier to the data :



```
In [475]: from sklearn.neighbors import KNeighborsClassifier
          # Create KNN classifier
          knn = KNeighborsClassifier(n_neighbors = 3)
          # Fit the classifier to the data
          knn.fit(X_train,y_train)

Out[475]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                               metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                               weights='uniform')
```

Figure 50

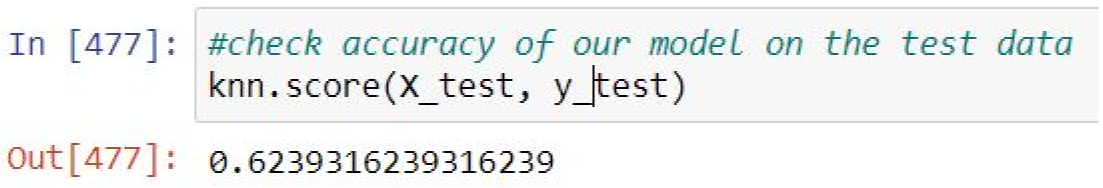
10.5 - To show first 5 model predictions on the test data :



```
In [476]: #show first 5 model predictions on the test data
          knn.predict(X_test)[0:5]

Out[476]: array([1, 1, 1, 1, 1], dtype=int64)
```

Figure 51

10.6 - To check accuracy of our model on the test data :

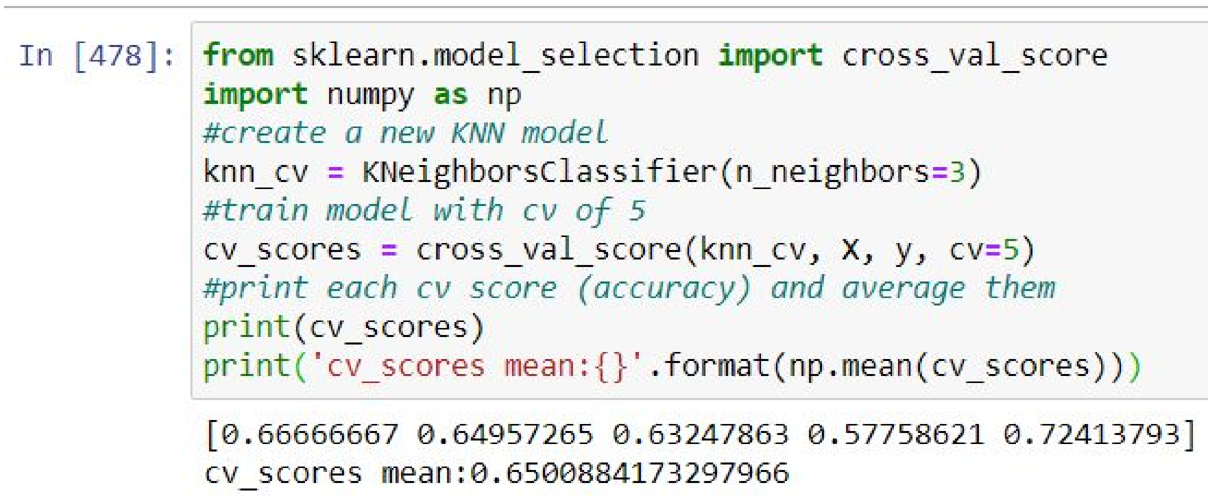


```
In [477]: #check accuracy of our model on the test data
          knn.score(X_test, y_test)

Out[477]: 0.6239316239316239
```

Figure 52

As the accuracy of our model is 62 % . To improve the accuracy we are applying the Grid CV search algorithm here in KNN.

10. 7 - Create a new KNN model :

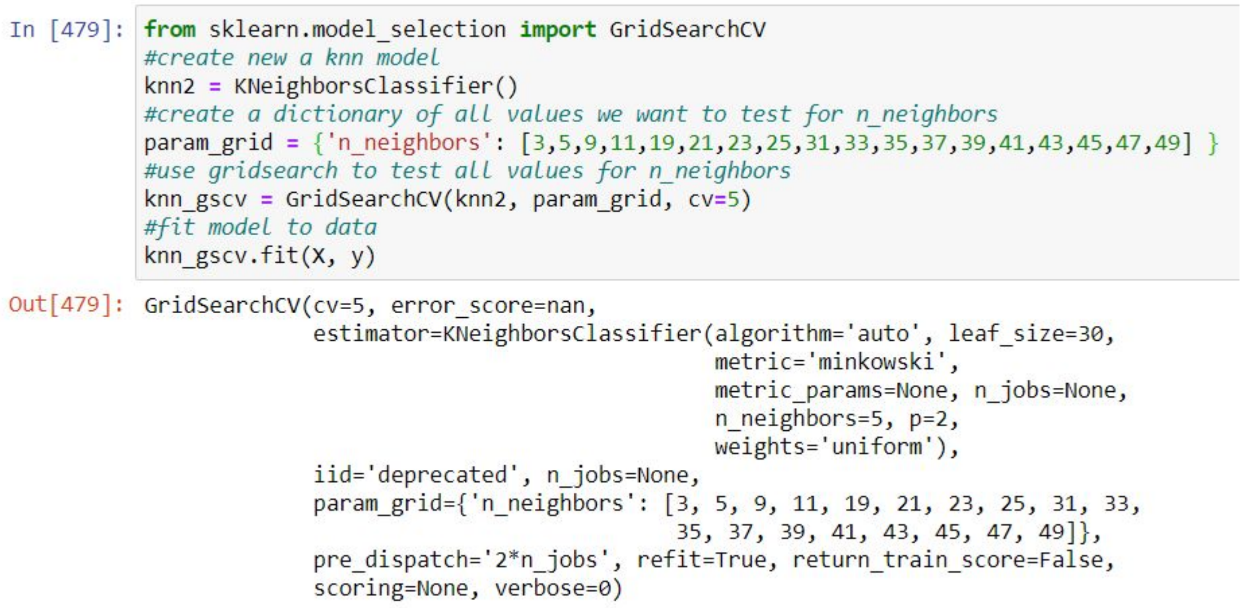


```
In [478]: from sklearn.model_selection import cross_val_score
          import numpy as np
          #create a new KNN model
          knn_cv = KNeighborsClassifier(n_neighbors=3)
          #train model with cv of 5
          cv_scores = cross_val_score(knn_cv, X, y, cv=5)
          #print each cv score (accuracy) and average them
          print(cv_scores)
          print('cv_scores mean:{}'.format(np.mean(cv_scores)))

          [0.66666667 0.64957265 0.63247863 0.57758621 0.72413793]
          cv_scores mean:0.6500884173297966
```

Figure 53

10.8 - Create a dictionary of all values we want to test for n_neighbors.

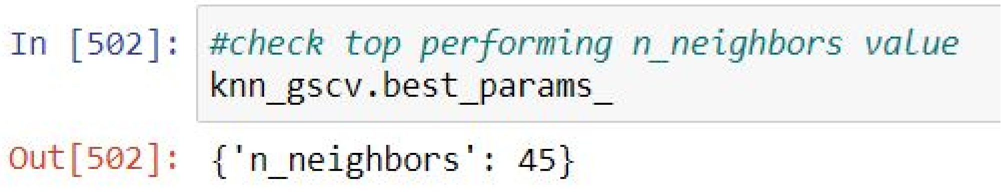
- **Fit model to data.**



```
In [479]: from sklearn.model_selection import GridSearchCV
          #create new a knn model
          knn2 = KNeighborsClassifier()
          #create a dictionary of all values we want to test for n_neighbors
          param_grid = {'n_neighbors': [3,5,9,11,19,21,23,25,31,33,35,37,39,41,43,45,47,49] }
          #use gridsearch to test all values for n_neighbors
          knn_gscv = GridSearchCV(knn2, param_grid, cv=5)
          #fit model to data
          knn_gscv.fit(X, y)

Out[479]: GridSearchCV(cv=5, error_score=nan,
                       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                                      metric='minkowski',
                                                      metric_params=None, n_jobs=None,
                                                      n_neighbors=5, p=2,
                                                      weights='uniform'),
                       iid='deprecated', n_jobs=None,
                       param_grid={'n_neighbors': [3, 5, 9, 11, 19, 21, 23, 25, 31, 33,
                                                   35, 37, 39, 41, 43, 45, 47, 49]},
                       pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                       scoring=None, verbose=0)
```

Figure 54

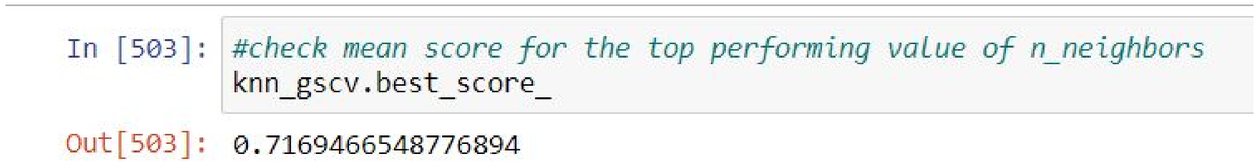
10.9 - To check top performing n_neighbors value :



```
In [502]: #check top performing n_neighbors value
          knn_gscv.best_params_

Out[502]: {'n_neighbors': 45}
```

Figure 55

- **To check mean score for the top performing value of n_neighbors :**



```
In [503]: #check mean score for the top performing value of n_neighbors
          knn_gscv.best_score_

Out[503]: 0.7169466548776894
```

Figure 56

By using grid search to find the optimal parameter for our model, we have improved our model accuracy by over 6% i.e., 71%

CHAPTER 11
RANDOM FOREST CLASSIFIER

11.1 - Random Forest Classifier :

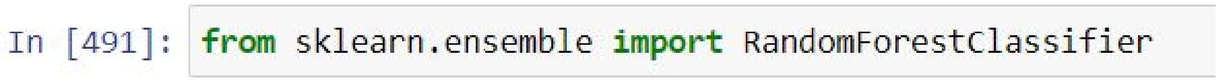
Random Forest Algorithm :

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

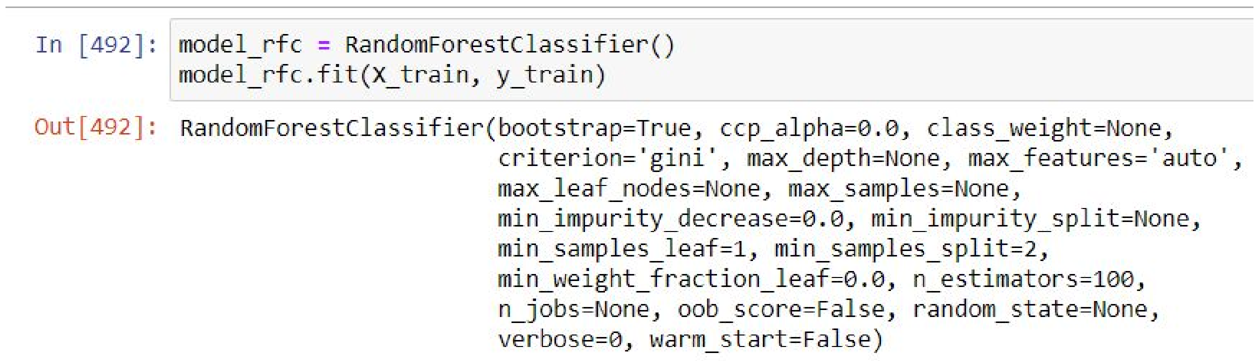
The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

11.2 - Importing RandomForestClassifier from sklearn.ensemble :



```
In [491]: from sklearn.ensemble import RandomForestClassifier
```

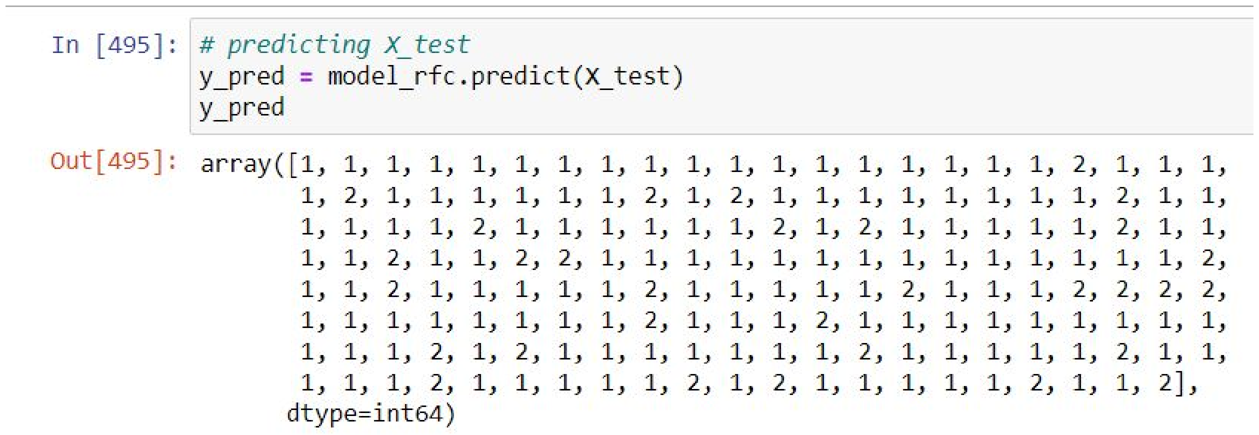
Figure 57



```
In [492]: model_rfc = RandomForestClassifier()
          model_rfc.fit(X_train, y_train)

Out[492]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                 criterion='gini', max_depth=None, max_features='auto',
                                 max_leaf_nodes=None, max_samples=None,
                                 min_impurity_decrease=0.0, min_impurity_split=None,
                                 min_samples_leaf=1, min_samples_split=2,
                                 min_weight_fraction_leaf=0.0, n_estimators=100,
                                 n_jobs=None, oob_score=False, random_state=None,
                                 verbose=0, warm_start=False)
```

Figure 58

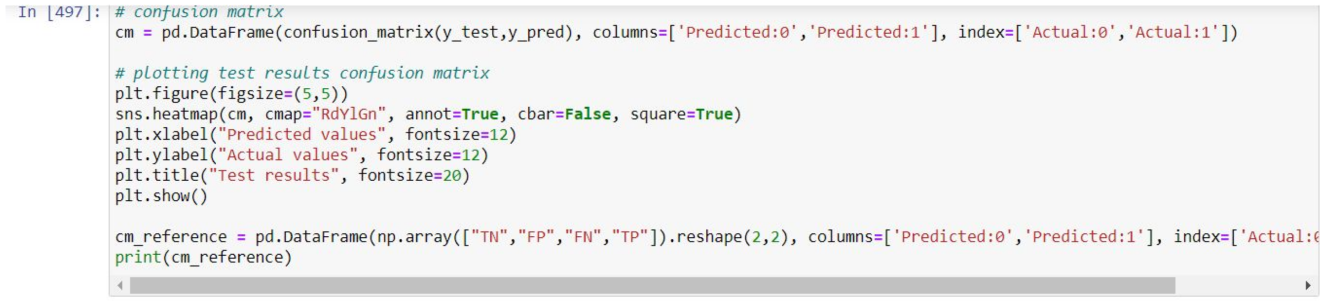
11.3 - Predicting X_test :



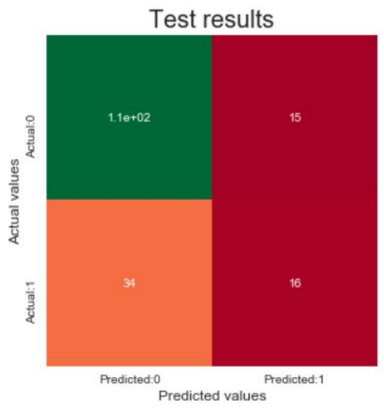
```
In [495]: # predicting X_test
          y_pred = model_rfc.predict(X_test)
          y_pred

Out[495]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
                 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1,
                 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1,
                 1, 1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
                 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 2, 2,
                 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1,
                 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 2],
                dtype=int64)
```

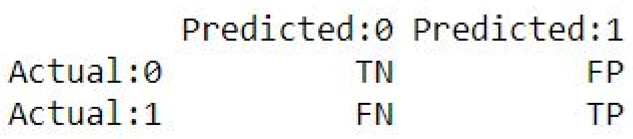
Figure 59

11.4 - Confusion matrix :



```
In [497]: # confusion matrix
          cm = pd.DataFrame(confusion_matrix(y_test,y_pred), columns=['Predicted:0','Predicted:1'], index=['Actual:0','Actual:1'])

          # plotting test results confusion matrix
          plt.figure(figsize=(5,5))
          sns.heatmap(cm, cmap="RdYlGn", annot=True, cbar=False, square=True)
          plt.xlabel("Predicted values", fontsize=12)
          plt.ylabel("Actual values", fontsize=12)
          plt.title("Test results", fontsize=20)
          plt.show()

          cm_reference = pd.DataFrame(np.array(["TN","FP","FN","TP"]).reshape(2,2), columns=['Predicted:0','Predicted:1'], index=['Actual:
          print(cm_reference)
```

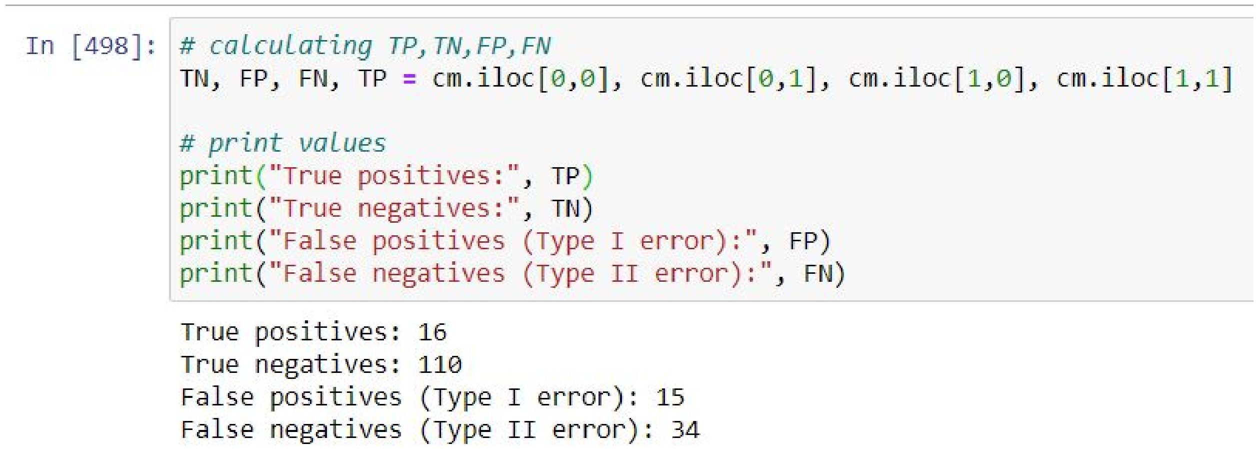




```
          Predicted:0 Predicted:1
Actual:0           TN          FP
Actual:1           FN          TP
```

Figure 60

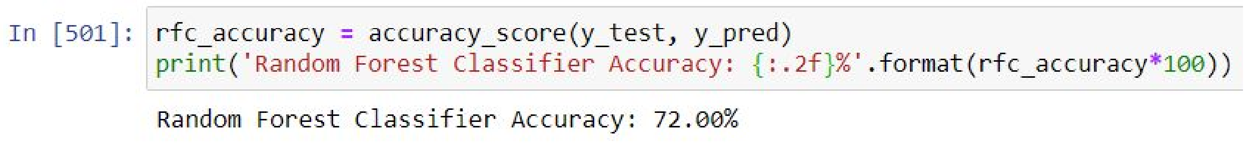
11.5 - Calculating TP,TN,FP,FN :



```
In [498]: # calculating TP,TN,FP,FN
          TN, FP, FN, TP = cm.iloc[0,0], cm.iloc[0,1], cm.iloc[1,0], cm.iloc[1,1]

          # print values
          print("True positives:", TP)
          print("True negatives:", TN)
          print("False positives (Type I error):", FP)
          print("False negatives (Type II error):", FN)

          True positives: 16
          True negatives: 110
          False positives (Type I error): 15
          False negatives (Type II error): 34
```

Figure 61

11.6 Accuracy of the model when Random Forest Classifier is used :



```
In [501]: rfc_accuracy = accuracy_score(y_test, y_pred)
          print('Random Forest Classifier Accuracy: {:.2f}%'.format(rfc_accuracy*100))

          Random Forest Classifier Accuracy: 72.00%
```

Figure 62

CONCLUSION:

It is concluded after performing thorough Exploratory Data analysis which include Stats models which are computed to get accuracy and also Heat maps which are computed to get a clear understanding of the data set (which parameter has the most abundant effect on the study case).