COLLEGE PLACEMENT PREDICTION

A report submitted in partial fulfilment of the requirements for the Award of Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

BOBBEPALLI VEERANJANEYULU

Regd. No.: 20B91A0536

Under Supervision of Mr. Gundala Nagaraju
Henotic Technology Pvt Ltd, Hyderabad
(Duration: 7th July 2022 to 6th September 2022)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SAGI RAMA KRISHNAM RAJUENGINEERING COLLEGE

(An Autonomous Institution)

Approved by AICTE, NEW DELHI and Affiliated to JNTUK, Kakinada

CHINNA AMIRAM, BHIMAVARAM,

ANDHRA PRADESH

Table of Contents

Abstract

Placement of students is one of the most important objective of an educational institution. Reputation and yearly admissions of an institution invariably depend on the placements it provides it students with. That is why all the institutions, arduously, strive to strengthen their placement department so as to improve their institution on a whole. Any assistance in this particular area will have a positive impact on an institution's ability to place its students. This will always be helpful to both the students, as well as the institution. In this study, the objective is to analyse previous year's student's data and use it to predict the placement chance of the current students. This model is proposed with an algorithm to predict the same. Data pertaining to the study were collected form the same institution for which the placement prediction is done, and also suitable data pre-processing methods were applied. This proposed model is also compared with other traditional classification algorithms such as Decision tree and Random forest with respect to accuracy, precision and recall. From the results obtained it is found that the proposed algorithm performs significantly better in comparison with the other algorithms mentioned.

Keywords: Classification, Decision tree, Random forest

1.0 INTRODUCTION

Placements are considered to be very important for each and every college. The basic success of the college is measured by the campus placement of the students. Every student takes admission to the colleges by seeing the percentage of placements in the college. Hence, in this regard the approach is about the prediction and analyses for the placement necessity in the colleges that helps to build the colleges as well as students to improve their placements .

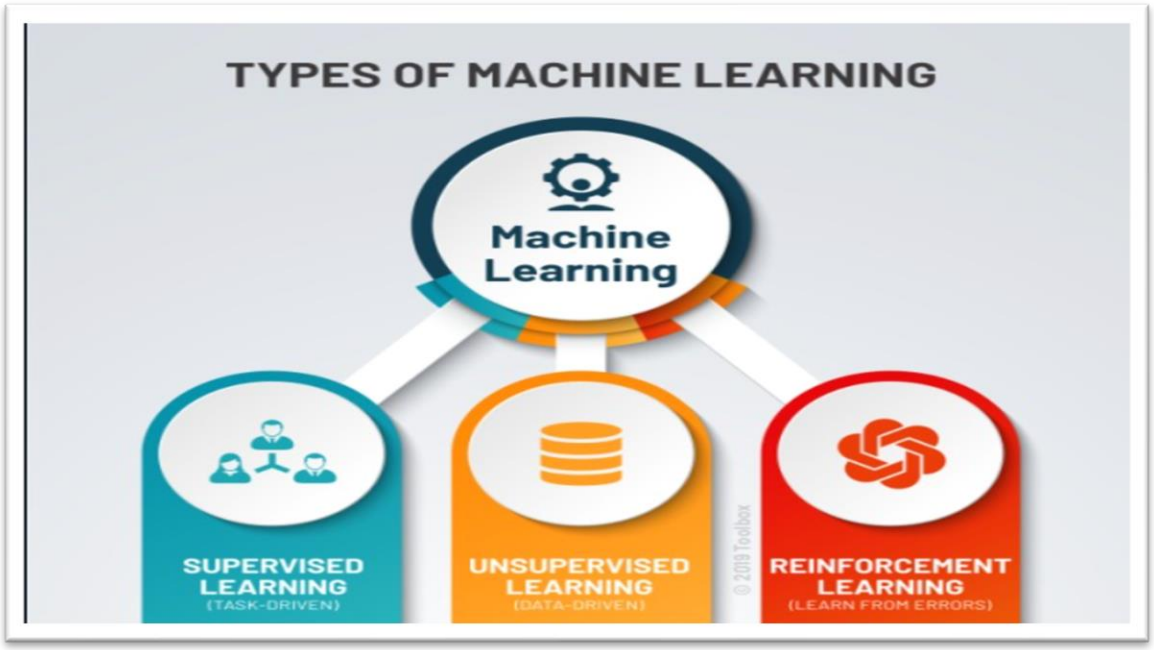
In Placement Prediction system predicts the probability of a undergrad students getting placed in a company by applying classification algorithms such as Decision tree and Random forest. The main objective of this model is to predict whether the student he/she gets placed or not in campus recruitment. For this the data consider is the academic history of student like overall percentage, backlogs, credits. The algorithms are applied on the previous years data of the students.

Oktariani Nurul Pratiwi used J48, Simple cart, kstar, SMO, NaiveBayes, OneR classification techniques on the data gathered from their high school. The results had shown that J48 and Simple Cart predicted well among them with an accuracy of 79.61% [5].
Ajay Kumar Pal and Saurabh Pal collected the data for the study and analysis of the student's educational performance basically for training and placement. The authors used different classification algorithm and used WEKA data mining tool [6]. They concluded that naive Bayes classification model is the better algorithm based on the placement data with found accuracy of 86.15% and overall time taken to build the model is at 0 sec. As compared with others Naïve Bayes classifier had lowest average error i.e. 0.28.
Ravi Tiwari and Awadhesh Kumar Sharma built the prediction model to improve the placement of the students [7]. They used WEKA as the data mining tool to build the model using random tree algorithm. They also used ID3, Bayes Net, RBF network, J48, algorithms on the student data set. They resolved that the RT (Random Tree) algorithm is more accurate with 73% for the classification/prediction of the model. The accuracy using ID3 and J48 is 71%. Bayes Net is 70%

1.1 What are the different types of Machine Learning?

As with any method, there are different ways to train machine learning algorithms, each with their own advantages and disadvantages. To understand the pros and cons of each type of machine learning, we must first look at what kind of data they ingest. In ML, there are two kinds of data — labelled data and unlabelled data.
Labelled data has both the input and output parameters in a completely machine-readable pattern but requires a lot of human labour to label the data, to begin with. Unlabelled data only has one or none of the parameters in a machine-readable form. This negates the need for human labour but requires more complex solutions.

There are also some types of machine learning algorithms that are used in very specific use-cases, but three main methods are used today.



Supervised Learning:

Supervised learning is one of the most basic types of machine learning. In this type, the machine learning algorithm is trained on labelled data. Even though the data needs to be labelled accurately for this method to work, supervised learning is extremely powerful when used in the right circumstances.

In supervised learning, the ML algorithm is given a small training dataset to work with. This training dataset is a smaller part of the bigger dataset and serves to give the algorithm a basic idea of the problem, solution, and data points to be dealt with. The training dataset is also very similar to the final dataset in its characteristics and provides the algorithm with the labelled parameters required for the problem.

The algorithm then finds relationships between the parameters given, essentially establishing a cause-and-effect relationship between the variables in the dataset. At the end of the training, the algorithm has an idea of how the data works and the relationship between the input and the output.This solution is then deployed for use with the final dataset, which it learns from in the

same way as the training dataset. This means that supervised machine learning algorithms will continue to improve even after being deployed, discovering new patterns and relationships as it trains itself on new data.

Categories of Supervised Machine Learning:

Supervised machine learning can be classified into two types of problems, which are given below:

- Classification
- Regression

✓ Classification

Classification algorithms are used to solve the classification problems in which the output variable is categorical, such as "Yes" or No, Male or Female, Red or Blue, etc. The classification algorithms predict the categories present in the dataset. Some real-world examples of classification algorithms are Spam Detection, Email filtering, etc.

Some popular classification algorithms are given below:

- Random Forest Algorithm
- Decision Tree Algorithm
- Logistic Regression Algorithm
- Support Vector Machine Algorithm

✓ Regression

Regression algorithms are used to solve regression problems in which there is a linear relationship between input and output variables. These are used to predict continuous output variables, such as market trends, weather prediction, etc.

Some popular Regression algorithms are given below:

- Simple Linear Regression Algorithm
- Multivariate Regression Algorithm
- Decision Tree Algorithm
- Lasso Regression

Unsupervised Learning:

Unsupervised machine learning holds the advantage of being able to work with unlabelled data. This means that human labour is not required to make the dataset machine-readable, allowing much larger datasets to be worked on by the program.

In supervised learning, the labels allow the algorithm to find the exact nature of the relationship between any two data points. However, unsupervised learning does not have labels to work off, resulting in the creation of hidden structures. Relationships between data points are perceived by the algorithm in an abstract manner, with no input required from human beings.The creation of these hidden structures is what makes unsupervised learning algorithms versatile. Instead of a defined and set problem statement, unsupervised learning algorithms can adapt to the data by dynamically changing hidden structures. This offers more post-deployment development than supervised learning algorithms.

Categories of Unsupervised Machine Learning:

Unsupervised Learning can be further classified into two types, which are given below:

- Clustering
- Association

✓ Clustering

The clustering technique is used when we want to find the inherent groups from the data. It is a way to group the objects into a cluster such that the objects with the most similarities remain in one group and have fewer or no similarities with the objects of other groups. An example of the clustering algorithm is grouping the customers by their purchasing behaviour.

Some of the popular clustering algorithms are given below:

- K-Means Clustering algorithm
- Mean-shift algorithm
- DBSCAN Algorithm
- Principal Component Analysis
- Independent Component Analysis

✓ Association

Association rule learning is an unsupervised learning technique, which finds interesting relations among variables within a large dataset. The main aim of this learning algorithm is to find the dependency of one data item on another data item and map those variables accordingly so that it can generate maximum profit. This algorithm is mainly applied in Market Basket analysis, Web usage mining, continuous production, etc.

Some popular algorithms of Association rule learning are:

- Apriori Algorithm
- Eclat
- FP-growth algorithm.

Applications of Unsupervised Learning:

- Network Analysis
- Recommendation Systems.
- Anomaly Detection
- Singular Value Decomposition

1.2 Benefits of Using Machine Learning in college placement prediction:

All students dream to obtain a job offer in their hands before they leave their college. A placement chance predictor helps students to have an idea about where they stand and what to be done to obtain a good placement.

The colleges can predict how many students are going to be placed and who are needed to be trained more. The students can evaluate themselves about their suitable job role which make organizations easy to give job role to the students.

Advantages of Machine Learning:

1. Automation of Everything:

Continuous Improvement:

Machine Learning algorithms are capable of learning from the data we provide. As new data is provided, the model's accuracy and efficiency to make decisions improve with subsequent training. Giants like Amazon, Walmart, college placement prediction etc collect a huge volume of new data every day. The accuracy of finding associated products or recommendation engine improves with this huge amount of training data available.

Automation for everything:

A very powerful utility of Machine Learning is its ability to automate various decision-making tasks. This frees up a lot of time for developers to use their time to more productive use. For example, some common use we see in our daily life is social media sentiment analysis and chatbots. The moment a negative tweet is made related to a product or service of a Company, a chatbot instantly replies as first-level customer support. Machine Learning is changing the world with its automation for almost everything we can think of.

Trends and patterns identification :

This advantage is a no brainer. All of us interested in Machine Learning technology are well aware of how the various Supervised, Unsupervised and Reinforced learning algorithms can be used for various classification and regression problems. We identify various trends and patterns with a huge amount of data using this technology. For example, Amazon analyzes the buying patterns and search trends of its customers and predicts products for them using Machine Learning algorithms.

Wide range of applications:

Machine Learning is used in every industry these days, for example from Defence to Education. Companies generate profits, cut costs, automate, predict the future, analyze trends and patterns from the past data, and many more. Applications like GPS Tracking for traffic, Email spam filtering, text prediction, spell check and correction, etc are a few used widely these days. Machine Learning is a branch of Artificial Intelligence, the latest trends and applications can be found in Artificial Intelligence Trends in 2020.

1.3 About Industry (COLLEGE PLACEMENT PREDICTION)

When it comes to business intelligence, AI can play a massive role. AI can introduce predictive features to this very industry. The AI-powered apps and tools are allowing business intelligence experts to make smarter decisions. It also enables businesses to use their data more effectively and insightfully to have improved results.

AI can help conquer several internal challenges to improve the manufacturing industry. AI can allow full automation, integration, decision-making, and channelizing info effortlessly and effectively. AI can transform manufacturing operations and

proceedings completely. According to a rough estimate by the experts, AI can help raise production by as much as 40% by the year 2035.

1.4. AI / ML Role in College Placement Prediction:

The scope of Artificial Intelligence in India is promising. Artificial Intelligence has immense potential to change each sector of the economy for the benefit of society. There is not just one technology under AI, but there are various useful technologies such as self-improving algorithms, machine learning, big data, pattern recognition. Soon, there would hardly be any industry or sector which would be untouched by this powerful tool in India. This is the reason why there has been an increasing demand for Artificial Intelligence online courses in India.

1. Education - Artificial Intelligence can help increase the effectiveness of our instructors via numerous AI applications such as text translation systems, real-time message to speech, automating mundane and also repeated jobs such as taking presence, automating grading, customising the discovering trip based upon ability, understanding, and also experience. The scope of Artificial Intelligence education and learning takes a look at the possibility of utilising AI ran rating machines that can evaluate unbiased solutions. This is something that is gradually being implemented in institutes. The various other applications of AI in the area of education are real-time text to speech as well as text translation systems.

2. Chatbots : In a country as varied as India, the combination of chatbots in the digital framework or availability via the IVRS system education domain can be transformational-they might be educated on the subject matter and a great percentage of doubts of the pupils could be responded to quickly, consequently lowering the current work of educators who could focus on even more imaginative tasks.

3. Automated grading : On a large scale, Machine Learning methods such as Natural Language Processing might be made use of for automated grading of assessments on systems such as E-PATHSHALA, SWAYAM (Study Webs of Active Learning for Young Aspiring Minds) and DIKSHA - not simply subjective ones however objective inquiries too. This is due to draft of National Education Policy 2019, prioritising on the internet understanding

Internship Project - Data Link

The internship project data has taken from Kaggle and the link is https://www.kaggle.com/datasets/tejashvi14/engineering-placements-prediction

3.0 AI / ML Modelling and Results

3.1 Your Problem of Statement

This dataset corresponds to an augmented version of the "Electrical Grid Stability Simulated Dataset", created by Vadim Arzamasov (Karlsruher Institute für Technologies, Karlsruhe, Germany) and donated to the University of California (UCI) Machine Learning Repository.Smart Grids represent a vision for the future of power distribution in which grid stability and reliability are enhanced through reconfigurable control schemes operating across a wide range of temporal and spatial scales. "Smart" grids use a network of computers to control the grid's generation, load, and distribution assets. These computers communicate over large-scale communication networks. These computer networks provide tighter and faster coordination of geographically distributed grid components in that *hopefully* enhances overall grid reliability.

- To predict the best smart grid using machine learning algorithms. Compare different algorithms and obtain the best one among them.

3.2 Data Science Project Life Cycle:

What is a Data Science Project Lifecycle?

In simple terms, a data science life cycle is nothing but a repetitive set of steps that you need to take to complete and deliver a project/product to your client. Although the data science

projects and the teams involved in deploying and developing the model will be different, every data science life cycle will be slightly different in every other company. However, most of the data science projects happen to follow a somewhat similar process.

In order to start and complete a data science-based project, we need to understand the various roles and responsibilities of the people involved in building, developing the project. Let us take a look at those employees who are involved in a typical data science project:

Who Are Involved in The Projects:

Business Analyst

Data Analyst

Data Scientists

Data Engineer

Data Architect

Machine Learning Engineer.

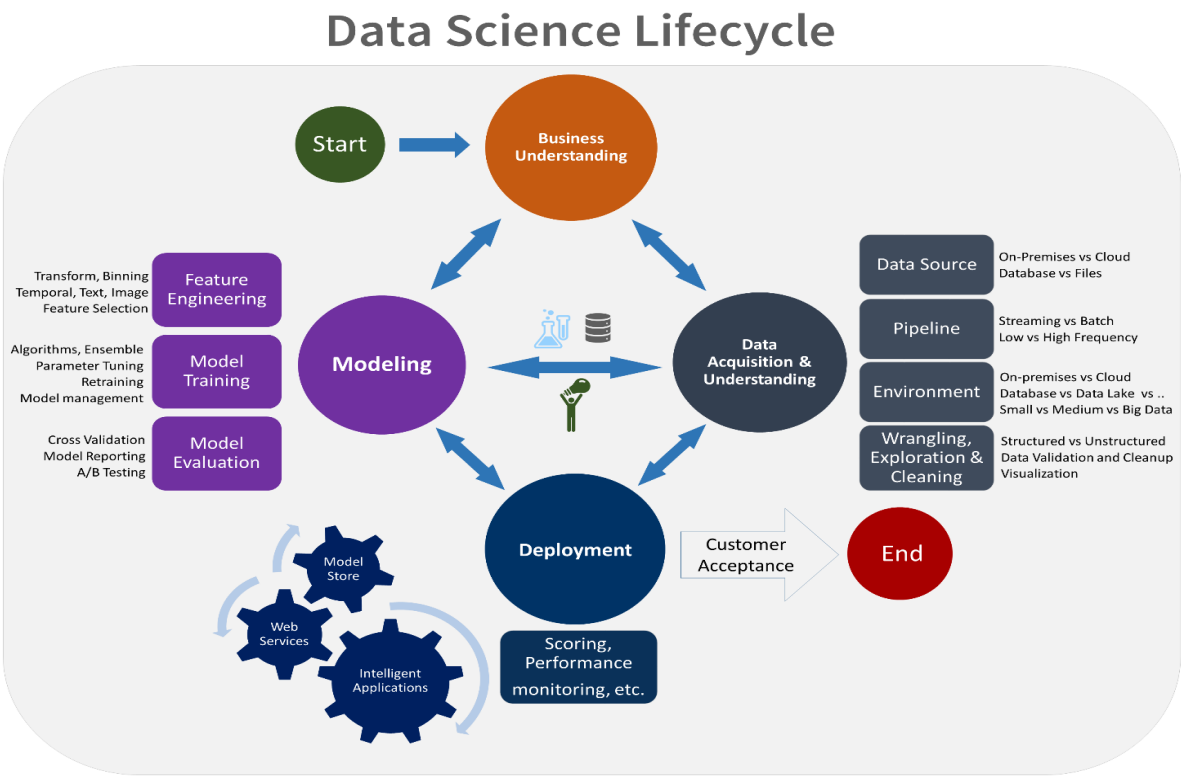
3.2.1 Data Exploratory Analysis:

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

3.2.2 Data Pre-processing:

Data preprocessing, a component of data preparation, describes any type of processing performed on raw data to prepare it for another data processing procedure. It has traditionally been an important preliminary step for the data mining process. More recently, data preprocessing techniques have been adapted for training machine learning models and AI models and for running inferences against them.

Data preprocessing transforms the data into a format that is more easily and effectively processed in data mining, machine learning and other data science tasks. The techniques are generally used at the earliest stages of the machine learning and AI.



3.2.2.1 Check the Duplicate and low variation data:

As companies collect more and more data about their customers, an increased amount of duplicate information starts appearing in the data as well, causing a lot of confusion among internal teams.Having duplicate data means that the model will be trained more on that type of data and thus will be biased. These duplicates add weight to the fit of those specific observations (cases). Whether this effect is big or small is hard to tell from the information we have provided. So that we checked the duplicates in our dataset as they are not required and also increase our computation time

3.2.2.2 Identify and address the missing variables

What are missing data

Missing data are values that are not recorded in a dataset. They can be a single value missing in a single cell or missing of an entire observation (row). Missing data can occur both in a continuous variable (e.g., height of students) or a categorical variable (e.g. gender of a population).

Missing data are common in any field of natural or behavioral science, but it is particularly commonplace in social sciences research data.

In programming languages, missing values are represented as NA or NaN or simply an empty cell in an object.

The origins of missing data

So where do the missing values come from, and why do they even exist?

Let's give an example. You are administering a questionnaire survey among a sample of respondents; and in the questionnaire, you are asking a question about household income. Now, what if a respondent refuses to answer that question? Would you make that up or rather leave the field empty? You'd probably leave that cell empty — creating an instance of missing value.

Problems caused

Missing values are definitely undesirable, but it is difficult to quantify the magnitude of effects in statistical and machine learning projects. If it's a large dataset and a very small percentage of data is missing the effect may not be detectable at all.

However, if the dataset is relatively small, every data point counts. In these situations, a missing data point means loss of valuable information.

In any case, generally missing data creates imbalanced observations, cause biased estimates, and in extreme cases, can even lead to invalid conclusion

How to deal with missing data?

f inevitable to avoid the following are some other common ways to deal with missing data:

Case deletion: if the dataset is relatively large delete the complete record with a missing value

Substitution: substitute missing cells with (a) column mean, (b) mean of nearest neighbors, (c) moving average, or (c) filling with the last observation

Statistical imputation: a regression can be an effective way to determine the value of missing cell given other information in the dataset

Sensitivity analysis: if the sample is small or missing values are relatively large then conduct a sensitivity analysis with multiple variations of outcomes

3.2.2.3 Identify objects and convert into numericals values

The datasets have both numerical and categorical features. Categorical features refer to string data types and can be easily understood by human beings. However, machines cannot interpret the categorical data directly. Therefore, the categorical data must be converted into numerical data for further processing.

There are many ways to convert categorical data into numerical data. Here in this article, we'll be discussing the two most used methods namely:

- Dummy Variable Encoding
- Label Encoding

Method 1: Dummy Variable Encoding

We will be using pandas.get_dummies function to convert the categorical string data into numeric.

Stepwise Implementation

Step 1: Importing Libraries

Step 2: Importing Data

Step 3: Converting Categorical Data Columns to Numerical.

Method 2: Label Encoding
We will be using. LabelEncoder() from sklearn library to convert categorical data to numerical data. We will use function fit_transform() in the process.

Stepwise Implementation

Step 1: Importing Libraries

Step 2: Importing Data

Step 3: Converting Categorical Data Columns to Numerical.

3.2.2.4. Handling of Outliers:

What is an outlier?

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population.

There is, of course, a degree of ambiguity. Qualifying a data point as an anomaly leaves it up to the analyst or model to determine what is abnormal—and what to do with such data points.

There are also different degrees of outliers:

- Mild outliers lie beyond an “inner fence” on either side.
- Extreme outliers are beyond an “outer fence.”

Why do outliers occur? According to Tom Bodenberg, chief economist, and data consultant at Unity Marketing, “It can be the result of measurement or recording errors, or the unintended and truthful outcome resulting from the set’s definition.”

Outliers may contain valuable information. Or be meaningless aberrations caused by measurement and recording errors. In any case, they can cause problems with repeatable A/B test results, so it’s important to question and analyze outliers.

3.2.2.5. Categorical data and Encoding Techniques

- What is Categorical Data?

Since we are going to be working on categorical variables in this article, here is a quick refresher on the same with a couple of examples. Categorical variables are usually represented as ‘strings’ or ‘categories’ and are finite in number. Here are a few examples:

1. The city where a person lives: Delhi, Mumbai, Ahmedabad, Bangalore, etc.
2. The department a person works in: Finance, Human resources, IT, Production.
3. The highest degree a person has: High school, Diploma, Bachelors, Masters, PhD.
4. The grades of a student: A+, A, B+, B, B- etc.

In the above examples, the variables only have definite possible values. Further, we can see there are two kinds of categorical data-

- Ordinal Data: The categories have an inherent order
- Nominal Data: The categories do not have an inherent order

- Label Encoding:

- We use this categorical data encoding technique when the categorical feature is ordinal. In this case, retaining the order is important. Hence encoding should reflect the sequence.

- In Label encoding, each label is converted into an integer value. We will create a variable that contains the categories representing the education qualification of a person.

- Binary Encoding:

- Binary encoding is a combination of Hash encoding and one-hot encoding. In this encoding scheme, the categorical feature is first converted into numerical using an ordinal encoder. Then the numbers are transformed in the binary number. After that binary value is split into different columns.

- Binary encoding works really well when there are a high number of categories. For example the cities in a country where a company supplies its products

3.2.2.6. Feature Scaling:

Why Feature Scaling?

Real Life Datasets have many features with a wide range of values like for example let's consider the house price prediction dataset. It will have many features like no. of. bedrooms, square feet area of the house, etc.

As you can guess, the no. of bedrooms will vary between 1 and 5, but the square feet area will range from 500-2000. This is a huge difference in the range of both features.

Many machine learning algorithms that are using Euclidean distance as a metric to calculate the similarities will fail to give a reasonable recognition to the smaller feature, in this case, the number of bedrooms, which in the real case can turn out to be an actually important metric.

E.g.: Linear Regression, Logistic Regression, KNN

There are several ways to do feature scaling. I will be discussing the top 5 of the most commonly used feature scaling techniques.

3.2.3 Selection of Dependent and Independent variables

The dependent or target variable here is satisfaction Target which tells us a

The independent variables are selected after doing exploratory data analysis. Which tells us whether the Customer is satisfied, neutral or dissatisfied.

3.2.4 Data Sampling Methods:

The data we have is balanced data so we no need for sampling method**s**

3.2.5 Models Used for Development

We built our predictive models by using the following ten algorithms

3.2.5.1 Logistic Regression:

Logistic uses logit link function to convert the likelihood values to probabilities so we can get a good estimate on the probability of a particular observation to be positive class or negative class. The also gives us p-value of the variables which tells us about significance of each independent variable.

3.2.5.2 Decision Tree Classifier:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed on the basis of features of the given dataset.

It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

3.2.5.3 Random Forest Classifier:

Random forest is an algorithm that consists of many decision trees. It was first developedby Leo Breiman and Adele Cutler. The idea behind it is to build several trees,to have the instance classified by each tree, and to give a "vote" at each class.The model uses a "bagging" approach and the random selection of features to build acollection of decision trees with controlled variance. The instance's class is to the classwith the highest number of votes, the class that occurs the most within the leaf in whichthe instance is placed.

The error of the forest depends on:

- Trees correlation: the higher the correlation, the higher the forest error rate.
- The strength of each tree in the forest. A strong tree is a tree with low error. By using trees that classify the instances with low error the error rate of the forest

3.2.5.4 Extra Tree Classifier:

This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting

3.2.5.5 KNN Classifier:

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

3.2.5.6 Gaussian Naive Bayes

Naïve Bayes is a probabilistic machine learning algorithm used for many classification functions and is based on the Bayes theorem. Gaussian Naïve Bayes is the extension of naïve Bayes. While other functions are used to estimate data distribution, Gaussian or normal distribution is the simplest to implement as you will need to calculate the mean and standard deviation for the training data.

What is the Naive Bayes Algorithm? Naive Bayes is a probabilistic machine learning algorithm that can be used in several classification tasks. Typical applications of Naive Bayes are classification of documents, filtering spam, prediction and so on. This algorithm is based on the discoveries of Thomas Bayes and hence its name.

The name "Naïve" is used because the algorithm incorporates features in its model that are independent of each other. Any modifications in the value of one feature do not directly impact the value of any other feature of the algorithm. The main advantage of the Naïve Bayes algorithm is that it is a simple yet powerful algorithm.

It is based on the probabilistic model where the algorithm can be coded easily, and predictions did quickly in real-time. Hence this algorithm is the typical choice to solve real-world problems as it can be tuned to respond to user requests instantly. But before we dive deep into Naïve Bayes and Gaussian Naïve Bayes, we must know what is meant by conditional probability.

3.2.5.7 XGB classifier

XGBoost is an implementation of Gradient Boosted decision trees. This library was written in C++. It is a type of Software library that was designed basically to improve speed

and model performance. It has recently been dominating in applied machine learning. XGBoost models majorly dominate in many Kaggle Competitions. In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost.

Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and the variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

3.2.5.8. LightGBM classifier:

LightGBM is a gradient boosting framework based on decision trees to increases the efficiency of the model and reduces memory usage.

It uses two novel techniques: Gradient-based One Side Sampling and Exclusive Feature Bundling (EFB) which fulfills the limitations of histogram-based algorithm that is primarily used in all GBDT (Gradient Boosting Decision Tree) frameworks. The two techniques of GOSS and EFB described below form the characteristics of LightGBM Algorithm. They comprise together to make the model work efficiently and provide it a cutting edge over other GBDT frameworks

Gradient-based One Side Sampling Technique for LightGBM:

Different data instances have varied roles in the computation of information gain. The instances with larger gradients (i.e., under-trained instances) will contribute more to the information gain.

GOSS keeps those instances with large gradients (e.g., larger than a predefined threshold, or among the top percentiles), and only randomly drop those instances with small gradients to retain the accuracy of information gain estimation. This treatment can lead to a more accurate gain estimation than uniformly random sampling, with the same target sampling rate, especially when the value of information gain has a large range.

3.2.5.9. Bagging classifier:

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a

way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

Each base classifier is trained in parallel with a training set which is generated by randomly drawing, with replacement, N examples(or data) from the original training dataset – where N is the size of the original training set.

Training set for each of the base classifiers is independent of each other. Many of the original data may be repeated in the resulting training set while others may be left out.

Bagging reduces overfitting (variance) by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance though.

3.2.5.10. SVM classifier:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

3.3. AI / ML ModelsAnalysis and Final Results

We used our train dataset to build the above models and used our test data to check the accuracy and performance of our models.

We used confusion matrix to check accuracy, Precision, Recall and F1 score of our models and compare and select the best model.

3.3.1. Logistic regression code:

•The Python code for models with stratified sampling technique as follows:

```
# To build the decision tree model with random sampling
from sklearn.linear_model import LogisticRegression
# Create an object for LR mode
modelLR = LogisticRegression()
```

```
# Train the model with training data
modelLR = modelLR.fit(x_train,y_train)
# Predict the model with test data set

y_pred = modelLR.predict(x_test)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# actual values
actual = y_test
# predicted values
predicted = y_pred
# confusion matrix
matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,
normalize=None)
print('Confusion matrix : \n', matrix)
# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)
# classification report for precision, recall f1-score and accuracy
C_Report = classification_report(actual,predicted,labels=[1,0])
print('Classification report : \n', C_Report)
# calculating the metrics
sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);
precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);
# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1 to
+1.
# A model with a score of +1 is a perfect model and -1 is a poor model
from math import sqrt
mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)
```

```
print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)
# Area under ROC curve
from sklearn.metrics import roc_curve, roc_auc_score
print('roc_auc_score:', round(roc_auc_score(y_test, y_pred), 3))
# ROC Curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test,modelLR.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
#plt.savefig('Log_ROC')
plt.show()
print('-------------------------------------------------------------------------------------------------
-----')
   new_row = {'Model Name' : modelLR,
  'True_Positive': tp,
  'False_Negative': fn,
  'False_Positive': fp,
```

```
        'True_Negative': tn,
        'Accuracy' : accuracy,
        'Precision' : precision,
        'Recall' : sensitivity,
        'F1 Score' : f1Score,
        'Specificity' : specificity,
        'MCC':MCC,
        'ROC_AUC_Score':roc_auc_score(y_test, y_pred),
        'Balanced Accuracy':balanced_accuracy}
SGresults = SGresults.append(new_row, ignore_index=True)
```

3.3.2. Extra tree Python Code

•The Python code for models with stratified sampling technique as follows:

```
from sklearn.ensemble import ExtraTreesClassifier
ModelET = ExtraTreesClassifier()
ModelET.fit(x_train, y_train)
    # Prediction the model with test dataset
    y_pred = ModelET.predict(x_test)
    y_pred_prob = ModelET.predict_proba(x_test)
    # confusion matrix in sklearn
    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import classification_report
    # actual values
    actual = y_test

# predicted values
```

```
    predicted = y_pred
    # confusion matrix
    matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None, normalize=None)
    print('Confusion matrix : \n', matrix)
    # outcome values order in sklearn
    tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
    print('Outcome values : \n', tp, fn, fp, tn)
    # classification report for precision, recall f1-score and accuracy
    C_Report = classification_report(actual,predicted,labels=[1,0])
    print('Classification report : \n', C_Report)
    # calculating the metrics
    sensitivity = round(tp/(tp+fn), 3);
    specificity = round(tn/(tn+fp), 3);
    accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
    balanced_accuracy = round((sensitivity+specificity)/2, 3);
    precision = round(tp/(tp+fp), 3);
    f1Score = round((2*tp/(2*tp + fp + fn)), 3);
    # Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1 to +1.
    # A model with a score of +1 is a perfect model and -1 is a poor model
    #from math import sqrt
    #mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
    #MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)
    print('Accuracy :', round(accuracy*100, 2),'%')
```

```
print('Precision :', round(precision*100, 2),'%')

print('Recall :', round(sensitivity*100,2), '%')

print('F1 Score :', f1Score)

print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )

print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')

#print('MCC :', MCC)

# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(actual, y_pred), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_curve

Model_roc_auc = roc_auc_score(actual, y_pred)

fpr, tpr, thresholds = roc_curve(actual, models.predict_proba(x_test)[:,1])

plt.figure()

#

plt.plot(fpr, tpr, label= 'Classification Model' % Model_roc_auc)

plt.plot([0, 1], [0, 1],'r--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic')

plt.legend(loc="lower right")
```

```
plt.savefig('Log_ROC')
plt.show()
new_row = {'Model Name' : models,
        'True Positive': tp,
          'False Negative': fn,
        'False Positive': fp,
        'True Negative': tn,
        'Accuracy' : accuracy,
        'Precision' : precision,
        'Recall' : sensitivity,
        'F1 Score' : f1Score,
        'Specificity' : specificity,
        'MCC': 'MCC',
        'ROC_AUC_Score':roc_auc_score(actual, y_pred),
        'Balanced Accuracy':balanced_accuracy}
        SGresults = SGresults.append(new_row, ignore_
          index=True)
```

3.3.3. Decision Tree Classifier Python code

```
# To build the 'Decision Tree' model with random sampling
from sklearn.tree import DecisionTreeClassifier
modelDT = DecisionTreeClassifier()
# Train the model with train data
modelDT.fit(x_train,y_train)
# Predict the model with test data set
```

```
y_pred = modelDT.predict(x_test)
y_pred_prob = modelDT.predict_proba(x_test)
# Confusion matrix in sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# actual values
actual = y_test
# predicted values
predicted = y_pred
# confusion matrix
matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,
normalize=None)
print('Confusion matrix : \n', matrix)
# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)
# classification report for precision, recall f1-score and accuracy
C_Report = classification_report(actual,predicted,labels=[1,0]
print('Classification report : \n', C_Report)
# calculating the metrics
sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);
precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);
# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1 to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model
from math import sqrt
mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)

print('Accuracy :', round(accuracy*100, 2),'%')
```

```
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)

# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(y_test, y_pred), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test, modelDT.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
print('----------------------------------------------------------------------------------------------------')
new_row = {'Model Name' : modelDT,
        'True_Positive': tp,
        'False_Negative': fn,
```

```
        'False_Positive': fp,
        'True_Negative': tn,

       'Accuracy' : accuracy,
        'Precision' : precision,
        'Recall' : sensitivity,
        'F1 Score' : f1Score,
        'Specificity' : specificity,
        'MCC':MCC,
        'ROC_AUC_Score':roc_auc_score(y_test, y_pred),
        'Balanced Accuracy':balanced_accuracy}
SGresults = SGresults.append(new_row, ignore_index=True)
```

3.3.4. Random Forest Classifier Python Code

The Python code for models with stratified sampling technique as follows:

```
# To build the 'Multinominal Decision Tree' model with random sampling
from sklearn.ensemble import RandomForestClassifier
ModelRF=RandomForestClassifier()
# Train the model with train data
ModelRF.fit(x_train,y_train)
# Predict the model with test data set
y_pred = ModelRF.predict(x_test)
y_pred_prob = ModelRF.predict_proba(x_test)
# Confusion matrix in sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# actual values
actual = y_test
```

```
# predicted values

predicted = y_pred

# confusion matrix

matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None, normalize=None)

print('Confusion matrix : \n', matrix)

# outcome values order in sklearn

tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)

print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy

C_Report = classification_report(actual,predicted,labels=[1,0])

print('Classification report : \n', C_Report)

# calculating the metrics


sensitivity = round(tp/(tp+fn), 3);

specificity = round(tn/(tn+fp), 3);

accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);

balanced_accuracy = round((sensitivity+specificity)/2, 3);

precision = round(tp/(tp+fp), 3);

f1Score = round((2*tp/(2*tp + fp + fn)), 3);

# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1 to +1.

# A model with a score of +1 is a perfect model and -1 is a poor model

from math import sqrt

mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
```

```
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)

print('Accuracy :', round(accuracy*100, 2),'%')

print('Precision :', round(precision*100, 2),'%')

print('Recall :', round(sensitivity*100,2), '%')

print('F1 Score :', f1Score)

print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )

print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')

print('MCC :', MCC)

# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(y_test, y_pred), 3))

# ROC Curve


from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_curve

logit_roc_auc = roc_auc_score(y_test, y_pred)

fpr, tpr, thresholds = roc_curve(y_test,ModelRF.predict_proba(x_test)[:,1])

plt.figure()

#-----------------------------------------------------------------

plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)

plt.plot([0, 1], [0, 1],'r--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
 new_row = {'Model Name' : models,
            'True_Positive' : tp,
            'False_Negative' : fn,
            'False_Positive' : fp,
            'True_Negative' : tn,
            'Accuracy' : accuracy,
            'Precision' : precision,
             'Recall' : sensitivity,
            'F1 Score' : f1Score,
            'Specificity' : specificity,
            'MCC':MCC,
            'ROC_AUC_Score':roc_auc_score(y_test, y_pred),
            'Balanced Accuracy':balanced_accuracy}
            SGresults = SGresults.append(new_row, ignore_
               index=True)
```

3.3.5. KNeighborsClassifier Python code

 # To build the 'Logistic Regression' model with random sampling

```
from sklearn.neighbors import KNeighborsClassifier
# Create model object
ModelKNN = KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto',
leaf_size=30,
                    p=2, metric='minkowski', metric_params=None, n_jobs=None)
# Fit the model
ModelKNN.fit(x_train, y_train)
# Predict the model with test data set
y_pred = ModelKNN.predict(x_test)
# confusion matrix in sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# actual values
actual = y_test
# predicted values
predicted = y_pred
# confusion matrix
matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,
normalize=None)
print('Confusion matrix : \n', matrix)
# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)
# classification report for precision, recall f1-score and accuracy
C_Report = classification_report(actual,predicted,labels=[1,0])
print('Classification report : \n', C_Report)
# calculating the metrics
sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);
precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);
# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1 to +1.
```

```
# A model with a score of +1 is a perfect model and -1 is a poor model
from math import sqrt
mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)
print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)
# Area under ROC curve
from sklearn.metrics import roc_curve, roc_auc_score
print('roc_auc_score:', round(roc_auc_score(y_test, y_pred), 3))
# ROC Curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test,ModelKNN.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
print('---------------------------------------------------------------------------------------------------')
#---------------------------------------------------------------------------------------------------
new_row = {'Model Name' : ModelKNN,
```

```
        'True_Positive': tp,
        'False_Negative': fn,
        'False_Positive': fp,
        'True_Negative': tn,
        'Accuracy' : accuracy,
        'Precision' : precision,
        'Recall' : sensitivity,
        'F1 Score' : f1Score,
        'Specificity' : specificity,
        'MCC':MCC,
        'ROC_AUC_Score':roc_auc_score(y_test, y_pred),
        'Balanced Accuracy':balanced_accuracy}
SGresults = SGresults.append(new_row, ignore_index=True)
```

3.3.6. GradientBoostingClassifier Python code

```
# Build the Calssification models and compare the results
from sklearn.ensemble import GradientBoostingClassifier
# Create objects of classification algorithm with default hyper-parameters
ModelGB = GradientBoostingClassifier(loss='deviance', learning_rate=0.1,
n_estimators=100, subsample=1.0,
                        criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1,
                        min_weight_fraction_leaf=0.0, max_depth=3,
min_impurity_decrease=0.0,
                         init=None, random_state=None,
                        max_features=None, verbose=0, max_leaf_nodes=None,
warm_start=False,
                        validation_fraction=0.1, n_iter_no_change=None, tol=0.0001,
ccp_alpha=0.0)
# Train the model with train data
ModelGB.fit(x_train,y_train)
# Predict the model with test data set
y_pred = ModelGB.predict(x_test)
```

```
y_pred_prob = ModelGB.predict_proba(x_test)
# Confusion matrix in sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# actual values
actual = y_test
# predicted values
predicted = y_pred
# confusion matrix
matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,
normalize=None)
print('Confusion matrix : \n', matrix)
# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)
# classification report for precision, recall f1-score and accuracy
C_Report = classification_report(actual,predicted,labels=[1,0])
print('Classification report : \n', C_Report)
# calculating the metrics
sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);
precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);
# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1 to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model
from math import sqrt
mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)
print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
```

```
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)
# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(y_test, y_pred), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test,ModelGB.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
print('---------------------------------------------------------------------------------------------------')
#---------------------------------------------------------------------------------------------------
new_row = {'Model Name' : ModelET,
        'True_Positive': tp,
        'False_Negative': fn,
        'False_Positive': fp,
        'True_Negative': tn,
```

```
        'Accuracy' : accuracy,
        'Precision' : precision,
        'Recall' : sensitivity,
        'F1 Score' : f1Score,
        'Specificity' : specificity,
        'MCC':MCC,
        'ROC_AUC_Score':roc_auc_score(y_test, y_pred),
        'Balanced Accuracy':balanced_accuracy}
SGresults = SGresults.append(new_row, ignore_index=True)
```

3.3.7. SVC classifier Python code

```
# Build the Calssification models and compare the results
from sklearn.svm import SVC
ModelSVM = SVC(probability=True)
# Train the model with train data
ModelGB.fit(x_train,y_train)
# Predict the model with test data set
y_pred = ModelGB.predict(x_test)
y_pred_prob = ModelGB.predict_proba(x_test)
# Confusion matrix in sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# actual values
actual = y_test
# predicted values
predicted = y_pred
# confusion matrix
matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,
normalize=None)
print('Confusion matrix : \n', matrix)
# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
```

```
print('Outcome values : \n', tp, fn, fp, tn)
# classification report for precision, recall f1-score and accuracy
C_Report = classification_report(actual,predicted,labels=[1,0])
print('Classification report : \n', C_Report)
# calculating the metrics
sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);
precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);
# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1 to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model
from math import sqrt
mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)
print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)
# Area under ROC curve
from sklearn.metrics import roc_curve, roc_auc_score
print('roc_auc_score:', round(roc_auc_score(y_test, y_pred), 3))
# ROC Curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test,ModelGB.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
```

```
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
print('-----------------------------------------------------------------------------------------------------')
#-----------------------------------------------------------------------------------------------------
new_row = {'Model Name' : ModelET,
        'True_Positive': tp,
        'False_Negative': fn,
        'False_Positive': fp,
        'True_Negative': tn,
        'Accuracy' : accuracy,
        'Precision' : precision,
        'Recall' : sensitivity,
        'F1 Score' : f1Score,
        'Specificity' : specificity,
        'MCC':MCC,
        'ROC_AUC_Score':roc_auc_score(y_test, y_pred),
        'Balanced Accuracy':balanced_accuracy}
SGresults = SGresults.append(new_row, ignore_index=True)
```

3.3.8. GaussianNB Classifier Python code

```
from sklearn.naive_bayes import GaussianNB
ModelGNB = GaussianNB()
# Train the model with train data

ModelET.fit(x_train,y_train)
```

```
# Predict the model with test data set
y_pred = ModelET.predict(x_test)
y_pred_prob = ModelET.predict_proba(x_test)
# Confusion matrix in sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# actual values
actual = y_test
# predicted values
predicted = y_pred
# confusion matrix
matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,
normalize=None)
print('Confusion matrix : \n', matrix)
# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)
# classification report for precision, recall f1-score and accuracy
C_Report = classification_report(actual,predicted,labels=[1,0])
print('Classification report : \n', C_Report)
# calculating the metrics
sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);
precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);
# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1 to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model
from math import sqrt
mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)
print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
```

```
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)
# Area under ROC curve
from sklearn.metrics import roc_curve, roc_auc_score
print('roc_auc_score:', round(roc_auc_score(y_test, y_pred), 3))
# ROC Curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test,ModelET.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
print('---------------------------------------------------------------------------------------------------')
#---------------------------------------------------------------------------------------------------
new_row = {'Model Name' : ModelET,
        'True_Positive': tp,
        'False_Negative': fn,
        'False_Positive': fp,
        'True_Negative': tn,
         'Accuracy' : accuracy,
        'Precision' : precision,
```

```
        'Recall' : sensitivity,
        'F1 Score' : f1Score,
        'Specificity' : specificity,
        'MCC':MCC,
        'ROC_AUC_Score':roc_auc_score(y_test, y_pred),
        'Balanced Accuracy':balanced_accuracy}
SGresults = SGresults.append(new_row, ignore_index=True)
```

3.3.9. BaggingClassifier Python code

```
from sklearn.ensemble import BaggingClassifier
modelBAG = BaggingClassifier(base_estimator=None, n_estimators=100,
max_samples=1.0, max_features=1.0,
                bootstrap=True, bootstrap_features=False, oob_score=False,
warm_start=False,
                n_jobs=None, random_state=None, verbose=0)
# Train the model with train data
modelBAG.fit(x_train,y_train)
# Predict the model with test data set
y_pred = modelBAG.predict(x_test)
y_pred_prob = modelBAG.predict_proba(x_test)
# Confusion matrix in sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# actual values
actual = y_test
# predicted values
predicted = y_pred
# confusion matrix

matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,
normalize=None)
print('Confusion matrix : \n', matrix)
```

```
# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)
# classification report for precision, recall f1-score and accuracy
C_Report = classification_report(actual,predicted,labels=[1,0])
print('Classification report : \n', C_Report)
# calculating the metrics
sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);
precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);
# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1 to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model
from math import sqrt
mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)
print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)
# Area under ROC curve
from sklearn.metrics import roc_curve, roc_auc_score
print('roc_auc_score:', round(roc_auc_score(y_test, y_pred), 3))
# ROC Curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test,modelBAG.predict_proba(x_test)[:,1])
plt.figure()
```

```
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
print('---------------------------------------------------------------------------------------------------')
#---------------------------------------------------------------------------------------------------
new_row = {'Model Name' : ModelET,
        'True_Positive': tp,
        'False_Negative': fn,
        'False_Positive': fp,
        'True_Negative': tn,
        'Accuracy' : accuracy,
        'Precision' : precision,
        'Recall' : sensitivity,
        'F1 Score' : f1Score,
        'Specificity' : specificity,
        'MCC':MCC,
        'ROC_AUC_Score':roc_auc_score(y_test, y_pred),
        'Balanced Accuracy':balanced_accuracy}
SGresults = SGresults.append(new_row, ignore_index=True)
```

3.3.10. LGBM Classifier Python Code

```
 # Training the lightgbm model on the Training set

import lightgbm as lgb

# Build the model
```

```
modelLGB = lgb.LGBMClassifier()

# Fit the model with train data

modelLGB.fit(x_train,y_train)

# Predict the model with test data set

y_pred = modelLGB.predict(x_test)

y_pred_prob = modelLGB.predict_proba(x_test)

# Confusion matrix in sklearn

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

# actual values

actual = y_test

# predicted values

predicted = y_pred

# confusion matrix

matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None, normalize=None)

print('Confusion matrix : \n', matrix)

# outcome values order in sklearn

tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)

print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy

C_Report = classification_report(actual,predicted,labels=[1,0])

print('Classification report : \n', C_Report)

# calculating the metrics

sensitivity = round(tp/(tp+fn), 3);
```

```
specificity = round(tn/(tn+fp), 3);

accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);

balanced_accuracy = round((sensitivity+specificity)/2, 3);

precision = round(tp/(tp+fp), 3);

f1Score = round((2*tp/(2*tp + fp + fn)), 3);

# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1 to +1.

# A model with a score of +1 is a perfect model and -1 is a poor model

from math import sqrt

mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)

MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)

print('Accuracy :', round(accuracy*100, 2),'%')

print('Precision :', round(precision*100, 2),'%')

print('Recall :', round(sensitivity*100,2), '%')

print('F1 Score :', f1Score)

print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )

print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')

print('MCC :', MCC)

# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(y_test, y_pred), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_curve

logit_roc_auc = roc_auc_score(y_test, y_pred)
```

```
        fpr, tpr, thresholds = roc_curve(y_test,modelLGB.predict_proba(x_test)[:,1])
        plt.figure()
        # plt.plot
        plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
        plt.plot([0, 1], [0, 1],'r--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
new_row = {'Model Name' : models,
           'True_Positive' : tp,
            'False_Negative' : fn,
           'False_Positive' : fp,
           'True_Negative' : tn,
           'Accuracy' : accuracy,
           'Precision' : precision,
           'Recall' : sensitivity,
           'F1 Score' : f1Score,
           'Specificity' : specificity,
           'MCC':MCC,
```

```
    'ROC_AUC_Score':roc_auc_score(y_test, y_pred),
    'Balanced Accuracy':balanced_accuracy}
     SGresults = SGresults.append(new_row, ignore_
index=True)
```

3.3.4 Hyper Parameter Python code

```
# Hyperparameter tuning with RandomizedSearchCV
    from sklearn.model_selection import RandomizedSearchCV
    solver = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
    penalty = ['none', 'l1', 'l2', 'elasticnet']
    C_space = np.logspace(-5, 8, 15)
    class_weight = ['None', 'dict', 'balanced']
    # Maximum number of iterations taken for the solvers to converge
    max_iter = [100, 1000, 2500, 5000
    # Create the random grid
    Random_Grid = {'solver': solver,
                'penalty': penalty,
                'C': C_space,
                'class_weight': class_weight,
                'max_iter' : max_iter
               }

    print(Random_Grid)

    # Prepare the cross-validation procedure

    from sklearn.model_selection import RepeatedKFold
```

```
RKCV = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

ModelLR = LogisticRegression()

ModelLR_RandomCV=RandomizedSearchCV(estimator=ModelLR,
param_distributions=Random_Grid, n_iter=100, cv=RKCV, verbose=2,
                                    random_state=100, n_jobs=-1)
### fit the randomized model

ModelLR_RandomCV.fit(x_train,y_train)

# best or the optimal values of the parameters for which the model performed the best

ModelLR_RandomCV.best_params_

# To build the 'Logistic Regression' model with random sampling (Hyper parameter
with RandomizedSearchCV)

from sklearn.linear_model import LogisticRegression

# Create model object

ModelLR = LogisticRegression(penalty='none', dual=False, tol=0.0001,
C=163789.3706954068, fit_intercept=True,
                    intercept_scaling=1, class_weight='balanced', random_state=None,
solver='newton-cg',
                    max_iter=1000, multi_class='auto', verbose=0, warm_start=False,
                    n_jobs=None, l1_ratio=None)

# Fit the model

ModelLR.fit(x_train, y_train)

# Predict the model with test data set
```

```
y_pred = ModelLR.predict(x_test)
y_pred_prob = ModelLR.predict_proba(x_test)

# confusion matrix in sklearn

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# actual values

actual = y_test

# predicted values

predicted = y_pred

# confusion matrix

matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,
normalize=None)
print('Confusion matrix : \n', matrix)

# outcome values order in sklearn

tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy

C_Report = classification_report(actual,predicted,labels=[1,0])

print('Classification report : \n', C_Report)
```

```
# calculating the metrics

sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);

precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);

# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1 to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model

from math import sqrt

mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)

print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)

# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(y_test, y_pred), 3))

# ROC Curve
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test,ModelLR.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
print('-------------------------------------------------------------------------------------------------
-----')
#-------------------------------------------------------------------------------------------------------
----
new_row = {'Model Name' : ModelLR,
        'True_Positive': tp,
        'False_Negative': fn,
        'False_Positive': fp,
        'True_Negative': tn,
        'Accuracy' : accuracy,
        'Precision' : precision,
        'Recall' : sensitivity,
        'F1 Score' : f1Score,
        'Specificity' : specificity,
        'MCC':MCC,
        'ROC_AUC_Score':roc_auc_score(y_test, y_pred),
        'Balanced Accuracy':balanced_accuracy}
```

```
SGresults = SGresults.append(new_row, ignore_index=True)
```

4.0 Conclusions and Future work

The campus placement activity is incredibly a lot of vital as institution point of view as well as student point of view. In this regard to improve the student's performance, a work has been analyzed and predicted using the classification algorithms Decision Tree and the Decision Tree on hypertuneing algorithm to validate the approaches. The algorithms are applied on the data set and attributes used to build the model. The accuracy obtained after analysis for Decision tree is 81.5% and for the Random Forest is 81%. Hence, from the above said analysis and prediction it's better if the DecisionTree algorithm is used to predict the placement results.

Model Name	False Negative	False Positive	True Negative	True Positive	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy
LogisticRegression()	44	50	86	162	0.725	0.764	0.786	0.775	0.632	0.422	0.709380	0.709
DecisionTreeClassifier()	40	46	90	166	0.749	0.783	0.806	0.794	0.662	0.471	0.733795	0.734
TreeClassifier(max_features='sqrt', r...	41	47	89	165	0.743	0.778	0.801	0.789	0.654	0.459	0.727691	0.728
ssifier(random_state=17799642), E...	49	48	88	157	0.716	0.766	0.762	0.764	0.647	0.409	0.704597	0.704
KNeighborsClassifier()	60	48	88	146	0.684	0.753	0.709	0.730	0.647	0.351	0.677898	0.678
SVC(probability=True)	49	35	101	157	0.754	0.818	0.762	0.789	0.743	0.498	0.752391	0.752
Classifier(random_state=210569038...	39	45	91	167	0.754	0.788	0.811	0.799	0.669	0.484	0.739899	0.740
eRegressor(criterion='friedman_ms...	43	22	114	163	0.810	0.881	0.791	0.834	0.838	0.618	0.814749	0.814
LGBMClassifier()	39	41	95	167	0.766	0.803	0.811	0.807	0.699	0.511	0.754605	0.755
GaussianNB()	45	41	95	161	0.749	0.797	0.782	0.789	0.699	0.478	0.740041	0.740
eRegressor(criterion='friedman_ms...	43	22	114	163	0.810	0.881	0.791	0.834	0.838	0.618	0.814749	0.814
GaussianNB()	49	30	106	157	0.769	0.840	0.762	0.799	0.779	0.532	0.770774	0.770

5.0 Reference:

1]. Mangasuli Sheetal B, Prof. Savita Bakare "Prediction of Campus Placement Using Data Mining Algorithm-Fuzzy logic and K nearest neighbour" International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 6, June 2016 .

[2]. Ajay Shiv Sharma, Swaraj Prince, Shubham Kapoor, Keshav Kumar "PPS-Placement prediction system using logistic regression" IEEE international conference on MOOC,innovation and Technology in Education(MITE), December 2014.

[3]. Jai Ruby, Dr. K. David "Predicting the Performance of Students in Higher Education Using Data Mining Classification Algorithms - A Case Study" International Journal for

Research in Applied Science & Engineering Technology (IJRASET) Vol. 2,Issue 11,November 2014.

[4]. Ankita A Nichat, Dr.Anjali B Raut "Predicting and Analysis of Student Performance Using Decision Tree Technique" International Journal of Innovative Research in Computer and Communication Engineering V0l. 5, Issue 4, April 2017.

[5]. Oktariani Nurul Pratiwi "Predicting Student Placement Class using Data Mining" IEEE International Conference 2013.

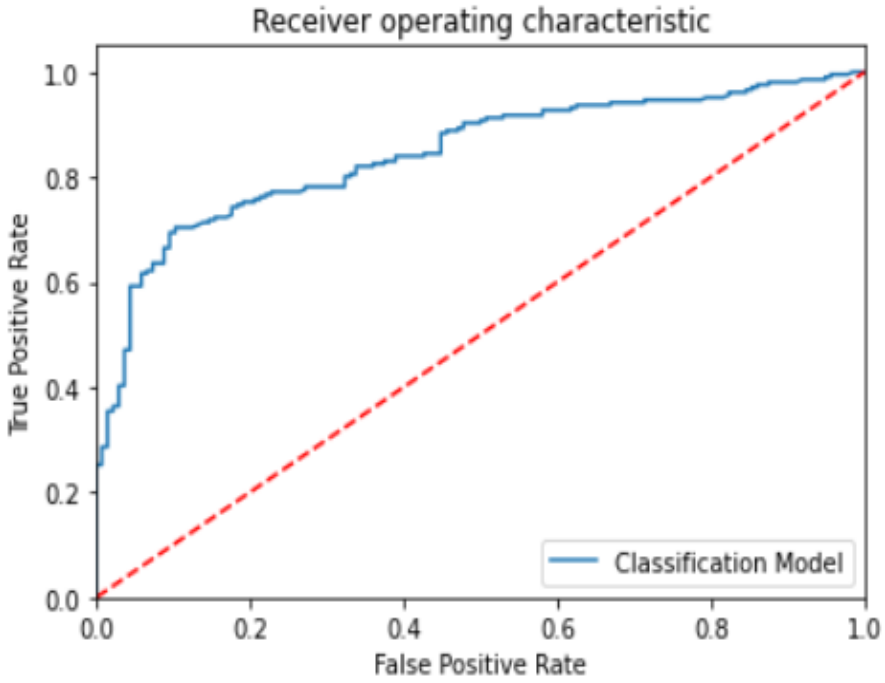
6.0. Appendices:

6.1 Python code Results:

- The result of logistic regression before hyper tuning as follows:

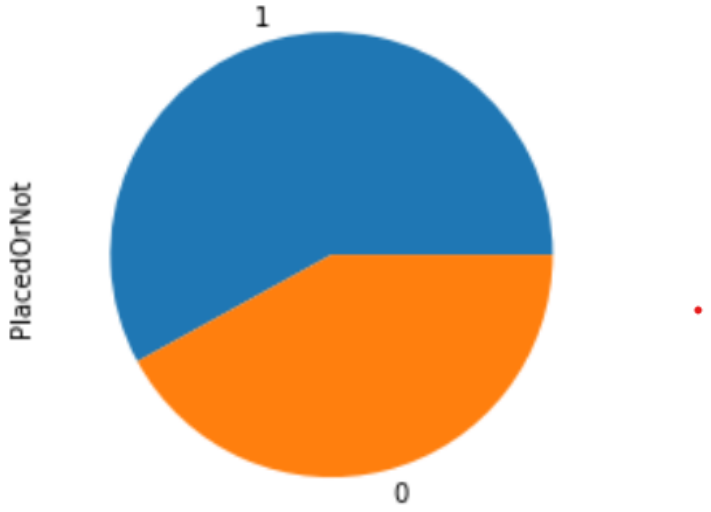
```
Confusion matrix :
 [[163  43]
 [ 22 114]]
Outcome values :
 163 43 22 114
Classification report :
              precision    recall  f1-score   support

           1       0.88      0.79      0.83       206
           0       0.73      0.84      0.78       136

    accuracy                           0.81       342
   macro avg       0.80      0.81      0.81       342
weighted avg       0.82      0.81      0.81       342

Precision : 88.1 %
Recall : 79.1 %
F1 Score : 0.834
Specificity or True Negative Rate : 83.8 %
Balanced Accuracy : 81.4 %
MCC : 0.618
Accuracy : 81.0 %
roc_auc_score: 0.815
```

DecisionTree hypertuneing results

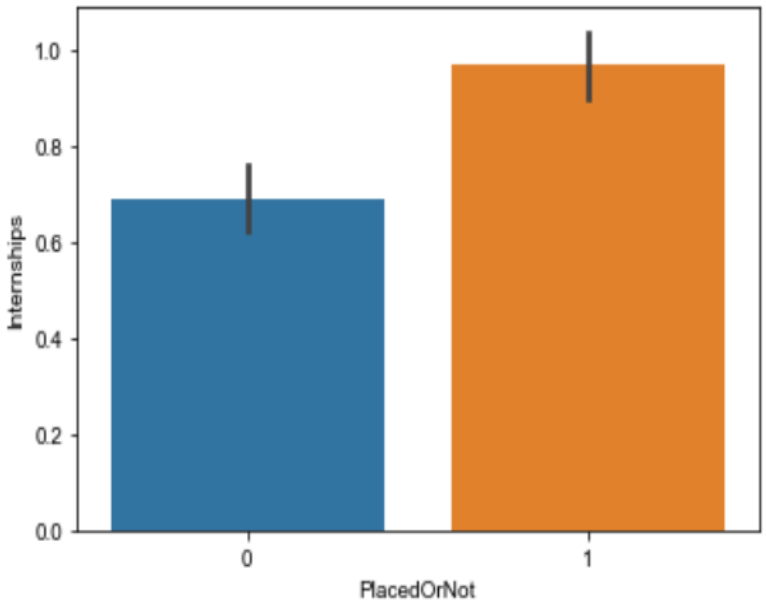


DecisionTree hypertuneing graph

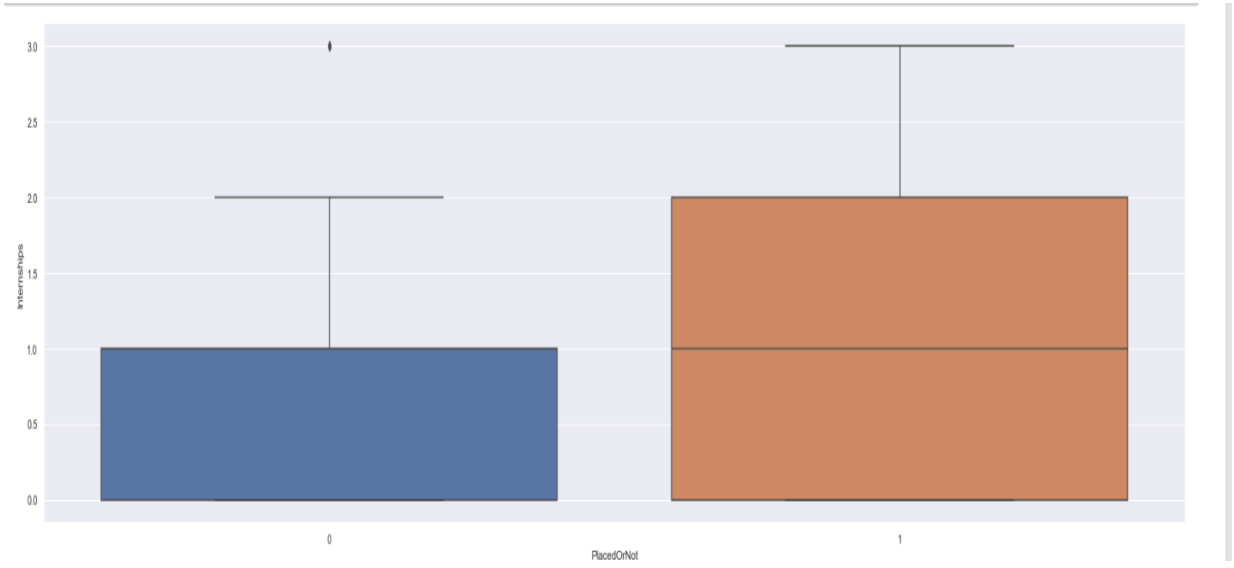
List of charts:



Pie chart



Bar graph



Box plot