DATA SCIENCE

(Classification of Liver Diseases)

Summer Internship Report submitted to

In partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

By

SANABOIANA HARI SARMISHTA

PIN No.: 221710309053

Under the guidance of

Assistant Professor

GITAM

(DEEMED TO BE UNIVERSITY)

VISAKHAPATNAM • HYDERABAD • BENGALURU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM (Deemed to be University)

Hyderabad-502329

JULY-2020

DECLARATION

I submit this Industrial 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 and Engineering**". I declare that it was carried out independently carried out by me under the guidance of _____, Asst. Professor, GITAM (Deemed to be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD

Date:

SANABOIANA HARI SARMISHTA

PIN# 221710309053

CERTIFICATE

This is to certify that the Industrial Training Report entitled "CLASSIFICATION OF LIVER DISEASES" is being submitted by SANABOIANA HARI SARMISHTA (221710309053) in partial fulfillment of the requirements for the award of the degree of "Bachelor of Technology" in "Computer Science and Engineering at GITAM University, Hyderabad for the Academic year 2020-21.

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

Mr.
Lecturer, Dept. of
Computer Science & Engineering

Mr. Prof. S. Phani Kumar
Professor and HOD, Dept. of
Computer Science & Engineering

ACKNOWLEDGEMENT

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

I would like to thank respected **Dr. N. Siva Prasad**, Pro Vice Chancellor, GITAM Hyderabad and **Prof. N. Seetha Ramaiah**, Principal, GITAM Hyderabad for providing necessary infrastructure and resources for the accomplishment of our internship.

I would like to thank respected **Prof. S. Phani Kumar**, 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. _____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.

SANABOIANA HARI SARMISHTA
PIN# 221710309053

ABSTRACT

Problems with liver patients are not easily discovered in an early stage as it will be functioning normally even when it is partially damaged. An early diagnosis of liver problems will increase a patient's survival rate. Liver failures are at a high rate of risk among Indians. It is expected that by 2025 India may become the World Capital for Liver Diseases. The widespread occurrence of liver infection in India is contributed due to deskbound lifestyle, increased alcohol consumption and smoking. There are about 100 types of liver infections. Therefore, developing a machine that will enhance the diagnosis of the disease will be of great advantage in the medical field. These systems will help the physicians in making accurate decisions on patients and also with the help of Automatic classification tools for liver diseases (probably mobile enabled or web enabled), one can reduce the patient queue at the liver experts such as endocrinologists.

Diagnosis of liver disease at a preliminary stage is important for better treatment. It is a very challenging task for medical researchers to predict the disease in the early stages owing to subtle symptoms. Often the symptoms become apparent when it is too late. To overcome this issue, this project aims to improve liver disease diagnosis using machine learning approaches. Classification techniques are much popular in medical diagnosis and predicting diseases. In this study, THREE classification algorithms Logistic Regression, Random Forest and K-Nearest Neighbor (KNN) have been considered for comparing their performance based on the liver patient data. The main objective of this project is to use classification algorithms to identify the liver patients from healthy individuals. This project also aims to compare the Classification algorithms based on their performance factors.

Table of Contents

TABLE OF FIGURES

1 DATA SCIENCE

1.1 Introduction of Data Science

Data Science has dominated almost all the industries of the world today. There is no industry in the world today that does not use data. As such, data science has become fuel for industries. There are various industries like banking, finance, manufacturing, transport, e-commerce, education, etc. that use data science. As a result, there are several Data Science Applications related to it.

1.1.1 What is Data Science

Data is one of the important features of every organization because it helps business leaders to make decisions based on facts, statistical numbers and trends. Due to this growing scope of data, data science came into picture which is a multidisciplinary field. It uses scientific approaches, procedure, algorithms, and framework to extract the knowledge and insight from a huge amount of data. The extracted data can be either structured or unstructured.

Data science is a concept to bring together ideas, data examination, Machine Learning, and their related strategies to comprehend and dissect genuine phenomena with data. Data science is an extension of various data analysis fields such as data mining, statistics, predictive analysis and many more. Data Science is a huge field that uses a lot of methods and concepts which belongs to other fields like information science, statistics, mathematics, and computer science. Some of the techniques utilized in Data Science encompasses machine learning, visualization, pattern recognition, probability model, data engineering, signal processing, etc.

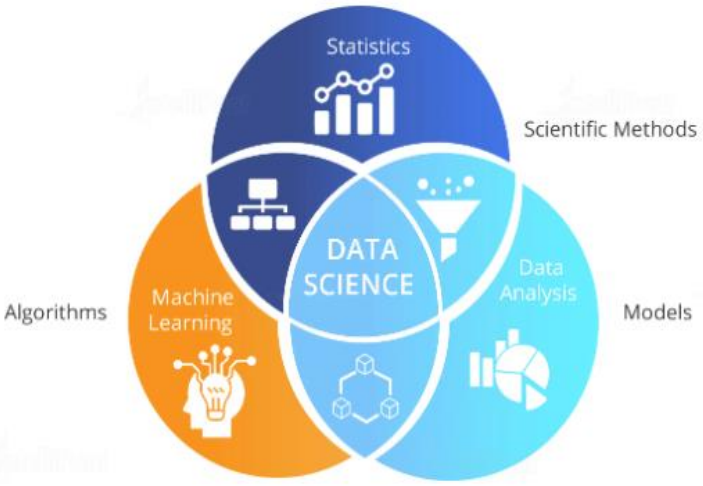


Figure 1.1.1.1: Data Science

Data science is an interdisciplinary field focused on extracting knowledge from data sets, which are typically large. Data science is the process of deriving knowledge and insights from a huge and diverse set of data through organizing, processing and analyzing the data. It involves many different disciplines like mathematical and statistical modelling, extracting data from its source and applying data visualization techniques. Often it also involves handling big data technologies to gather both structured and unstructured data.

Practitioners of data science use programming skills, statistics knowledge, and machine learning techniques to mine large data sets for patterns that can be used to analyze the past or even predict the future. Although every data science job is different, here's one way to visualize the data science workflow, with some examples of typical tasks a data scientist might perform at each step.

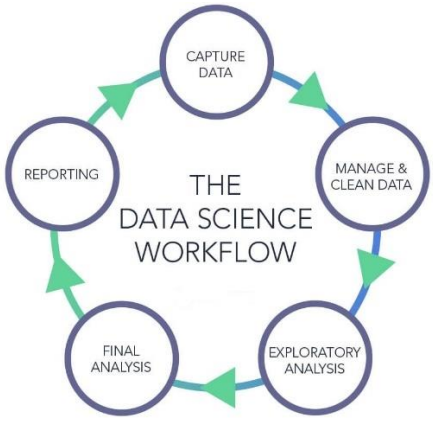


Figure 1.1.2.2: Data Science Workflow

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.

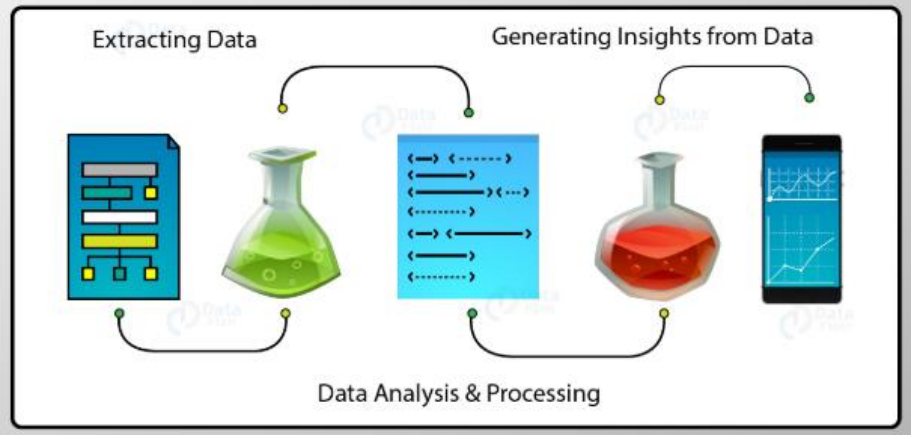


Figure 1.1.2.1: Data Analysis & Processing

- With the help of Data Science, the companies will be able to recognize their client in a more improved and enhanced way. Clients are the foundation of any product and play an essential role in their success and failure. Data Science enables companies to connect with their customers in a modified manner and thus confirms the better quality and power of the product.
- Data Science allows products to tell their story powerfully and engagingly. This is one of the reasons which makes it popular. When products and companies use this data inclusively, they can share their story with their viewers and thus create better product connections.
- One of the important features of Data Science is that its results can be applied to almost all types of industries such as travel, healthcare and education. With the help of Data Science, the industries can analyze their challenges easily and can also address them effectively.
- Presently, data science is available in almost all the fields and there is a vast amount of data present in the world today and if it is used properly it can lead the product to success or failure. If data is used correctly it will hold the importance for achieving goals for the product in the future.
- Big data is continuously emerging and growing. Using various tools which are developed regularly, big data helps the organization to resolve complex issues related to IT, human resource and resource management efficiently and successfully.
- Data science is gaining popularity in every industry and thus playing a significant role in functioning and growth of any product. Therefore, the requirement of data scientists is also increased as they have to perform an

important task of handling data and delivering solutions for the specific problems.

- Data science also influenced the retail industries. Let's take an example to understand this, the older people were having a fantastic interaction with the local seller. The seller was also capable of fulfilling the requirements of the clients in a personalized way. But now due to the emergence and increase of supermarkets, this attention got lost. But with the help of data analytics, it is possible for the sellers to connect with their clients.
- Data Science helps organizations to build this connection with the clients. With the help of data science, organizations and their products will be able to create a better and deep understanding of how customers can utilize their products.

Regardless of the industry vertical, Data Science is likely to play a key role in the organization's success. The below infographic will emphasize on how Data Science is creating its impression.

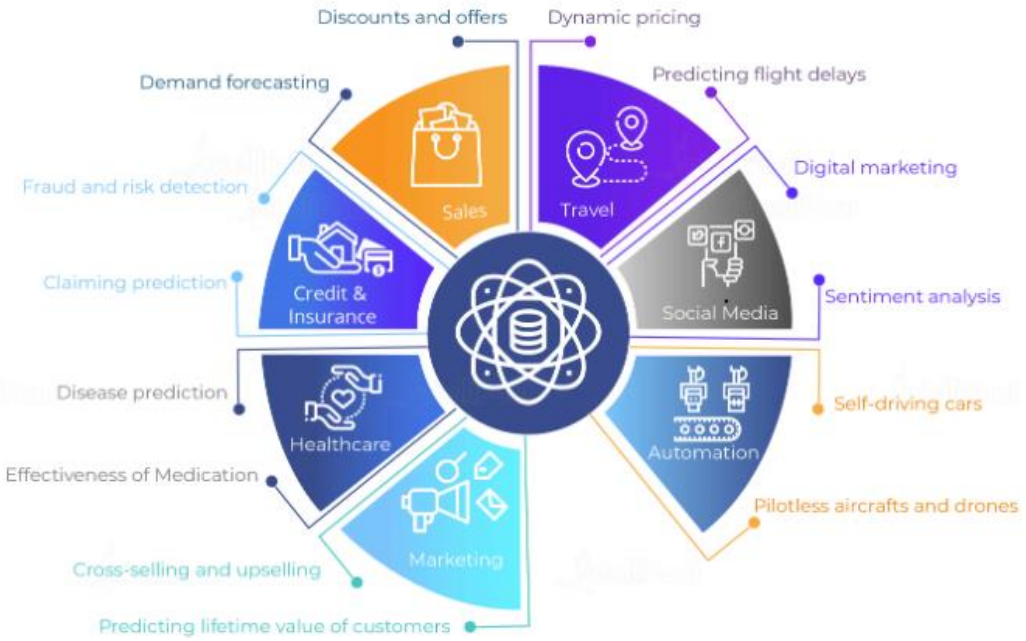


Figure 1.1.2.2: Data Science & its impression

1.1.3 Uses of Data Science

- **Recommendation systems** - As online shopping becomes more prevalent, the e-commerce platforms are able to capture users' shopping preferences as well as the performance of various products in the market. This leads to creation of recommendation systems which

create models predicting the shoppers needs and show the products the shopper is most likely to buy.

- **Financial Risk management** - The financial risk involving loans and credits are better analyzed by using the customers past spend habits, past defaults, other financial commitments and many socio-economic indicators. These data are gathered from various sources in different formats. Organizing them together and getting insight into customers' profiles needs the help of Data science. The outcome is minimizing loss for the financial organization by avoiding bad debt.
- **Improvement in Health Care services** - The healthcare industry deals with a variety of data which can be classified into technical data, financial data, patient information, drug information and legal rules. All this data needs to be analyzed in a coordinated manner to produce insights that will save cost both for the health care provider and care receiver while remaining legally compliant.
- **Computer Vision** - The advancement in recognizing an image by a computer involves processing large sets of image data from multiple objects of the same category. For example, Face recognition. These data sets are modelled, and algorithms are created to apply the model to newer images to get a satisfactory result. Processing of these huge data sets and creation of models need various tools used in Data science.
- **Efficient Management of Energy** - As the demand for energy consumption soars, the energy producing companies need to manage the various phases of the energy production and distribution more efficiently. This involves optimizing the production methods, the storage and distribution mechanisms as well as studying the customers consumption patterns. Linking the data from all these sources and deriving insight seems a daunting task. This is made easier by using the tools of data science.

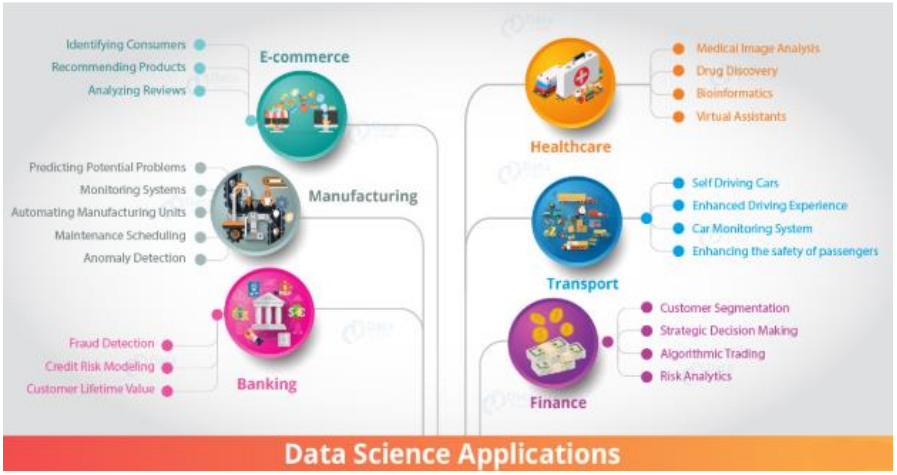


Figure 1.1.3.1: Data Science Applications in various Industries

2 MACHINE LEARNING

2.1 Introduction

Machine learning is a term closely associated with data science. It refers to a broad class of methods that revolve around data modeling to (1) algorithmically make predictions, and (2) algorithmically decipher patterns in data.

2.2 Importance of Machine Learning

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

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

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

The process flow depicted here represents how machine learning works

Figure 2.2.1: Machine Learning - Process Flow

2.3 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 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.

2.4 Types of Machine Learning

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.

2.4.1 Supervised Learning

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

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

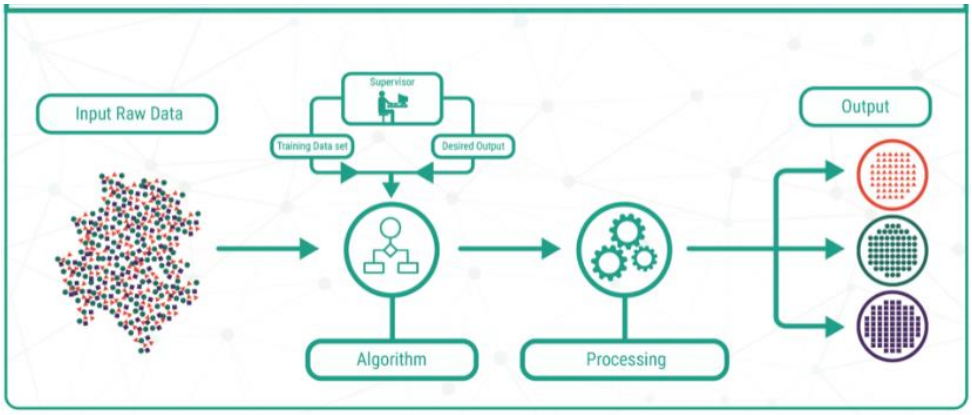


Figure 2.4.1.1: Supervised Learning

2.4.2 Unsupervised Learning

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

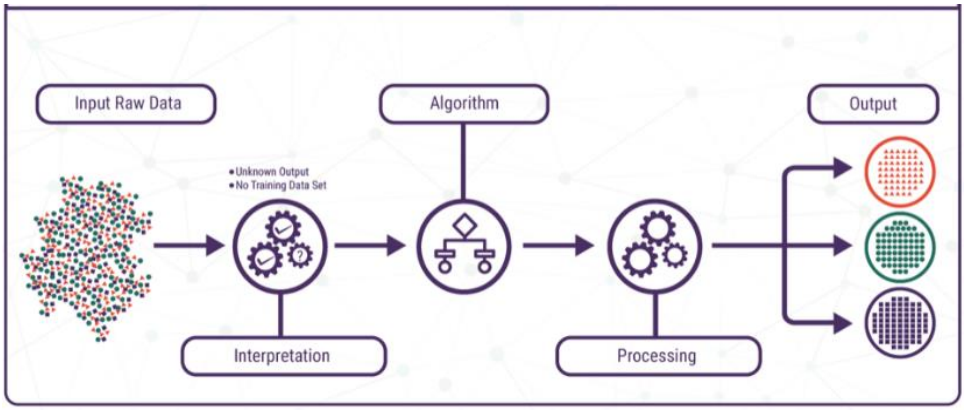


Figure 2.4.2.1: Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

2.4.3 Semi Supervised Learning

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

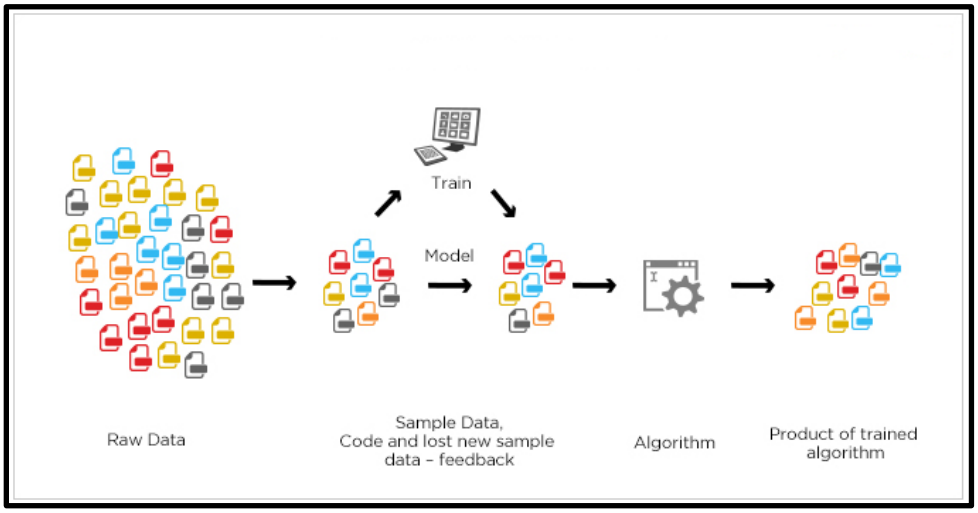


Figure 2.4.3.1: Semi Supervised Learning

2.5 Relation between Data Mining, Machine Learning and Deep Learning

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

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

3 PYTHON

Python is a programming language with simple syntax that is commonly used for data science. There are a number of python libraries that are used in data science including NumPy, pandas, and SciPy.

3.1 Introduction to Python

Basic programming language used for machine learning is: PYTHON

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

3.2 History of Python

- Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989.
- In January 2019, active Python core developers elected Brett Cannon, Nick Coghlan, Barry Warsaw, Carol Willing and Van Rossum to a five-member "Steering Council" to lead the project.
- Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.
- Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3.0.
- Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

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.

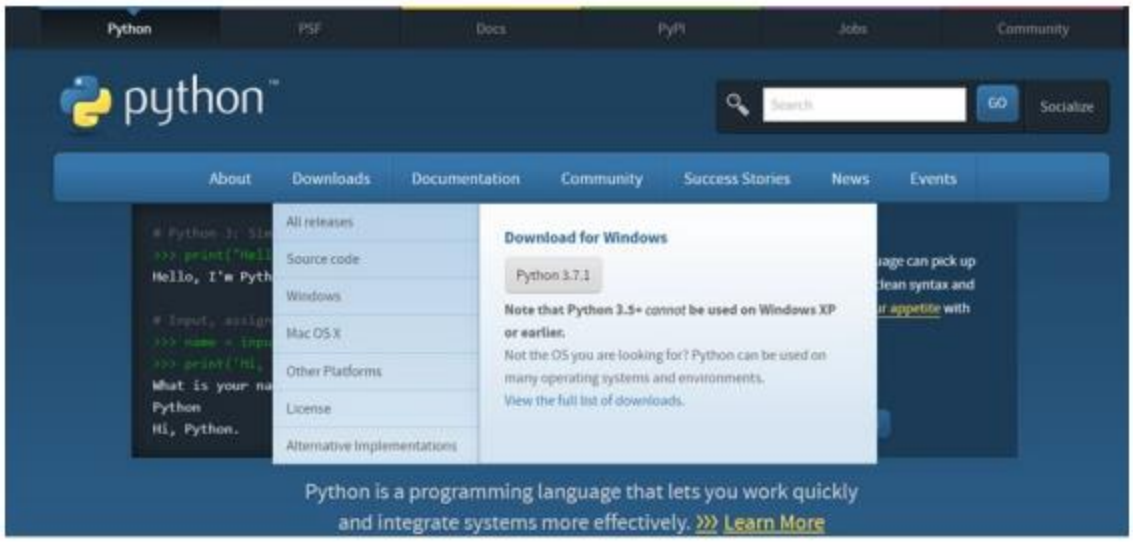


Figure 3.4.1.1: Python download

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

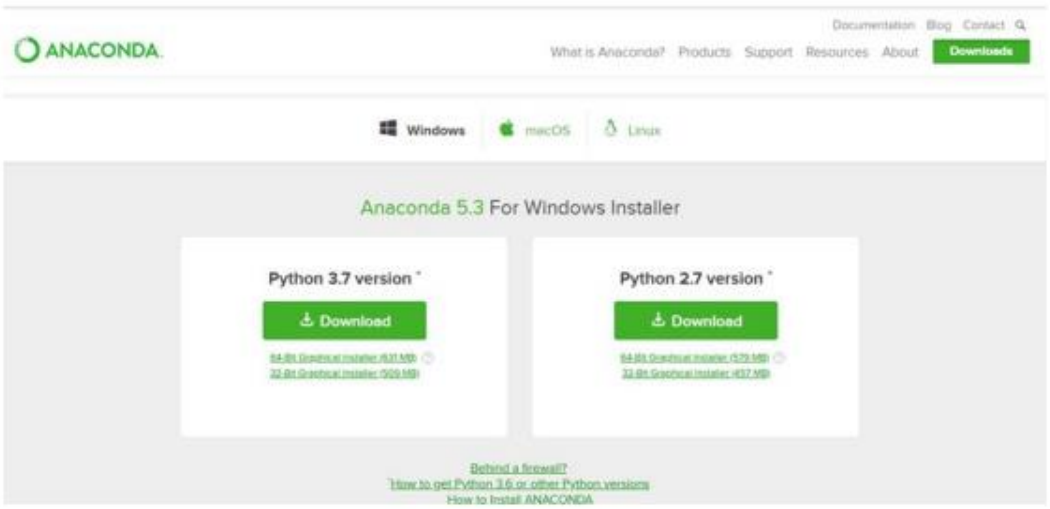


Figure 3.4.2.1: Anaconda download

Figure 3.4.2.2 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 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. A dictionary 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:
    - We define a class in a very similar way how we define a function.

Just like a function, we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is indented like a functions body is.

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

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

Figure 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__().

4 CASE STUDY – Classification of Liver Diseases

4.1 Project Requirements

Diagnosis of liver disease at a preliminary stage is important for better treatment. It is a very challenging task for medical researchers to predict the disease in the early stages owing to subtle symptoms. Often the symptoms become apparent when it is too late. To overcome this issue, this project aims to improve liver disease diagnosis using machine learning approaches. The main objective of this research is to use classification algorithms to identify the liver patients from healthy individuals. This project also aims to compare the classification algorithms based on their performance factors. This data can further be leveraged to develop a graphical user interface using python to serve the medicinal community for the diagnosis of liver disease among patients. The GUI can be readily utilized by doctors and medical practitioners as a screening tool for the liver disease.

4.2 Packages Used

- **pandas** - is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- **NumPy** - is a Python package which stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object, and provides tools for integrating C, C++ etc. It is also useful in linear algebra, random number capability etc.
- **matplotlib** - is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. Sub packages used:
    - pyplot
    - colors
        - ListedColormap
- **Seaborn** - is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **Scikit-learn** - is a free machine learning library for Python. It features various algorithms like Support Vector Machine, Random Forests, and K-Neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy. Sub packages used:
    - metrics
        - confusion_matrix
        - accuracy_score

        - f1_score
        - classification_report
    - model_selection
        - train_test_split
        - GridSearchCV
        - RandomizedSearchCV
        - cross_val_score
    - linear_model
        - LogisticRegression
    - ensemble
        - RandomForestClassifier
    - neighbors
        - KNeighborsClassifier
    - feature_selection
        - RFE
- **missingno** - is a Python library and compatible with Pandas. Missingno library offers a very nice way to visualize the distribution of NaN values.

4.3 Versions of the Packages

Package Name	Version
pandas	0.24.2
NumPy	1.16.2
matplotlib	3.0.3
Seaborn	0.9.0
Scikit-learn	0.23.1
missingno	0.4.2

4.4 Algorithms Used

1. **Logistic regression** - is basically a Supervised Classification algorithm. In a classification problem, the target variable (or output), y, can take only discrete values for a given set of features (or inputs), X.

   Contrary to popular belief, logistic regression IS a regression model. The model builds a regression model to predict the probability that a given data entry belongs to the category numbered as “1”. Just like Linear regression assumes that the data follows a linear function, Logistic regression models the data using the sigmoid function.

   $$g(z) = \frac{1}{1+e^{-z}}$$

2. **Random Forest Classifier**- is a flexible, easy to use Supervised Machine Learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It can be used for both classification and regression tasks. It is also one of the most used algorithms, because of its simplicity and diversity. The "forest" it builds is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

   Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model’s prediction

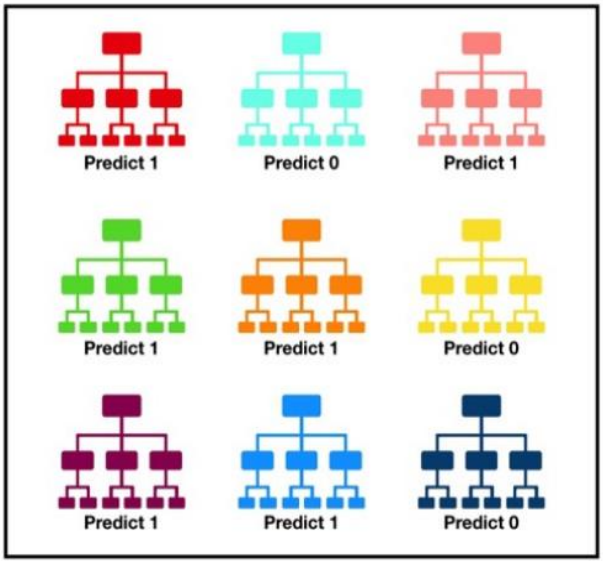


Figure 4.4.1: Random Forest Classifier Algorithm

3. **K-Nearest Neighbor (KNN) -** is a type of Supervised Machine Learning 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.

    K-Nearest Neighbor (KNN) algorithm uses 'feature similarity' to predict the values of new data points which further means that the new data point will be assigned a value based on how closely it matches the points in the training set.

4.5 Problem Statement

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.

4.6 Dataset Description

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 patients (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".

Columns:

- Age of the patient
- Gender of the patient
- Total Bilirubin
- Direct Bilirubin
- Alkaline Phosphatase
- Alamine Aminotransferase
- Aspartate Aminotransferase
- Total Proteins

- Albumin
- Albumin and Globulin Ratio
- Dataset: field used to split the data into two sets (patient with liver disease, or no disease)

4.7 Objective of the Case Study

The main objective of this project is to use classification algorithms to identify the liver patients from healthy individuals. Thus, this uses the patient records to determine which patients have liver disease and which ones do not.

In this study, THREE classification algorithms Logistic Regression, Random Forest Classifier, K-Nearest Neighbor (KNN) have been considered for comparing their performance based on the liver patient data.

Further, the model with the highest Accuracy can be considered for implementing as a user-friendly Graphical User Interface (GUI) using the Tkinter package in python.

5 MODEL BUILDING

5.1 Data Collection

Getting the dataset – We can either get the Data from Database or from the Client

Dataset from Kaggle: https://www.kaggle.com/uciml/indian-liver-patient-records

Importing the Libraries

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

Importing the Libraries

```
#Import all required libraries for reading data, analysing and visualizing data
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import LabelEncoder

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
```

Figure 5.1.1: Importing Libraries

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 accessed using the dataframe.

Any missing value or NaN value has to be cleaned.

Importing the Data Set

```
#Reading the data (Training & Test)
liver_df = pd.read_csv('classification_of_liver_diseases.csv')
```

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).

Figure 5.1.2: Importing Dataset

```
liver_df.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin
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

```
liver_df.info()
```

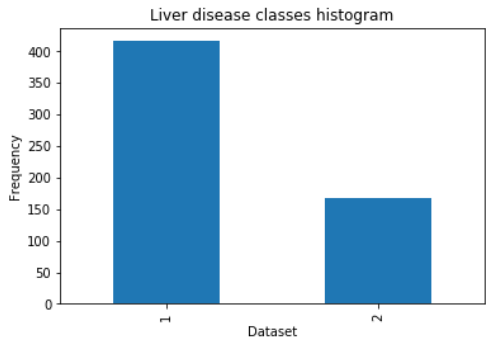
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
Age                         583 non-null int64
Gender                      583 non-null object
Total_Bilirubin             583 non-null float64
Direct_Bilirubin            583 non-null float64
Alkaline_Phosphotase        583 non-null int64
Alamine_Aminotransferase    583 non-null int64
Aspartate_Aminotransferase  583 non-null int64
Total_Protiens              583 non-null float64
Albumin                     583 non-null float64
Albumin_and_Globulin_Ratio  579 non-null float64
Dataset                     583 non-null int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

Figure 5.1.3: Dataset Column Info

Now I would like to check if the data is balanced or not by plotting a histogram

```
# checking the stats from the kaggle 416 liver disease patients and 167 non liver disease patients
# need to remap the classes liver disease:=1 and no liver disease:=0 (normal convention to be followed)
count_classes = pd.value_counts(liver_df['Dataset'], sort = True).sort_index()
count_classes.plot(kind = 'bar')
plt.title("Liver disease classes histogram")
plt.xlabel("Dataset")
plt.ylabel("Frequency")
```

```
: Text(0, 0.5, 'Frequency')
```



Check from the Kaggle: 416 liver disease patients and 167 normal patients

```
liver_df['Dataset'].value_counts()
```

```
1    416
2    167
Name: Dataset, dtype: int64
```

Figure 5.1.4: Checking if the data is balanced or not

Here is the observation from the dataset:

1) Only gender is non-numeric variable. All others are numeric.

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

5.2 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is understanding the data sets by summarizing their main characteristics often plotting them visually. This step is very important especially when we arrive at modeling the data in order to apply Machine learning. Plotting in EDA consists of Histograms, Box plot, Scatter plot and many more. It often takes much time to explore the data.

There is no one method or common methods in order to perform EDA. There are some common plots that would be used in EDA process.

Steps involved:

- Statistical Analysis
- Understanding the data by summarizing their main characteristics often plotting them visually
- Visualize data to help detect relevant relationships between variables/columns

5.2.1 Statistical Analysis

Checking Dimensions of Data - It is always a good practice to know how much data, in terms of rows and columns, we are having for our ML project. The reasons behind are −

- Suppose if we have too many rows and columns then it would take long time to run the algorithm and train the model.
- Suppose if we have too less rows and columns then it, we would not have enough data to well train the model.

```
#Describe gives statistical information about NUMERICAL columns in the dataset
liver_df.describe(include='all')
#We can see that there are missing values for Albumin_and_Globulin_Ratio as only 579 entries have valid values indicatin
#Gender has only 2 values - Male/Female
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens
count	583.000000	583	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000
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
std	16.189833	NaN	6.209522	2.808498	242.937989	182.620356	288.918529	1.085451
min	4.000000	NaN	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000
25%	33.000000	NaN	0.800000	0.200000	175.500000	23.000000	25.000000	5.800000
50%	45.000000	NaN	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000
75%	58.000000	NaN	2.600000	1.300000	298.000000	60.500000	87.000000	7.200000
max	90.000000	NaN	75.000000	19.700000	2110.000000	2000.000000	4929.000000	9.600000

Figure 5.2.1.1: Statistical Analysis of Data

Following is a Python script for getting the total number of rows and columns in it.

```
liver_df.shape
```

```
(583, 11)
```

These are the Data Set features available.

```
#Which features are available in the dataset?
liver_df.columns
```

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

Figure 5.2.1.2: Features available in Dataset

```
#Check for any null values
liver_df.isnull().sum()
```

```
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 5.2.1.3: Checking for Null values in Dataset

Let's see from the above observation dataset.

- Only gender column is non-numeric variable. All others columns 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.
- The only data that is null is the Albumin_and_Globulin_Ratio - Only 4 rows are null. Let's see whether this is an important feature

5.2.2 Distribution of Categorical data

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

- Categorical Variables are of two types: Nominal and Ordinal
    - Nominal: The categories do not have any numeric ordering in between them. They don't have any ordered relationship between each of them. Examples: Male or Female, any color.
    - Ordinal: The categories have a numerical ordering in between them. Example: Graduate is less than Post Graduate, Post Graduate is less than Ph.D. customer satisfaction survey, high low medium.
- Categorical data can be handled by using dummy variables, which are also called as indicator variables.

5.2.3 Data Visualization

Data Visualization is the discipline of trying to understand **data** by placing it in a visual context so that patterns, trends and correlations that might not otherwise be detected can be exposed. **Python** offers multiple great graphing libraries that come packed with lots of different features.

Matplotlib is the most popular data visualization library of Python and is a 2D plotting library. It is the most widely-used library for plotting in the Python community and is more than a decade old.

Because matplotlib was the first Python data visualization library, many other libraries are built on top of it or designed to work in tandem with-it during analysis. Some libraries like pandas and Seaborn are “wrappers” over matplotlib. They allow you to access a number of matplotlib's methods with less code.

Identifying Missing Values

```
# Visualizing the missing values with heatmap
sns.heatmap(liver_df.isna())
```

```
9]: <matplotlib.axes._subplots.AxesSubplot at 0x1ab7d88d7f0>
```

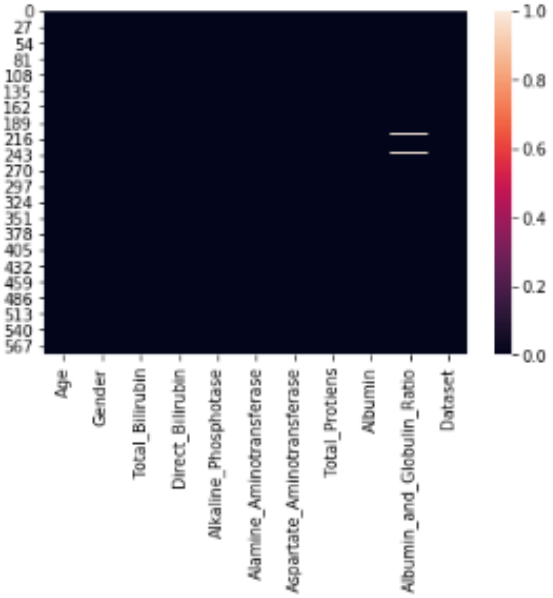


Figure 5.2.3.1: Missing Data Heat Map

```
## missingno--> It's specially used for checking(visualizing) the missing values
import missingno as msno
msno.bar(liver_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ab7dc3fcc0>
```

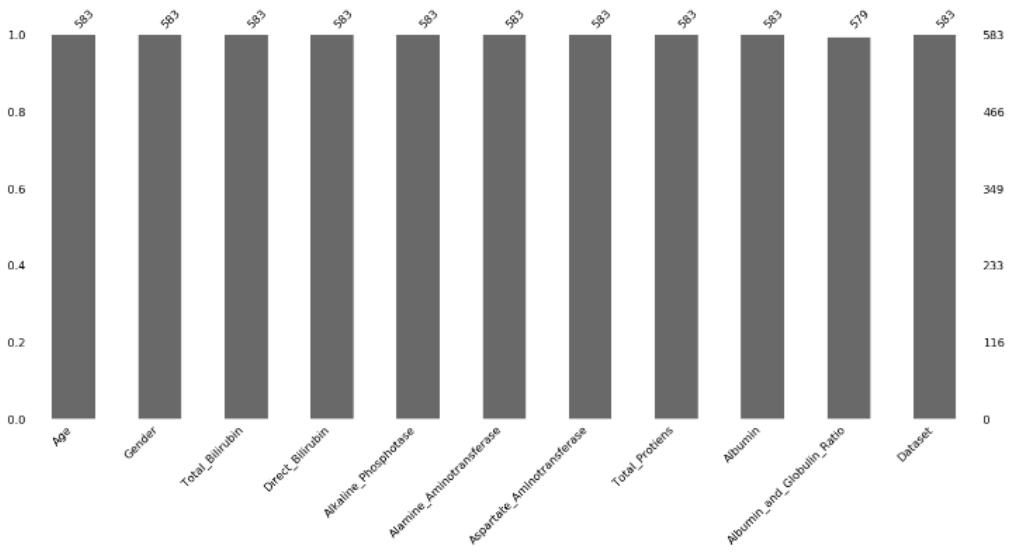


Figure 5.2.3.2: Missing Values Data Visualization in Bar Format

```
liver_df.hist(figsize=(10,10))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DCE2CF8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DD84908>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DDABE80>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DDDB438>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DE039B0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DE2CEF0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DE5C4A8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DE84A58>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DE84A90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DEDD550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DF04AC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001AB7DF38080>]],
      dtype=object)
```

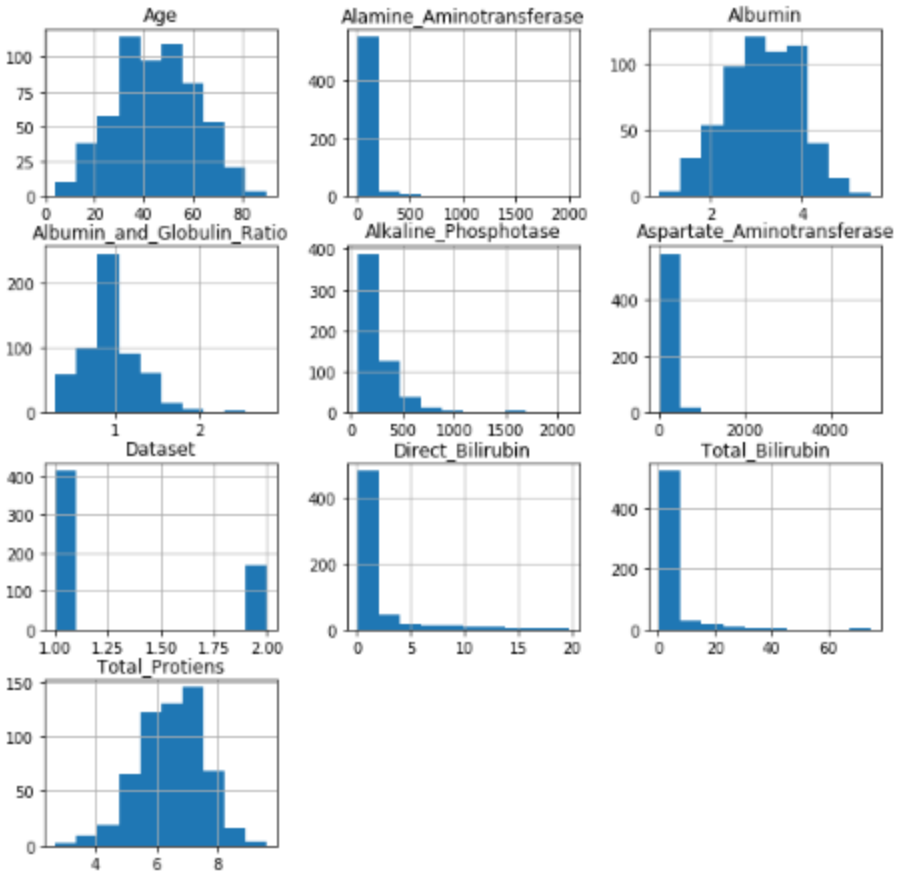


Figure 5.2.3.3: Data Visualization - Numeric Data using Histogram

To check the countplot of the Dataset column

```
# To check the countplot of the Dataset column
sns.countplot(data=liver_df, x = 'Dataset', label='Count')

LD, NLD = liver_df['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
```

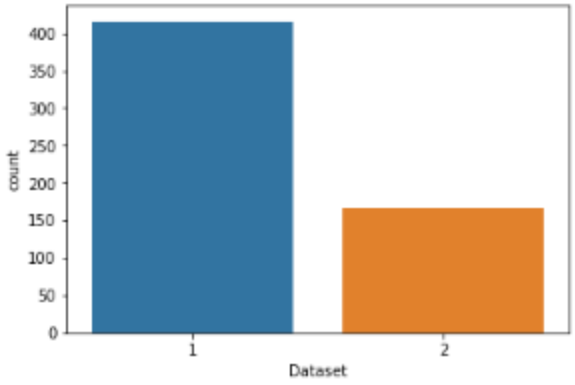


Figure 5.2.3.4: Data Visualization – Patient Diagnosis

To check the countplot of the Gender column

```
#To check the countplot of Gender column
sns.countplot(data=liver_df, x = 'Gender', label='Count')

M, F = liver_df['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
```

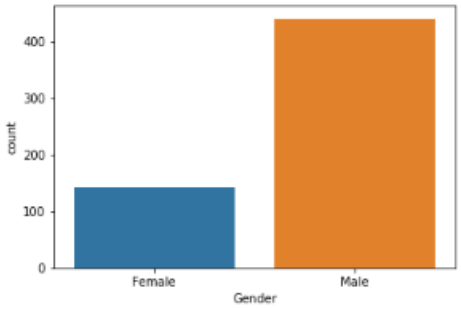


Figure 5.2.3.5: Data Visualization - Gender Diagnosis

```
sns.catplot(x="Age", y="Gender", hue="Dataset", data=liver_df);
```

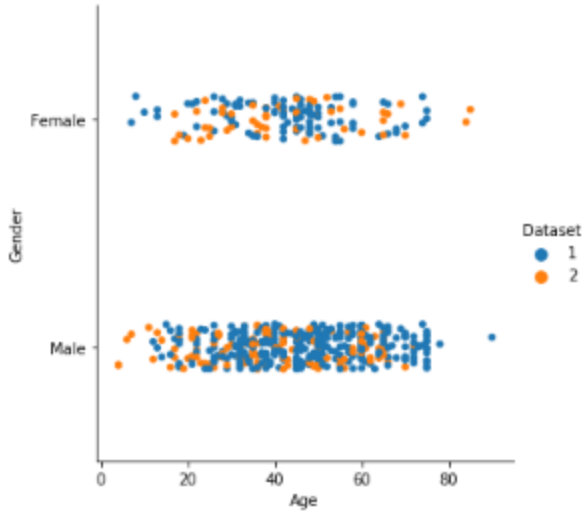


Figure 5.2.3.6: Data Visualization – Gender Categorization using CatPlot

Age seems to be a factor for liver disease for both male and female gender.

```
liver_df[['Gender', 'Dataset','Age']].groupby(['Dataset','Gender'],
                                              as_index=False).count().sort_values(by='Dataset', ascending=False)
```

]:

	Dataset	Gender	Age
2	2	Female	50
3	2	Male	117
0	1	Female	92
1	1	Male	324

```
liver_df[['Gender', 'Dataset','Age']].groupby(['Dataset','Gender'],
                                              as_index=False).mean().sort_values(by='Dataset', ascending=False)
```

:

	Dataset	Gender	Age
2	2	Female	42.740000
3	2	Male	40.598291
0	1	Female	43.347826
1	1	Male	46.950617

```
g = sns.FacetGrid(liver_df, col="Dataset", row="Gender", margin_titles=True)
g.map(plt.hist, "Age", color="red")
plt.subplots_adjust(top=0.9)
g.fig.suptitle('Disease by Gender and Age');
```

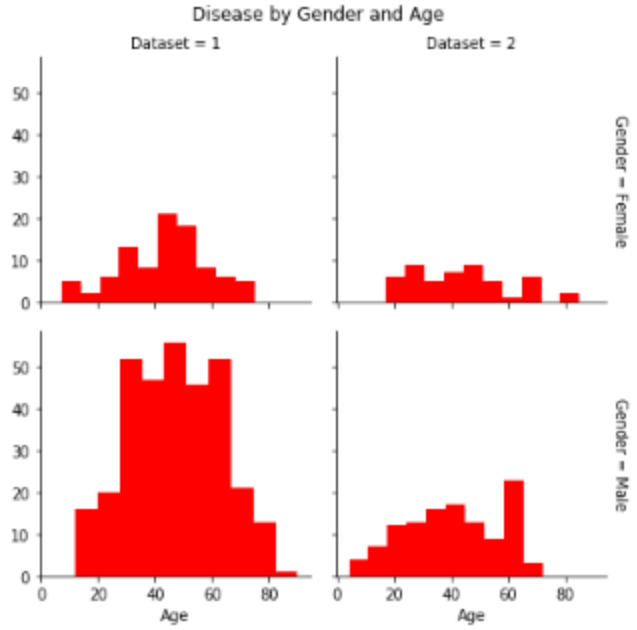


Figure 5.2.3.7: Data Visualization - Disease by Gender & Age

```
g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Direct_Bilirubin", "Total_Bilirubin", edgecolor="w")
plt.subplots_adjust(top=0.9)
```

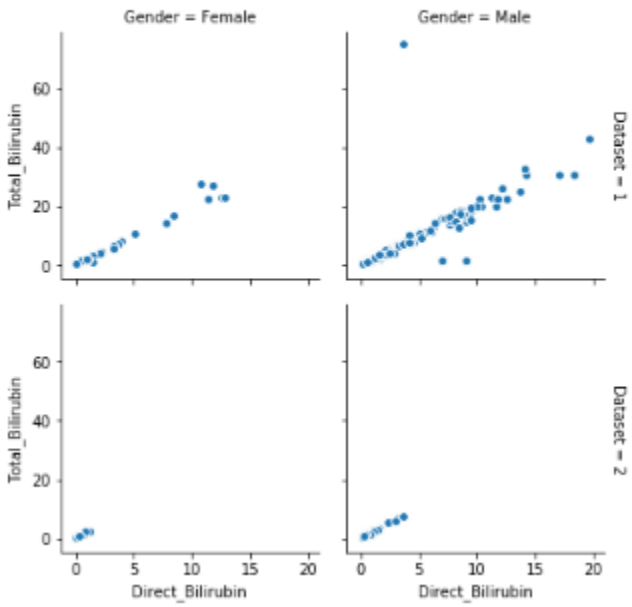
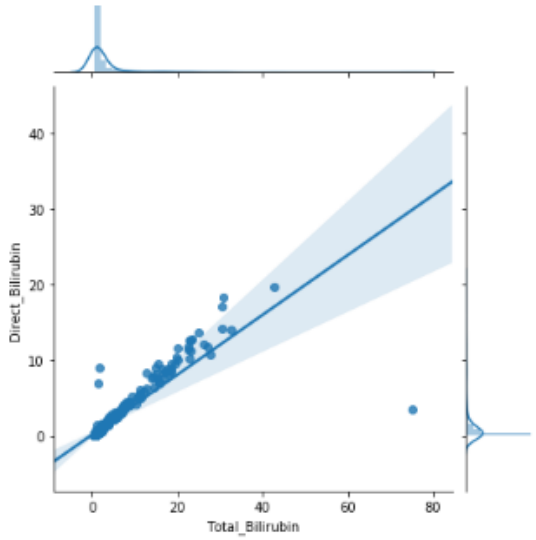


Figure 5.2.3.8: Data Visualization - Relation b/w Total Bilirubin & Direct Bilirubin

There seems to be direct relationship between Total_Bilirubin and Direct_Bilirubin. We have the possibility of removing one of these features.

```
1 sns.jointplot("Total_Bilirubin", "Direct_Bilirubin", data=liver_df, kind="reg")
```

```
]: <seaborn.axisgrid.JointGrid at 0x1ab7eba5518>
```




```
1 g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)
2 g.map(plt.scatter,"Aspartate_Aminotransferase", "Alamine_Aminotransferase",  edgecolor="w")
3 plt.subplots_adjust(top=0.9)
```

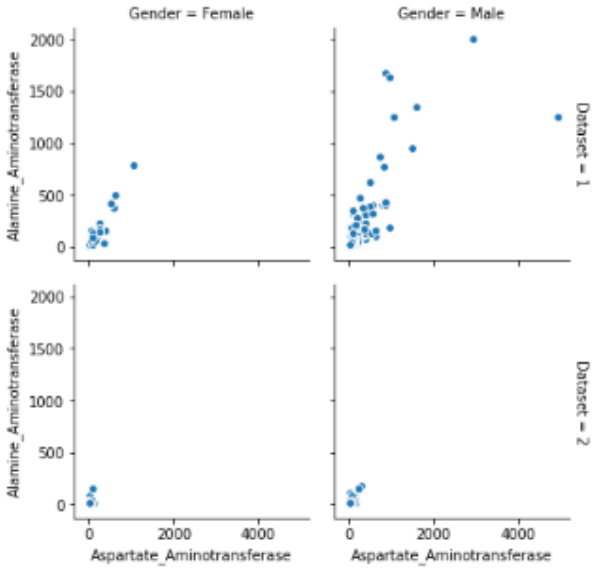
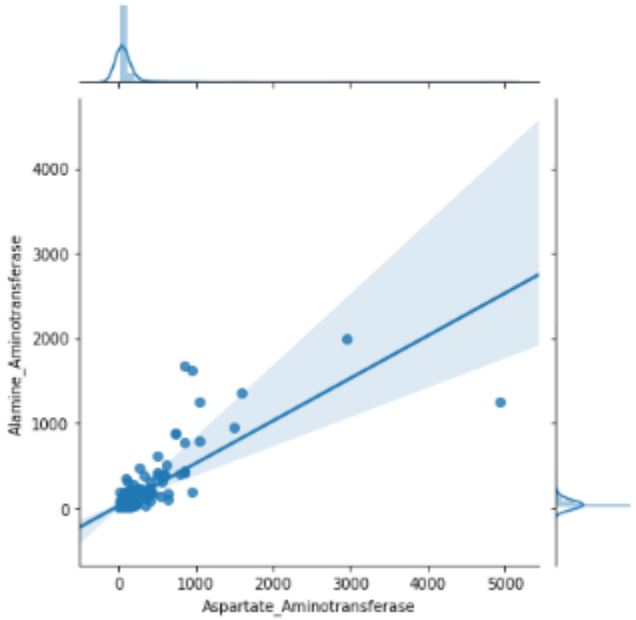


Figure 5.2.3.9: Data Visualization - Relation b/w Aspartate_Aminotransferase & Alamin_ Aminotransferase

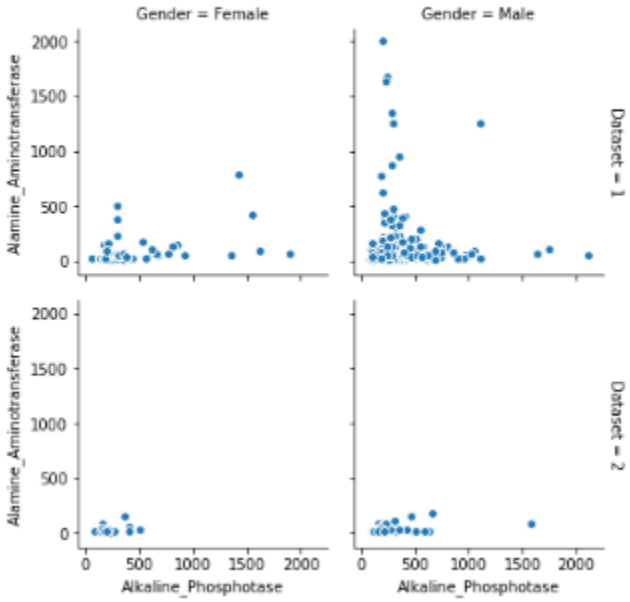
There is linear relationship between Aspartate_Aminotransferase and Alamine_Aminotransferase and the gender. We have the possibility of removing one of these features.

```
sns.jointplot("Aspartate_Aminotransferase", "Alamine_Aminotransferase",
              data=liver_df, kind="reg")
```

```
<seaborn.axisgrid.JointGrid at 0x1ab7ec54320>
```




```
g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Alkaline_Phosphotase", "Alamine_Aminotransferase",  edgecolor="w")
plt.subplots_adjust(top=0.9)
```



```
sns.jointplot("Alkaline_Phosphotase", "Alamine_Aminotransferase",
              data=liver_df, kind="reg")
```

```
<seaborn.axisgrid.JointGrid at 0x1ab7e150940>
```

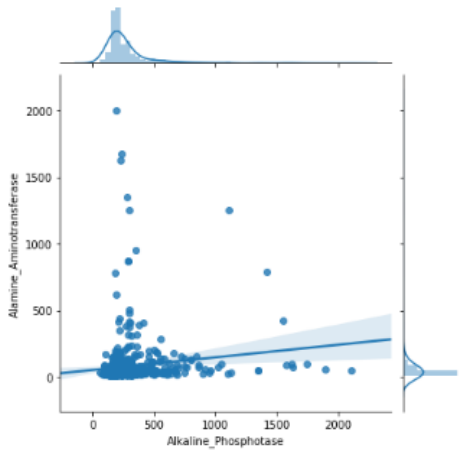


Figure 5.2.3.10: Data Visualization - No linear correlation between Alkaline_Phosphotase and Alamine_Aminotransferase

No linear correlation between Alkaline_Phosphotase and Alamine_Aminotransferase.

```
g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Total_Protiens", "Albumin",  edgecolor="W")
plt.subplots_adjust(top=0.9)
```

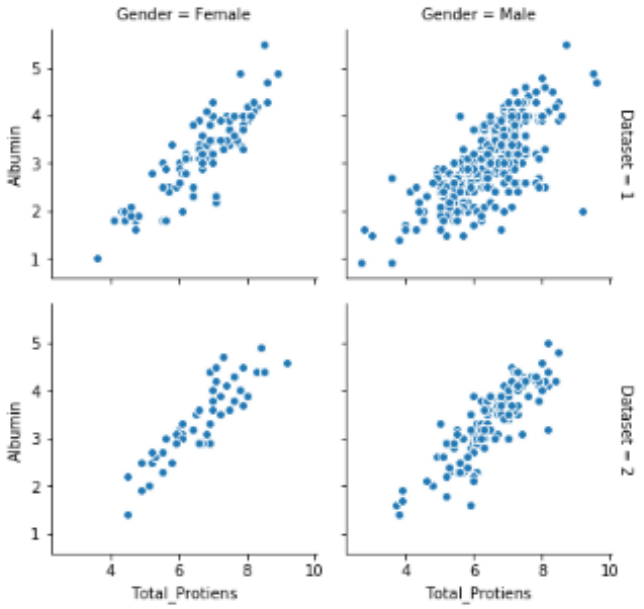
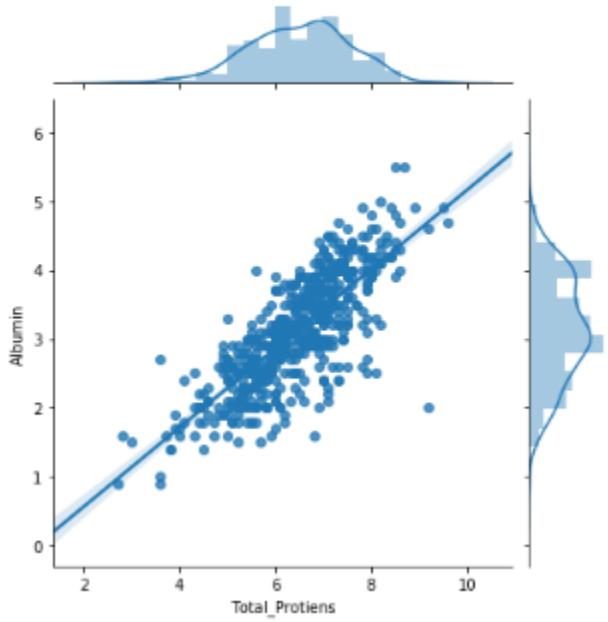


Figure 5.2.3.11: Data Visualization - Linear relationship b/w Total_Protiens and Albumin and Gender

There is linear relationship between Total_Protiens and Albumin and the gender. We have the possibility of removing one of these features.

```
1 sns.jointplot("Total_Protiens", "Albumin", data=liver_df, kind="reg")
```

```
<seaborn.axisgrid.JointGrid at 0x1ab7ff13cc0>
```



```
g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Albumin", "Albumin_and_Globulin_Ratio",  edgecolor="w")
plt.subplots_adjust(top=0.9)
```

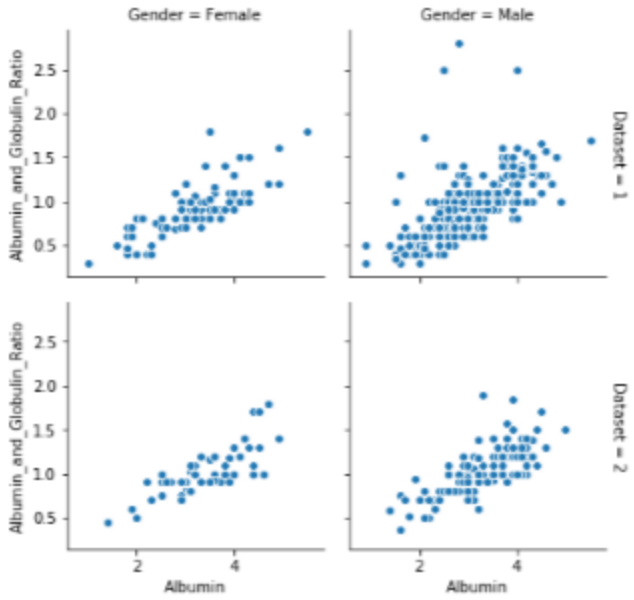
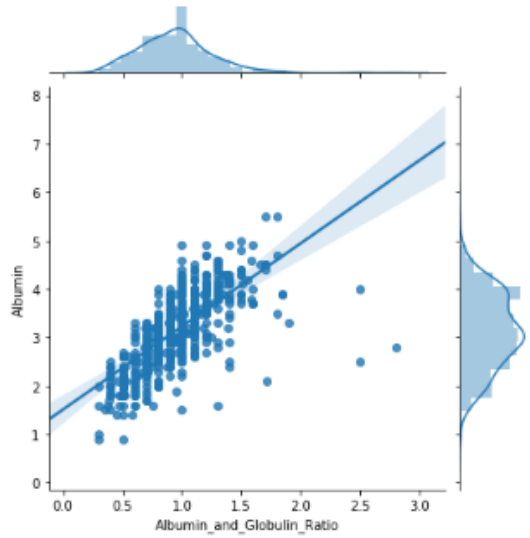


Figure 5.2.3.6: Data Visualization - Linear relationship b/w Albumin_and_Globulin_Ratio and Albumin

There is linear relationship between Albumin_and_Globulin_Ratio and Albumin. We have the possibility of removing one of these features.

```
sns.jointplot("Albumin_and_Globulin_Ratio", "Albumin", data=liver_df, kind="reg")
```

```
<seaborn.axisgrid.JointGrid at 0x1ab0225ca90>
```



```
g = sns.FacetGrid(liver_df, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Albumin_and_Globulin_Ratio", "Total_Protiens",  edgecolor="w")
plt.subplots_adjust(top=0.9)
```

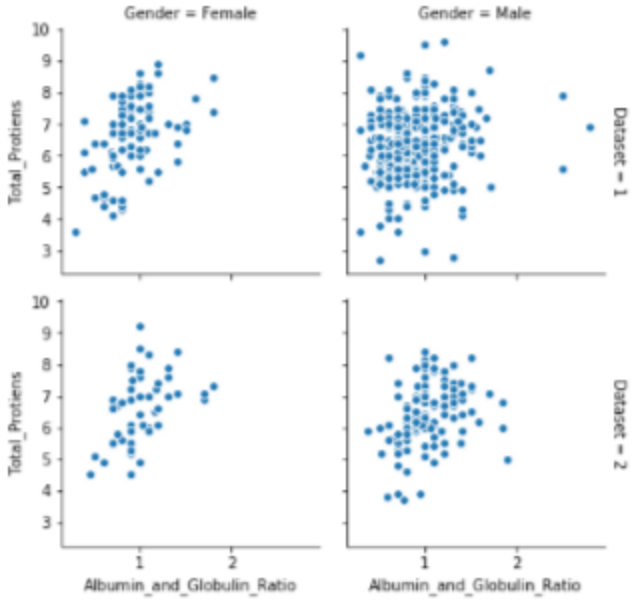


Figure 5.2.3.7: Data Visualization - Relation b/w Albumin_and_Globulin_Ratio and Total_Proteins

Overall Observation:

- From the above joint plots and scatterplots, we find direct relationship between the following features:
    - Direct_Bilirubin & Total_Bilirubin
    - Aspartate_Aminotransferase & Alamine_Aminotransferase
    - Total_Protiens & Albumin
    - Albumin_and_Globulin_Ratio & Albumin

- Hence, we can very well find that we can omit one of the features. I'm going to keep the following features:
    - Total_Bilirubin
    - Alamine_Aminotransferase
    - Total_Protiens
    - Albumin_and_Globulin_Ratio
    - Albumin

```
1 liver_df.head(3)
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin
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

5.3 Data Pre-processing

Data cleaning and preparation is a critical first step in any machine learning project. The Indian Liver Patient Dataset comprised of 10 different attributes of 583 patients. The patients were described as either 1 or 2 on the basis of liver disease. The detailed description of the dataset is shown in Statistical Analysis.

As clearly visible from it, all the features except Gender are real valued integers. The feature Gender is converted to numeric value (0 and 1) in the data pre-processing step.

5.3.1 Handling Missing Values

- There is a method called isnull() which gives the number of missing values in each and every column.
- Using fillna() method each and every missing value is replaced by 0.

```
1 #Check for any null values
2 liver_df.isnull().sum()
```

```
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 5.3.1.1: Data Pre-processing - Before handling the missing values

The only data that is null is the Albumin_and_Globulin_Ratio - Only 4 rows are null.

```
liver_df[liver_df['Albumin_and_Globulin_Ratio'].isnull()]
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin
209	45	Female	0.9	0.3	189	23	33	6.6	3.9
241	51	Male	0.8	0.2	230	24	46	6.5	3.1
253	35	Female	0.6	0.2	180	12	15	5.2	2.7
312	27	Male	1.3	0.6	106	25	54	8.5	4.8

```
liver_df["Albumin_and_Globulin_Ratio"] =
    liver_df.Albumin_and_Globulin_Ratio.fillna(liver_df['Albumin_and_Globulin_Ratio'].mean())
```

5.3.2 Encoding Categorical columns

- Dealing with categorical data
- Using label encoder from preprocessing package which is present in scikit learn, we can get dummies in place of categorical data
- Once we get dummies we need to fit and transform that to our dataframe

Convert categorical variable "Gender" to indicator variables

```
pd.get_dummies(liver_df['Gender'], prefix = 'Gender').head()
```

	Gender_Female	Gender_Male
0	1	0
1	0	1
2	0	1
3	0	1
4	0	1

```
liver_df = pd.concat([liver_df,pd.get_dummies(liver_df['Gender'], prefix = 'Gender')], axis=1)
```

```
liver_df.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin
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 5.3.2.1: Converting "Gender" Categorical Variable to Indicator Variable

```
liver_df.describe()
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Album
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.0000
mean	44.746141	3.298799	1.486106	290.576329	80.713551	109.910806	6.483190	3.1418
std	16.189833	6.209522	2.808498	242.937989	182.620356	288.918529	1.085451	0.7955
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000	0.9000
25%	33.000000	0.800000	0.200000	175.500000	23.000000	25.000000	5.800000	2.6000
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000	3.1000
75%	58.000000	2.600000	1.300000	298.000000	60.500000	87.000000	7.200000	3.8000
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	9.600000	5.5000

5.3.3 Dropping irrelevant columns

```
# The input variables/features are all the inputs except Dataset.
# The prediction or label is 'Dataset' that determines whether the patient has liver disease or not.
X = liver_df.drop(['Gender','Dataset'], axis=1)
X.head(3)
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_
0	65	0.7	0.1	187	16	18	6.8	3.3	
1	62	10.9	5.5	699	64	100	7.5	3.2	
2	62	7.3	4.1	490	60	68	7.0	3.3	

Figure 5.3.3.1: Dropping Irrelevant Column

```
y = liver_df['Dataset'] # 1 for liver disease; 2 for no liver disease
```

5.3.4 Correlations

- Finally, let's take a look at the relationships between numeric features and other numeric features.
- Correlation is a value between -1 and 1 that represents how closely values for two separate features move in unison.
- Positive correlation means that as one feature increases, the other increases; e.g. a child's age and her height.
- Negative correlation means that as one feature increases, the other decreases; e.g. hours spent studying and number of parties attended.
- Correlations near -1 or 1 indicate a strong relationship.
- Those closer to 0 indicate a weak relationship.
- 0 indicates no relationship.

To get the correlation we have a method called corr() which gives the correlation between each and every column

```
# Correlation
liver_corr = X.corr()
```

```
liver_corr
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Tot
Age	1.000000	0.011763	0.007529	0.080425	-0.086883	-0.019910	
Total_Bilirubin	0.011763	1.000000	0.874618	0.206669	0.214065	0.237831	
Direct_Bilirubin	0.007529	0.874618	1.000000	0.234939	0.233894	0.257544	
Alkaline_Phosphotase	0.080425	0.206669	0.234939	1.000000	0.125680	0.167196	
Alamine_Aminotransferase	-0.086883	0.214065	0.233894	0.125680	1.000000	0.791966	
Aspartate_Aminotransferase	-0.019910	0.237831	0.257544	0.167196	0.791966	1.000000	
Total_Protiens	-0.187461	-0.008099	-0.000139	-0.028514	-0.042518	-0.025645	
Albumin	-0.265924	-0.222250	-0.228531	-0.165453	-0.029742	-0.085290	
Albumin_and_Globulin_Ratio	-0.216089	-0.206159	-0.200004	-0.233960	-0.002374	-0.070024	
Gender_Female	-0.056560	-0.089291	-0.100436	0.027496	-0.082332	-0.080336	
Gender_Male	0.056560	0.089291	0.100436	-0.027496	0.082332	0.080336	

```
plt.figure(figsize=(30, 30))
sns.heatmap(liver_corr, cbar = True,  square = True, annot=True, fmt= '.2f',annot_kws={'size': 15},
            cmap= 'coolwarm')
plt.title('Correlation between features');
```

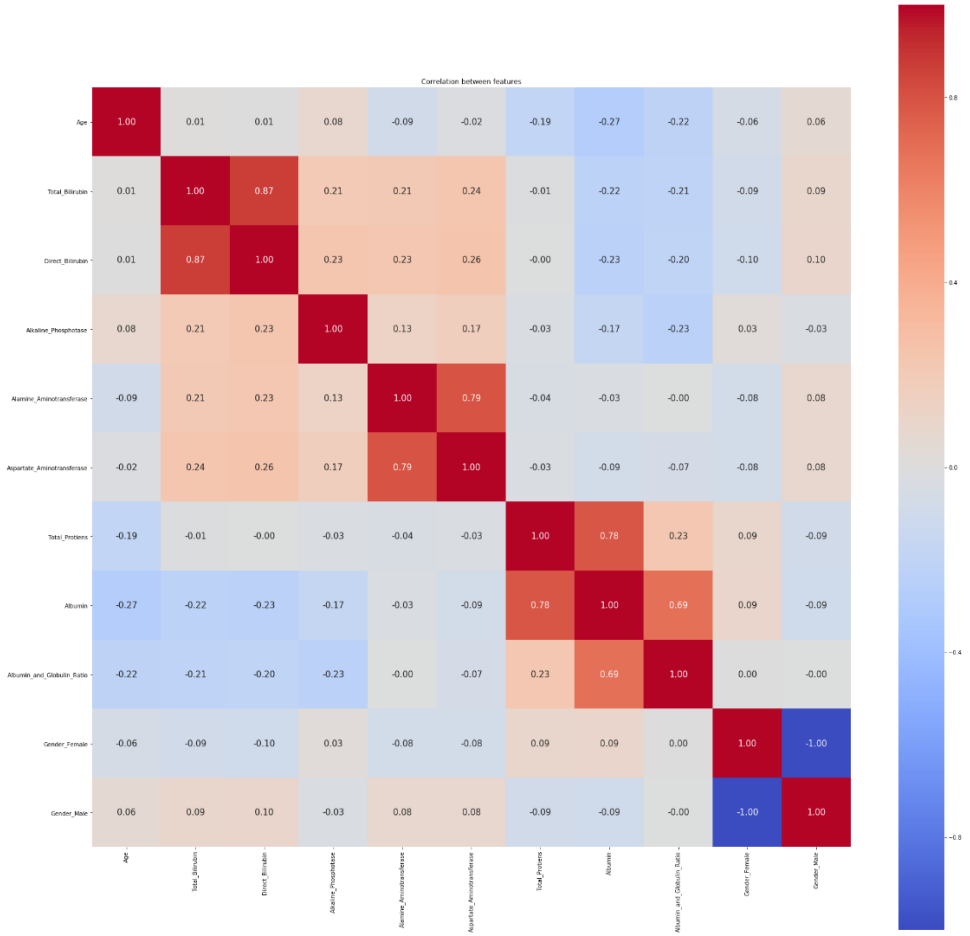


Figure 5.3.4.1: Correlation between Features

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 it's not as high as Total_Protiens & Albumin

5.3.5 Detection of Outliers

Outliers are extreme values that deviate from other observations on data, they may indicate a variability in a measurement, experimental errors or a novelty. In other words, an outlier is an observation that diverges from an overall pattern on a sample.

Types of outliers - Outliers can be of two kinds:

- univariate
- multivariate.

5.4 Pick a Model/Algorithm

In machine learning, Classification refers to a predictive modeling problem where a class label is predicted for a given example of input data.

- Binary classification refers to predicting one of two classes and multi-class classification involves predicting one of more than two classes.
- Multi-label classification involves predicting one or more classes for each example and imbalanced classification refers to classification tasks where the distribution of examples across the classes is not equal.

An easy to understand example is classifying emails as “spam” or “not spam.” From a modeling perspective, classification requires a training dataset with many examples of inputs and outputs from which to learn.

A model will use the training dataset and will calculate how to best map examples of input data to specific class labels. As such, the training dataset must be sufficiently representative of the problem and have many examples of each class label.

Class labels are often string values, e.g. “spam,” “not spam,” and must be mapped to numeric values before being provided to an algorithm for modeling. This is often referred to as label encoding, where a unique integer is assigned to each class label, e.g. “spam” = 0, “no spam” = 1.

There are many different types of classification algorithms for modeling classification predictive modeling problems.

Binary Classification

Binary classification refers to those classification tasks that have two class labels. Typically, binary classification tasks involve one class that is the normal state and another class that is the abnormal state.

Here “Lung disease not detected” is the normal state of a task that involves a medical test and “Lung disease detected” is the abnormal state.

The class for the normal state is assigned the class label 0 and the class with the abnormal state is assigned the class label 1.

Popular algorithms that can be used for binary classification include:

- Logistic Regression
- k-Nearest Neighbors
- Random Forest Classifier

So, we are going to use these 3 Algorithm to solve this problem.

```
df.columns
```

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

```
# Create separate object for target variable
y = df.Dataset

# Create separate object for input features
X = df.drop('Dataset', axis=1)
```

5.5 Training the Model

After choosing the model to use, we need to train the model. So feed the model with the data which we have collected. The goal of training is to answer a question or make a prediction correctly as often as possible.

- **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

```
# Importing modules
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

Figure 5.5.1: Importing Modules for Machine Algorithm Implementation

- This method splits the input and output data to train and test based on the percentage specified by the user and assigns them to four different variables (we need to mention the variables)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40, random_state=111)
print (X_train.shape)
print (y_train.shape)
print (X_test.shape)
print (y_test.shape)
```

```
(349, 11)
(349,)
(234, 11)
(234,)
```

Figure 5.5.8: Splitting Dataset into Training Data and Test Data

- Next step is to perform Data Standardization. In Data Standardization we perform zero mean centering and unit scaling; i.e. we make the mean of all the features as zero and the standard deviation as 1. Thus, we use mean and standard deviation of each feature.
- It is very important to save the mean and standard deviation for each of the feature from the training set, because we use the same mean and standard deviation in the test set.

5.5.1 Method 1 – Logistic Regression

- We need to import linear regression method from linear model package from scikit learn library
- We need to train the model based on our train set (that we have obtained from splitting)
- Then we have to test the model for the test set that is done as follows.
    - We have a method called predict, using this method we need to predict the output for input test set and we need to compare the out but with the output test data.
    - If the predicted values and the original values are close then we can say that model is trained with good accuracy

```
# Create Logistic regression object
logreg = LogisticRegression()
# Train the model using the training sets and check score
logreg.fit(X_train, y_train)
#Predict Output
log_predicted= logreg.predict(X_test)

logreg_score = round(logreg.score(X_train, y_train) * 100, 2)
logreg_score_test = round(logreg.score(X_test, y_test) * 100, 2)
logreg_acc = accuracy_score(y_test,log_predicted)
#Equation coefficient and Intercept
print('Logistic Regression Training Score: \n', logreg_score)
print('Logistic Regression Test Score: \n', logreg_score_test)
print('Coefficient: \n', logreg.coef_)
print('Intercept: \n', logreg.intercept_)
print('Accuracy: \n', logreg_acc)
print('Confusion Matrix: \n', confusion_matrix(y_test,log_predicted))
print('Classification Report: \n', classification_report(y_test,log_predicted))

sns.heatmap(confusion_matrix(y_test,log_predicted),annot=True,fmt="d")
```

Figure 5.5.1.1: Logistic Regression Algorithm Implementation

```
Logistic Regression Training Score:
 71.63
Logistic Regression Test Score:
 69.66
Coefficient:
 [[-1.38864503e-02  1.49987013e-01 -6.40184824e-01 -4.74249185e-04
  -1.25183763e-02 -1.87307568e-03 -7.22487424e-02  2.82957445e-01
   1.52715449e-02  2.51359045e-01 -4.67555325e-02]]
Intercept:
 [0.2155177]
Accuracy:
 0.6965811965811965
Confusion Matrix:
 [[153  12]
 [ 59  10]]
Classification Report:
              precision    recall  f1-score   support

           1       0.72      0.93      0.81       165
           2       0.45      0.14      0.22        69

    accuracy                           0.70       234
   macro avg       0.59      0.54      0.52       234
weighted avg       0.64      0.70      0.64       234
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d975596208>
```

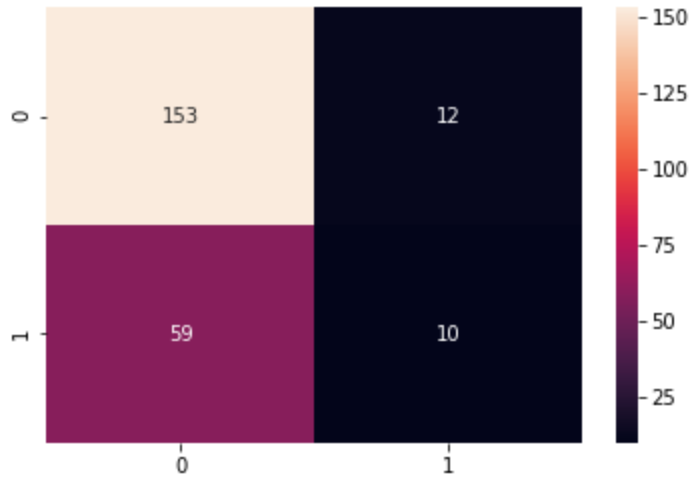


Figure 5.5.1.2: Logistic Regression - Scores, Accuracy, Confusion Matrix, Classification Report

```python
coeff_df = pd.DataFrame(X.columns)
coeff_df.columns = ['Feature']
coeff_df["Correlation"] = pd.Series(logreg.coef_[0])
pd.Series(logreg.coef_[0])

coeff_df.sort_values(by='Correlation', ascending=False)
```

	Feature	Correlation
7	Albumin	0.282957
9	Gender_Female	0.251359
1	Total_Bilirubin	0.149987
8	Albumin_and_Globulin_Ratio	0.015272
3	Alkaline_Phosphotase	-0.000474
5	Aspartate_Aminotransferase	-0.001873
4	Alamine_Aminotransferase	-0.012518
0	Age	-0.013886
10	Gender_Male	-0.046756
6	Total_Protiens	-0.072249
2	Direct_Bilirubin	-0.640185

5.5.2 Method 2 – Random Forest Classification

- First, start with the selection of random samples from a given dataset.
- Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree.
- In this step, voting will be performed for every predicted result.
- At last, select the most voted prediction result as the final prediction result.

```
# Create Random Forest object
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)

#Predict Output
rf_predicted = random_forest.predict(X_test)

random_forest_score = round(random_forest.score(X_train, y_train) * 100, 2)
random_forest_score_test = round(random_forest.score(X_test, y_test) * 100, 2)
random_forest_acc = accuracy_score(y_test,rf_predicted)

print('Random Forest Training Score: \n', random_forest_score)
print('Random Forest Test Score: \n', random_forest_score_test)
print('Accuracy: \n', random_forest_acc)
print('Confusion Matrix: \n',confusion_matrix(y_test,rf_predicted))
print('Classification Report: \n',classification_report(y_test,rf_predicted))

sns.heatmap(confusion_matrix(y_test,rf_predicted),annot=True,fmt="d")
```

Figure 5.5.2.1: Random Forest Classifier Algorithm Implementation

```
Random Forest Training Score:
 100.0
Random Forest Test Score:
 74.79
Accuracy:
 0.7478632478632479
Confusion Matrix:
 [[153  12]
 [ 47  22]]
Classification Report:
              precision    recall  f1-score   support

           1       0.77      0.93      0.84       165
           2       0.65      0.32      0.43        69

    accuracy                           0.75       234
   macro avg       0.71      0.62      0.63       234
weighted avg       0.73      0.75      0.72       234
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d975637a58>
```

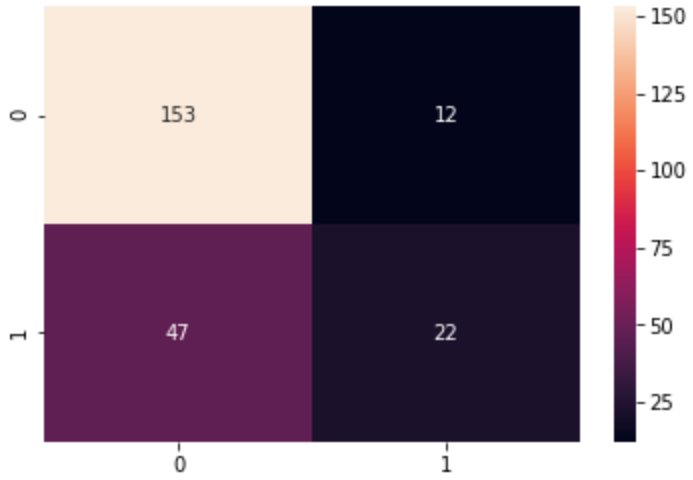


Figure 5.5.2.2: Random Forest - Scores, Accuracy, Confusion Matrix, Classification Report

After applying Hyperparameters:

```
grid_param = {
    'criterion': ['gini', 'entropy'],
    'n_estimators': range(1,200,10),
    'max_depth' : range(1,170,1),
    'min_samples_leaf' : range(1,50,1),
}
```

```
from sklearn.model_selection import RandomizedSearchCV
clf=RandomForestClassifier()
key = RandomizedSearchCV(estimator=clf,param_distributions=grid_param)
key.fit(X_train, y_train)
key.best_params_
```

```
{'n_estimators': 121,
 'min_samples_leaf': 47,
 'max_depth': 115,
 'criterion': 'entropy'}
```

```
clf = RandomForestClassifier(n_estimators=121,min_samples_leaf=47,max_depth= 115,criterion= 'entropy')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
accuracy_score(y_test,y_pred)
# f1_score(y_test,y_pred)
```

```
0.7051282051282052
```

Figure 5.5.2.3: Applying Hyperparameters on Random Forest Classifier

5.5.3 Method 3 – K-Nearest Neighbor Classification

- For implementing any algorithm, we need dataset. So, during the first step of KNN, we must load the training as well as test data.
- Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.
- For each point in the test data do the following −

- Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
- Now, based on the distance value, sort them in ascending order.
- Next, it will choose the top K rows from the sorted array.
- Now, it will assign a class to the test point based on most frequent class of these rows.

```
# KNN Model generation
from sklearn.neighbors import KNeighborsClassifier
# creating odd list of K for KNN
neighbors = list(range(1,20,2))
# empty list that will hold cv scores
cv_scores = []
from sklearn.model_selection import cross_val_score
#  10-fold cross validation , 9 datapoints will be considered for training and
# 1 for cross validation (turn by turn) to determine value of k
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=5, scoring='accuracy')
    cv_scores.append(scores.mean())

# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

knn_classifier = KNeighborsClassifier(n_neighbors = optimal_k)
knn_classifier.fit(X_train, y_train)

#Predict Output
knn_predicted = knn_classifier.predict(X_test)

knn_classifier_score = round(knn_classifier.score(X_train, y_train) * 100, 2)
knn_classifier_score_test = round(knn_classifier.score(X_test, y_test) * 100, 2)
knn_classifier_acc = accuracy_score(y_test,knn_predicted)

print('KNN Classifier Training Score: \n', knn_classifier_score)
print('KNN Classifier Test Score: \n', knn_classifier_score_test)
print('Accuracy: \n', knn_classifier_acc)
print('Confusion Matrix: \n',confusion_matrix(y_test,knn_predicted))
print('Classification Report:\n',classification_report(y_test,knn_predicted))

sns.heatmap(confusion_matrix(y_test,knn_predicted),annot=True,fmt="d")
```

Figure 5.5.3.1: KNN Classifier Algorithm Implementation

```
The optimal number of neighbors is 7.
KNN Classifier Training Score:
 75.36
KNN Classifier Test Score:
 73.08
Accuracy:
 0.7307692307692307
Confusion Matrix:
 [[150  15]
 [ 48  21]]
Classification Report:
              precision    recall  f1-score   support

           1       0.76      0.91      0.83       165
           2       0.58      0.30      0.40        69

    accuracy                           0.73       234
   macro avg       0.67      0.61      0.61       234
weighted avg       0.71      0.73      0.70       234
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d975179320>
```

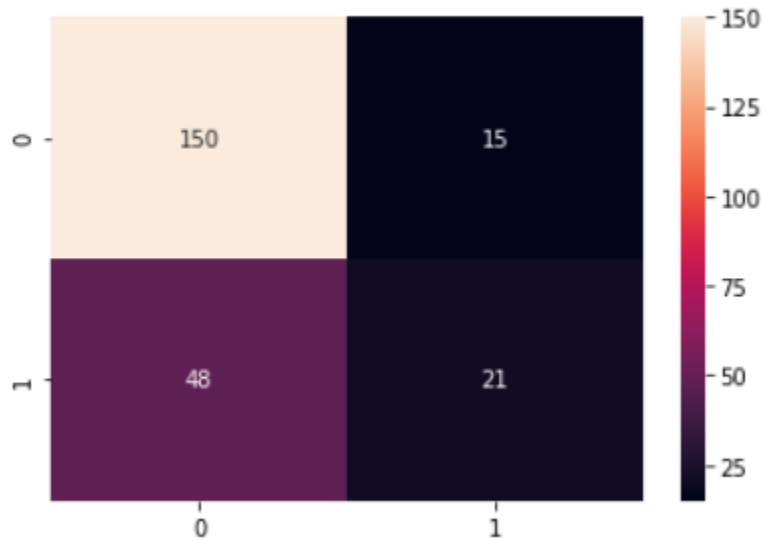


Figure 9.5.3.2: KNN Classifier - Scores, Accuracy, Confusion Matrix, Classification Report

5.6 Model Evaluation

In the Model Evaluation, we search for any trends, relations & correlations among the models used. We now rank our evaluation of all the models to choose the best one for our problem.

```
#We can now rank our evaluation of all the models to choose the best one for our problem.
models = pd.DataFrame({
    'Model': [ 'Logistic Regression', 'Random Forest Classifier', 'KNN Classifier'],
    'Training Score': [ logreg_score,  random_forest_score,knn_classifier_score],
    'Test Score': [ logreg_score_test, random_forest_score_test, knn_classifier_score_test],
    'Accuracy': [ logreg_acc, random_forest_acc, knn_classifier_acc]})
models.sort_values(by='Test Score', ascending=False)
```

	Model	Training Score	Test Score	Accuracy
1	Random Forest Classifier	100.00	74.79	0.747863
2	KNN Classifier	75.36	73.08	0.730769
0	Logistic Regression	71.63	69.66	0.696581

Figure 5.6.1: Model Evaluation - Logistic Regression, Random Forest Classifier, KNN Classifier

Our main goal going into this project was to predict liver disease using various machine learning techniques. We predicted using Logistic Regression, Random Forest Classifier and K-Nearest Neighbor (K-NN). All of them predicted with better results. With Each algorithm, we have observed Accuracy, Precision, Sensitivity and Specificity which can be defined as follows:

- Accuracy: The accuracy of a classifier is the percentage of the test set tuples that are correctly classified by the classifier.
- Sensitivity: Sensitivity is also referred as True positive rate i.e. the proportion of positive tuples that are correctly identified.
- Precision: precision is defined as the proportion of the true positives against all the positive results (both true positives and false positives)
- Specificity: Specificity is the True negative rate that is the proportion of negative tuples that are correctly identified

5.7 Feature Selection

```
linear = linear_model.LinearRegression()
# Train the model using the training sets and check score
linear.fit(X_train, y_train)
#Predict Output
lin_predicted = linear.predict(X_test)

linear_score = round(linear.score(X_train, y_train) * 100, 2)
linear_score_test = round(linear.score(X_test, y_test) * 100, 2)

#Equation coefficient and Intercept
print('Linear Regression Training Score: \n', linear_score)
print('Linear Regression Test Score: \n', linear_score_test)
print('Coefficient: \n', linear.coef_)
print('Intercept: \n', linear.intercept_)

from sklearn.feature_selection import RFE
rfe =RFE(linear, n_features_to_select=3)
rfe.fit(X,y)
```

```
Linear Regression Training Score:
 10.57
Linear Regression Test Score:
 10.52
Coefficient:
 [-2.37339280e-03  9.86352744e-03 -4.12675187e-02 -1.66881221e-04
 -3.78214852e-04  5.41378290e-05 -5.69342015e-02  1.51288967e-01
 -1.67674875e-01  4.11372174e-02 -4.11372174e-02]
Intercept:
 1.5625028576977165

RFE(estimator=LinearRegression(), n_features_to_select=3)
```

```
for i in range(len(rfe.ranking_)):
    if rfe.ranking_[i] == 1:
        print(X.columns.values[i])
```

```
Total_Protiens
Albumin
Gender_Male
```

```
#I'm considering seven important features based on recursive feature elimination
#finX = liver_df[['Age','Direct_Bilirubin','Total_Protiens','Albumin', 'Gender_Female', 'Gender_Male']]
finX = liver_df[['Total_Protiens','Albumin', 'Gender_Male']]
finX.head(4)
```

	Total_Protiens	Albumin	Gender_Male
0	6.8	3.3	0
1	7.5	3.2	1
2	7.0	3.3	1
3	6.8	3.4	1

```
X_train, X_test, y_train, y_test = train_test_split(finX, y, test_size=0.40, random_state=101)
```

Figure 5.7.1: Feature selection

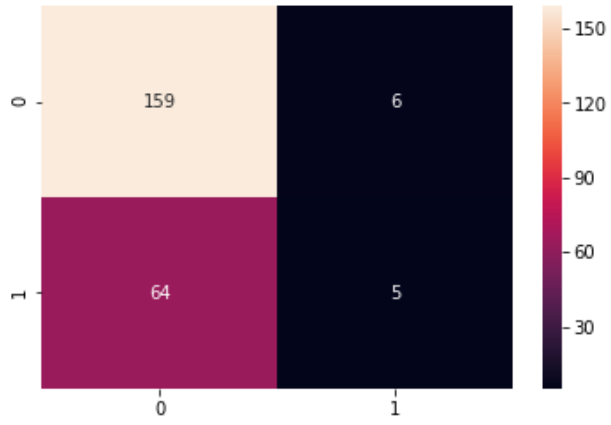
5.8 Drawing an inference

- In this step, we have reused the Logistic Regression and trained the model using the training sets and rechecked the score.

```
#Logistic Regression
logreg = LogisticRegression()
# Train the model using the training sets and check score
logreg.fit(X_train, y_train)
#Predict Output
log_predicted= logreg.predict(X_test)

logreg_score = round(logreg.score(X_train, y_train) * 100, 2)
logreg_score_test = round(logreg.score(X_test, y_test) * 100, 2)
#Equation coefficient and Intercept
print('Logistic Regression Training Score: \n', logreg_score)
print('Logistic Regression Test Score: \n', logreg_score_test)
print('Coefficient: \n', logreg.coef_)
print('Intercept: \n', logreg.intercept_)
print('Accuracy: \n', accuracy_score(y_test,log_predicted))
print('Confusion Matrix: \n', confusion_matrix(y_test,log_predicted))
print('Classification Report: \n', classification_report(y_test,log_predicted))

sns.heatmap(confusion_matrix(y_test,log_predicted),annot=True,fmt="d")

```

```
Logistic Regression Training Score:
 72.21
Logistic Regression Test Score:
 70.09
Coefficient:
 [[-0.65406998  1.1573853  -0.56380772]]
Intercept:
 [0.04339508]
Accuracy:
 0.7008547008547008
Confusion Matrix:
 [[159   6]
 [ 64   5]]
Classification Report:
              precision    recall  f1-score   support

           1       0.71      0.96      0.82       165
           2       0.45      0.07      0.12        69

    accuracy                           0.70       234
   macro avg       0.58      0.52      0.47       234
weighted avg       0.64      0.70      0.61       234
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x24906ad7518>
```



6 CONCLUSION

In this project, we have proposed methods for diagnosing liver disease in patients using Machine learning techniques. The three machine learning techniques that were used include Logistic Regression, Random Forest Classifier and KNN Classifier. The system was implemented using all the 3 models and their performance was evaluated.

Performance evaluation was based on performance metrics – Accuracy and Precision. The predictions using the Random Forest Classifier predicted with better results giving an accuracy of 75% when compared to other models. Comparing this work with the previous research works, it was discovered that Random Forest Classifier model proved highly efficient model.

References

- https://www.kaggle.com/uciml/indian-liver-patient-records
- https://intellipaat.com/blog/what-is-data-science/
- https://www.tutorialspoint.com/python_data_science/index.htm
- https://www.tutorialspoint.com/machine_learning/index.htm
- https://en.wikipedia.org/wiki/Python_(programming_language)#:~:text=Python%20was%20conceived%20in%20the%20late%201980s%20by%20Guido%20van,implementation%20began%20in%20December%201989.