# ILLINOIS INSTITUTE OF TECHNOLOGY


## Personalized Placement Predictor for Job Aspirants

**Team Members**

| | |
|---|---|
| **Subramanya Ganesh** | **A20516250** |
| **Vinod Krishna Selpol** | **A20511584** |
| **Manoja Krishna Damele** | **A20521267** |

**GitHub Link :** https://github.com/subramanyaganesh/CS584_MachineLearning

## Table of Contents


1. Abstract 4
2. Problem Statement 4
3. Proposed Solution 5
    - 3.1 Logistic Regression 6
    - 3.2 Decision Trees 6
    - 3.3 Random Forests 6
    - 3.4 Support Vector Machines 7
    - 3.5 Naive Bayes 7
    - 3.6 K-nearest Neighbor 7
    - 3.7 Gradient Boosting 8
4. Exploratory Data Analysis (EDA) 8
5. Data Preprocessing 14
    - 5.1 Splitting Data 14
    - 5.2 Missing Values 14
    - 5.3 Outliers 14
    - 5.4 Feature Selection 15
    - 5.5 Pipeline 15
6. Model Definition 16
    - 6.1 Model Training 16
    - 6.2 Model Evaluation 17
        - 6.2.1 SVC Model 17

6.2.2 Decision Trees 17

6.2.3 Random Forest 18

6.2.4 KNN 18

6.2.5 Naive Bayes 19

6.2.6 Gradient Boosting 19

6.2.7 Logistic Regression 20

6.3 Confusion Matrix 21

7. Model Improvements 22

8. Overall analysis model evaluation 24

9. Conclusion 25

10. Reference 26

## 1. Abstract

The global workforce is constantly evolving, and organizations are increasingly recognizing the need for educated and skilled individuals. In developing countries, where the demand for such individuals is particularly high, organizations routinely recruit fresh graduates to meet their staffing needs. However, traditional recruitment methods and selection processes can be prone to errors, leading to suboptimal hiring decisions and inefficiencies in the recruitment process.

To address these issues, innovative methods are needed to optimize the recruitment process. One such method is the use of machine learning to predict the success of a candidate in the recruitment process. By analyzing past data, machine learning algorithms can identify patterns and make predictions about which candidates are likely to be successful hires.The use of machine learning in recruitment has several potential benefits, including increased efficiency and accuracy in the selection process, reduced bias in candidate selection, and improved candidate experience. However, to achieve these benefits, accurate prediction models must be developed and validated. This requires access to high-quality data and expertise in data analytics and machine learning. This paper explores the development of machine learning algorithms for accurate prediction to ensure efficient hiring process and it also compares the different models with its accuracy and efficiency. In this project we gathered different features to build a machine learning model that can predict if the applicant can be placed or not. Furthermore, we propose future applications of machine learning approaches to job placements.

## 2. Problem statement

Organizations routinely hire recent graduates because of the growing demand for educated and talented people, particularly in developing nations. Traditional hiring practices and selection procedures can be prone to mistakes, so some novel approaches are required to optimize the entire process.

In this project, we have built different classification ML models that can predict if the applicant can be placed or not based on the data we have gathered.

The job placement dataset provided contains various attributes related to the educational background and work experience of candidates. These attributes can be used to develop a machine learning model to predict the likelihood of a candidate being successfully placed in a job. Below is a detailed description of each attribute in the dataset:

**Gender**: This attribute indicates the gender of the candidate.

**SSC Percentage**: This attribute indicates the percentage of marks obtained by the candidate in their Senior Secondary Certificate (SSC) examination, which is usually taken at the end of 10th grade.

**SSC Board**: This attribute indicates the board of education for the SSC examination.

**HSC Percentage**: This attribute indicates the percentage of marks obtained by the candidate in their Higher Secondary Certificate (HSC) examination, which is usually taken at the end of 12th grade.

**HSC Board**: This attribute indicates the board of education for the HSC examination.

**HSC Subject**: This attribute indicates the subject of study for the candidate in their HSC examination.

**Degree Percentage**: This attribute indicates the percentage of marks obtained by the candidate in their undergraduate degree.

**Undergrad Degree**: This attribute indicates the majors that the candidate pursued in their undergraduate degree.

**Work Experience**: This attribute indicates the past work experience of the candidate, if any.

**Employment Test Percentage**: This attribute indicates the percentage of marks obtained by the candidate in an aptitude test conducted by the employer.

**Specialization**: This attribute indicates the majors that the candidate pursued in their postgraduate degree, specifically an MBA specialization.

**MBA Percent**: This attribute indicates the percentage of marks obtained by the candidate in their MBA degree.

**Status (TARGET)**: This attribute is the target variable and indicates whether the candidate was successfully placed or not. This is the attribute that the machine learning model is trained to predict.

## 3. Proposed solution

In this project, different classification models are developed and trained. The comparison of those different models is done to evaluate the accuracy and efficiency of the models.

In machine learning, classification is a supervised learning technique that involves predicting the categorical class label of a given input data point. The goal of classification is to learn a model from a given set of labeled training data that can be used to classify new, unseen data into one of the pre-defined categories or classes.

In a classification problem, the input data can be represented as a set of features, which are used to predict the output class label. The classification model uses various algorithms and techniques to map the input features to the output class label. The input data is divided into two sets, namely training data and testing data. The training data is used to train the classification model, while the testing data is used to evaluate the performance of the model.

The different types of classification algorithms are trained in this project:

### 3.1. Logistic regression

Logistic regression is a statistical and machine learning technique that is commonly used for classification problems. It is a type of supervised learning algorithm that predicts the probability of a binary outcome (i.e., a categorical response variable that takes only two possible values) based on one or more predictor variables, which can be either continuous or categorical.

In logistic regression, the goal is to find a mathematical relationship between the predictor variables and the binary outcome variable, by estimating the probability of the outcome variable being in a particular category, given the values of the predictor variables. The logistic regression model produces a predicted probability value for each data point, which is then converted into a class label based on a threshold value. Typically, the threshold value is set at 0.5, meaning that if the predicted probability is greater than or equal to 0.5, the data point is classified as belonging to one category, and if it is less than 0.5, it is classified as belonging to the other category.

### 3.2. Decision Trees

Decision Trees are a popular machine learning algorithm used for both classification and regression problems. Decision trees work by recursively partitioning the data into subsets based on the value of a chosen attribute, in a way that maximizes the separation of the target variable. This results in a tree-like model, where each internal node represents a test on a particular attribute, each branch represents the outcome of the test, and each leaf node represents a predicted value or class.

In the classification setting, decision trees can be used to predict the class of a given instance based on the values of its features. The tree is constructed by recursively selecting the best feature to split the data on, until a stopping criterion is met (e.g., a minimum number of instances per leaf node, or a maximum depth of the tree). The criterion for selecting the best feature can vary, but commonly used metrics include information gain, Gini impurity, and chi-squared tests.

### 3.3. Random Forests

Random forests are an ensemble learning method in machine learning that is widely used for both classification and regression tasks. It is an extension of the decision tree algorithm that builds multiple decision trees and then combines them to improve the overall prediction accuracy.

Random forests work by building multiple decision trees on different subsets of the training data, using random subsets of the features at each split. The trees are grown to maximum depth, and the final prediction is made by taking the majority vote (in classification) or the average (in regression) of the predictions of the individual trees. The randomization of both the data and features helps to reduce overfitting and improve the generalization performance of the model.

### 3.4. Support Vector Machines

SVM, or Support Vector Machines, is a powerful supervised learning algorithm in machine learning that is used for classification and regression analysis. It is particularly useful for solving binary classification problems, i.e., problems where we need to classify the input data into two distinct classes.

The basic idea behind SVM is to find the best possible decision boundary that can separate the two classes. In other words, SVM tries to find a hyperplane that can best separate the data into two classes. However, not all hyperplanes are created equal; some may be too close to one class or too far from the other, leading to poor classification accuracy. To find the optimal hyperplane, SVM uses a mathematical optimization technique that maximizes the margin between the two classes. The margin is defined as the distance between the hyperplane and the closest data points from each class.

### 3.5. Naive Bayes

Naive Bayes is a classification algorithm in machine learning that is based on Bayes' theorem. The algorithm is called "naive" because it makes a simplifying assumption that the features in the input data are conditionally independent of each other given the class label.

In a classification problem, the goal is to predict the class label of an input data point based on its feature values. In Naive Bayes, we calculate the probability of the input data point belonging to each class label, and then choose the class label with the highest probability as the prediction.

To calculate the probability of a data point belonging to a particular class label, Naive Bayes uses Bayes' theorem. Bayes' theorem states that the probability of a hypothesis (class label) given the observed evidence (input data point) is proportional to the probability of the evidence given the hypothesis, multiplied by the prior probability of the hypothesis.

### 3.6. K-nearest Neighbor

K-Nearest Neighbors (KNN) is a popular algorithm for classification and regression tasks in machine learning. It is a type of instance-based learning where the algorithm memorizes the training data instances and predicts the class or value of new instances based on the similarity with the nearest K training instances.

In KNN, K refers to the number of nearest neighbors that are used to make a prediction for a new data point. The similarity between two instances is typically calculated using a distance metric, such as Euclidean distance or Manhattan distance. The KNN algorithm then finds the K training instances that are closest to the new data point in terms of the distance metric, and assigns the majority class label (in classification) or the average value (in regression) of these K instances to the new data point.

### 3.7. Gradient Boosting

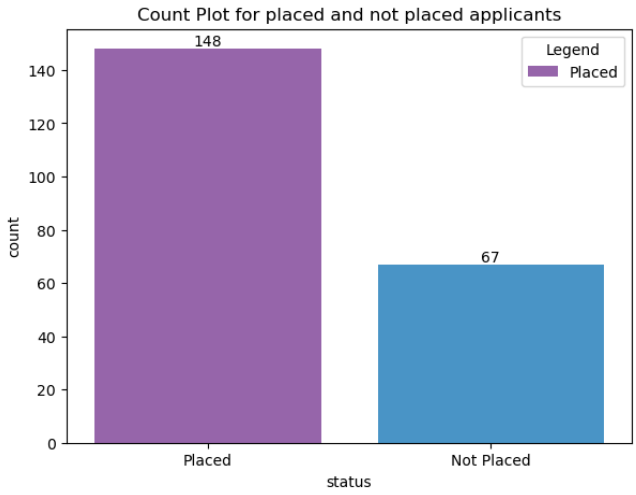
Gradient Boosting is a machine learning algorithm that is used for both regression and classification tasks. It is a type of ensemble learning method that combines multiple weak learners into a single strong learner. The basic idea behind gradient boosting is to iteratively add new weak learners to the ensemble, with each new learner attempting to correct the errors made by the previous learners.

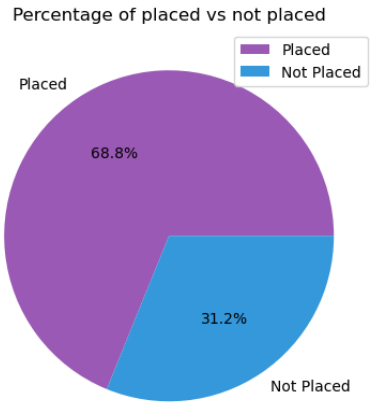
The process starts with the creation of a single decision tree as the first weak learner. This decision tree makes predictions based on the input features, and the errors between its predictions and the true values are calculated. Then, a new decision tree is trained to predict these errors, which is added to the first tree to create a better model. This process is repeated iteratively until the desired accuracy is achieved or no further improvement is possible.

## 4. Exploratory Data Analysis (EDA)

Gaining insights into the data, comprehending its structure, spotting patterns and linkages, and spotting any anomalies or outliers that could need additional examination are the objectives of EDA. EDA is frequently used as the initial stage of data analysis before using more sophisticated statistical or machine learning approaches.

Below are the graphs used to describe the different features that will determine the classification of an applicant as placed or not placed.
The below graph describes the overall plot for number of people placed vs not placed



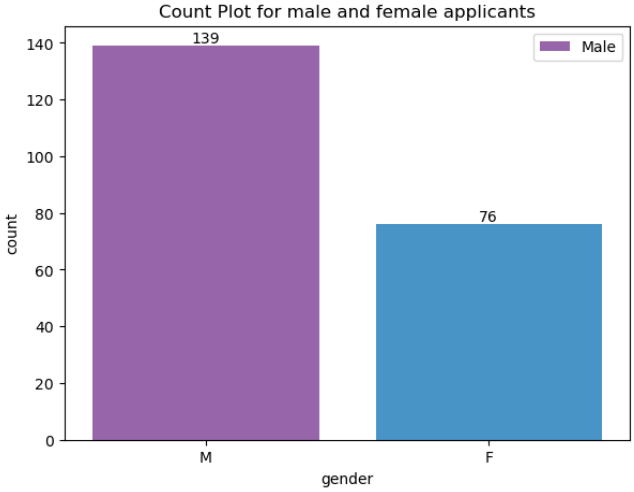

***Feature descriptions***

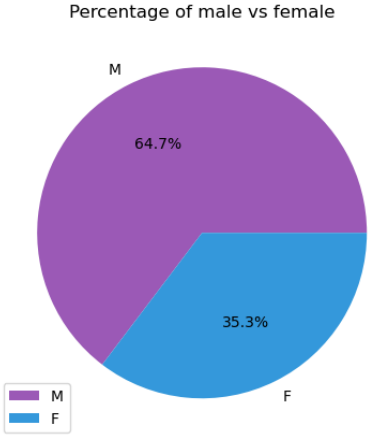
In order to efficiently develop a machine learning model for making an accurate prediction of job placement for an applicant, we need to figure out all the features that attribute to an applicant acquiring a job. There has to be a significant number of features for the model to give accurate prediction, but this does not mean that more features result in better prediction as it is more likely to overfit the model and would lead to low output rate for testing data. Hence selecting the number of features is a crucial step in developing a model. In this project, after careful consideration, we have considered the following features as our initial set of features.

1. Gender
2. SSC percentage or $10^{th}$ grade marks
3. HSC percentage or $12^{th}$ grade marks
4. HSC subjects or specialization in $12^{th}$ grade
5. Undergrad percentage
6. Degree percentage
7. Work experience
8. Employee test percentage
9. Specialization
10. MBA Percentage

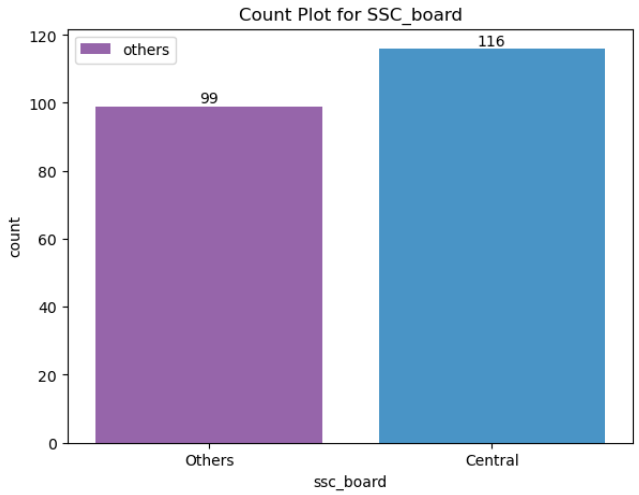
The below mentioned graphs provide the statistics required to determine the weightage of each of these features in job placement. The final graph provides the summary of these statistics and uses *Regularisation* to determine the most significant features that will be used to generate a model equation.

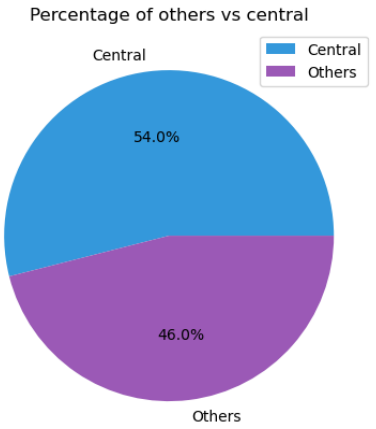
- *Gender*



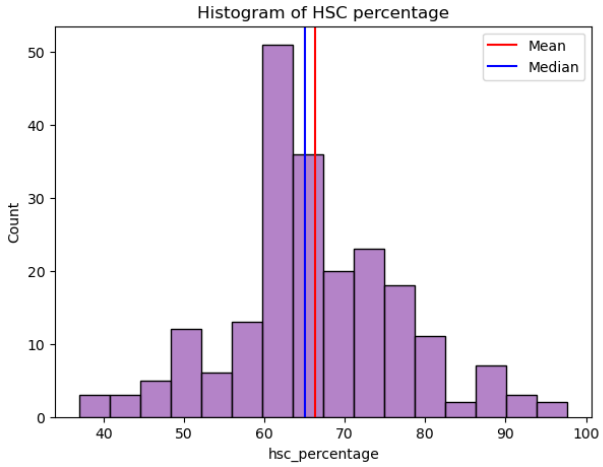



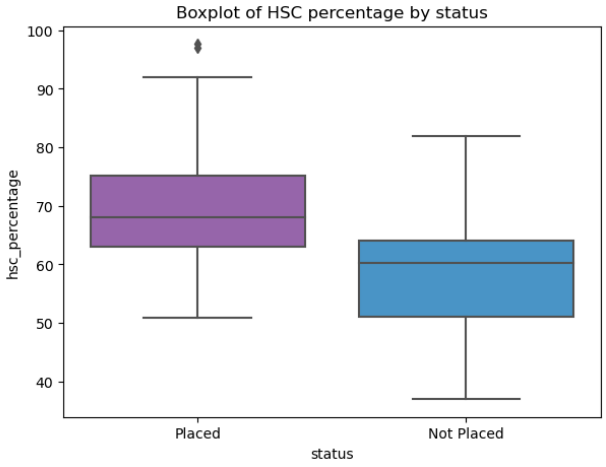

- *SSC percentage*



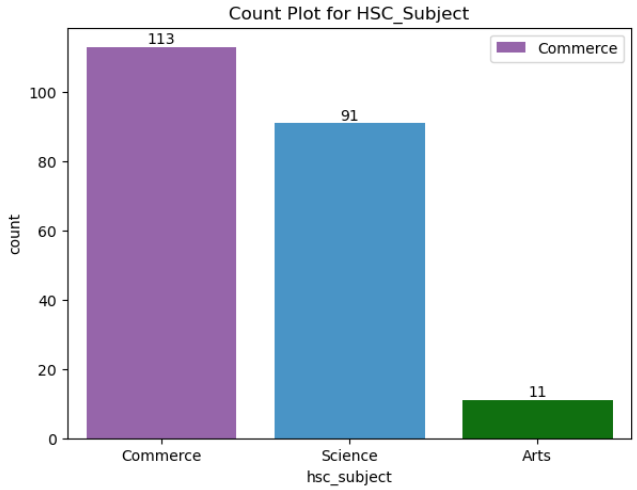



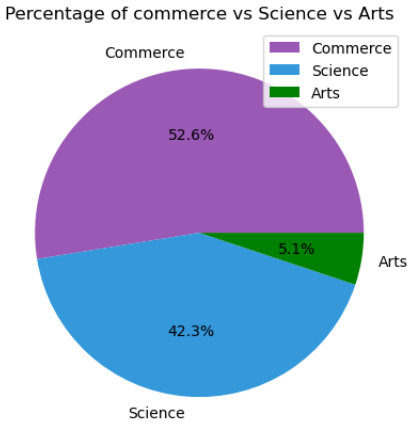

- *HSC percentage*



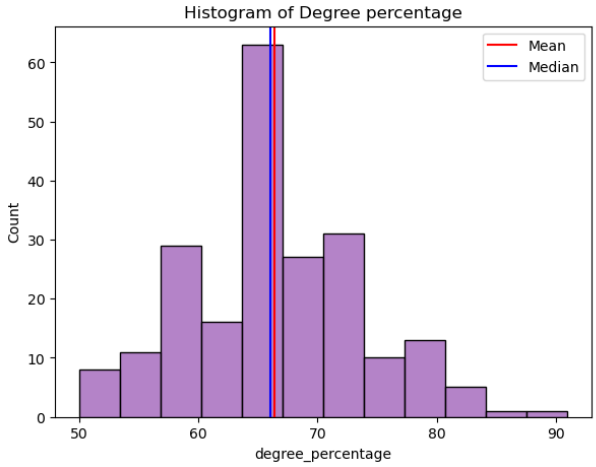



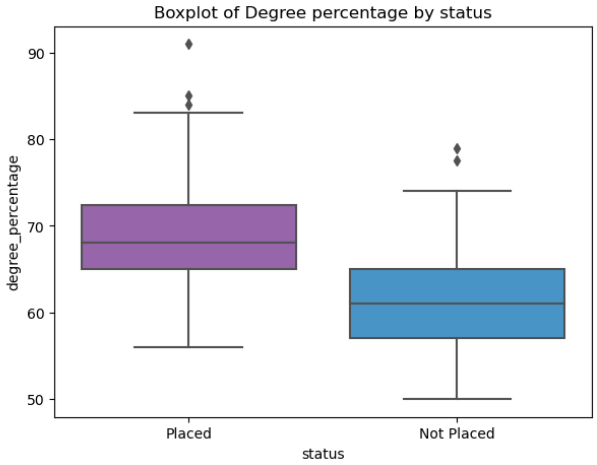

- *HSC subject*



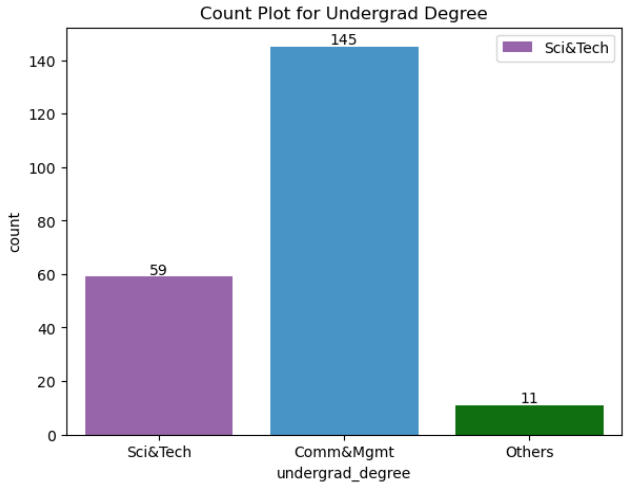



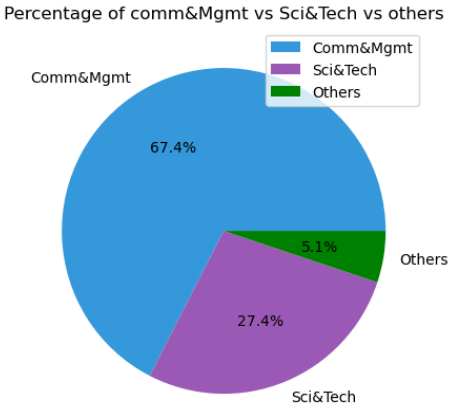
- *Degree Percentage*



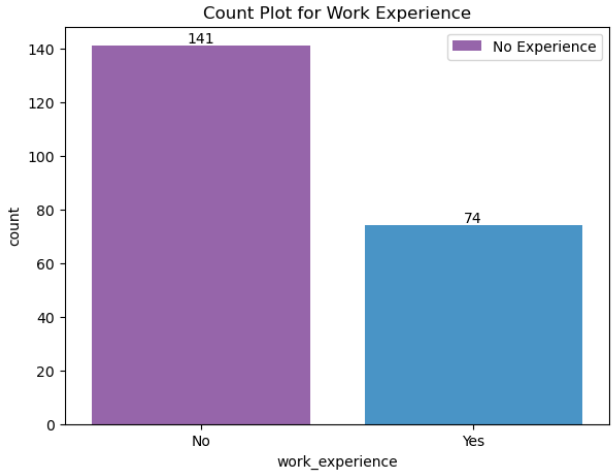



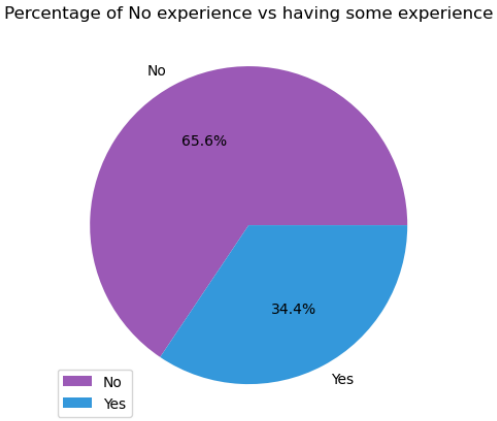

- *Undergrad Percentage*



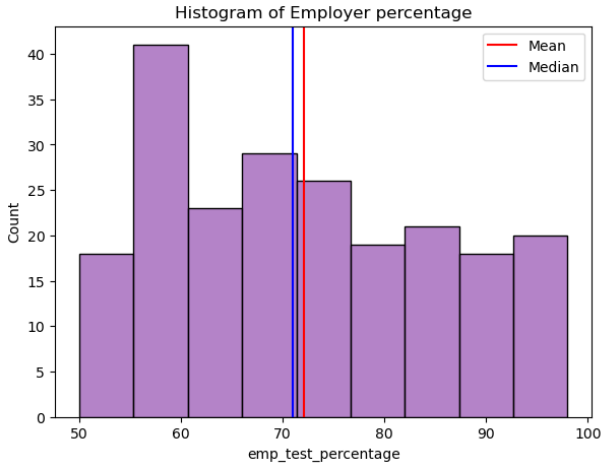



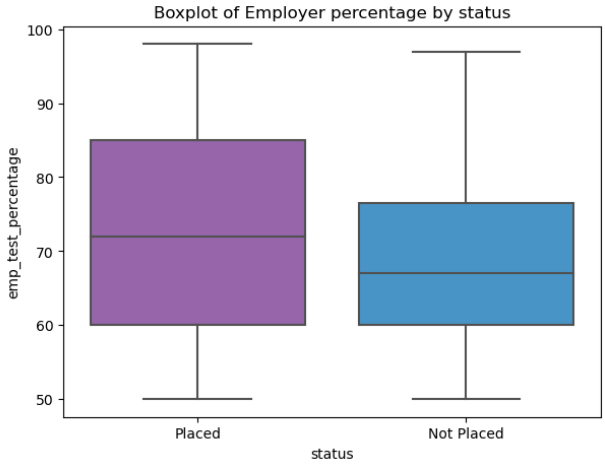

- *Work Experience*



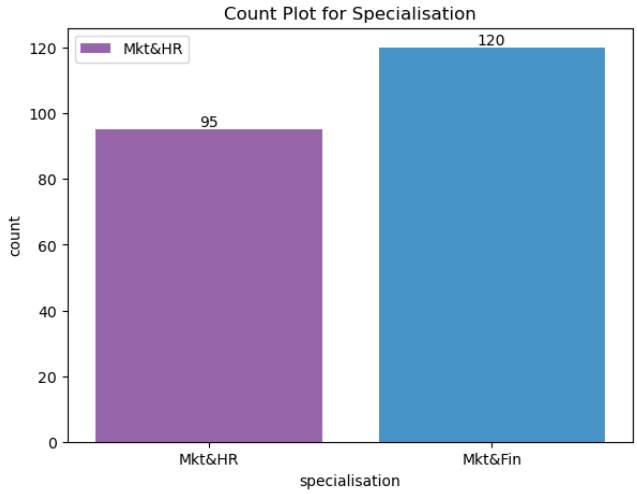



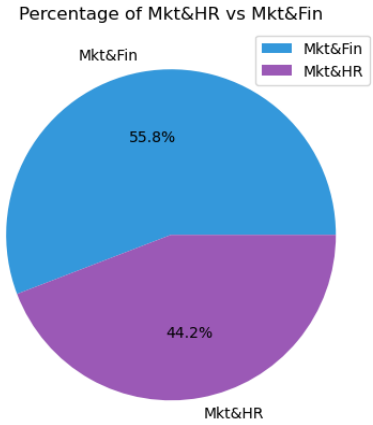
- *Employee test Percentage*



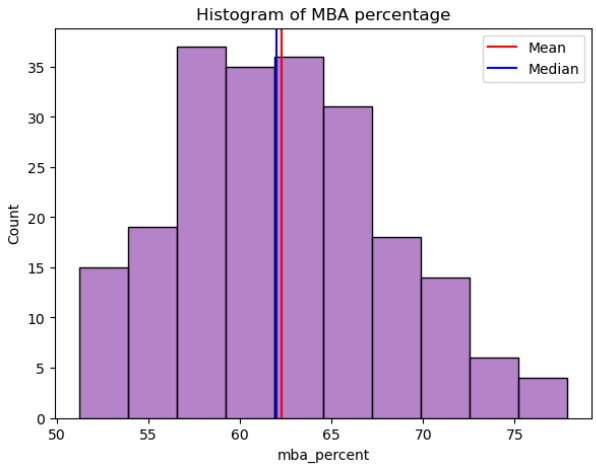



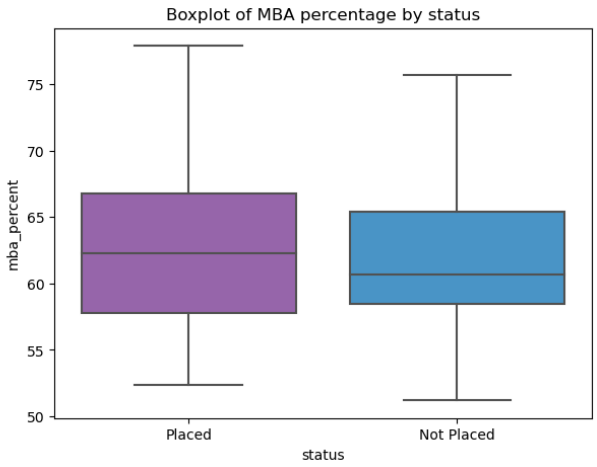

- *Specialization*



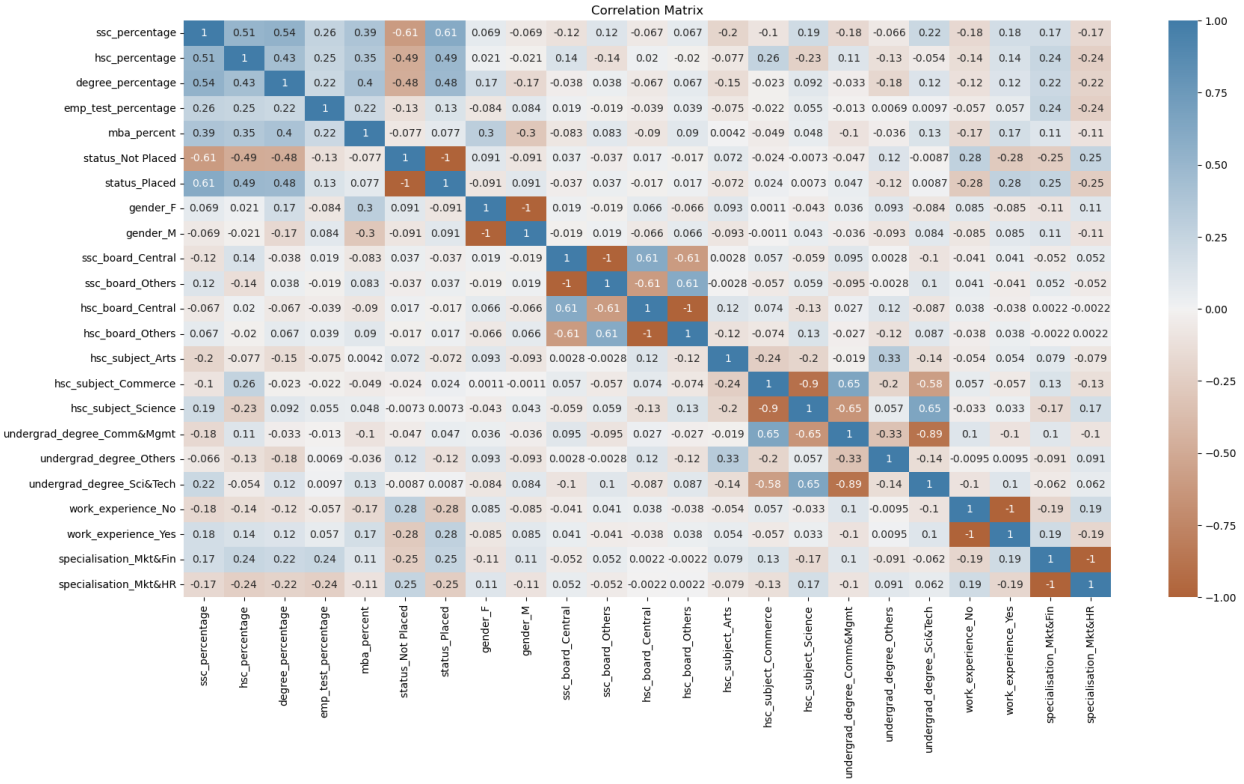




- *MBA Percent*




The below graph shows the summary of the weights of each parameter and specifies which set of features are prominent for developing the model equation.




From the correlation matrix it can be seen that the target column has a correlation with the

- SSC_percentage
- HSC_percentage
- Degree_percentage
- Work_experience
- Specialization

these columns will be used as features

## 5. Data Preprocessing :

Data preprocessing is the process of preparing raw data for use with a machine learning model. It is the first and most important step in developing a machine learning model. When developing a machine learning project, we do not always come across clean and formatted data.

### 5.1. Splitting Data :

Splitting data in machine learning refers to dividing the available dataset into two or more subsets that can be used for different purposes during machine learning model training, validation, and testing. The most common method for splitting data is to divide it into training, validation, and testing sets.

In this section, data separation is performed between train data and test data, with a 70% train data to 30% test data ratio.

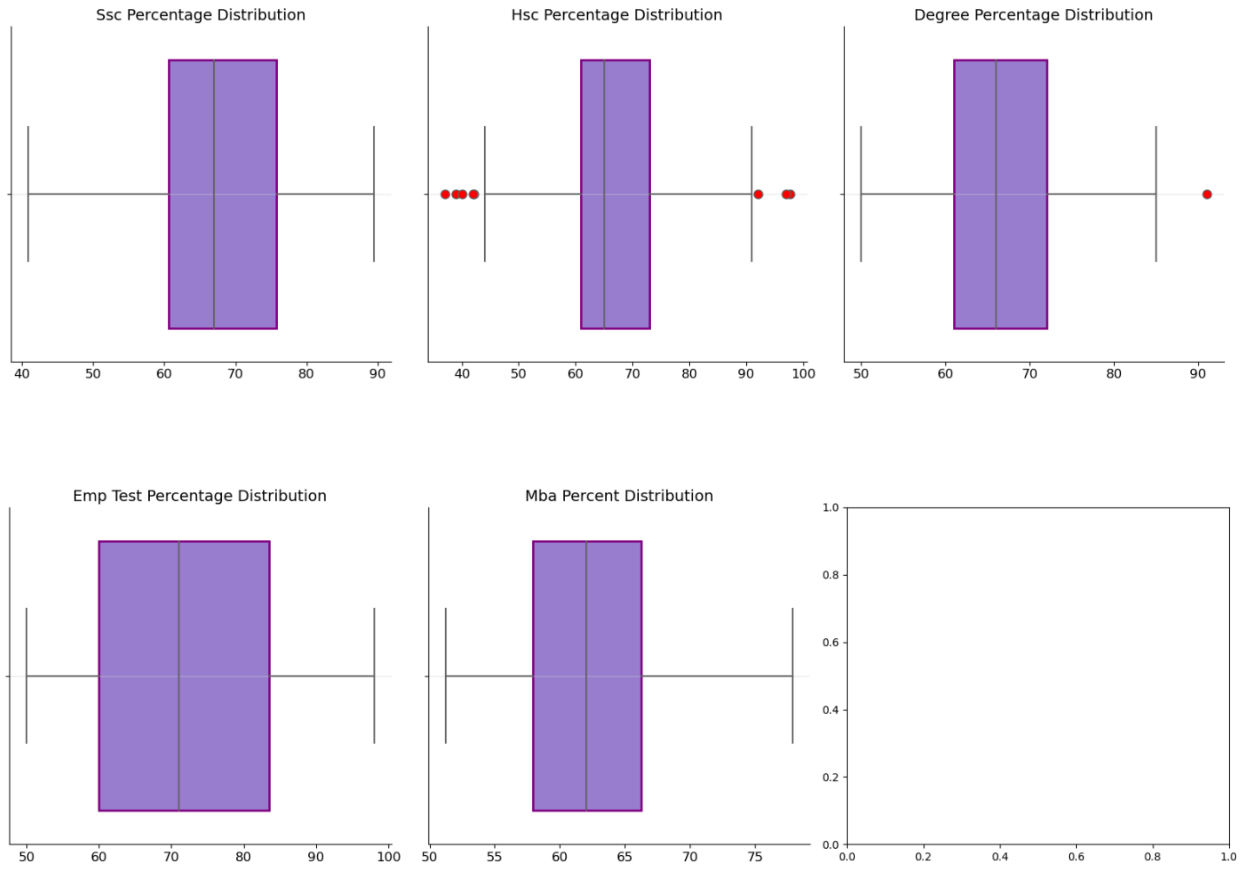
### 5.2. Missing Values :

The absence of a specific value or feature in a dataset is referred to as missing values. Missing values in machine learning can cause issues during data preprocessing and model training.
In the dataset of our project, we do not have any missing values.

### 5.3. Outliers :

Outliers in machine learning are data points or observations that deviate significantly from the normal range of values in a dataset. Outliers can appear for a variety of reasons, including measurement errors, data entry errors, and natural variations in the data.Outliers can have a significant impact on machine learning model accuracy and reliability. They can skew the data distribution, affect the mean and variance, and cause the model to overfit or underfit. As a result, it is critical to detect and handle outliers appropriately during data preprocessing.

In the below mentioned boxplot , There are 2 columns that are having outliers namely the hsc_percentage and degree_percentage columns. It turns out that the outlier in the hsc_percentage column is data that has an hsc value of less than 40 and in the degree_percentage column that makes this column have an outlier is the value 91 in degree_percentage




### 5.4. Feature Selection :

The process of selecting a subset of relevant features (variables or attributes) from a larger set of features in a dataset is known as feature selection. The goal of feature selection is to improve machine learning model performance by reducing the dimensionality of the input data and removing irrelevant, redundant, or noisy features that may have a negative impact on the model's accuracy, interpretability, or generalization ability.

### 5.5. Pipeline:

A pipeline in machine learning is a chained together sequence of data processing components (also known as stages) that transform and prepare data for machine learning algorithms. A pipeline is used to automate and streamline the process of developing, training, and deploying machine learning models by encapsulating all necessary data transformations and model development steps into a single object that can be easily reused and modified.
In this section , Pipeline is made to handle outliers using winsorizer with the capping method using IQR and fold 1.5 in the hsc_percentage and degree_percentage columns, then scaling using StandardScaler, then encoding using OneHotEncoding, then combining the results from handling outliers, scaling and encoding using ColumnTransformer with preprocessor variable.

The preprocessing pipeline is important because it allows you to transform your data in a consistent and automated way before fitting a machine learning model.
Raw data often has missing values, outliers, or categorical variables that need to be transformed before a model can be trained. The preprocessing pipeline allows you to apply different transformations to different columns of data in a way that can be easily replicated on new data.
By using a pipeline, you can ensure that the same preprocessing steps are applied to both your training data and your test data. This is important because it helps to avoid overfitting to the training data and ensures that your model generalizes well to new, unseen data.

## 6. Model Definition:

After building the model, a pipeline is established, comprising 7 individual models: SVM, Decision Tree, Random Forest, K-Nearest Neighbor, Naive Bayes, Gradient Boosting, and Logistic Regression. This allows for comprehensive analysis and comparison of the different models, aiding in the selection of the most suitable approach.

```
# Model building
svc_pipe = Pipeline([('preprocessing', preprocessor),('classifier', SVC())])
tree_pipe = Pipeline([('preprocessing', preprocessor),('classifier', DecisionTreeClassifier())])
rf_pipe = Pipeline([('preprocessing', preprocessor),('classifier', RandomForestClassifier())])
knn_pipe = Pipeline([('preprocessing', preprocessor),('classifier', KNeighborsClassifier())])
bayes_pipe = Pipeline([('preprocessing', preprocessor),('classifier', GaussianNB())])
gb_pipe = Pipeline([('preprocessing', preprocessor),('classifier', GradientBoostingClassifier())])
logistic_pipe = Pipeline([('preprocessing', preprocessor),('classifier', LogisticRegression())])
```

```
# Rename variables
svc_base = svc_pipe
tree_base = tree_pipe
rf_base = rf_pipe
knn_base = knn_pipe
bayes_base = bayes_pipe
gb_base = gb_pipe
logistic_base = logistic_pipe
```

### 6.1. Model Training:

Within this segment, training of each model is executed. This involves the development of individual models, each of which utilizes distinct algorithms and techniques, and is tailored to the specific task at hand. Through this process, the models are refined and optimized for optimal performance.

```
# Train Model
svc_base.fit(X_train, y_train)
tree_base.fit(X_train, y_train)
rf_base.fit(X_train, y_train)
knn_base.fit(X_train, y_train)
bayes_base.fit(X_train, y_train)
gb_base.fit(X_train, y_train)
logistic_base.fit(X_train, y_train)
```

### 6.2. Model Evaluation:

The following section involves an evaluation of the models to determine which one performs most effectively for the task at hand. The evaluation involves a comparison of each model's output with the actual data to determine its accuracy, precision, and recall, among other performance metrics. By analyzing these metrics, it is possible to identify the model that produces the most accurate and reliable results. This process is essential in selecting the best-performing model, which can be further refined and optimized for improved outcomes. Through rigorous evaluation and selection, the chosen model will provide valuable insights and enable efficient decision-making.

*6.2.1. SVC Model:*

```
print('Training Score:', svc_base.score(X_train, y_train))
print('Test Score:', svc_base.score(X_test, y_test))
print('-----------------------------------------------------')
print(classification_report(y_test, svc_base.predict(X_test)))
```

```
Training Score: 0.9
Test Score: 0.8461538461538461
-----------------------------------------------------
              precision    recall  f1-score   support

  Not Placed       0.78      0.70      0.74        20
      Placed       0.87      0.91      0.89        45

    accuracy                           0.85        65
   macro avg       0.83      0.81      0.81        65
weighted avg       0.84      0.85      0.84        65
```

The support vector classifier (SVC) model exhibits a train score of 0.9 and a test score of 0.84. However, there is a slight overfitting present. The model has an accuracy of 0.85, a recall of 0.81, and a precision of 0.83. This suggests that the model's performance is good, but not excellent. The overfitting may be due to the model being too complex for the given dataset. Careful analysis and modification of the model may help to improve its performance and reduce overfitting, leading to more accurate and reliable predictions.

*6.2.2. Decision Trees:*

```
print('Training Score:', tree_base.score(X_train, y_train))
print('Test Score:', tree_base.score(X_test, y_test))
print('-----------------------------------------------------')
print(classification_report(y_test, tree_base.predict(X_test)))
```

```
Training Score: 1.0
Test Score: 0.7846153846153846
-----------------------------------------------------
              precision    recall  f1-score   support

  Not Placed       0.65      0.65      0.65        20
      Placed       0.84      0.84      0.84        45

    accuracy                           0.78        65
   macro avg       0.75      0.75      0.75        65
weighted avg       0.78      0.78      0.78        65
```

The Decision Tree model displays a train score of 1.0 and a test score of 0.78, indicating an overfitting issue. Despite having an accuracy of 0.8, the model's recall and precision scores are relatively low at 0.73 and 0.75, respectively. This highlights the model's inability to generalize to new data, which is a common issue with overfitting. To address this, various techniques such as regularization, pruning, and feature selection may be applied to improve the model's generalization performance. By reducing overfitting and increasing generalization, the model's accuracy, recall, and precision can be improved, resulting in more reliable predictions.

*6.2.3. Random Forest:*

```
print('Training Score:', rf_base.score(X_train, y_train))
print('Test Score:', rf_base.score(X_test, y_test))
print('-----------------------------------------------------')
print(classification_report(y_test, rf_base.predict(X_test)))
```

```
Training Score: 1.0
Test Score: 0.8615384615384616
-----------------------------------------------------
              precision    recall  f1-score   support

  Not Placed       0.82      0.70      0.76        20
      Placed       0.88      0.93      0.90        45

    accuracy                           0.86        65
   macro avg       0.85      0.82      0.83        65
weighted avg       0.86      0.86      0.86        65
```

The Random Forest model demonstrates a train score of 1.0 and a test score of 0.84, indicating overfitting. Despite an accuracy score of 0.86, the model's recall and precision scores are relatively low at 0.79 and 0.83, respectively. The overfitting issue may be addressed by reducing the number of trees in the forest or by pruning the individual decision trees. This would aid in improving the model's generalization capability, resulting in better performance on new, unseen data. By addressing the overfitting problem, the model's accuracy, recall, and precision can be enhanced, allowing for more dependable predictions.

*6.2.4. KNN:*

```
print('Training Score:', knn_base.score(X_train, y_train))
print('Test Score:', knn_base.score(X_test, y_test))
print('-----------------------------------------------------')
print(classification_report(y_test, knn_base.predict(X_test)))
```

```
Training Score: 0.9133333333333333
Test Score: 0.7846153846153846
-----------------------------------------------------
              precision    recall  f1-score   support

  Not Placed       0.67      0.60      0.63        20
      Placed       0.83      0.87      0.85        45

    accuracy                           0.78        65
   macro avg       0.75      0.73      0.74        65
weighted avg       0.78      0.78      0.78        65
```

The K-Nearest Neighbor (KNN) model displays a train score of 0.91 and a test score of 0.78, suggesting overfitting. Despite an accuracy score of 0.78, the model's recall and precision scores are relatively low, standing at 0.73 and 0.75, respectively. This indicates that the model's performance is not optimal and may be affected by the overfitting issue. To address this, various techniques such as adjusting the value of K or applying feature selection may be employed to improve the model's generalization performance. By reducing overfitting and increasing generalization, the model's accuracy, recall, and precision can be improved, resulting in more dependable predictions.

*6.2.5. Naive Bayes:*

```
print('Training Score:', bayes_base.score(X_train, y_train))
print('Test Score:', bayes_base.score(X_test, y_test))
print('-----------------------------------------------------')
print(classification_report(y_test, bayes_base.predict(X_test)))
```

```
Training Score: 0.84
Test Score: 0.8461538461538461
-----------------------------------------------------
              precision    recall  f1-score   support

  Not Placed       0.78      0.70      0.74        20
      Placed       0.87      0.91      0.89        45

    accuracy                           0.85        65
   macro avg       0.83      0.81      0.81        65
weighted avg       0.84      0.85      0.84        65
```

The Naive Bayes model exhibits a slight underfit, as the training score is lower than the test score at 0.84 and 0.846, respectively. The model's accuracy, recall, and precision scores are relatively high, standing at 0.85, 0.81, and 0.83, respectively. While underfitting is less common than overfitting, it may still negatively impact the model's performance by reducing its ability to capture the complexity of the data. To address this, techniques such as increasing the model's complexity or utilizing alternative algorithms may be applied to enhance its performance. By reducing underfitting, the model's accuracy, recall, and precision can be further improved, resulting in more dependable predictions.

*6.2.6. Gradient Boosting:*

```
print('Training Score:', gb_base.score(X_train, y_train))
print('Test Score:', gb_base.score(X_test, y_test))
print('-----------------------------------------------------')
print(classification_report(y_test, gb_base.predict(X_test)))
```

```
Training Score: 1.0
Test Score: 0.8307692307692308
-----------------------------------------------------
              precision    recall  f1-score   support

  Not Placed       0.76      0.65      0.70        20
      Placed       0.85      0.91      0.88        45

    accuracy                           0.83        65
   macro avg       0.81      0.78      0.79        65
weighted avg       0.83      0.83      0.83        65
```

The Gradient Boosting model displays a train score of 1.0 and a test score of 0.83, suggesting overfitting. The model's accuracy score is 0.83, while its recall and precision scores are relatively low at 0.78 and 0.81, respectively. Overfitting may be addressed by reducing the learning rate, increasing the regularization parameter, or using early stopping. These techniques can help prevent the model from fitting to the noise in the training data, allowing it to generalize better on new, unseen data. By reducing overfitting, the model's accuracy, recall, and precision can be enhanced, resulting in more reliable predictions.

*6.2.7. Logistic Regression:*

```
print('Training Score:', logistic_base.score(X_train, y_train))
print('Test Score:', logistic_base.score(X_test, y_test))
print('-----------------------------------------------------')
print(classification_report(y_test, logistic_base.predict(X_test)))
```

```
Training Score: 0.8733333333333333
Test Score: 0.8461538461538461
-----------------------------------------------------
              precision    recall  f1-score   support

  Not Placed       0.78      0.70      0.74        20
      Placed       0.87      0.91      0.89        45

    accuracy                           0.85        65
   macro avg       0.83      0.81      0.81        65
weighted avg       0.84      0.85      0.84        65
```

The Logistic Regression model displays a training score of 0.87 and a test score of 0.84, indicating a good fit. The model's accuracy, recall, and precision scores are relatively high, standing at 0.85, 0.81, and 0.83, respectively. A good fit suggests that the model has found a balance between underfitting and overfitting, allowing it to generalize well to new data. However, it is still important to evaluate the model's performance using additional metrics and techniques such as cross-validation to ensure its reliability. By assessing the model's performance through various means, the accuracy, recall, and precision scores can be further improved.

Out of the seven base models, the logistic regression model appears to perform the best since it is the only model that displays a good fit. This indicates that it can generalize well to new data, suggesting that it may be the most reliable model for making predictions. However, it is still crucial to consider the results of all seven models to gain a better understanding of their performance.

The Support Vector Machine (SVM) model shows a slight overfit with accuracy, recall, and precision scores of 0.85, 0.81, and 0.83, respectively. The Decision Tree model exhibits overfitting, with accuracy, recall, and precision scores of 0.80, 0.73, and 0.75, respectively. The Random Forest model also appears to be overfit with accuracy, recall, and precision scores of

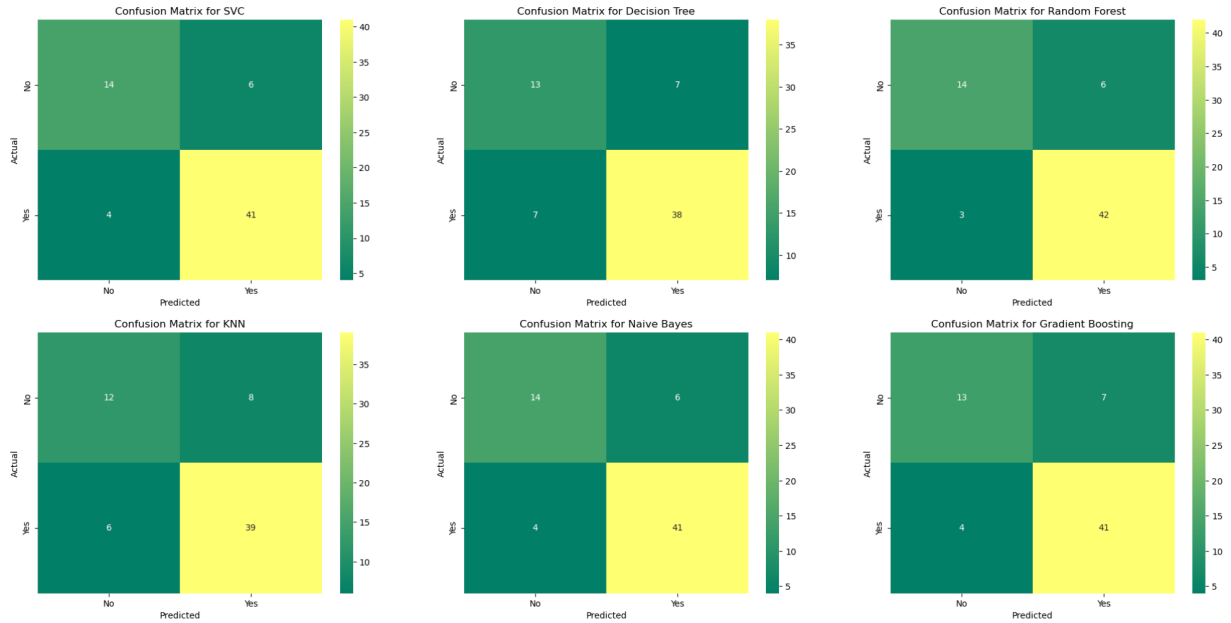
0.86, 0.79, and 0.83, respectively. The K-Nearest Neighbor (KNN) model is also overfit with accuracy, recall, and precision scores of 0.78, 0.73, and 0.75, respectively.

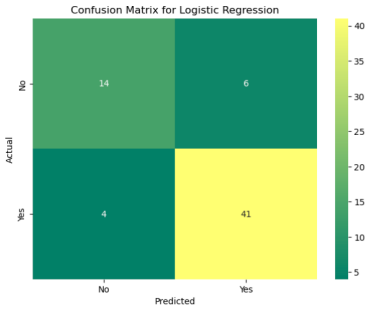
The Naive Bayes model displays a slight underfit, as the training score is lower than the test score, but the accuracy, recall, and precision scores are relatively high at 0.85, 0.81, and 0.83, respectively. Finally, the Gradient Boosting model shows overfitting, with accuracy, recall, and precision scores of 0.83, 0.78, and 0.81, respectively.

Overall, while the logistic regression model is the best fit, it is still vital to consider the other models' performance to ensure that the chosen model is reliable and produces accurate predictions. Techniques such as cross-validation and parameter tuning can also be employed to enhance the models' performance and prevent overfitting or underfitting. By assessing the models' results using various means, a more dependable and accurate model can be developed, which can be utilized to make predictions with greater certainty.

### 6.3. Confusion Matrix:

A table called a confusion matrix is frequently used to assess how well a classification model is working. In a table format, it displays the proportion of true positives, true negatives, false positives, and false negatives. The instances where the model accurately predicted the positive and negative classes, respectively, are referred to as true positives and true negatives. The instances where the model incorrectly predicted the class are known as false positives and false negatives. Precision, recall, and F1-score are a few performance metrics that can be computed using the confusion matrix.






Based on the confusion matrix shown above, it appears that the logistic regression model, Support Vector Machine (SVM), and Naive Bayes models all have identical data. This similarity could be due to the relatively small size of the dataset. Since the dataset is small, it is possible that the models are not able to capture the nuances of the data, leading to identical confusion matrices. However, it is still important to evaluate the models' performance using various metrics to ensure that the selected model can produce reliable predictions.

## 7. Model Improvments:

The next step in the analysis involves improving the models by adding hyperparameter tuning. This process aims to find the optimal combination of hyperparameters for each model, which can lead to better performance and more accurate predictions. By fine-tuning the hyperparameters, the models can be adjusted to better capture the nuances of the data, resulting in more reliable and accurate outcomes. This section is crucial in optimizing the models' performance and ensuring that they are capable of producing high-quality predictions.

Hyperparameter tuning is an essential step in improving the performance of machine learning models. The process involves adjusting the hyperparameters of the models to identify the optimal combination that produces the best results. By fine-tuning these parameters, models can better capture the underlying patterns in the data, leading to more accurate and reliable predictions.

Hyperparameter tuning is a crucial step in machine learning as it ensures that the selected model is optimized for the given dataset. By adjusting the hyperparameters, the models can be adapted to handle specific challenges within the dataset, such as imbalanced classes, noisy data, or complex patterns. This process is essential in ensuring that the models can generalize well to new data and produce accurate predictions.

There are different techniques that can be used for hyperparameter tuning, including grid search, random search, and Bayesian optimization. These methods aim to find the optimal hyperparameters for the models by searching through a predefined parameter space or using a probabilistic approach. It is essential to validate the performance of the models using different metrics, such as accuracy, precision, recall, and F1 score, to determine the best combination of hyperparameters for the given dataset.

The next step in the model improvement process is to use GridSearchCV with 5 cross-validations for all models used. GridSearchCV is a powerful tool for hyperparameter tuning in machine learning. It explores the hyperparameter space exhaustively to find the best combination of hyperparameters that produces optimal performance for a given model and dataset.

GridSearchCV requires a set of hyperparameters to search over, a model, and a training dataset as input. It returns the best set of hyperparameters that maximizes the cross-validated score on the training data. By using GridSearchCV, we can fine-tune the hyperparameters of the models, and achieve the best possible performance from each model.

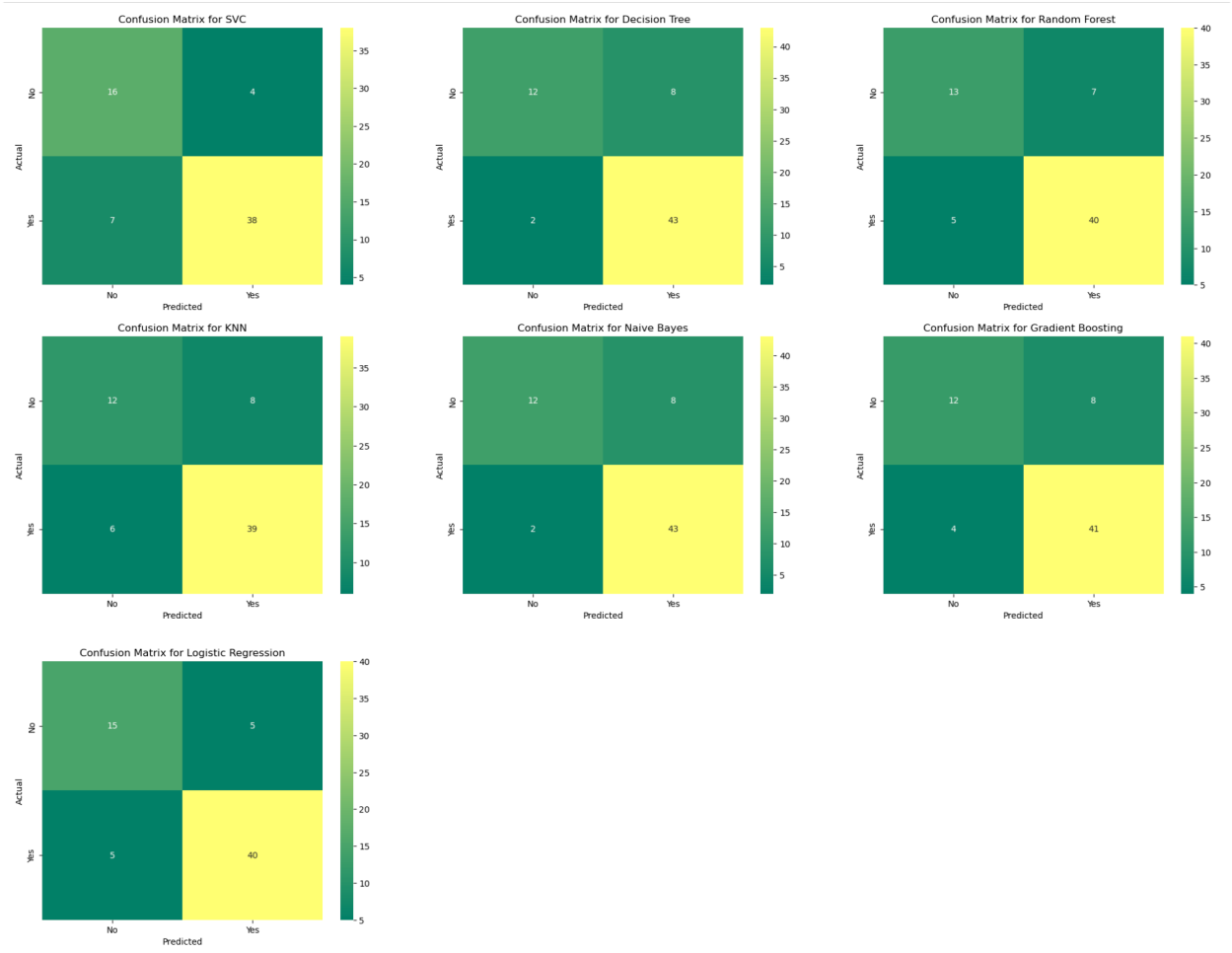
The process of hyperparameter tuning is critical in machine learning, as it can significantly improve the model's accuracy, precision, and recall. Fine-tuning the hyperparameters helps to optimize the model, and results in better prediction results on unseen data.

Overall, using GridSearchCV for hyperparameter tuning is a crucial step in the machine learning pipeline. It enables us to optimize the model's performance and achieve better results, which is especially important when dealing with complex and large datasets. By tuning the hyperparameters, we can achieve better results in a shorter time, and make more accurate predictions on new data.

This section discusses the model evaluation results after hyperparameter tuning to find out which model has good performance. The classification report presents the results of tuning each model. Among the models, the tuned Logistic regression model had the highest accuracy score of 0.85, with a train score of 0.86 and a test score of 0.84. The model shows good fitting with a recall score of 0.82 and a precision score of 0.82. Then Decision Tree model also performed well, with an accuracy score of 0.80, a train score of 0.80 and a test score of 0.80. It had a recall score of 0.72 and a precision score of 0.79.

The Random Forest model had a train score of 1.0 and a test score of 0.84, indicating overfitting. It had an accuracy score of 0.85, a recall score of 0.81 and a precision score of 0.83. The KNN model had a train score of 0.91 and a test score of 0.78, indicating overfitting. It had an accuracy score of 0.78, a recall score of 0.73 and a precision score of 0.75. The Naive Bayes model had a train score of 0.83 and a test score of 0.884, indicating underfitting. It had an accuracy score of 0.85, a recall score of 0.78 and a precision score of 0.85. The gradient boosting model had a train score of 0.98 and a test score of 0.84, indicating overfitting. It had an accuracy score of 0.85, a recall score of 0.79 and a precision score of 0.83. Lastly, the Logistic Regression model had a train score of 0.86 and a test score of 0.84, indicating good fitting. It had an accuracy score of 0.85, a recall score of 0.82 and a precision score of 0.82.

Overall, the hyperparameter tuning improved the performance of the models, but the Logistic Regression model appeared to be the most promising for use in making predictions for the given dataset. However, it is important to note that the model evaluation and selection process should be based on more than just the accuracy score, but also on the specific requirements and objectives of the problem being solved.




## 8. Overall analysis model evaluation:

The model evaluation results for both the base model and the improvement model show that logistic regression had the best performance among the base models, with a train score of 0.87 and a test score of 0.84. This model was the only one that showed good fitting, as the other five models suffered from overfitting. Naive Bayes, on the other hand, showed underfitting but had a train score and test score of 0.84 and 0.846 respectively, where the train score was slightly lower than the test score.

After hyperparameter tuning, most of the models in the improvement model demonstrated performance improvements. For instance, the SVC and Decision Tree models, which were overfitting in the base model, became good fitting after tuning. The KNN and Naive Bayes models did not show any improvement in their performance. The gradient boosting model improved its performance but remained overfitting, while logistic regression remained good fitting with a narrow difference in train scores and test scores.

Overall, hyperparameter tuning helped improve the performance of the majority of the models, with some of them transitioning from overfitting to good fitting. However, Naive Bayes remained underfitting, and logistic regression was the only model that showed good fitting in both the base and improvement models.

## 9. Conclusion:

The dataset used in this project is the job placement dataset where companies want to recruit but want quality human resources who have good grades and majors that are appropriate to their job position. It can be seen from the EDA that when someone has a high score on ssc_percentage, hsc_percentage, The degree_percentage level of acceptance tends to be high, on the other hand work_experience and specialization also affect the acceptance rate of applicants, because these factors can make a person accepted for work or not, so these columns are used as features to be trained on machine models learning.

In this case, the company wants applicants who have the best quality, so a classification model is needed that is focused on recall value, because if the company gets applicants whose quality is below the company's standards and whose quality is predicted to be good, the company will experience losses such as late assignments due to employee incompetence, so recall or false negative values should be increased (predictive error values are minimized or eliminated)

then made 7 different models with hyperparameter tuning namely SVC, KNN, Naive Bayes, Logistic Regression, Decision Tree, Random Forest, Gradient Boosting where the model with the best performance is Logistic Regression with hyperparameters (C = 2, max_iter = 300, penalty = l2, and solver = lbfgs) this model has a train score of 0.87 and a test score of 0.84, an accuracy of 0.85 and an average value of precision and recall 0.82

## 10. References :


1. “Dataset”: https://www.kaggle.com/datasets/ahsan81/job-placement-dataset

2. "Machine Learning for Job Matching in Online Recruitment." Journal of Big Data, by C. Yang, X. Zhang, and S. Zhu (2018). https://doi.org/10.1186/s40537-018-0164-4

3. "Predicting Job Placement for College Graduates Using Machine Learning." IEEE Transactions on Education, by D. E. Holmes, D. R. Johnson, and R. E. Beck (2019). https://doi.org/10.1109/TE.2019.2955286

4. "Applicant screening using machine learning techniques." International Journal of Human Resource Management, by H. P. Tseng, C. H. Chen, and Y. T. Wang (2010). https://doi.org/10.1080/09585191003611629

5. "Predictive Analytics in Human Resource Management: A Review of the Literature and Directions for Future Research." Human Resource Management Review, by M. H. Bondarouk, M. Ruel, T. J. Olivas-Lujan, and H. A. Guiderdoni-Jourdain (2020). https://doi.org/10.1016/j.hrmr.2019.07.002