

Kheshini BUDHNA

C0909662

## Table of contents

Project overview .....	3
Dataset description.....	3
Preprocessing steps .....	4
Data structure and addressing missing values. ....	4
Data visualization .....	5
Train and test data split .....	9
Categorical data encoding .....	9
Class ratio.....	10
SMOTE implementation .....	11
Standardization of the numerical columns .....	12
Model Selection.....	13
Model training.....	14
Random Forest.....	14
XGBoost Model.....	16
Logistic Regression.....	19
Voting classifier .....	22
Conclusion.....	27

## Project overview

This project essentially aims at predicting whether a student gets recruited during campus placements based on the different attributes and features available in the student dataset. this would generally encompass attributes of their academic performance, skills, extracurricular activities, work experience, among other factors that might influence the employability of a student. the project analyzes this data to develop a machine learning model that can predict the chances of a student getting employed during campus recruitment.

## Dataset description

The dataset has been sourced from kaggle, and can be found on the following link: [campus recruitment prediction \(course project\) | kaggle](#). the dataset (train.csv) contains the following columns:

Column name:	Description of column content:	Options if categorical data:
ssc_p	Secondary Education percentage	
ssc_b	Secondary Education board	Central or Others
hsc_p	Higher Secondary Education percentage	
<b>hsc_b</b>	Higher Secondary board	Central or Others
hsc_s	Higher Secondary specialization	Commerce, Science, Arts
degree_p	Degree percentage	
degree_t	Degree type	Sci&Tech, Comm&Mgmt
workex	Work experience	Yes or No
etest_p	e-test percentage	
specialisation	MBA specialisation	Mkt&HR, Mkt&Fin
mba_p	MBA percentage	
status	Placement status	Placed or Not Placed
salary	Salary offered	

## Preprocessing steps

### Data structure and addressing missing values.

First step of the data preprocessing was EDA, which is the process of analyzing and visualizing the data to summarize its big characteristics, identify patterns, relationships, and potential anomalies, and inform the selection of appropriate data preprocessing techniques. To start with it, the data structure of the dataset was explored. The dataset had 215 rows and 15 columns. Using the `info()` function of the pandas, the summary information of the dataset was outputted.

```
[203] # Get summary information about the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 15 columns):
 #   Column                Non-Null Count  Dtype  
---  --
 0   sl_no                 215 non-null   int64  
 1   gender                215 non-null   int64  
 2   ssc_p                 215 non-null   float64 
 3   ssc_b                 215 non-null   object  
 4   hsc_p                 215 non-null   float64 
 5   hsc_b                 215 non-null   object  
 6   hsc_s                 215 non-null   object  
 7   degree_p              215 non-null   float64 
 8   degree_t              215 non-null   object  
 9   workex                215 non-null   object  
10  etest_p               215 non-null   float64 
11  specialisation         215 non-null   object  
12  mba_p                 215 non-null   float64 
13  status                 215 non-null   object  
14  salary                148 non-null   float64 
dtypes: float64(6), int64(2), object(7)
memory usage: 25.3+ KB
```

Figure 1. use of `info()` on dataset

Using the `describe()` function from pandas to output the descriptive statistics of the dataset, it was found that the 'Salary' column had missing values. Since the 'Salary' column was dependent on the 'Status' column and was not needed as an input for predicting the target variable, it was dropped.

## Data visualization

Following that, the data visualization of the dataset was done, where the categorical and numerical features were plotted according to their characteristics. Below are examples of the categorical data that was plotted:

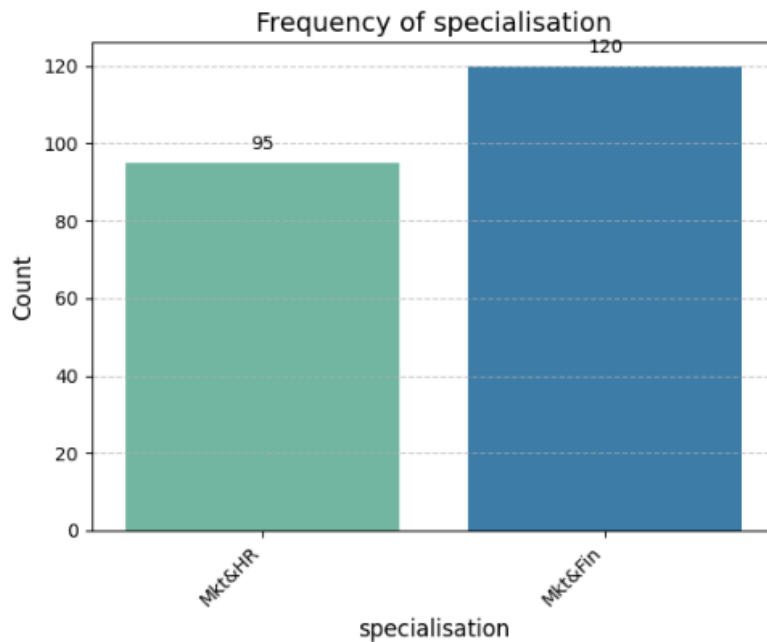


Figure 2. Specialisation column plotted

This bar graph represents the distribution of two sets of specializations, Mkt&HR and Mkt&Fin, represented by 95 and 120 students, respectively. Mkt&Fin appears to be the more popular specialization, having 120 students and comprising about 56% of the total population. Mkt&HR makes up the rest of the 44% of the students and comprises a total of 95 students. Clear labeling and an uncluttered two-bar format make it easy to see that there are 25 students in total between the two specializations. The relatively close numbers suggest that both specializations are well-subscribed, though with a slight preference for the Finance combination over Human Resources.

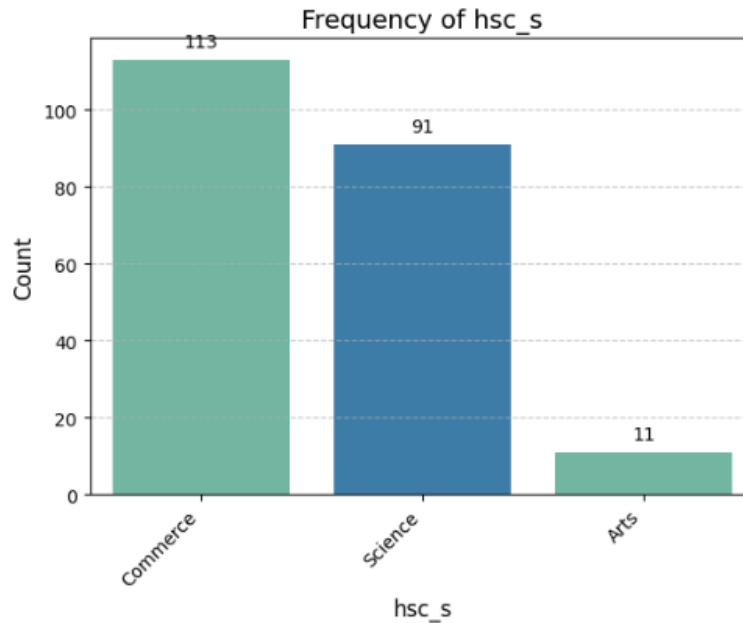
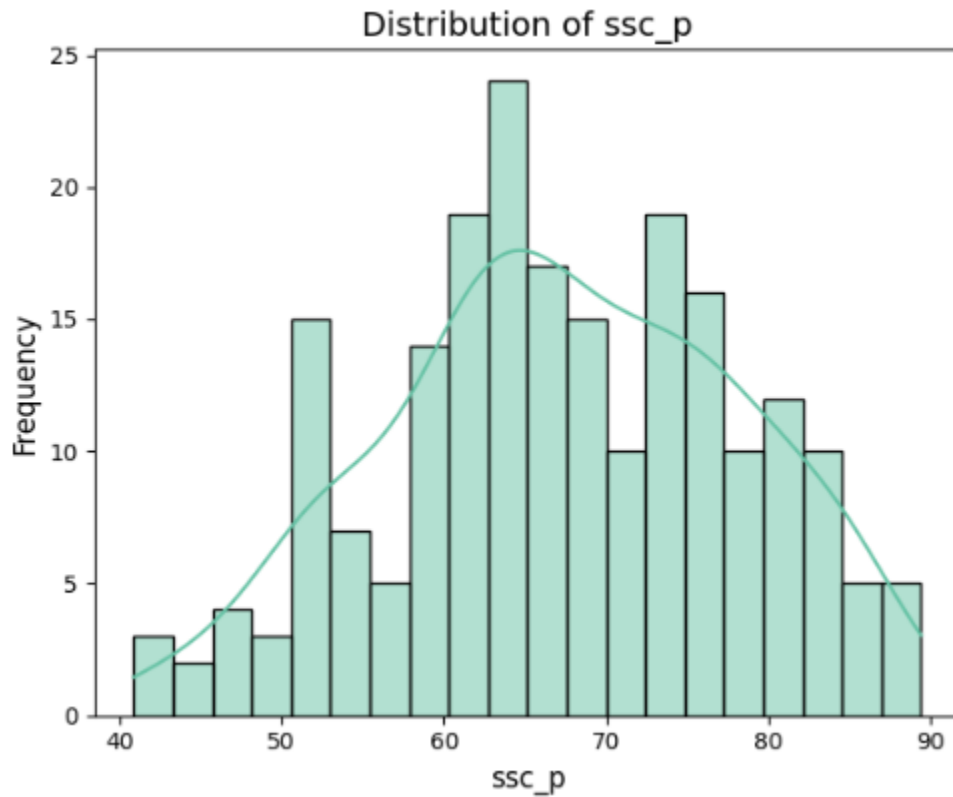


Figure 3. hsc\_s column plotted

This bar chart depicts the distribution of HSC streams into three sets of categories: Commerce, Science, and Arts. In this aspect, Commerce is leading the race with a frequency of 113 students, approximately 53% of the total population. Science comes out to be the second popular stream in this regard with 91 students, composing about 42% of the total. On the other hand, Arts have an extremely low intake of 11 students or about 5% of the total number of students. Looking at this huge gap between Arts and the other two streams, one may say this is data from an institution or program which focuses primarily upon Commerce and Science disciplines. What this means is that the combined strength of Commerce and Science, at 204, far outweighs that of Arts at 11. The reasonably close numbers between Commerce and Science do show that both segments are well established and enjoy a popularity among students in almost equal measure. This distribution might reflect either student preference, admission policies, or the nature of the subsequent higher education program these students get into.

Below are examples of the numerical columns that were plotted:



*Figure 4. Secondary school percentage plotted*

This histogram represents the distribution of secondary school percentage scores, ranging from 40 to 90. The distribution appears roughly bell-shaped with multiple peaks, with the major peak occurring between 65 and 70, indicating that most students score within this range. The overall distribution is somewhat right-skewed, with higher-performing outliers pulling the mean above the median. Most scores fall between 55 and 85, and fewer students score in the tails, either below 50 or above 85. The presence of multiple modes in the histogram suggests distinct groups of students performing differently.

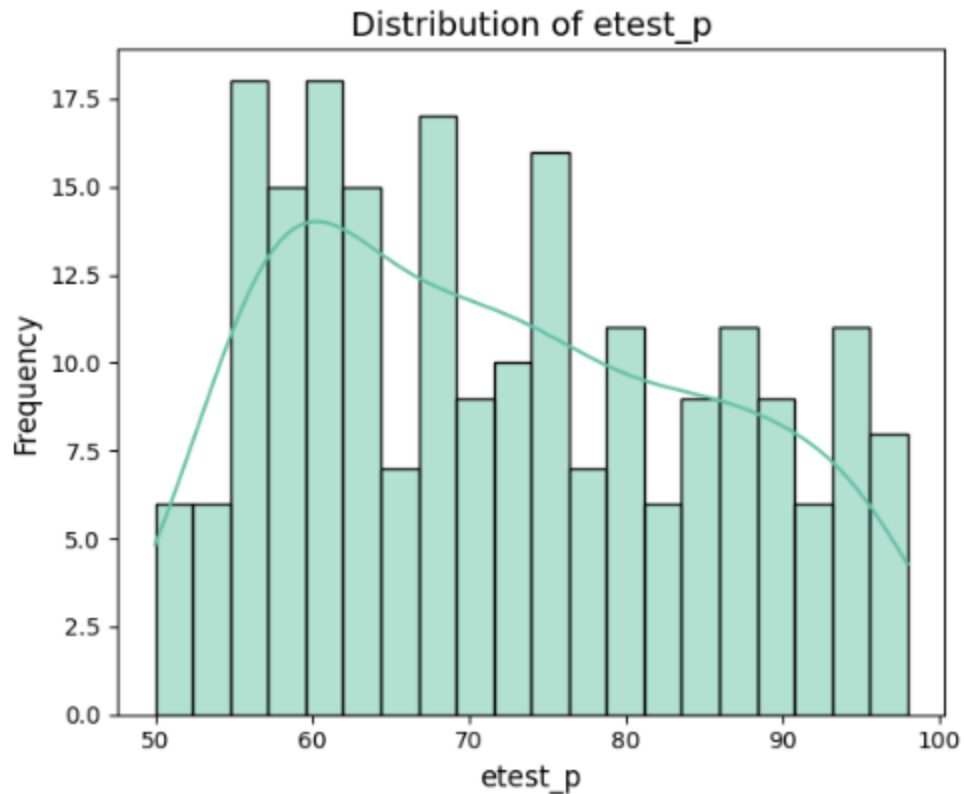


Figure 5. e-test distribution

This histogram shows the distribution of etest\_p, the entrance test percentage, ranging from 50 to 100 percent. The distribution of the data is left-skewed, where in the lower range, the frequencies are the highest. There is a noticeable peak frequency of about 18 students scoring in this lower range, immediately followed by a gradual decline of frequency as the scores increase. There are several smaller peaks in this distribution, suggesting multiple performance clusters among test-takers. There are scores at the very lowest and very highest possible values in the distribution, but there is a strong concentration of students scoring in the 55 to 75 range. Relatively few students achieved scores above 85, which indicates that higher scores were much harder to achieve. The overall shape suggests that this may be a difficult test where very high scores were less common but passing scores - above 50% - were readily achievable.



## Train and test data split

After data visualisation, the dataset was split into test and train in the ratio of 30% to 70% as per the guidelines of the project.

```
[210] # Split data into features (X) and target (y)
      X = df.drop('status', axis=1)
      y = df['status']

      # Split into training and test sets (30% test, 70% train)
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Figure 6. train-test split of the dataset

## Categorical data encoding

Next, the columns of categorical data needed to be encoded using Label Encoding. Label Encoding will just map each category in a column to a unique integer number, hence converting categorical values into a numerical data format. It allows machine learning algorithms, which essentially work with numbers, to process the information accordingly. One must be aware that Label Encoding does not keep any semantic relation between categories beyond numeric order. It could mean that, in this way, the model is considering categorical variables to be quantitative, hence would affect the outcome based on the nature of the data.

```
3.4. Encoding the Categorical Columns (categorical data to numerical data)

Generate randomly select 5 items from a list

[211] from sklearn.preprocessing import LabelEncoder

      # Columns to encode in the features (X)
      columns_to_encode = ['ssc_b', 'hsc_b', 'hsc_s', 'degree_t', 'workex', 'specialisation']

      # Initialize an empty dictionary to store mappings
      label_mappings = {}

      # Apply LabelEncoder and store mappings for feature columns
      for column in columns_to_encode:
          le = LabelEncoder()
          # Fit the encoder on the training data and transform it
          X_train[column] = le.fit_transform(X_train[column])
          # Apply the same transformation to the test data
          X_test[column] = le.transform(X_test[column]) # Use transform only, not fit

          # Store the mapping for each column
          label_mappings[column] = dict(zip(le.classes_, le.transform(le.classes_)))

      # Encode the target variable 'status' separately
      le_status = LabelEncoder()
      y_train = le_status.fit_transform(y_train) # Fit and transform on training set
      y_test = le_status.transform(y_test) # Transform test set using the same encoder

      # Store the mapping for 'status'
      label_mappings['status'] = dict(zip(le_status.classes_, le_status.transform(le_status.classes_)))

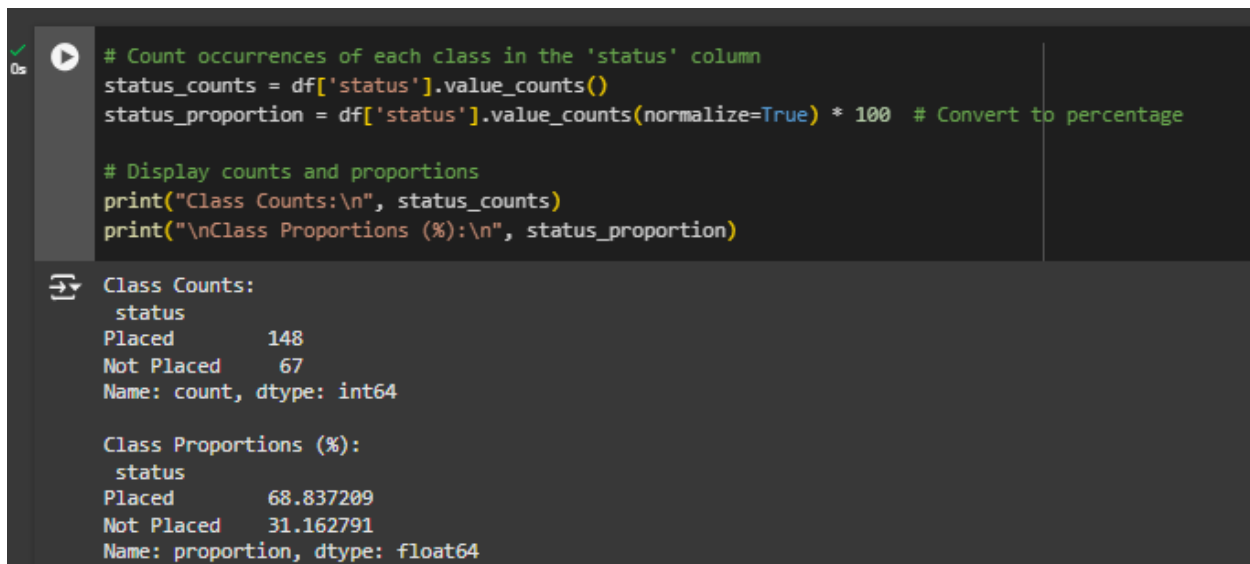
      # Print mappings for each column
      for column, mapping in label_mappings.items():
          print(f"{column} - {mapping}")

ssc_b - {'Central': 0, 'Others': 1}
hsc_b - {'Central': 0, 'Others': 1}
hsc_s - {'Arts': 0, 'Commerce': 1, 'Science': 2}
degree_t - {'Comm&Mgmt': 0, 'Others': 1, 'Sci&Tech': 2}
workex - {'No': 0, 'Yes': 1}
specialisation - {'Mkt&Fin': 0, 'Mkt&HR': 1}
status - {'Not Placed': 0, 'Placed': 1}
```

Figure 7. Categorical columns encoding

## Class ratio

This distribution of the placement status in the dataset signifies that the number of placed students amounts to almost threefold over that of unplaced students, reflected in the class ratio 2:1. Real-world placement data take this structure, but this usually needs cautious handling by oversampling, undersampling, or SMOTE to avoid predicting biases for the majority class. This disproportion suggests that, although overall the place programme has a very high rate of placement, any predictive modeling would need to make provision for this in order not to simply predict the majority class as a default.



```
# Count occurrences of each class in the 'status' column
status_counts = df['status'].value_counts()
status_proportion = df['status'].value_counts(normalize=True) * 100 # Convert to percentage

# Display counts and proportions
print("Class Counts:\n", status_counts)
print("\nClass Proportions (%):\n", status_proportion)
```

Class Counts:

status	
Placed	148
Not Placed	67

Name: count, dtype: int64

Class Proportions (%):

status	
Placed	68.837209
Not Placed	31.162791

Name: proportion, dtype: float64

Figure 8. Target variable ratio exploration

## SMOTE implementation

This code applies the SMOTE algorithm to balance the class distribution of the training data. A random state for reproducibility is set at 42. Originally, the distribution in the training set consisted of 104 instances of class 1 representing placed and 46 instances of class 0 representing not placed, so that there is clear imbalance in the training set. After applying SMOTE, the distribution is perfectly balanced, with 104 instances of each class where additional synthetic examples of the minority class-so-called dummy examples-have been generated to reach the same count as the majority class. This transformation from an imbalanced dataset, 104:46, to a balanced one, 104:104, indicates that SMOTE is truly effective in creating a more balanced dataset to train machine learning models.

```
✓ 3.6. Applying SMOTE to deal with class imbalance.

[213] # Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Convert y_train and y_train_smote back to pandas Series for value_counts
y_train_series = pd.Series(y_train)
y_train_smote_series = pd.Series(y_train_smote)

# Verify the class distribution before and after applying SMOTE
print(f"Class distribution before SMOTE: \n{y_train_series.value_counts()}")
print(f"Class distribution after SMOTE: \n{y_train_smote_series.value_counts()}")
```

```
↔ Class distribution before SMOTE:
1    104
0     46
Name: count, dtype: int64
Class distribution after SMOTE:
0    104
1    104
Name: count, dtype: int64
```

Figure 9. SMOTE implementation.

## Standardization of the numerical columns

At this point, StandardScaler comes in handy, since all these numerical features, ssc\_p, hsc\_p, degree\_p, etest\_p, and mba\_p, are scaled to zero mean and unit variance to prevent features with larger magnitudes-for instance, percentage scores-from dominating the model's learning process. This standardization is important in view of the fact that these percentage scores are on different scales, and many institutions might grade more strictly than others. Most of the machine learning algorithms, such as logistic regression, SVM, or neural networks, are sensitive to the scale of input features. By doing this, we ensure that all features are on an equal footing in contributing to the model's predictions and generally converge faster during training.

### ✓ 3.7. Standardization of columns with integers and floats.

```
[214] # Identify the numerical columns (you can specify them manually or select them automatically)
      numerical_columns = ['ssc_p', 'hsc_p', 'degree_p', 'etest_p', 'mba_p']

      # Initialize the StandardScaler
      scaler = StandardScaler()

      # Apply StandardScaler to the numerical columns in the training data
      X_train[numerical_columns] = scaler.fit_transform(X_train[numerical_columns])
      X_test[numerical_columns] = scaler.transform(X_test[numerical_columns])
```

Figure 10. Standardization of numerical data

## Model Selection

In predicting a categorical target "status" - values include "Placed" and "Not Placed" - Random Forest Classifier is a strong choice, since this algorithm can handle both numerical and categorical features very well. Random Forest is another ensemble method where multiple decision trees are created and then improve accuracy by averaging, with less risk of overfitting. It is robust to complex relationships and, as an added benefit, feature importance gives deeper insight into the model's decisions. Random Forest works well in situations where high accuracy and stability from a wide range of data are important, making the algorithm useful for classification tasks.

XGBoost is the optimized version of gradient boosting. It is a well-recognized algorithm in classification tasks of structured data, which constructs decision trees one after another and rectifies the deficiencies of previous ones. For its extraordinary performance in several competitions and industrial applications, especially with imbalanced datasets, XGBoost can be very effective for tasks where high predictive performance is needed. In this regard, it is ideal for situations requiring maximum accuracy; thus, it works great where complex patterns come into play, making it suitable for tough datasets.

Logistic Regression is another good candidate, especially when the relationship between the features and the target variable is approximately linear. It is a simple, interpretable model that works well as a baseline in binary classification tasks. Although Logistic Regression may not capture complex non-linear relationships, it is computationally efficient and easy to interpret. This model is beneficial for quickly assessing how well the data can be separated along a linear decision boundary, and it provides a solid starting point before applying more complex models like Random Forest or XGBoost.

# Model training

## Random Forest

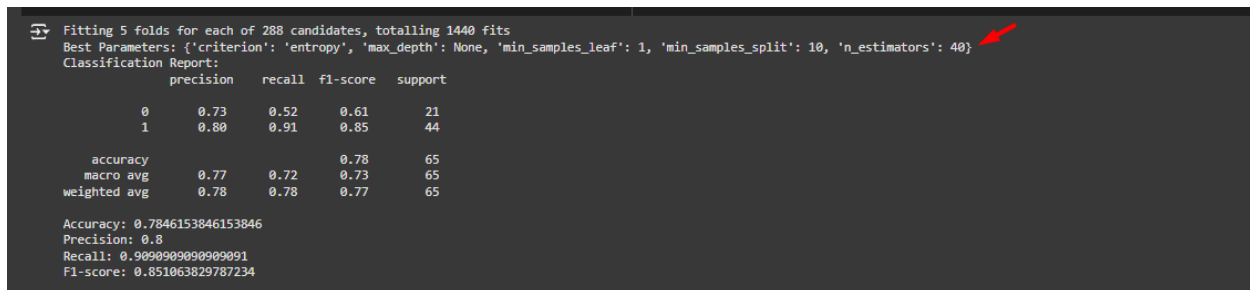


Figure 11: Random Forest metrics and hyper tuning

The model thus was cross validated with 5 folds for each of the 288 candidate parameter sets for a total of 1440 fits. The best set of hyperparameters proved to be { 'criterion': 'entropy', 'max\_depth': None, 'min\_samples\_leaf': 1, 'min\_samples\_split': 10, 'n\_estimators': 40 }. These parameters were picked based on performance metrics from the cross-validation process and resulted in a classification model with an overall accuracy of about 78%.

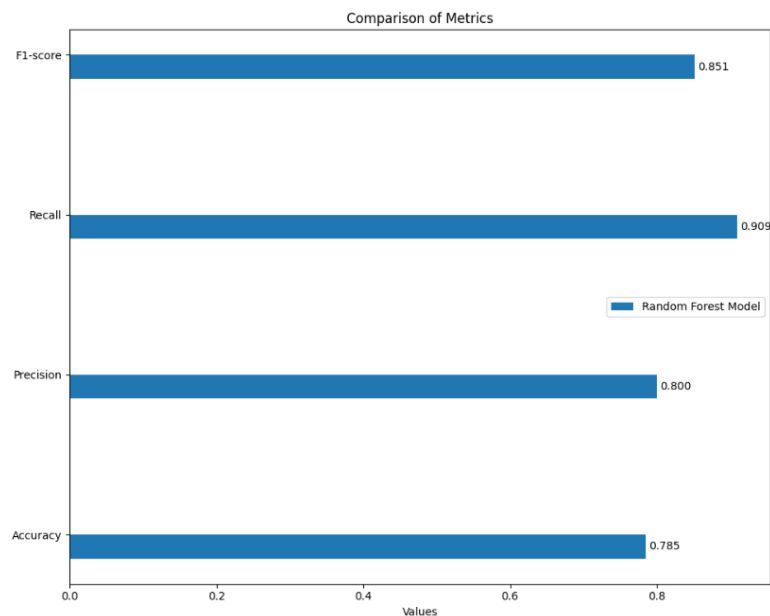


Figure 12. Metrics for Random Forest

The classification report shows the model performance, with considerable variance in class "1" (e.g., "Placed"), which, again, is considered a positive class. In this class, the model's precision is 0.80; that is, 80% of the model's predictions for "Placed" are correct. Also, the recall for the "1" class is 0.91, showing that out of all the actual instances of "Placed", the model correctly predicted 91%. This results in a high F1-score of 0.85 for

the "1" class, being a balanced measure of both precision and recall; hence, this model is effective in predicting this class with very few false positives and false negatives.

On the contrary, the model performance for the "0" class-for example, "Not Placed"-is comparative and weaker. Recall for the "0" class is 0.52, which means that the model identifies only 52% of actual instances of "Not Placed"; hence, there are more false negatives for this class. Hence, its F1-score turns out to be 0.61 for the "0" class, which shows the low accuracy in the prediction of this class. Despite this variance within classes, the overall accuracy of the model remains 0.78, correctly classifying 78% of all instances. It turns out that a weighted average F1-score is 0.77, which suggests that the model is skewed but the performance is still alright.

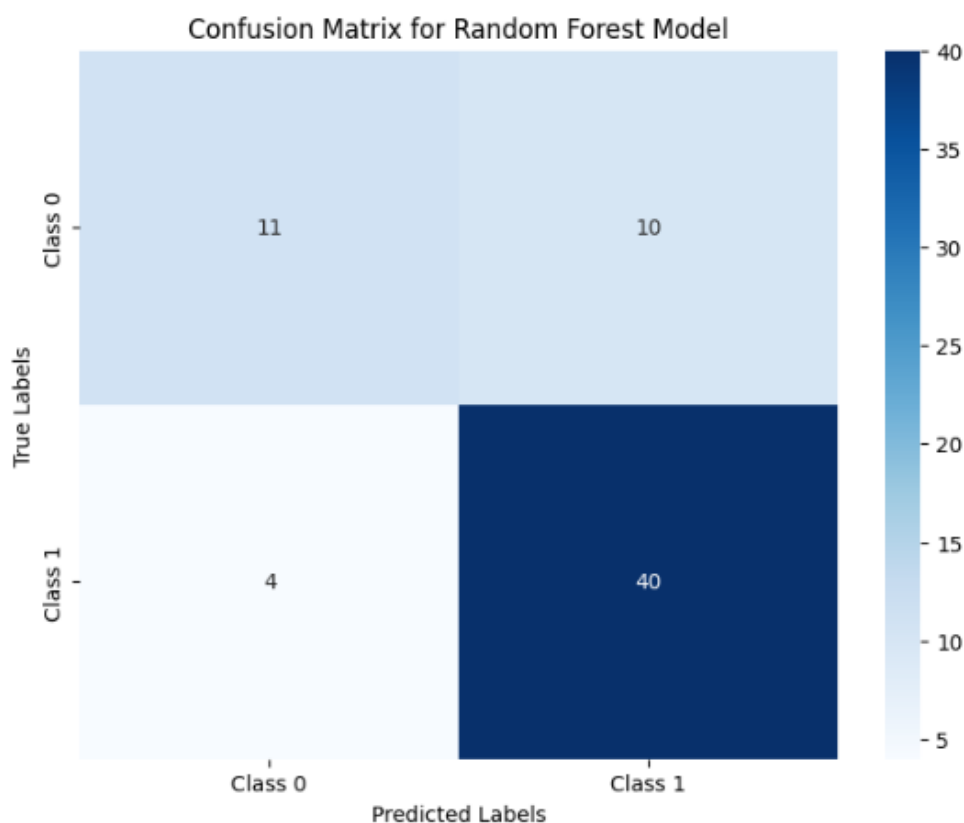


Figure 13. Confusion matrix of testing data (Random Forest)

The confusion matrix in this case will indicate that the model is relatively good at picking out instances of "Placed" (True Positives). With its high number of 40 correctly predicted "Placed" students, the model really does excel in that class. However, the number of False Negatives, corresponding to 4, accounts for those "Placed" students who are not correctly predicted and hence missed by the model. The results are not that good for the "Not Placed" category. While it rightly predicts "Not Placed" for 11 instances, it has also wrongly classified 10 "Not Placed" students as "Placed." This shows that the model is biased

toward placing students who have not been placed, which may raise several concerns in situations where such misjudgment can have serious repercussions.

## XGBoost Model

```
Fitting 5 folds for each of 2916 candidates, totalling 14580 fits
Best Parameters: {'colsample_bytree': 1.0, 'gamma': 0.1, 'learning_rate': 0.2, 'max_depth': 3, 'min_child_weight': 3, 'n_estimators': 50, 'subsample': 0.8}
Classification Report:
      precision    recall  f1-score   support

     0       0.76      0.62      0.68        21
     1       0.83      0.91      0.87        44

 accuracy: 0.8153846153846154
 macro avg: 0.80      0.76      0.78        65
 weighted avg: 0.81      0.82      0.81        65

Accuracy: 0.8153846153846154
Precision: 0.8333333333333334
Recall: 0.9090909090909091
F1-score: 0.8695652173913043
```

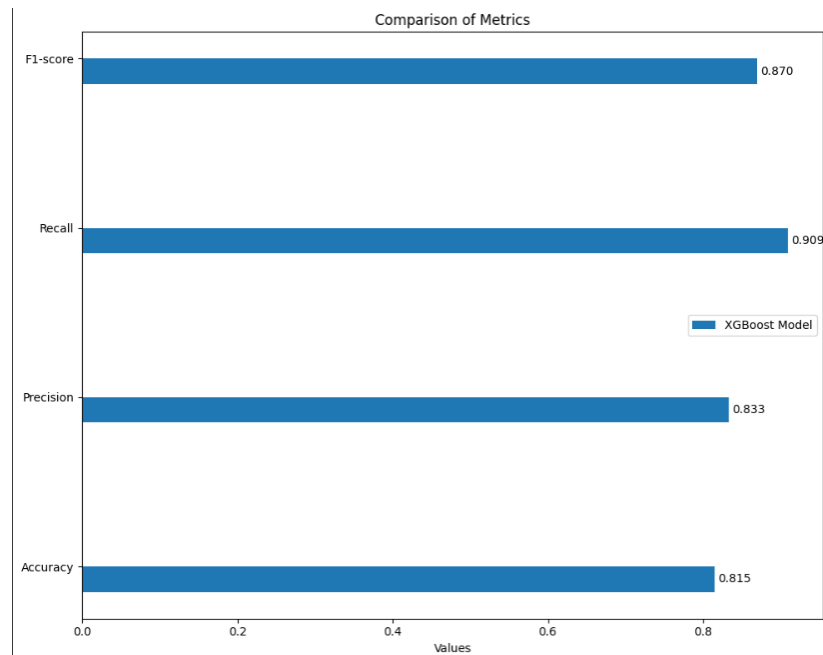
Figure 14. XGBoost Model hyper tuning and metrics

The model thus was cross validated with 5 folds for each of the 2916 candidate parameter sets for a total of 14580 fits. The best set of hyperparameters proved to be { 'colsample\_bytree': 1.0, 'gamma': 0.1, 'learning\_rate': 0.2, 'max\_depth': 3, 'min\_child\_weight': 3, 'n\_estimators': 50, 'subsample': 0.8 }. These parameters were picked based on performance metrics from the cross-validation process and resulted in a classification model with an overall accuracy of about 81.5%. The classification report shows precision, recall, and F1-scores as follows:

- Precision: 0.76 for class 0, and 0.83 for class 1
- Recall: 0.62 for class 0, and 0.91 for class 1
- F1-score: 0.68 for class 0, and 0.87 for class 1.

The model's accuracy was 81.5%, with a weighted average F1-score of 0.81, showing good overall performance across both classes.





*Figure 15. XGBoost Metrics plotted*

The classification report shows the performance of the models, with very strong results in both classes. In the "1" class-as one might refer to as the positive class, for example-the precision reaches a value of 0.82, therefore 82% of its predictions for this class are correct. Its recall was 0.91, which means that 91% of actual instances of this class will be correctly identified by this model. This results in a very high F1-score of 0.86, indicating a balanced measure of precision and recall of the class in question; this class is predicted quite well by this model with a minimum of false positives and false negatives.

In the "0" class, the model has performed a little weaker. While precision is good at 0.76, recall is lower at 0.62; hence, it identifies only 62% of the actual instances of the class "0". This results in a lower F1-score of 0.68 for class "0", thus reflecting lesser accuracy in its prediction with a greater degree of false negatives.

Therefore, with such variability across class performance, the model will have an overall accuracy of 80%, meaning that it will classify 80% of all instances correctly. The weighted average F1-score is 0.81, which already shows that the performance of the model is quite balanced between both classes, with slight bias toward class "1" because of higher recall.

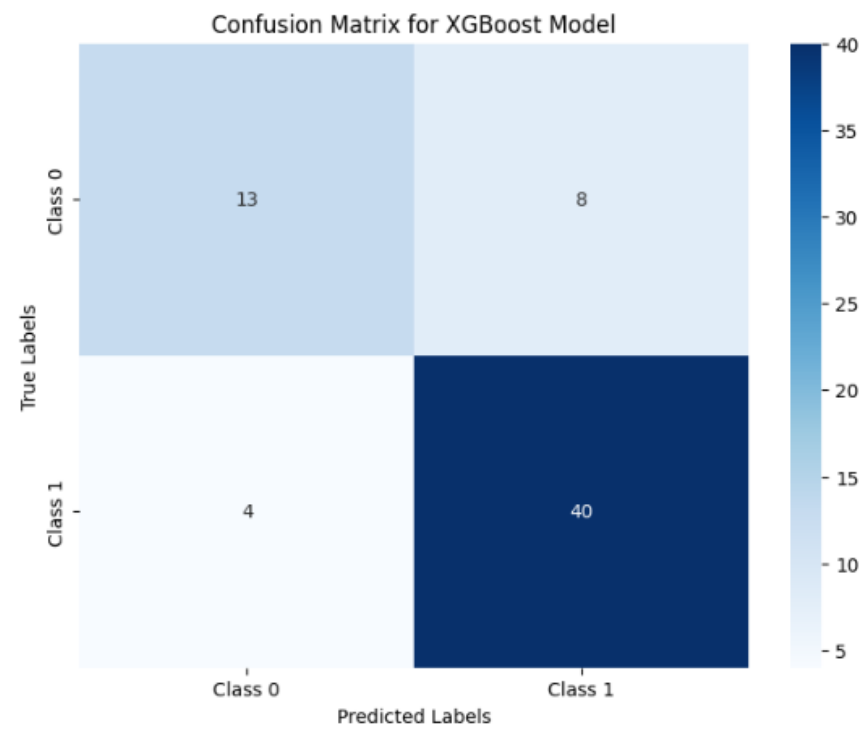


Figure 16. Confusion matrix for XGBoost

The Confusion matrix shows that this model has been performant enough in terms of identifying the "Placed" instances as true positives. Given the scenario that several students get placed, the model is super performant and has correctly predicted 40 "Placed" students. The count of False Negatives is 4 - where the model will fail to identify some "Placed" students, which will miss the predictions.

Regarding this, the model has a good and bad performance in terms of "Not Placed." For instance, 13 "Not Placed" students were classified as "Not Placed," while 8 "Not Placed" students were wrongfully classified as "Placed." This shows that the model is somehow biased to predict "Placed," which, in many scenarios, can be problematic. These False Positives can lead to unnecessary interventions or misclassifications of students from the "Not Placed" group.

## Logistic Regression

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Best Parameters: {'C': 1, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}
Classification Report:
      precision    recall  f1-score   support

     0       0.75      0.57      0.65        21
     1       0.82      0.91      0.86        44

 accuracy          0.80          65
  macro avg       0.78      0.74      0.75        65
 weighted avg     0.79      0.80      0.79        65

Accuracy: 0.8
Precision: 0.8163265306122449
Recall: 0.9090909090909091
F1-score: 0.8602150537634409
```

Figure 17. Logistic regression hyper tuning and metrics

Tuning was performed using cross-validation with 5 folds, testing 48 candidate parameter sets for a total of 240 fits. The best set of hyperparameters selected was {'C': 1, 'max\_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}. These parameters are really good to get sparse feature selection, particularly the regularization parameter being  $C=1$  and the L1 penalty and reduce overfitting. The 'liblinear' solver selected is very efficient on small datasets or models and involves L1 regularization, hence suitable for the classification problem at hand.

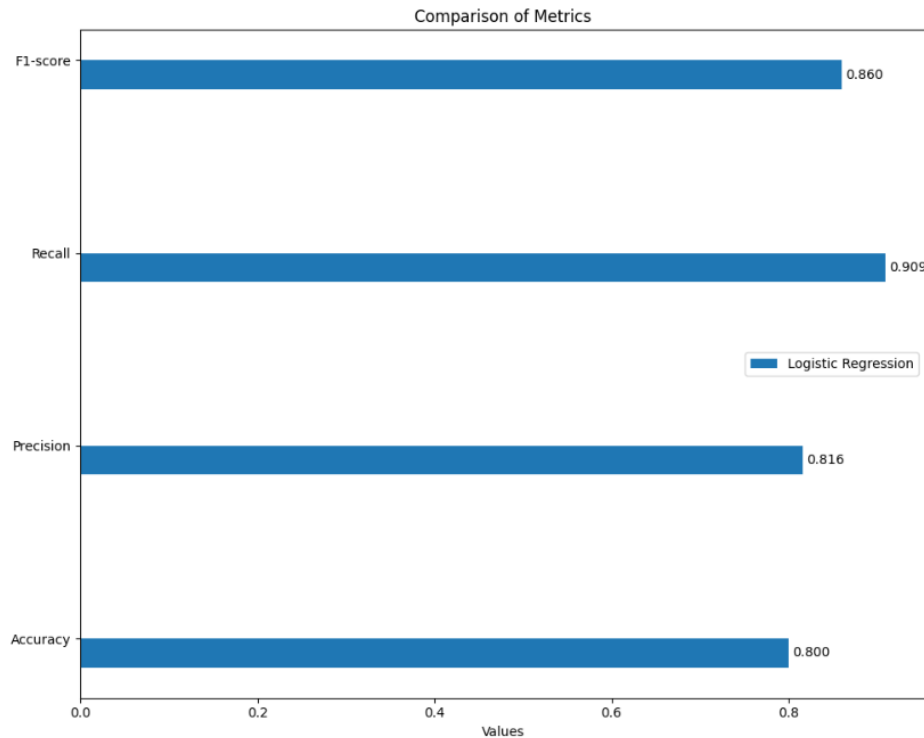


Figure 18. Metrics for logistic regression

The model's evaluation metrics for logistic regression seem to perform quite well, especially for identifying the positive class, which we have assumed is "Placed."

This **accuracy** of 0.80 in the overall model correctly classified instances, showing generally reliable performance throughout the dataset. Further, with a **precision** of approximately 0.82, this would mean that when the model predicts a student as "Placed," it is correct about 82% of the time. That would indicate this model is good at minimizing false positives-meaning that it has relatively low tendencies of taking "Not Placed" students and classifying them as "Placed."

This would mean that with a recall of about 0.91, the model correctly identifies 91% of actual "Placed" students and is hence highly sensitive in capturing the students who are "Placed." This could be useful in cases where it is important to capture as many instances of "Placed" as possible. Finally, the **F1-score**-close to 0.86-stands for a balance between precision and recall, confirming that the model keeps a strong balance between the accuracy of the positive predictions it makes and the completeness of true positives it captures. These metrics taken together suggest that the logistic regression model is especially well-placed for making accurate predictions on "Placed" instances with a balanced trade-off between precision and recall; hence, this model has proven effective for binary classification in this context.

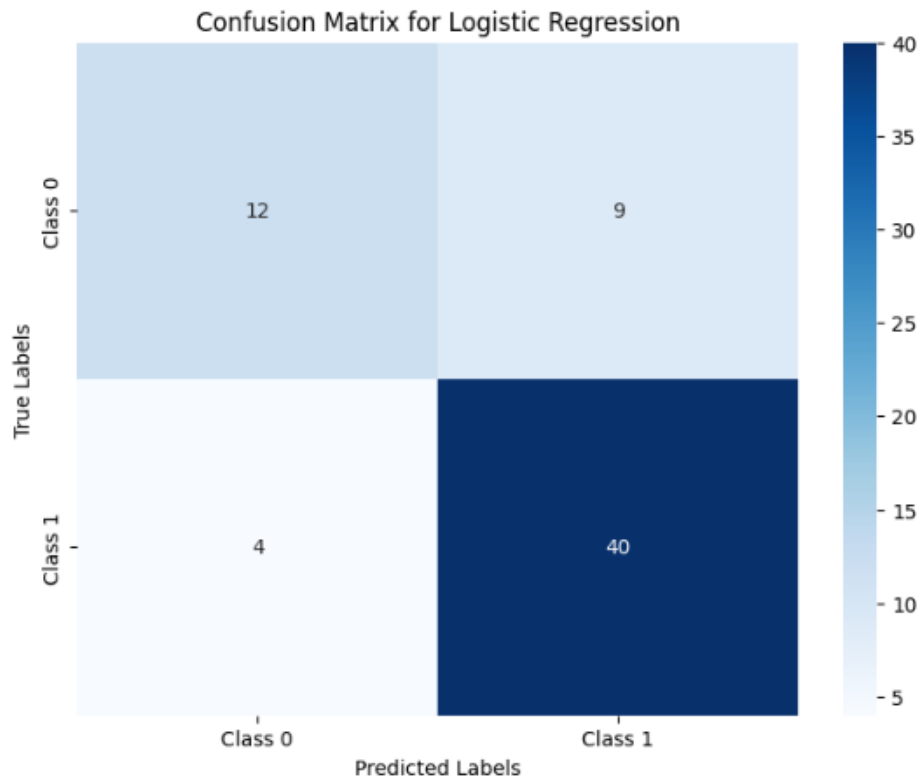


Figure 19. Logistic regression confusion matrix

The confusion matrix also shows that the model performs exceptionally well in predicting the "Placed" instances as True Positives. Because 40 of the "Placed" students were correctly predicted, it seems that in this category, the model is highly accurate and really performs a good job of identifying students that have been placed. However, because of the presence of 4 False Negatives, it also means that some of the "Placed" students can be missed by the model, which might imply missed opportunities for finding all the placed students.

In the "Not Placed" category, performance from the model is relatively weaker. It correctly identified 12 "Not Placed" students as such, while it had 9 "Not Placed" students misidentified as "Placed." This would indicate some bias toward classifying a student as "Placed" rather than "Not Placed," which is something to be worried about when correct identification of "Not Placed" students becomes vital. Hence, the False Positives would result in possibly unnecessary follow-ups or interventions for students predicted to be "Placed" who are not, which could deflate the overall accuracy and reliability of the model in differentiating these two classes.

# Voting classifier

Classification Report:				
	precision	recall	f1-score	support
0	0.75	0.57	0.65	21
1	0.82	0.91	0.86	44
accuracy			0.80	65
macro avg	0.78	0.74	0.75	65
weighted avg	0.79	0.80	0.79	65
Accuracy: 0.8				
Precision: 0.8163265306122449				
Recall: 0.9090909090909091				
F1-score: 0.8602150537634409				

Figure 20. Voting Classifier metrics

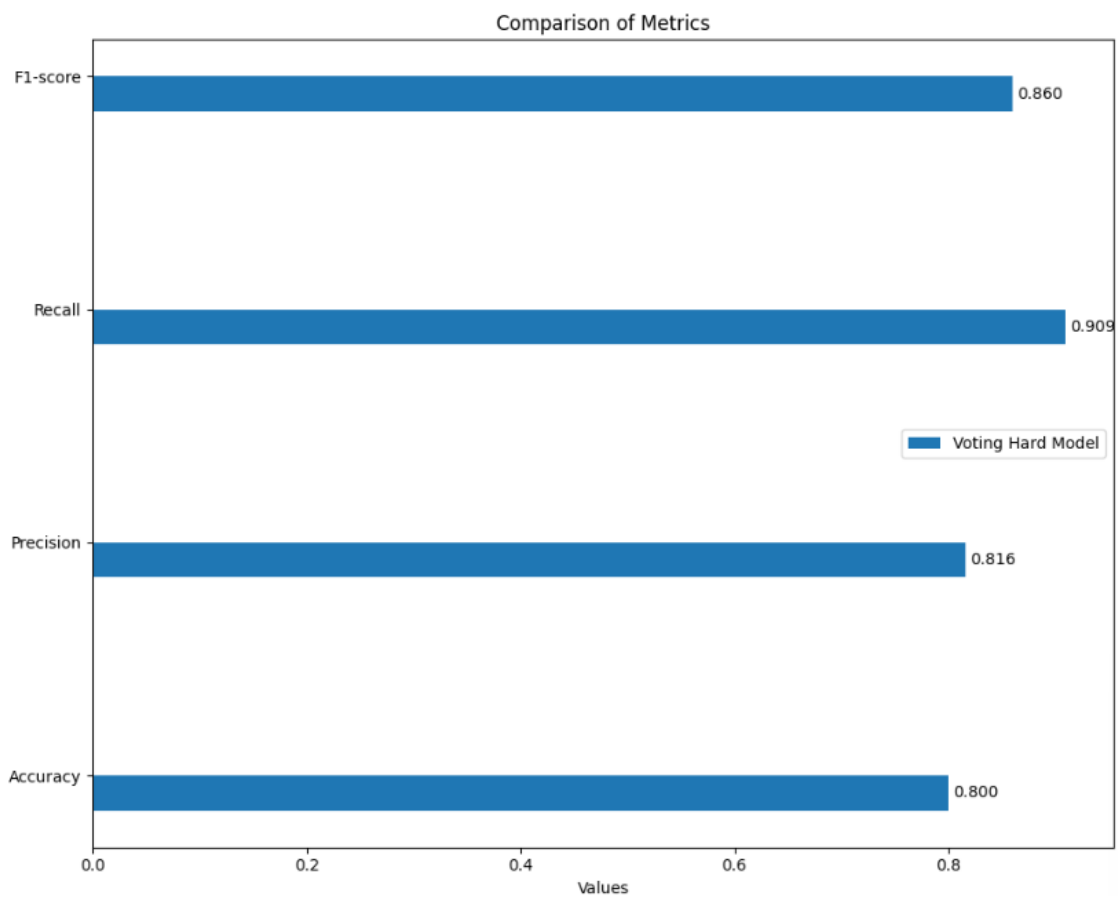


Figure 21. Voting classifier metrics

The Voting Classifier combines the predictive power of three individual models: Logistic Regression, XGBoost, and Random Forest using a majority vote-hard voting-for making the final classification for each instance. In integrating these models, the classifier draws on relative strengths of each approach: the simplicity and interpretability of Logistic Regression, advanced feature-handling in XGBoost, and the robustness of the Random Forest. The Voting Classifier is trained on the training dataset, and each model contributes to the final decision; hence, it potentially will improve overall performance by reducing various biases of individual models.

The classification report after that shows that the overall accuracy for the Voting Classifier stands at 80%, while the precision is 0.82 for the positive class ("Placed") with a recall of 0.91, hence a strong F1-score of 0.86. This high recall for the "Placed" class suggests that the model is particularly effective at identifying placed students, as it captures 91% of all true positives. On the other hand, in the case of the "Not Placed" class, the recall is 0.57, which is relatively low and thus prone to missing instances of this class. This weighted average F1-score of 0.79 reflects an overall good balance across classes and makes the Voting Classifier a sound model in this context for binary classification.

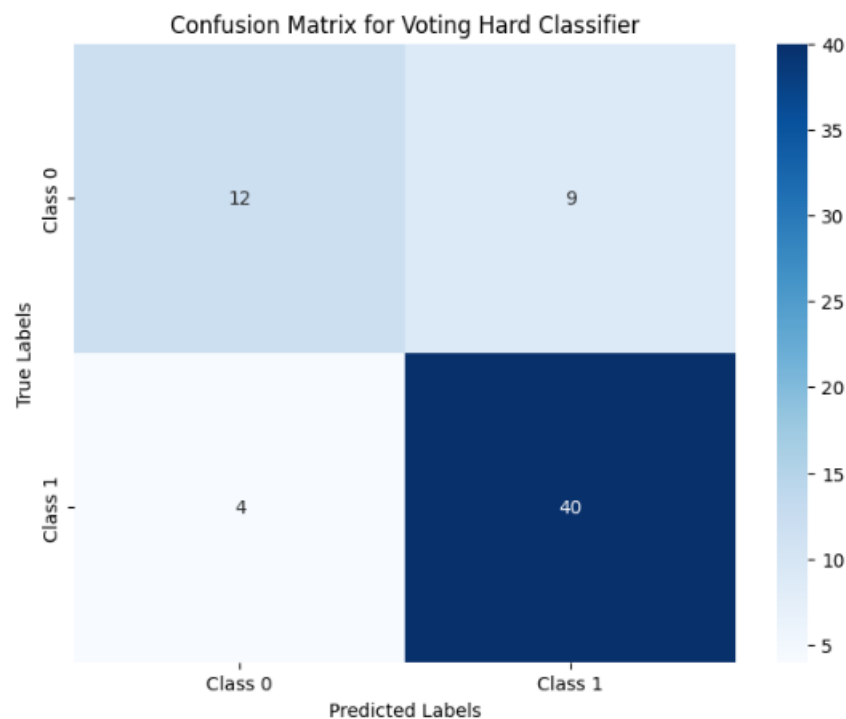


Figure 22. Confusion matrix for Voting Classifier

The confusion matrix infers that the Voting Classifier is good at predicting "Placed" instances of TP. For 40 "Placed" students, the model performs good in identifying students who are, in fact, placed. Whereas the False Negatives are 4, indicating a few "Placed" students are not identified by the model and may result in missed instances from this category.

The performance in the "Not Placed" category is more mixed, with the model correctly classifying 12 students as "Not Placed" and misclassifying a further 9 "Not Placed" students as "Placed". This may show a slight bias toward the "Placed" rather than the "Not Placed" prediction, which might be a limitation where the correct identification of "Not Placed" students is necessary. This may lead to incorrect predictions for the students who are not placed, getting predicted as placed just because of the higher relatively count of False Positives, which again may result in unnecessary interventions or misclassifications. Overall, this model is effective in predicting the instances of "Placed," while there is room for improvement in the prediction regarding "Not Placed".



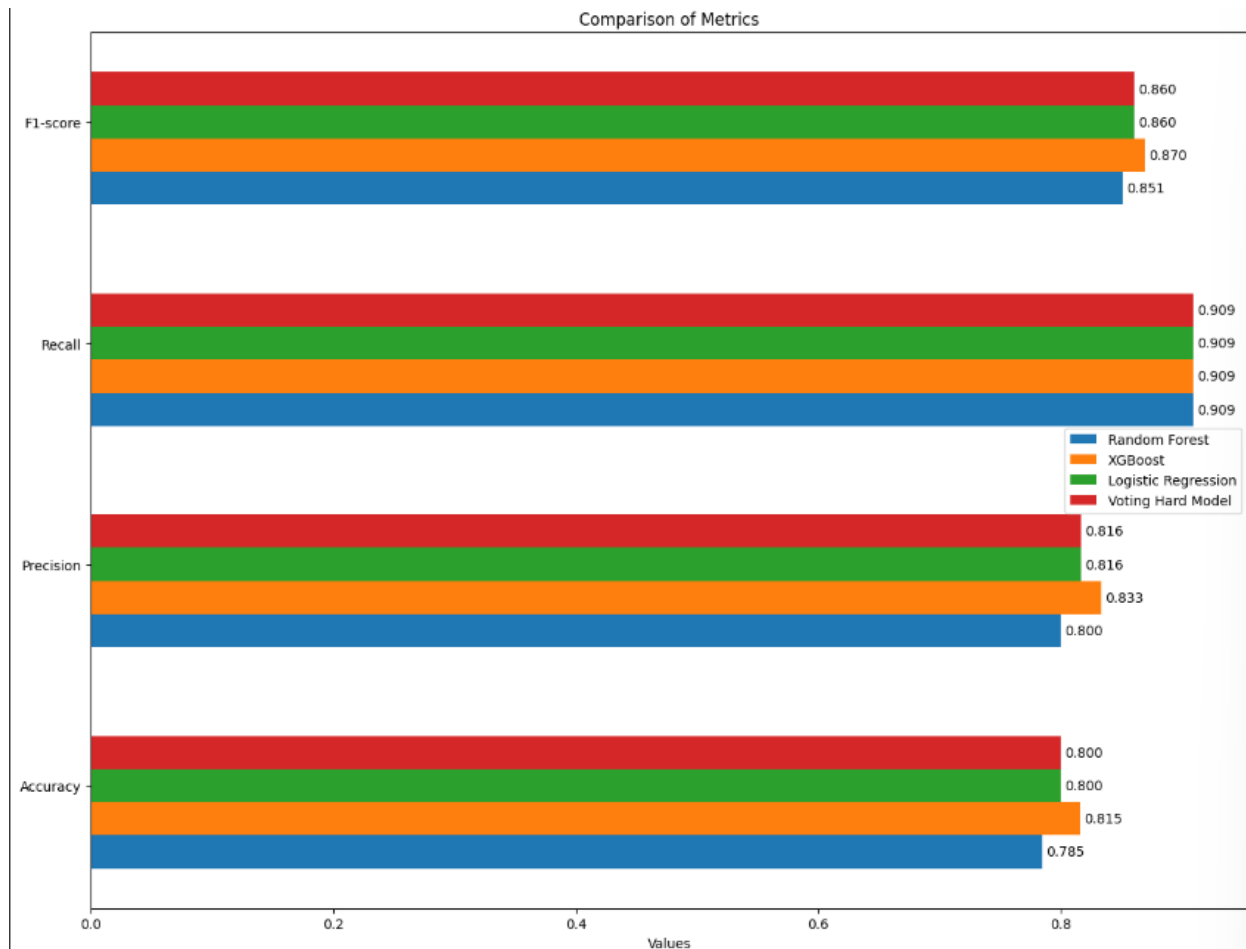


Figure 23: Bar chart for all metrics

The above bar chart is used to compare accuracy, precision, recall, and F1-score performance metrics for four different models: Random Forest, XGBoost, Logistic Regression, and a Voting Hard Model (ensemble). Each metric provides insight into how well each model classifies the binary target variable, assumed to be "Placed" versus "Not Placed."

It does so first by accuracy, where the Voting Hard Model takes a slight lead at 0.815, closely followed by Logistic Regression and XGBoost, each at an accuracy of 0.800. Random Forest has the lowest at an accuracy of 0.785. This means that all the models, while performing well in the correct classification of instances, tend to have their overall accuracy enhanced a little by the ensemble method of Voting Hard Model.

Precision concentrates on the preciseness of the positive predictions and points to how often a model makes a correct prediction for a student that it predicts will be "Placed". Logistic Regression has the highest precision value with 0.833, thus being more selective in labeling instances as "Placed," reducing the false positive count. Random Forest and

XGBoost come in at 0.816 precision, while the Voting Hard Model achieved 0.800 precision. This indicates that Logistic Regression is better at balancing true positives without overcommitting to misclassifying negatives as true positives.

Since recall essentially calculates how well each model captures all true instances of "Placed," all three models have an equal recall of 0.909. This consistent value in their recalls means the models are similarly able to catch most "Placed" instances without letting too many slips by, which can be very important for applications where capturing positive cases is highly important.

Lastly, F1-score, which is the harmonic mean of precision and recall, gives a balanced measure regarding the performance of each model when it comes to precision and recall. At 0.860, both the XGBoost model and the Hard Voting Ensemble Model present a strong balance between precision and recall, whereas Logistic Regression is slightly better with an F1-score of 0.870, while Random Forest lags at 0.851. This result shows that out of these algorithms, Logistic Regression gives the best result, but the Voting Hard Model and XGBoost are also strong choices, especially for those who prefer ensemble methods.

## Conclusion

That would be the "best" of these in this comparison, which is dependent on specific objectives for the classification task. Each model performs slightly different across the key metrics; thus, the choice of model should reflect the priority of these metrics.

1. If the key drivers are high overall accuracy and balance, the Voting Hard Model may be the best option. It has the highest accuracy at 0.815 and is also very sound on all other metrics since it combines the power of multiple algorithms to enhance robustness and stability.
2. For high precision, such that the number of false positives (i.e., "Not Placed" misclassified as "Placed") is as minimal as possible, the best model will be Logistic Regression. It yields the highest precision of 0.833, which means this model is choosier and provides fewer wrong placed labels to non-placed students.
3. If one prioritizes a strong balance between precision and recall—that is, the F1-score—working with either a balanced or insignificantly imbalanced dataset, then the model performing impressively well is again Logistic Regression with an F1-score of 0.870, closely followed by XGBoost and the Voting Hard Model at 0.860. This balance just makes Logistic Regression a little bit more favorable if interpretability and a linear decision boundary are also desirable.

In general, Logistic Regression will be advantageous when precision or interpretability is the key factor, whereas the Voting Hard Model would provide the best performance if accuracy and robustness of the ensemble is more important.