# ABSTRACT

Student Placements Prediction is a process of predicting whether a student gets placed or not in campus recruitment. Using the student data from prior years, classification techniques like Decision Tree and Random Forest are applied. The algorithms consider factors like USN, tenth-grade, PUC/diploma, and CGPA results, as well as technical and aptitude skills. The information was gathered during a two-year period from the years 2013 and 2014. These statistics on college placements were gathered over a two-year period. Using this information, we examine and anticipate a student's placement based on their past. Also conduct thorough EDAs and generate insights. At last, creating a solid model using ML or neural networks.

# INTRODUCTION

Student Placements Prediction is a process of predicting whether a student gets placed or not in campus recruitment. Using the student data from prior years, classification techniques like Decision Tree and Random Forest are applied. The algorithms take into account factors like USN, tenth-grade, PUC/diploma, and CGPA results, as well as technical and aptitude skills. The information was gathered during a two-year period from the years 2013 and 2014. These statistics on college placements were gathered over a two-year period. Using this information, we examine and anticipate a student's placement based on their past. Also conduct thorough EDAs and generate insights. At last creating a solid model using ML or neural networks.

The PROJECT aims to achieve several objectives related to the placement of engineering students. The first objective is to identify the factors that have an impact on the placement of engineering students. The second objective is to develop a predictive model that accurately predicts the placement of engineering students based on their characteristics and academic performance. The third objective is to evaluate the performance of the developed predictive model and compare it with other existing models. Moreover, the fourth objective is to identify the limitations of the developed predictive model and suggest ways to improve it.

In addition to the above objectives, this research also aims to investigate the performance of hostel students among engineering graduates and how it affects their placement. Furthermore, the research aims to examine the role of internships and co-op programs in improving the placement of engineering students. The achievement of these objectives will contribute to the understanding of the factors

affecting the placement of engineering students and will provide insights into strategies that can be adopted to improve their placement prospects.

When assessing whether a student can be placed, the general Placement Prediction System simply takes academic performance into account.

It would be unjust to judge a student purely on his or her academic performance when that kid may have a great aptitude, technical skill set, and communication ability. It would be unjust to grade a kid solely on his academic performance because many different aspects must be taken into account when determining where a student will be placed. To be selected for the campus interview, a student's technical and aptitude abilities need to be excellent. Academic success is important, but it is not the main determinant of where students are placed. Assignments on campus are typically determined by a single factor. The first three years of engineering's academic performance are used to evaluate pupils. Yet, student awareness and academic success are both necessary for passing aptitude tests and interviews. Certain data mining algorithms deceive students by interpreting the probability of a student being chosen as more than 100%, which is not realistic. Algorithms for negative probability often misinterpreted by students. Grades alone are insufficient to assess a pupil. The student's future should also be determined by technical and aptitude tests.

# LITERATURE SURVEY

The objective of this study[1] is to analyse previous year’s student’s data and use it to predict the placement chance of the current students. In terms of accuracy, precision, and recall, the proposed model is compared to traditional classification algorithms like Decision tree and Random forest. Based on the results, it is clear that the proposed algorithm does a much better job than the other algorithms mentioned. The accuracy of the Decision tree is 84% and the accuracy of the Random Forest is 86%. So, based on the analysis and predictions above, it's better to use the Random Forest algorithm to predict placement results.

In the next study [2], the system predicts the most likely placement status that Btech students will have at the end of their final year placements. With tested real-life data, the system was found to be accurate 71.66% of the time. This shows that the system is reliable for achieving its main goals, which are to help teachers and the placement cell in an institution find potential students and give them better training so they can do well in placement processes by different companies. The

system helps an institution improve its placement rate, which can be a key part of improving the institution's reputation. From the analysis, it's clear that the way the system was put together is good enough to make a big difference in the classification techniques that have been used so far in the field of placement prediction.

The paper [3] suggests a few supervised machine learning classifiers that can be used to predict where a student will work in the IT industry based on their academic performance in tenth grade, twelfth grade, college, and up to this point in college. The classifiers are made by using the classification algorithms Support Vector Machine, Gaussian Naive Bayes, K-Nearest Neighbour, Random Forest, Decision Tree, Stochastic Gradient Descent, Logistic Regression, and Neural Network. Through developed classifiers, the paper also suggests a way to rank the academic performance parameters that are important for student placement. In terms of academic performance, the percentage in tenth grade is the most important. This is followed by the percentage in twelfth grade and the number of backlogs in B Tech. B: The percentage of tech is the least important of the four inputs. The classifiers are compared based on their classification report using the sub parameters precision, recall, f1-score, and support. By making different binary classifiers, the performance of different supervised machine learning classification algorithms for predicting placement is compared. AUC and ROC curve are used to measure performance. With an AUC value of 0.86, Support Vector Machine and Logistic Regression do the best. In the future, classifiers will be built that can predict where students will be placed by considering both academic performance and skills.

The paper [4] suggests using seven models to solve the problem of predicting where students will be placed: the linear regression model, the K- neighbor regression model, the decision tree regression model, the XGBoost regression model, the gradient boost regression model, the light GBM regression model, and the random tree classifier model. This study looks at the accuracy of predictions and the root mean square error to measure performance (RMSE). The performance of these seven models is analysed by putting them to use with two sets of data: a simple set of data and an extended set of data that has more information about the students. For dataset-I, it was found that the K-neighbor regression model did better than other models because it had a higher R2 score (0.94), and its RMSE value was lower (103263.25). But in dataset-II, which has more features, the linear regression model and the K-neighbor regression model come out on top with high R2 scores (0.89). By looking at the RMSE value of the

XGBoost regression model, it was found that dataset-II had a very low value (12970.41). This work is a first attempt to figure out how to use machine learning models for student prediction problems. As such, it is not perfect in many ways. To improve this work in the future, we will add more features to the data set and try to predict a few more variables that are more important for student placement.

The paper [5] suggests a Placement Prediction System that uses the machine learning model of k-nearest neighbours' classification to figure out how likely it is that an undergraduate student will get a job in an IT company. For this, the student's academic history and skills, such as their ability to programme, communicate, analyse, and work in a team, are taken into account. The Placement Statistics of the PES Institute of Technology, Bangalore South Campus for the last two academic batches are used for this. The placement office can try to figure out what the students' weaknesses are and help them get better so that they can overcome their weaknesses and do the best they can. So, the key is to test the student's abilities in the right places and give them the right training.

The paper [6] is about how important internships are in colleges and universities. The college's placement cell oversees finding well-paying jobs for their students. Some of the things that organisations look at when deciding where to place a student are the student's profile, including the percentage of SSC, Intermediate, and undergraduate courses, technical skills, programming skills, aptitude, and reasoning. But the personal profile of the student is not considered in this process. The proposal is mostly about managed learning, which is a more direct form of predictive testing that can be used to make predictions about the future. ML algorithms like support vector machine, linear logistic regression, and decision tree are used to classify the students' data. However, the Random Forest algorithm is the most accurate of all of them. The main goal of this model is to predict whether the student will be put in campus recruitment. The algorithms are based on information about students from the year before. The model or classifier's accuracy is measured by the number of times it correctly predicted or grouped events. The accuracy of the two methods is used to compare how well they work. Use the following equation to figure out how accurate it is: Precision= 𝑇𝑃 𝑇𝑃 +

𝐹𝑃. In this case, "True positive" means that the model correctly identifies the positive class. In the same way, a true negative outcome is one where the model correctly predicts that the class will be negative. False positive is a result that happens when the model predicts the positive class wrongly. On the other hand, a false negative is a result that happens when the model predicts the negative class wrongly.

# PROBLEM STATEMENT

The general Placement Prediction System only considers academic performance when determining whether a student can be placed. A student might have strong aptitude, technical skills, and communication abilities but unfavourably perform poorly in academics, thus judging him or her solely on that basis would be unfair to the student.

Since several factors must be considered when predicting a student's placement, it would be unfair to grade a student just on his academic performance. However, a student's technical and aptitude skills must be strong in order to be chosen for the campus interview. Academic achievements are crucial, but they are not the most crucial factor in how students are placed. Campus assignments are usually based on one parameter. Academic achievement in the first three years of engineering is utilized to evaluate students. However, passing aptitude tests and interviews depends on student awareness as well as academic performance. Some Data Mining algorithms interpret the chance of a student being selected as more than 100%, which is unfeasible and misleads the student. Students misinterpret negative probability algorithms. Grades are not enough to judge a student. Aptitude and technical assessments should also be considered to determine the student's future.

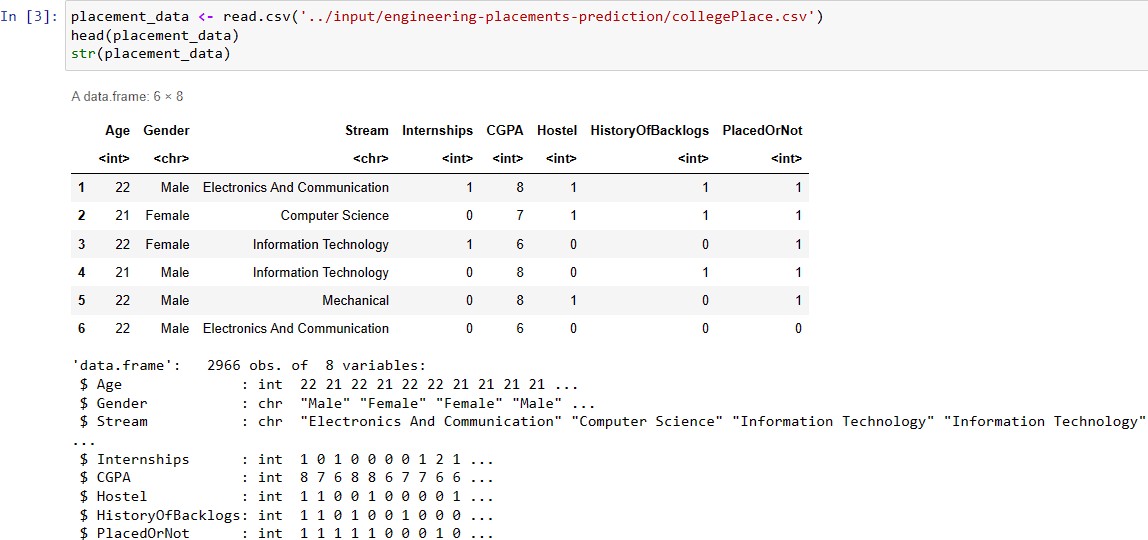
# DATASET

We have taken a Kaggle dataset with the attributes like

* Age: The age of the student,
* Gender: The gender of the student,
* Stream: The stream that the student is studying, such as Electronics and Communication, Computer Science, etc.,
* Internships: The number of internships that the student has completed,
* CGPA: The Cumulative Grade Point Average of the student,
* Hostel: Whether the student is staying in a hostel or not (1 = Yes, 0 = No),
* HistoryOfBacklogs: Whether the student has a history of backlogs or not (1

= Yes, 0 = No),

* PlacedOrNot: Whether the student has been placed in a job or not (1 = Yes, 0 = No).



# PROPOSED SYSTEM

**Step 1: Problem formulation** : Problem formulation is the process of clearly defining a problem in a way that it can be effectively addressed and solved. It involves identifying the problem, analysing the current situation, determining the root cause of the problem, and defining the objectives to be achieved.

**Step2: Data collection** : The system is designed to collect and store data from students, including their academic records.

**Step 3: Data cleansing** : data cleaning involves tasks, such as removing duplicate records, correcting misspellings and typos, filling in missing values, standardizing data formats, and addressing inconsistencies in data formatting.

**Step 4: Exploratory Data Analysis** : The system will perform data analysis on the collected data. It uses EDA algorithms and machine learning models to analyse the data and identify patterns that can help predict if a student is placed or not.

**Step 5: Classification algorithm identification and selection** : There are various classification algorithms available, and selecting the most appropriate one for a given problem depends on several factors such as the size and complexity of the data, the type of features available, and the desired level of accuracy.

**Step 6: Implementation of algorithm and development of classification models through training:** The implementation of a classification algorithm

involves using programming tools and libraries to train the algorithm on a dataset, and then applying it to new data to make predictions.

Here we have used Decision tree, Random Forest, AdaBoost, Gradient Boosting, Extra Trees Classifier for predicting.

**RANDOM FOREST**

* The random forest algorithm is an ensemble method in machine learning. It takes a dataset consisting of records with attributes as input and creates random subsets of the input. For each random subset, a decision tree is constructed. The final class of a test record is determined using the majority vote technique.
* The algorithm makes use of the out-of-bag error technique. Each tree is constructed using the following algorithm:
* The number of training cases is N, and the number of variables in the classifier is M.
* The number of input variables to be used to determine the decision at a node of the tree is m. m should be much less than M.
* A training set for this tree is chosen by selecting N times with replacement from all N available training cases (i.e., taking a bootstrap sample). The rest of the cases are used to estimate the error of the tree by predicting their classes.
* For each node in the tree, m variables are randomly chosen on which to base the decision at that node. The best split based on these m variables in the training set is calculated.
* Each tree is fully grown and not pruned (as may be done in constructing a normal tree classifier).

**DECISION TREE**

* The decision tree technique is a classification algorithm that constructs a tree to model the classification process. However, there are several issues faced by most decision tree algorithms, such as choosing splitting attributes,

ordering of splitting attributes, number of splits to take, balance of tree structure and pruning, and stopping criteria.

* The ID3 algorithm is a classification algorithm based on Information Entropy. Its basic idea is to map all examples to different categories based on different values of the condition attribute set. The algorithm chooses information gain as attribute selection criteria, and the attribute with the highest information gain is selected as the splitting attribute of the current node. This is done to minimize the information entropy that the divided subsets need.
* Branches are established based on the different values of the attribute, and each branch creates other nodes and branches until all the samples in a branch belong to the same category. The concepts of Entropy and Information Gain are used to select the splitting attributes.
* The ID3 algorithm is a decision tree algorithm that uses information gain as attribute selection criteria to determine the best classification attribute from condition attribute sets. It constructs a tree by establishing branches based on the different values of the attribute and selecting the attribute with the highest information gain as the splitting attribute of the current node.

### ADABOOST

* AdaBoost (Adaptive Boosting) is a popular ensemble learning algorithm that combines multiple "weak" classifiers to create a more accurate "strong" classifier. The idea behind AdaBoost is to iteratively train a series of classifiers on the same dataset, with each new classifier giving more weight to the misclassified data points from the previous classifier.
* In each iteration, the algorithm assigns weights to each training example in the dataset, and then trains a "weak" classifier on the weighted data. The weights of the misclassified examples are then increased, and the weights of the correctly classified examples are decreased. This process is repeated for a predetermined number of iterations or until a certain level of accuracy is achieved.
* The final classifier is a weighted combination of the weak classifiers, where the weights are determined based on the accuracy of each classifier. The

algorithm is particularly effective when used with decision trees as the base classifier.

* AdaBoost has been shown to be effective in a wide range of applications, including face detection, object recognition, and speech recognition. One of the key advantages of AdaBoost is that it can handle noisy data and outliers well, while still producing accurate results.

### GRADIENT BOOSTING

* Gradient Boosting Classifier is another ensemble learning algorithm that works by building a series of decision trees. However, unlike AdaBoost, it trains each tree sequentially by optimizing the gradient of the loss function. The objective of the algorithm is to minimize the overall error or loss of the model.
* In Gradient Boosting Classifier, each new tree is built to correct the errors made by the previous trees in the sequence. At each iteration, the algorithm fits a new tree to the residuals of the previous tree. The residuals are the differences between the predicted and actual target values.
* The learning rate is another important parameter in Gradient Boosting Classifier that controls the contribution of each tree to the final ensemble. A smaller learning rate results in slower convergence but can lead to better generalization performance.
* One of the key advantages of Gradient Boosting Classifier is its ability to handle different types of data, including categorical and continuous variables. It can also handle missing values by using surrogate splits. Moreover, it can capture complex non-linear relationships between the features and the target variable.
* Gradient Boosting Classifier has become a popular algorithm for various applications such as anomaly detection, fraud detection, and click-through rate prediction. However, it can be computationally expensive and sensitive to overfitting, especially when the dataset is noisy, or the number of features is large.

### EXTRA TREES CLASSIFIER

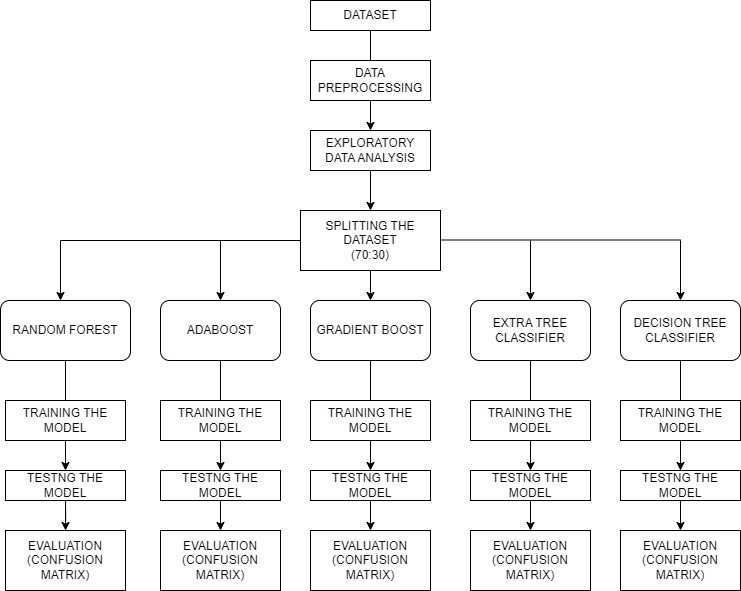
* Extra Trees Classifier (or Extremely Randomized Trees) is an ensemble learning algorithm that belongs to the family of decision tree-based models. It is like Random Forests in that it builds multiple decision trees and combines them to make predictions, but with some important differences.
* In Extra Trees Classifier, instead of choosing the optimal split point for each feature, the algorithm selects random split points within the feature's range. This randomization process makes the algorithm less sensitive to noisy data and reduces overfitting. Moreover, Extra Trees Classifier typically requires fewer trees to achieve comparable or even better accuracy than Random Forests.
* Another difference between Extra Trees Classifier and Random Forests is the way the algorithm handles the bootstrap samples. While Random Forests draw bootstrap samples from the original dataset to train each tree, Extra Trees Classifier uses the entire dataset to train each tree. This means that Extra Trees Classifier can be faster than Random Forests for smaller datasets.
* Extra Trees Classifier has been shown to be effective in a variety of applications such as image recognition, text classification, and fraud detection. One of the key advantages of Extra Trees Classifier is its simplicity and ease of implementation. However, it may not perform as well as other algorithms on certain types of data or with high-dimensional datasets.

**Step 7: Testing the models** : Testing a machine learning model is the process of evaluating the model's performance on a dataset that it has not seen during training. This is done to assess the model's ability to generalize to new data and to ensure that it is not overfitting the training data.

**Step8: Evaluating the accuracy of the models** : The predictive models are evaluated using accuracy scores after testing with test dataset and the best model is selected. Evaluating the performance of the predictive model using various metrics such as accuracy, precision, recall, F1 score, and ROC curve analysis.

**Step 9: Comparing the accuracy and performance of the model**.

# FLOW CHART



**MODULES USED**

### tidyverse:

Tidyverse is a collection of R packages designed for data science. It provides a consistent set of tools for data manipulation, transformation, and visualization. Some of the packages included in tidyverse are ggplot2, dplyr, tidyr, readr, and purrr. The tidyverse package is a collection of packages and does not have a specific algorithm associated with it.

### caret:

The caret package provides a unified interface to train and evaluate machine learning models in R. It supports a wide range of algorithms and provides tools for preprocessing, feature selection, and model tuning. The caret package provides an interface for a wide range of machine learning algorithms, including decision trees, random forests, support vector machines, and neural networks.

### C5.0:

The C5.0 package is an implementation of the C5.0 decision tree algorithm in

R. C5.0 is a popular algorithm for decision tree classification and regression tasks. It uses an entropy-based splitting criterion to partition the data and create a tree structure. The C50 package implements the C5.0 decision tree algorithm.

1. Data Preprocessing: The first step is to preprocess the data by removing any missing values or outliers, encoding categorical variables, and splitting the data into training and testing sets.
2. Training Set Creation: A decision tree is trained on the training set, where each node of the tree represents a feature, and each branch represents a possible value of that feature. The tree is built recursively by selecting the best feature to split the data at each node, based on some splitting criterion such as information gain or gain ratio.
3. Pruning: After the tree has been built, it may be overfitted to the training data, meaning that it is too complex and captures noise rather than true patterns in the data. To prevent overfitting, the tree is pruned by removing branches that do not improve the overall accuracy of the tree on a validation set.
4. Testing Set Evaluation: Once the tree has been pruned, it is evaluated on the testing set to measure its accuracy in predicting new data. The accuracy is calculated by comparing the predicted class labels of the test set to the actual class labels.
5. Model Selection: If the accuracy of the C5.0 decision tree is better than other models tested, it is selected as the best model for predicting the placement of engineering students.
6. Overall, the C5.0 decision tree algorithm is an efficient and powerful algorithm for classification tasks, and it is often used in the field of machine learning for its interpretability and ability to handle both numerical and categorical data.

### Entropy:

If you know the probabilities p1, p2, …, ps, where Σpi = 1, you can define entropy as H(p1, p2, …, ps) = Σ - (pi log pi) Entropy finds out how much order there is in a given state of a database. A set that has a value of H = 0 is perfectly classified. In other words, the more entropy there is, the more likely it is that the classification process can be made better.

### Gain of Knowledge

The Dtree chooses the splitting attribute that gives the most information. Information gain is the difference between how much information is needed after the split and how much information is needed before the split. The difference between the entropies of the original dataset and the weighted sum of the entropies of each of the subdivided datasets is used to figure this out. G(D, S) = H(D) - ΣP(Di)H(Di) is the formula that is used for this (Di)

### randomForest:

The randomForest package provides an implementation of the random forest algorithm in R. Random Forest is an ensemble learning method that builds multiple decision trees and combines their predictions. It is a popular algorithm for classification and regression tasks and is known for its high accuracy and robustness. The randomForest package implements the random forest ensemble learning algorithm.

A random forest algorithm needs a set of records with attributes, which is called a dataset. There are made random subsets of the input. A decision tree will be made for each of the random subsets that are made. The algorithm that uses the majority vote method will decide what class a test record will end up in. The out of bag error method is used by the random forest algorithm. The following algorithm is used to build each tree:

1. Let's say that there are N training cases and M variables in the classifier.
2. We are given the number m of input variables that will be used to make a decision at a node of the tree; m should be much less than M.
3. Pick a training set for this tree by picking N times with replacement from the N training cases that are available (i.e. take a bootstrap sample). Use the rest of the cases to estimate the tree's error by guessing what class they belong to.
4. For each node in the tree, pick m variables at random to use as the basis for the decision at that node. Use these m variables in the training set to figure out the best way to split the set.
5. Every tree is full-grown and hasn't been cut down (as may be done in constructing a normal tree classifier).

### corrplot:

The corrplot package provides tools for creating correlation matrices and visualizing them as heatmaps. It supports a variety of correlation coefficients, including Pearson, Spearman, and Kendall. The corrplot package provides tools for creating correlation matrices and heatmaps but does not implement any specific algorithm.

### ROCR:

The ROCR package provides tools for visualizing and evaluating the performance of binary classification models. It supports a variety of metrics, including accuracy, precision, recall, F1 score, and ROC curves. The ROCR package provides tools for visualizing and evaluating the performance of binary classification models but does not implement any specific algorithm.

### gmodels:

The gmodels package provides tools for creating and displaying statistical models. It includes functions for creating contingency tables, calculating chi- squared tests, and generating model summaries. The gmodels package provides tools for creating and displaying statistical models but does not implement any specific algorithm.

# RESULTS AND DISCUSSION

### RANDOM FOREST

RANDOM FOREST CONFUSION MATRIX

|  |  |  |  |
| --- | --- | --- | --- |
| **actual**  **/predicted** | **Output** | | |
| ***no*** | ***yes*** | ***Row total*** |
| no | 377 | 21 | 398 |
| yes | 92 | 399 | 491 |
| Column total | 469 | 420 | 889 |

The random forest model has been evaluated on a dataset with a binary target variable, where the positive class is labelled as "yes".

The accuracy of the random forest model is reported to be 0.8729, which is higher than the accuracy reported for the previous model. This suggests that the random forest model is better at predicting the target variable than the previous model.

The 95% confidence interval for the accuracy is (0.8492, 0.8941), which means that if the same model were applied to multiple datasets of the same size and composition, 95% of the time the accuracy would fall within this range.

The p-value for the accuracy being greater than the null hypothesis accuracy (NIR) is < 2.2e-16, which means that the random forest model's accuracy is significantly better than chance.

The balanced accuracy for the random forest model is reported to be 0.8769, which suggests that the model performs well in predicting both the positive and negative classes, taking into account the imbalance in the dataset.

Overall, based on the results, the random forest model appears to perform well in predicting the positive class of the binary target variable, with a higher accuracy than the previous model.

### ADABOOST CLASSIFIER

ADABOOST CLASSIFIERCONFUSION MATRIX

|  |  |  |  |
| --- | --- | --- | --- |
| **actual /predicted** | **Output** | | |
| ***no*** | ***yes*** | ***Row total*** |
| no | 390 | 8 | 398 |
| yes | 110 | 381 | 491 |
| Column total | 500 | 389 | 889 |

There are a total of 889 instances that were classified into two categories, "actual" and "predicted”. Out of 398 instances that were labelled as "no", the model predicted "no" for 390 of them, and "yes" for 8 of them. Out of 491 instances that were labelled as "yes", the model predicted "no" for 110 of them, and "yes" for 381 of them. Overall, the model made 771 correct predictions (390 "no" and 381 "yes") out of 889 instances. The model's accuracy can be calculated by dividing the number of correct predictions by the total number of instances, which is 771/889 = 0.867 or approximately 87%.

In summary, the table provides information on the model's performance in predicting two categories ("no" and "yes") based on the actual labels. It indicates that the model correctly predicted most instances, achieving an accuracy of approximately 87%. However, it also made some incorrect predictions, particularly in cases where the actual label was "yes" but the model predicted "no" for 110 instances.

### GRADIENT BOOSTING

GRADIENT BOOSTING CONFUSION MATRIX

|  |  |  |  |
| --- | --- | --- | --- |
| **actual**  **/predicted** | **Output** | | |
| ***no*** | ***yes*** | ***Row total*** |
| no | 380 | 18 | 398 |
| yes | 121 | 370 | 491 |
| Column total | 501 | 388 | 889 |

There are a total of 889 instances that were classified into two categories, "actual" and "predicted”. Out of 398 instances that were actually labelled as "no", the model predicted "no" for 380 of them, and "yes" for 18 of them. Out of 491 instances that were actually labelled as "yes", the model predicted "no" for 121 of them, and "yes" for 370 of them. Overall, the model made 750 correct predictions (380 "no" and 370 "yes") out of 889 instances. The model's accuracy can be calculated by dividing the number of correct predictions by the total number of instances, which is 750/889 = 0.843 or approximately 84%.

In summary, the table provides information on the model's performance in predicting two categories ("no" and "yes") based on the actual labels. It indicates that the model correctly predicted most instances, achieving an accuracy of approximately 84%. However, it also made some incorrect predictions, particularly in cases where the actual label was "yes" but the model predicted "no" for 121 instances.

### EXTRA TREES CLASSIFIER

EXTRA TREES CLASSIFIERCONFUSION MATRIX

|  |  |  |  |
| --- | --- | --- | --- |
| **actual**  **/predicted** | **Output** | | |
| ***no*** | ***yes*** | ***Row total*** |
| no | 387 | 11 | 398 |
| yes | 110 | 381 | 491 |
| Column total | 497 | 392 | 889 |

There are a total of 889 instances that were classified into two categories, "actual" and "predicted".

Out of 398 instances that were actually labelled as "no", the model predicted "no" for 387 of them, and "yes" for 11 of them. Out of 491 instances that were labelled as "yes", the model predicted "no" for 110 of them, and "yes" for 381 of them. Overall, the model made 768 correct predictions (387 "no" and 381 "yes") out of 889 instances.

The model's accuracy can be calculated by dividing the number of correct predictions by the total number of instances, which is 768/889 = 0.864 or approximately 86%.

In summary, the table provides information on the model's performance in predicting two categories ("no" and "yes") based on the actual labels. It indicates that the model correctly predicted most instances, achieving an accuracy of approximately 86%. However, it also made some incorrect predictions, particularly in cases where the actual label was "yes" but the model predicted "no" for 110 instances.

### DECISION TREE CLASSIFIER

DECSION TREE CONFUSION MATRIX

|  |  |  |  |
| --- | --- | --- | --- |
| **actual**  **/predicted** | **Output** | | |
| ***no*** | ***yes*** | ***Row total*** |
| no | 393 | 5 | 398 |
| yes | 112 | 379 | 491 |
| Column total | 505 | 384 | 889 |

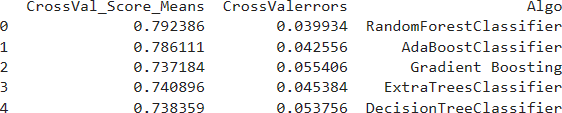
The reported accuracy of 0.8684 indicates that the Decision model correctly predicted the class of 86.84% of the instances in the dataset. The 95% confidence interval (CI) of (0.8444, 0.8899) indicates that if the same model were applied to other similar datasets, we would expect the accuracy to fall within this range with 95% confidence.

The p-value of < 2.2e-16 suggests that the accuracy of the model is significantly better than chance (i.e., better than the null accuracy rate).

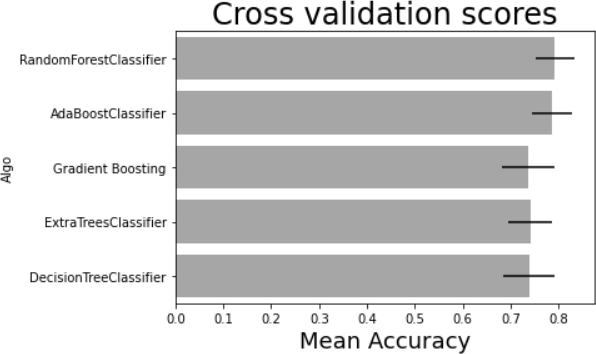
The balanced accuracy of 0.8826 indicates that the model performs well on both the positive and negative classes, considering the imbalance in class distribution.

Finally, based on the results, it appears that the "Positive" class is labeled as "yes".

# CONCLUSION



### In the plot we ca see that Random Forest Classifier is performing the best with a accuracy of 87.29%. The model has a cross validation score of 79.23% and cross validation error of 3.99%



Based on the accuracy score of 87.29%, it seems that the random forest classifier performed better than other models that were tested. However, it's important to note that accuracy alone may not be the only metric to consider when evaluating a model's performance. Other factors such as precision, recall, F1 score, and the specific requirements of the task at hand should also be considered. Additionally, it may be useful to perform cross-validation or tune hyperparameters to ensure that the model's performance is robust and can generalize well to new data. The model has a cross validation score of 79.23% and cross validation error of 3.99%.

Based on the given table, there were 889 observations in total. The rows represent the actual values of a certain event (yes or no) and the columns represent the predicted values of the same event (yes or no).

Out of 398 cases where the actual value was "no", 377 were correctly predicted as "no" and 21 were falsely predicted as "yes". Similarly, out of 491 cases where the actual value was "yes", 399 were correctly predicted as "yes" and 92 were falsely predicted as "no".

From this, we can see that the model's overall accuracy was (377+399) / 889 = 0.79, meaning that it correctly predicted the outcome in about 79% of the cases.

Additionally, we can compute other metrics such as precision, recall, and F1-score to better understand the performance of the model. For example, precision measures the proportion of true positives among all predicted positives, while recall measures the proportion of true positives among all actual positives. These metrics can be useful in different contexts depending on the specific goals of the analysis.

**APPENDIX**

# CODE

## Will you get placed ?

April 10, 2023

[1]:

*# This R environment comes with many helpful analytics packages installed*

*# It is defined by the kaggle/rstats Docker image: https://github.com/kaggle/*

↪*docker-rstats*

*# For example, here's a helpful package to load*

library(tidyverse) *# metapackage of all tidyverse packages*

*# Input data files are available in the read-only "../input/" directory*

*# For example, running this (by clicking run or pressing Shift+Enter) will list*␣

↪*all files under the input directory*

list.files(path = "../input")

*# You can write up to 20GB to the current directory (/kaggle/working/) that*␣

↪*gets preserved as output when you create a version using "Save & Run All"*

*# You can also write temporary files to /kaggle/temp/, but they won't be saved*␣

↪*outside of the current session*

**Attaching pack**t**a**i**g**d**e**y**s**verse 1.3.1

ggplot2 3.3.4 purrr 0.3.4

tibble 3.1.2 dplyr 1.0.7

tidyr 1.1.3 stringr 1.4.0

readr 1.4.0 forcats 0.5.1

**Conflicts**

tidyverse\_conflicts()

dplyr::filter() masks stats::filter() dplyr::lag() masks stats::lag()

’engineering-placements-prediction’

[2]:

library(caret) library(C50) library(randomForest) library(corrplot)

library(ROCR) library(gmodels)

Loading required package: lattice Attaching package: ‘caret’

The following object is masked from ‘package:purrr’: lift

The following object is masked from ‘package:httr’: progress

randomForest 4.6-14

Type rfNews() to see new features/changes/bug fixes. Attaching package: ‘randomForest’

The following object is masked from ‘package:dplyr’: combine

The following object is masked from ‘package:ggplot2’: margin

corrplot 0.88 loaded

[3]:

placement\_data <- read.csv('../input/engineering-placements-prediction/

↪collegePlace.csv')

head(placement\_data) str(placement\_data)

A data.frame: 6 × 8

Age Gender

<int> <chr>

Stream

<chr>

Electronics And Communication Computer Science

Information Technology Information Technology Mechanical

Electronics And Communication

Internships CGPA Hostel His

<int> 1

0

1

0

0

0

<int> <int> <i

1 22

2 21

3 22

4 21

5 22

6 22

Male Female Female Male Male Male

8

7

6

8

8

6

1

1

0

0

1

0

1

1

0

1

0

0

[4]:

'data.frame': 2966 obs. of 8 variables:

$ Age : int 22 21 22 21 22 22 21 21 21 21 …

$ Gender : chr "Male" "Female" "Female" "Male" …

$ Stream : chr "Electronics And Communication" "Computer Science" "Information Technology" "Information Technology" …

$ Internships : int 1 0 1 0 0 0 0 1 2 1 …

$ CGPA : int 8 7 6 8 8 6 7 7 6 6 …

$ Hostel : int 1 1 0 0 1 0 0 0 0 1 …

$ HistoryOfBacklogs: int 1 1 0 1 0 0 1 0 0 0 …

$ PlacedOrNot : int 1 1 1 1 1 0 0 0 1 0 …

* 1. **EXPLORATORY DATA ANALYSIS**

aggregate(PlacedOrNot~Age, data=placement\_data, FUN=mean) placement\_data %>%

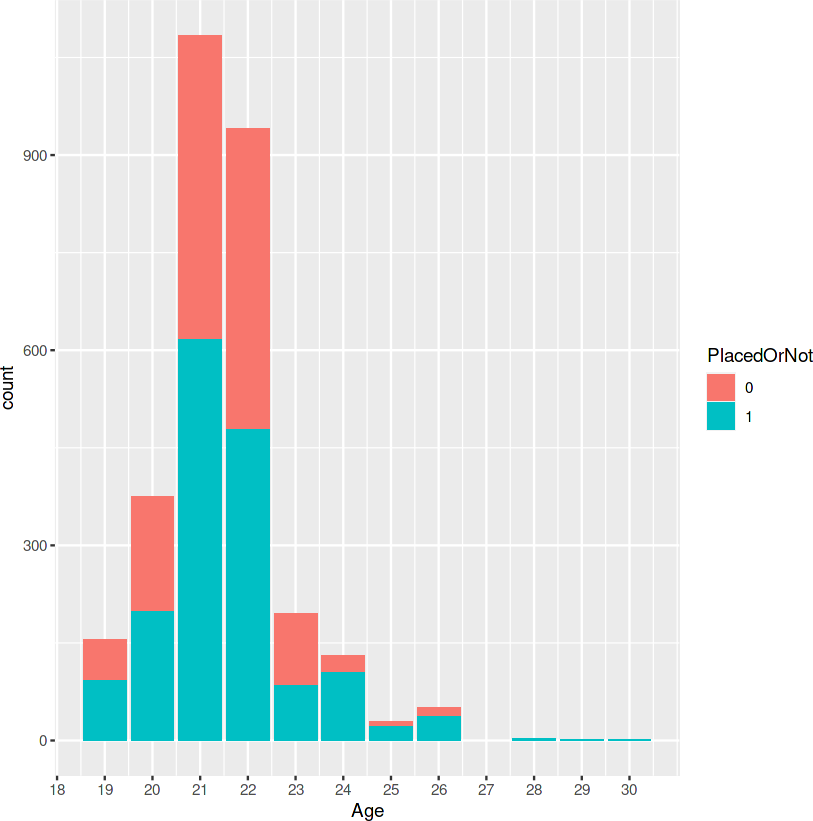
ggplot(aes(Age, fill=factor(PlacedOrNot))) + geom\_bar() + scale\_x\_continuous(breaks=seq(18,30, by=1)) + labs(fill="PlacedOrNot")

A data.frame: 11 × 2

Age PlacedOrNot

<int> <dbl>

|  |  |
| --- | --- |
| 19 | 0.5897436 |
| 20 | 0.5306667 |
| 21 | 0.5691882 |
| 22 | 0.5079702 |
| 23 | 0.4358974 |
| 24 | 0.7938931 |
| 25 | 0.7586207 |
| 26 | 0.7400000 |
| 28 | 1.0000000 |
| 29 | 1.0000000 |
| 30 | 1.0000000 |



[5]:

[6]:

Gender PlacedOrNot

aggregate(PlacedOrNot~Gender, data=placement\_data, FUN=mean)

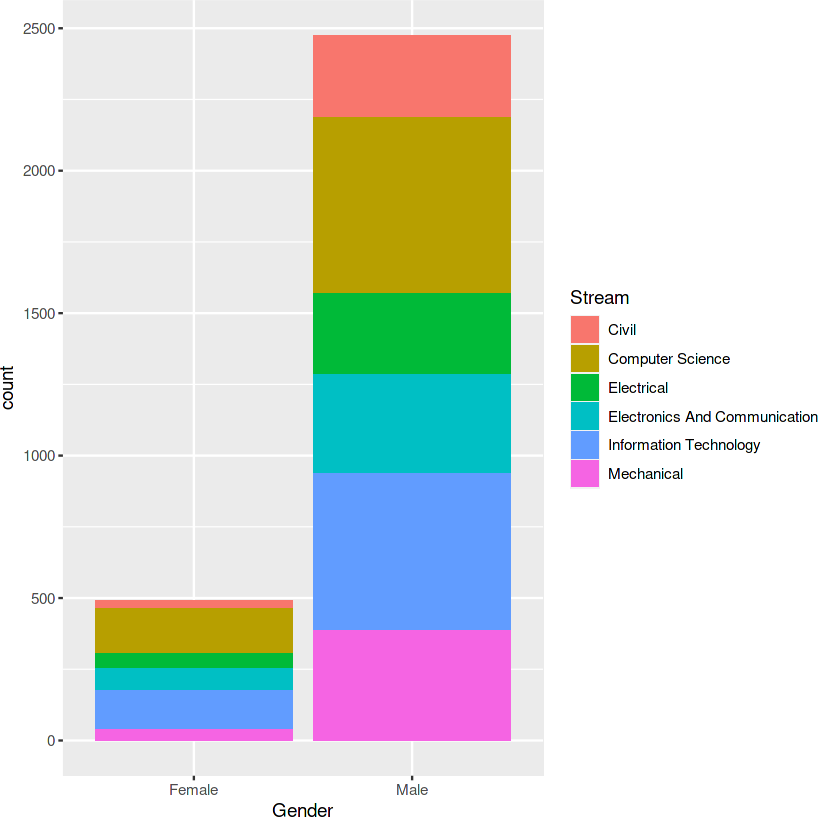
<chr> <dbl> Female 0.5600815

A data.frame: 2 × 2

Male 0.5511111

placement\_data %>%

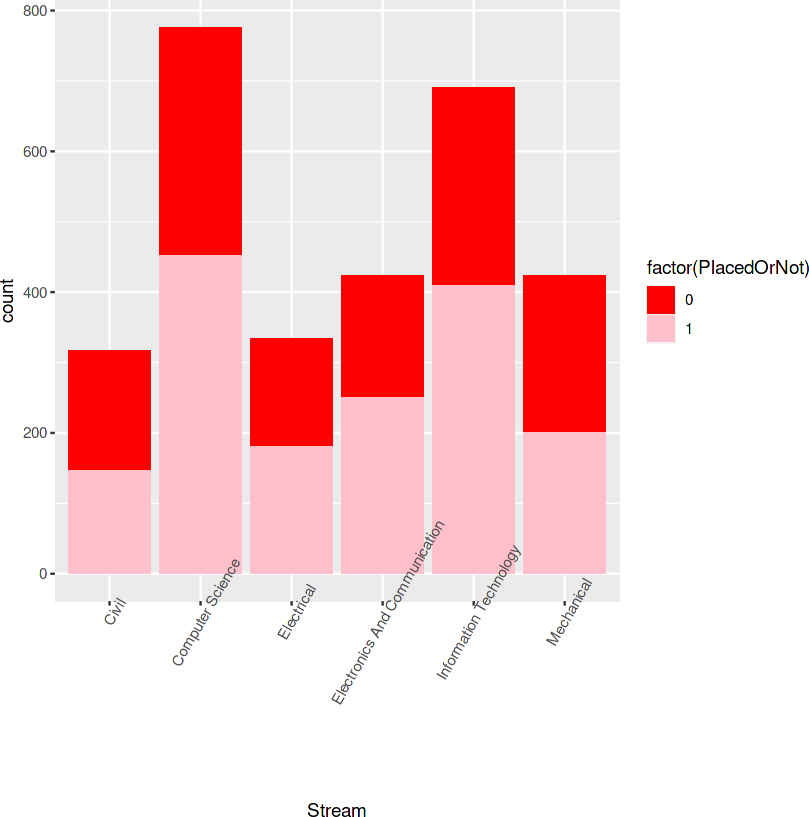
ggplot(aes(Gender, fill=Stream)) + geom\_bar()



[7]:

aggregate(PlacedOrNot~Stream, data=placement\_data, FUN=mean) placement\_data %>%

ggplot(aes(Stream)) + geom\_bar(aes(fill=factor(PlacedOrNot))) + theme(axis.text.x=element\_text(angle=60)) + scale\_fill\_manual(values=c("red", "pink"))

[8]:

num <- function(x){

x <- as.numeric(x)

}

p <- as.data.frame(lapply(placement\_data[,-c(2,3)], FUN = num))

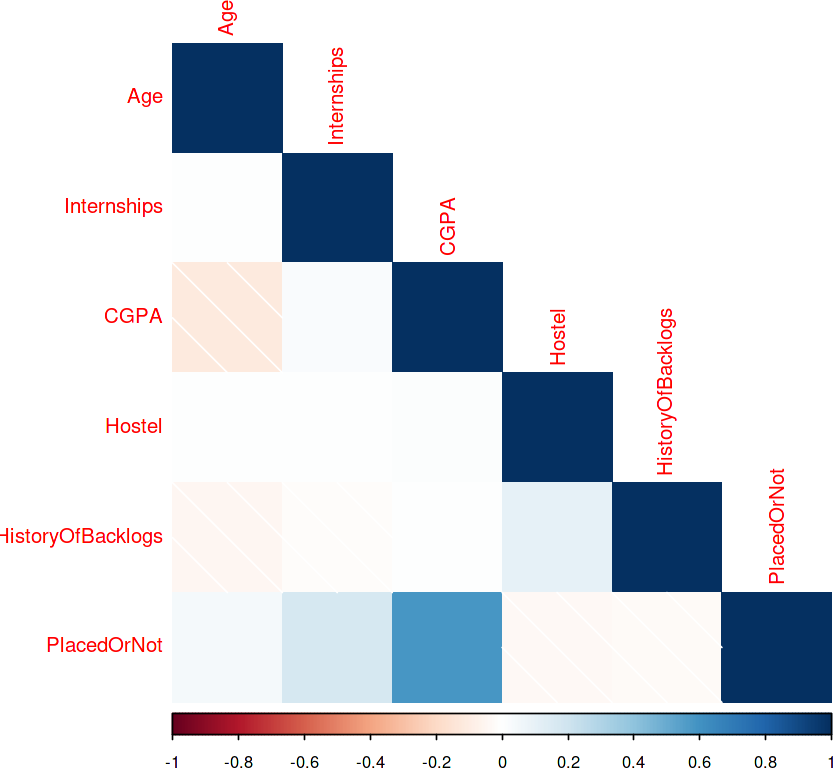
A data.frame: 6 × 2

Stream PlacedOrNot

<chr> <dbl>

|  |  |
| --- | --- |
| Civil | 0.4605678 |
| Computer Science | 0.5824742 |
| Electrical | 0.5419162 |
| Electronics And Communication | 0.5919811 |
| Information Technology | 0.5918958 |
| Mechanical | 0.4716981 |

corrplot(cor(p), method="shade", type="lower")

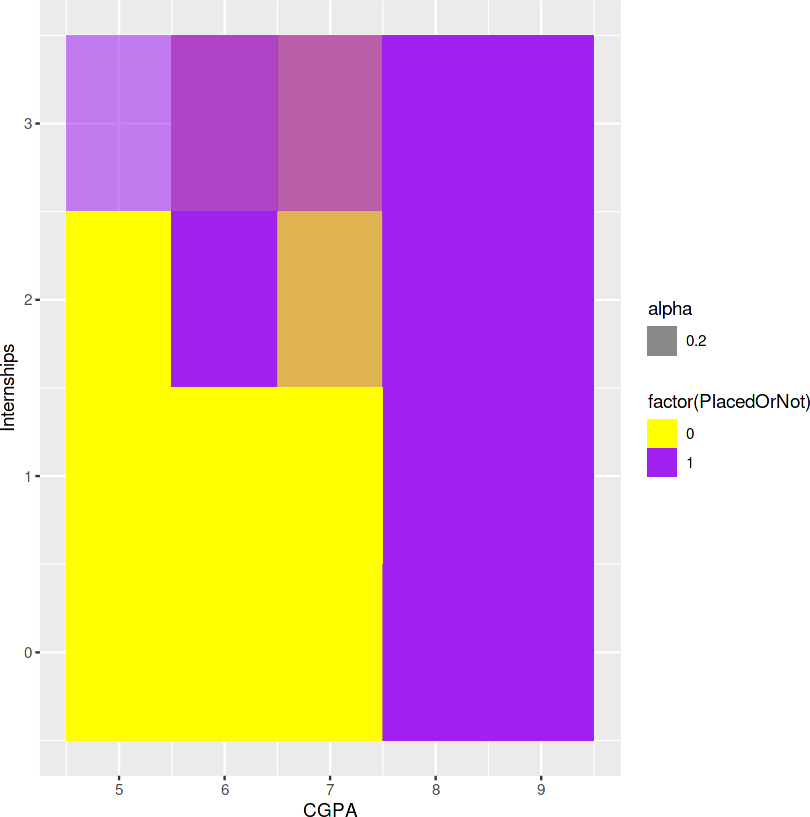


[9]:

placement\_data %>%

ggplot(aes(CGPA, Internships, fill=factor(PlacedOrNot), alpha=0.2)) + geom\_tile() +

scale\_fill\_manual(values=c("yellow", "purple"))



[10]:

aggregate(PlacedOrNot~Hostel+HistoryOfBacklogs, data=placement\_data, FUN=mean) placement\_data %>%

ggplot(aes(factor(Hostel), factor(HistoryOfBacklogs),␣

↪color=factor(PlacedOrNot))) +

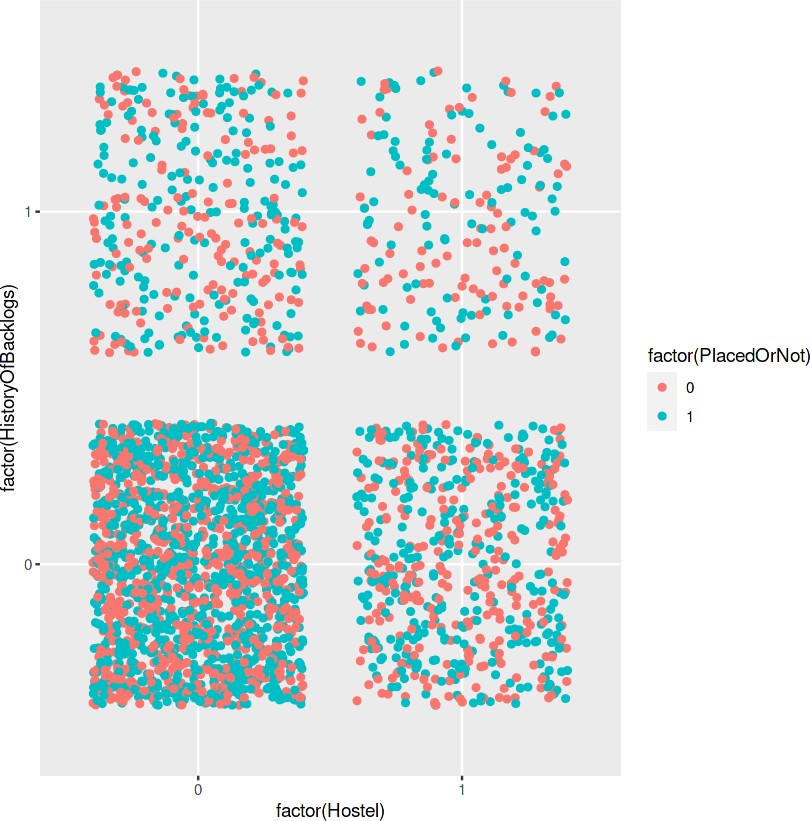
geom\_jitter()

Hostel HistoryOfBacklogs PlacedOrNot

<int> <int> <dbl>

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0.5678670 |
| 1 | 0 | 0.5279188 |
| 0 | 1 | 0.5454545 |
| 1 | 1 | 0.5024155 |

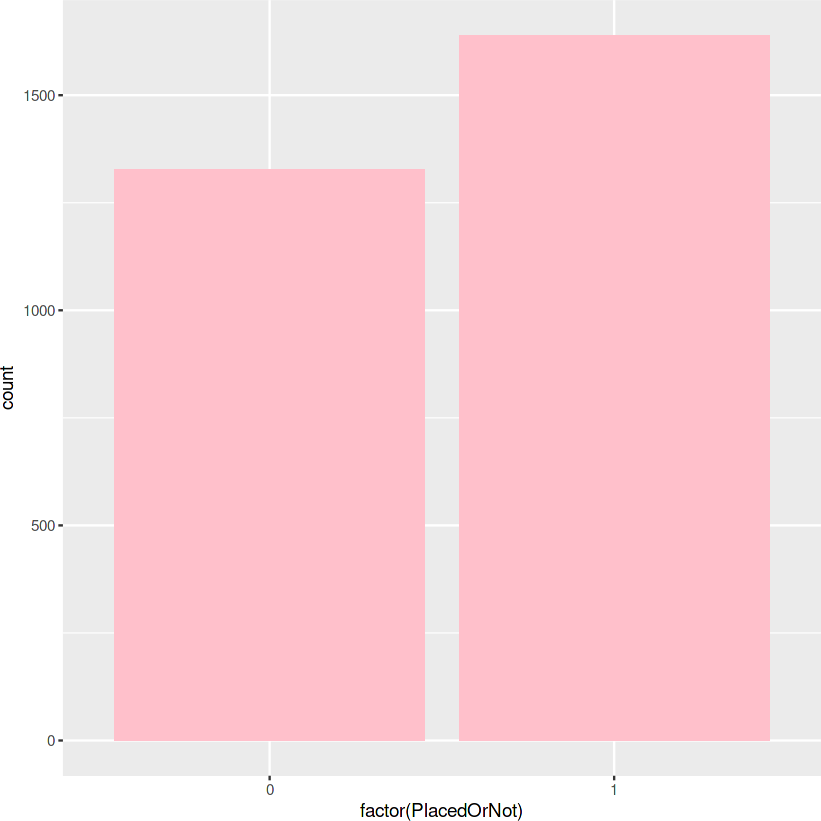
A data.frame: 4 × 3



[11]:

placement\_data %>%

ggplot(aes(factor(PlacedOrNot))) + geom\_bar(fill="pink")



[12]:

prop.table(table(placement\_data$PlacedOrNot))

0 1

0.4474039 0.5525961

* + 1. **CONCLUSIONS:**
       - Engineering students are most probably aged between 21-22.
       - Age group between 28-30 years is more likely to be placed.
       - Male and female have approximately same chances of getting placed and have more or less likely same choices of stream, that is, Computer Science and Electronics and Communication Engineering.

[13]:

* There is very less proportion of female data may be because female are less likely to take up engineering.
* Electronics and communication engineering students are more often placed followed by Infor- mation technology and Computer science.
* Higher CGPA or more internships are required to land up a job.
* Students living in hostel have less backlogs but students not living in hostel have more ten- dency to get placed even after backlogs.
  1. **MODELLING**

**Decision tree**

placement\_data$PlacedOrNot <- factor(placement\_data$PlacedOrNot,␣

↪levels=c("0","1"), labels=c("no","yes"))

[14]:

set.seed(123)

train\_in <- createDataPartition(placement\_data$PlacedOrNot, p=0.70, list=**FALSE**) train\_data <- placement\_data[train\_in,]

test\_data <- placement\_data[-train\_in,]

[15]:

model1 <- C5.0(x=train\_data[-8], train\_data$PlacedOrNot) model1

Call:

C5.0.default(x = train\_data[-8], y = train\_data$PlacedOrNot)

Classification Tree Number of samples: 2077 Number of predictors: 7

Tree size: 14

Non-standard options: attempt to group attributes

[16]:

predict1 <- predict(model1, test\_data[-8])

[17]:

CrossTable(test\_data$PlacedOrNot, predict1, prop.chisq = **FALSE**, prop.c = **FALSE**, prop.r = **FALSE**, dnn = c('actual', 'predicted'))

Cell Contents

|-------------------------|

| N |

| N / Table Total |

|-------------------------|

Total Observations in Table: 889

| predicted

actual | no | yes | Row Total |

-------------|-----------|-----------|-----------|

|  |  |  |  |
| --- | --- | --- | --- |
| no | | 393 | | 5 | | 398 | |
| | | 0.442 | | 0.006 | | | |

-------------|-----------|-----------|-----------|

|  |  |  |  |
| --- | --- | --- | --- |
| yes | | 112 | | 379 | | 491 | |
| | | 0.126 | | 0.426 | | | |

-------------|-----------|-----------|-----------|

Column Total | 505 | 384 | 889 |

-------------|-----------|-----------|-----------|

[18]:

confusionMatrix(test\_data$PlacedOrNot, predict1, positive="yes")

Confusion Matrix and Statistics

Reference Prediction no yes

no 393 5

yes 112 379

|  |  |
| --- | --- |
| Accuracy : | 0.8684 |
| 95% CI : | (0.8444, 0.8899) |
| No Information Rate : | 0.5681 |
| P-Value [Acc > NIR] : | < 2.2e-16 |
| Kappa : | 0.7405 |
| Mcnemar's Test P-Value : | < 2.2e-16 |
| Sensitivity : | 0.9870 |
| Specificity : | 0.7782 |
| Pos Pred Value : | 0.7719 |
| Neg Pred Value : | 0.9874 |
| Prevalence : | 0.4319 |
| Detection Rate : | 0.4263 |
| Detection Prevalence : | 0.5523 |
| Balanced Accuracy : | 0.8826 |
| 'Positive' Class : | yes |

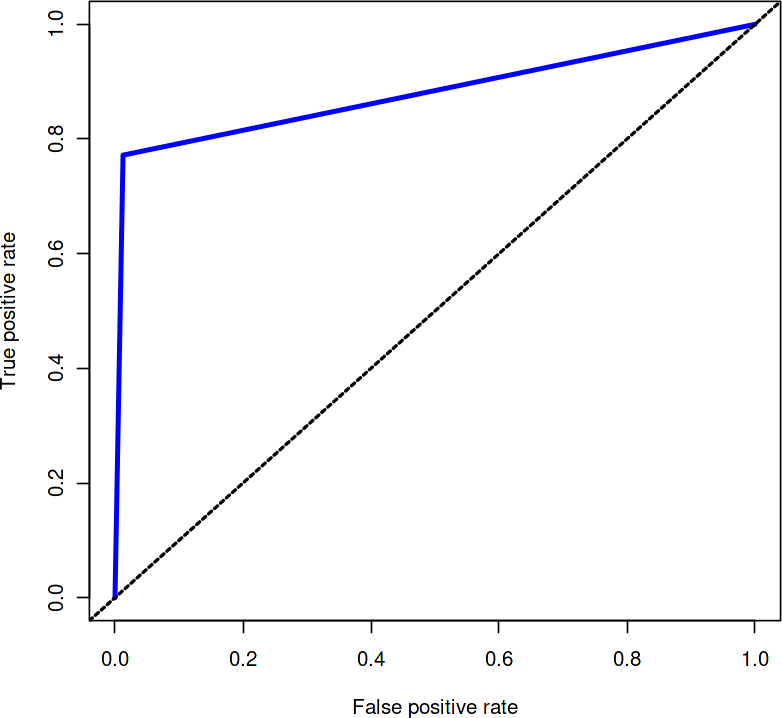
[19]:

pred <- prediction(predictions=as.data.frame(c(predict1)),␣

↪labels=test\_data$PlacedOrNot)

perf <- performance(pred, measure = "tpr", x.measure="fpr") plot(perf, lwd="3", col="blue")

abline(a = 0, b = 1, lwd = 2, lty = 2)



**Tuning the parameters**

[20]:

ctrl2 <- trainControl(method="cv", number = 10, selectionFunction="best") model2 <- train(PlacedOrNot~., data=train\_data, method="C5.0", trcontrol=ctrl2) model2

Warning message:

“'trials' should be <= 9 for this object. Predictions generated using 9 trials”

Warning message:

“'trials' should be <= 9 for this object. Predictions generated using 9 trials” Warning message:

“'trials' should be <= 7 for this object. Predictions generated using 7 trials” Warning message:

“'trials' should be <= 9 for this object. Predictions generated using 9 trials” Warning message:

“'trials' should be <= 9 for this object. Predictions generated using 9 trials”

C5.0

2077 samples

7 predictor

2 classes: 'no', 'yes'

No pre-processing

Resampling: Bootstrapped (25 reps)

Summary of sample sizes: 2077, 2077, 2077, 2077, 2077, 2077, … Resampling results across tuning parameters:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| model | winnow | trials | Accuracy | Kappa |
| rules | FALSE | 1 | 0.8848599 | 0.7700795 |
| rules | FALSE | 10 | 0.8723363 | 0.7431016 |
| rules | FALSE | 20 | 0.8774194 | 0.7541316 |
| rules | TRUE | 1 | 0.8860885 | 0.7724958 |
| rules | TRUE | 10 | 0.8713451 | 0.7410522 |
| rules | TRUE | 20 | 0.8777825 | 0.7548332 |
| tree | FALSE | 1 | 0.8825897 | 0.7655632 |
| tree | FALSE | 10 | 0.8808979 | 0.7617196 |
| tree | FALSE | 20 | 0.8822204 | 0.7645296 |
| tree | TRUE | 1 | 0.8835644 | 0.7674999 |
| tree | TRUE | 10 | 0.8813033 | 0.7626561 |
| tree | TRUE | 20 | 0.8825752 | 0.7653149 |

[21]:

[22]:

Accuracy was used to select the optimal model using the largest value.

The final values used for the model were trials = 1, model = rules and winnow

= TRUE.

predict2 <- predict(model2, test\_data[-8])

confusionMatrix(test\_data$PlacedOrNot, predict2, positive = "yes")

Confusion Matrix and Statistics

Reference Prediction no yes

no 387 11

yes 110 381

|  |  |
| --- | --- |
| Accuracy : | 0.8639 |
| 95% CI : | (0.8396, 0.8858) |
| No Information Rate : | 0.5591 |
| P-Value [Acc > NIR] : | < 2.2e-16 |
| Kappa : | 0.7311 |
| Mcnemar's Test P-Value : | < 2.2e-16 |
| Sensitivity : | 0.9719 |
| Specificity : | 0.7787 |
| Pos Pred Value : | 0.7760 |
| Neg Pred Value : | 0.9724 |
| Prevalence : | 0.4409 |
| Detection Rate : | 0.4286 |
| Detection Prevalence : | 0.5523 |
| Balanced Accuracy : | 0.8753 |
| 'Positive' Class : | yes |

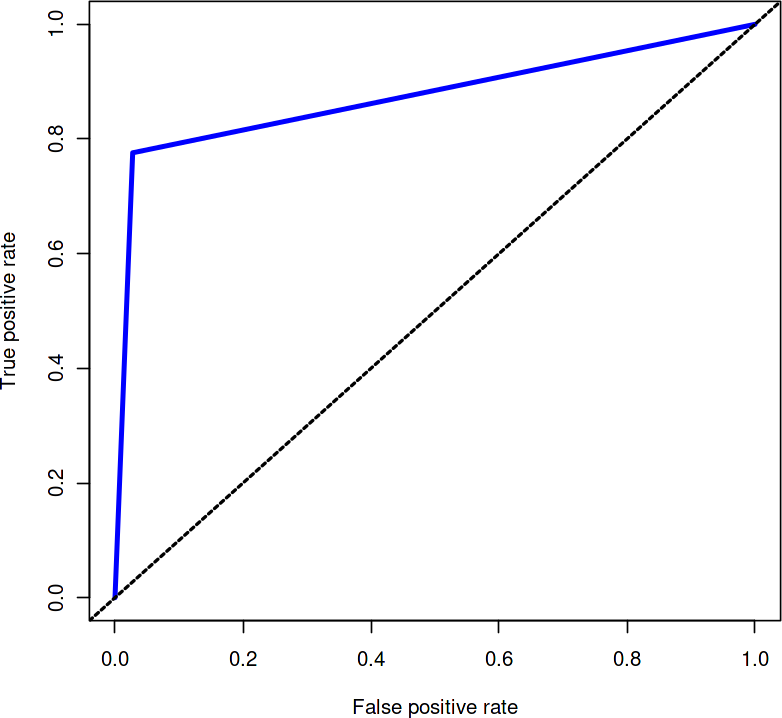
[23]:

pred <- prediction(predictions=as.data.frame(c(predict2)),␣

↪labels=test\_data$PlacedOrNot)

perf <- performance(pred, measure = "tpr", x.measure="fpr") plot(perf, lwd="3", col="blue")

abline(a = 0, b = 1, lwd = 2, lty = 2)



[24]:

model3 <- randomForest(PlacedOrNot~., data=train\_data, mtry=sqrt(7)) model3

**Random Forests**

Call:

randomForest(formula = PlacedOrNot ~ ., data = train\_data, mtry = sqrt(7)) Type of random forest: classification

Number of trees: 500 No. of variables tried at each split: 3

OOB estimate of error rate: 11.22% Confusion matrix:

[25]:

[26]:

[27]:

pred <- prediction(predictions=as.data.frame(c(predict3)),␣

↪labels=test\_data$PlacedOrNot)

perf <- performance(pred, measure = "tpr", x.measure="fpr") plot(perf, lwd="3", col="blue")

abline(a = 0, b = 1, lwd = 2, lty = 2)

no yes class.error no 878 51 0.05489774

yes 182 966 0.15853659

predict3 <- predict(model3, test\_data[-8])

confusionMatrix(test\_data$PlacedOrNot, predict3, positive="yes")

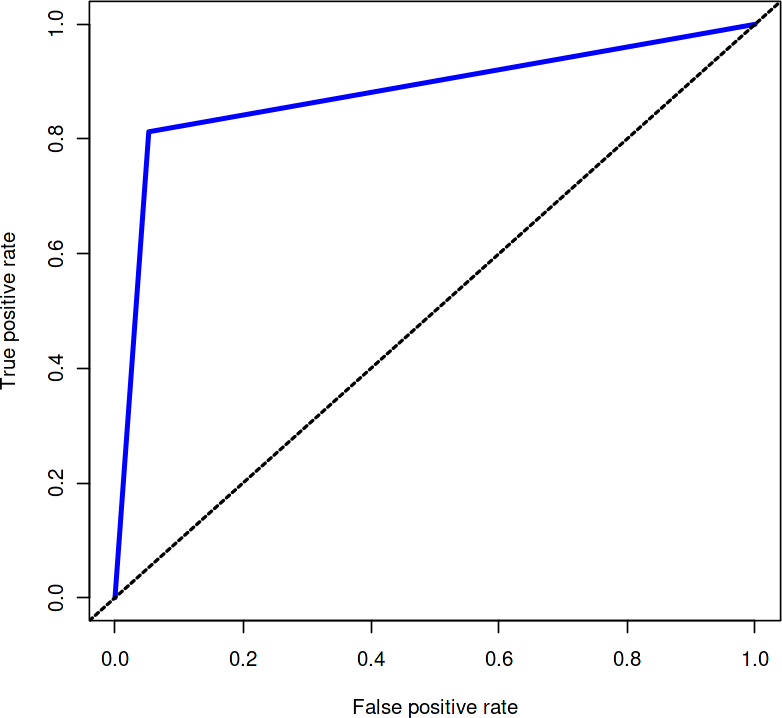
Confusion Matrix and Statistics

Reference Prediction no yes

no 377 21

yes 92 399

|  |  |
| --- | --- |
| Accuracy : | 0.8729 |
| 95% CI : | (0.8492, 0.8941) |
| No Information Rate : | 0.5276 |
| P-Value [Acc > NIR] : | < 2.2e-16 |
| Kappa : | 0.7472 |
| Mcnemar's Test P-Value : | 4.547e-11 |
| Sensitivity : | 0.9500 |
| Specificity : | 0.8038 |
| Pos Pred Value : | 0.8126 |
| Neg Pred Value : | 0.9472 |
| Prevalence : | 0.4724 |
| Detection Rate : | 0.4488 |
| Detection Prevalence : | 0.5523 |
| Balanced Accuracy : | 0.8769 |
| 'Positive' Class : | yes |



**Tuning the Parameters**

[28]:

ctrl4 <- trainControl(method="repeatedcv", number=10, repeats=10) grid4 <- expand.grid(.mtry=c(2,4,8,16))

model4 <- train(PlacedOrNot~., data=train\_data, method="rf",␣

↪trControl=ctrl4,tuneGrid=grid4)

model4

Random Forest

2077 samples

7 predictor

2 classes: 'no', 'yes'

No pre-processing

Resampling: Cross-Validated (10 fold, repeated 10 times) Summary of sample sizes: 1869, 1870, 1870, 1870, 1869, 1869, … Resampling results across tuning parameters:

|  |  |
| --- | --- |
| mtry Accuracy | Kappa |
| 2 0.8774718 | 0.7556445 |
| 4 0.8871513 | 0.7748720 |
| 8 0.8857996 | 0.7715762 |
| 16 0.8848859 | 0.7698060 |

[29]:

[30]:

[31]:

pred <- prediction(predictions=as.data.frame(c(predict4)),␣

↪labels=test\_data$PlacedOrNot)

Accuracy was used to select the optimal model using the largest value. The final value used for the model was mtry = 4.

predict4 <- predict(model4, test\_data[-8])

confusionMatrix(test\_data$PlacedOrNot, predict4, positive="yes")

Confusion Matrix and Statistics

Reference Prediction no yes

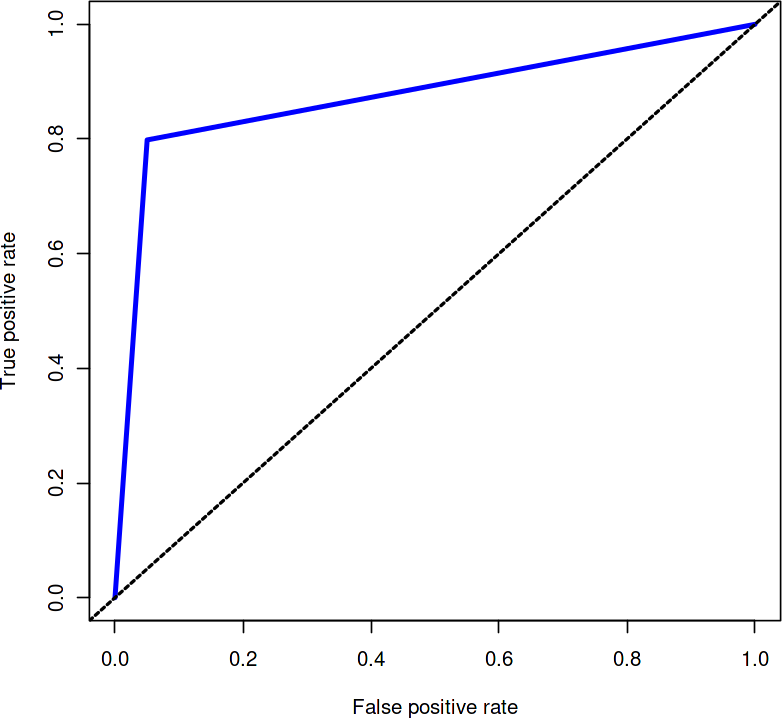
no 378 20

yes 99 392

|  |  |
| --- | --- |
| Accuracy : | 0.8661 |
| 95% CI : | (0.842, 0.8878) |
| No Information Rate : | 0.5366 |
| P-Value [Acc > NIR] : | < 2.2e-16 |
| Kappa : | 0.7343 |
| Mcnemar's Test P-Value : | 8.662e-13 |
| Sensitivity : | 0.9515 |
| Specificity : | 0.7925 |
| Pos Pred Value : | 0.7984 |
| Neg Pred Value : | 0.9497 |
| Prevalence : | 0.4634 |
| Detection Rate : | 0.4409 |
| Detection Prevalence : | 0.5523 |
| Balanced Accuracy : | 0.8720 |
| 'Positive' Class : | yes |

perf <- performance(pred, measure = "tpr", x.measure="fpr") plot(perf, lwd="3", col="blue")

abline(a = 0, b = 1, lwd = 2, lty = 2)



This shows that Random Forest works best here with 87.29% accuracy on testing data

## Extended Models Building

April 10, 2023

[2]:

*#IMPORT THE LIBRARIES....*

**import numpy as np** *# linear algebra....*

**import pandas as pd** *# data processing, CSV file I/O (e.g. pd.read\_csv)....*

**from matplotlib import** pyplot **as** plt *#Visualization of the data....*

**import warnings** warnings.filterwarnings("ignore") pd.set\_option("display.max\_columns",**None**) pd.set\_option("display.max\_rows",**None**) **import plotly.graph\_objs as go**

**import matplotlib as mpl**

**import matplotlib.patches as mpatches import seaborn as sns**

**import plotly.express as px**

**from sklearn.preprocessing import** StandardScaler

**from plotly import** tools

**from plotly.subplots import** make\_subplots

**from plotly.offline import** iplot

[3]:

**import os**

**for** dirname, \_, filenames **in** os.walk('/kaggle/input'):

**for** filename **in** filenames: print(os.path.join(dirname, filename))

/kaggle/input/engineering-placements-prediction/collegePlace.csv

[4]:

df=pd.read\_csv("../input/engineering-placements-prediction/collegePlace.csv") df.head(10).style.set\_properties(\*\*{"background-color": "black","color":␣

↪"white", "border-color": "black","font-size":"11.5pt",'width': 200})

[4]: <pandas.io.formats.style.Styler at 0x73b61222f690> [5]:

*#The shape of the dataset.....*

df.shape

[5]: (2966, 8)

[6]:

*#The dimensions of the dataset.......*

df.ndim

[6]: 2

[7]:

*#The size of the dataset.......*

df.size

[7]: 23728

[8]:

*#The columns we have in the dataset.....*

df.columns

1. : Index(['Age', 'Gender', 'Stream', 'Internships', 'CGPA', 'Hostel', 'HistoryOfBacklogs', 'PlacedOrNot'],

dtype='object')

[9]:

*#The dtypes we have in the dataset.....*

df.dtypes

1. : Age int64

Gender object

Stream object

Internships int64

CGPA int64

Hostel int64

HistoryOfBacklogs int64 PlacedOrNot int64 dtype: object

[10]:

*#The Information of the Dataset*

df.info()

[12]:

<class 'pandas.core.frame.DataFrame'> RangeIndex: 2966 entries, 0 to 2965 Data columns (total 8 columns):

# Column Non-Null Count Dtype

1. Age 2966 non-null int64
2. Gender 2966 non-null object
3. Stream 2966 non-null object
4. Internships 2966 non-null int64
5. CGPA 2966 non-null int64
6. Hostel 2966 non-null int64
7. HistoryOfBacklogs 2966 non-null int64
8. PlacedOrNot 2966 non-null int64 dtypes: int64(6), object(2)

memory usage: 185.5+ KB

df.duplicated().sum()

[12]: 1829

[13]:

df.drop\_duplicates(inplace=**True**)

[14]:

df.skew()

|  |  |  |
| --- | --- | --- |
| [14]: | Age | 0.817990 |
|  | Internships | 0.697572 |
|  | CGPA | 0.122766 |
|  | Hostel | 0.699651 |
|  | HistoryOfBacklogs | 0.892883 |
|  | PlacedOrNot | -0.322920 |
|  | dtype: float64 |  |

[15]:

*# statistics on Categorical data......*

round(df.describe(exclude = 'object'), 2).style.

↪set\_properties(\*\*{"background-color": "black","color": "white",␣

↪"border-color": "black","font-size":"11.5pt",'width': 200})

1. : <pandas.io.formats.style.Styler at 0x73b60973ab90> [16]:

*# statistics on numerical data*

round(df.describe(exclude = ['float', 'int64']),2).style.

↪set\_properties(\*\*{"background-color": "black","color": "white",␣

↪"border-color": "black","font-size":"11.5pt",'width': 200})

1. : <pandas.io.formats.style.Styler at 0x73b609714550> [17]:

*# list of numerical variables............*

numerical\_features = [feature **for** feature **in** df.columns **if** df[feature].dtypes !

↪= 'O']

print('Number of numerical variables: ', len(numerical\_features)) print('**\n**')

print('Numeric Column name',numerical\_features) print('**\n**')

*# visualise the numerical variables........*

df[numerical\_features].head().style.set\_properties(\*\*{"background-color":␣

↪"black","color": "white", "border-color": "black","font-size":"11.

↪5pt",'width': 200})

Number of numerical variables: 6

Numeric Column name ['Age', 'Internships', 'CGPA', 'Hostel', 'HistoryOfBacklogs', 'PlacedOrNot']

1. : <pandas.io.formats.style.Styler at 0x73b60916f590> [19]:

df\_nunique = {var: pd.DataFrame(df[var].value\_counts())

**for** var **in** {'Age'}}

multi\_table([ df\_nunique['Age'].style.set\_properties(\*\*{"background-color":␣

↪"black","color": "white", "border-color": "black","font-size":"11.

↪5pt",'width': 200})])

[19]: <IPython.core.display.HTML object>

[20]:

print('Maximum age of the students:',df['Age'].max()) print('Manimum age of the students:',df['Age'].min()) print('Average age of the students:',df['Age'].mean())

[54]:

Maximum age of the students: 30 Manimum age of the students: 19

Average age of the students: 21.641160949868073

*# Changing the Gender to 0 and 1 (Hidden Input/Output)*

df["Gender"] = df["Gender"].map({"Male":1, "Female":0})

[55]:

df.drop(['Age'], axis = 1,inplace = **True**)

[56]:

df.head(5).style.set\_properties(\*\*{"background-color": "black","color":␣

↪"white", "border-color": "black","font-size":"11.5pt",'width': 200})

[56]: <pandas.io.formats.style.Styler at 0x73b6085b9b90> [58]:

corrmat = df.corr() top\_corr\_features = corrmat.index

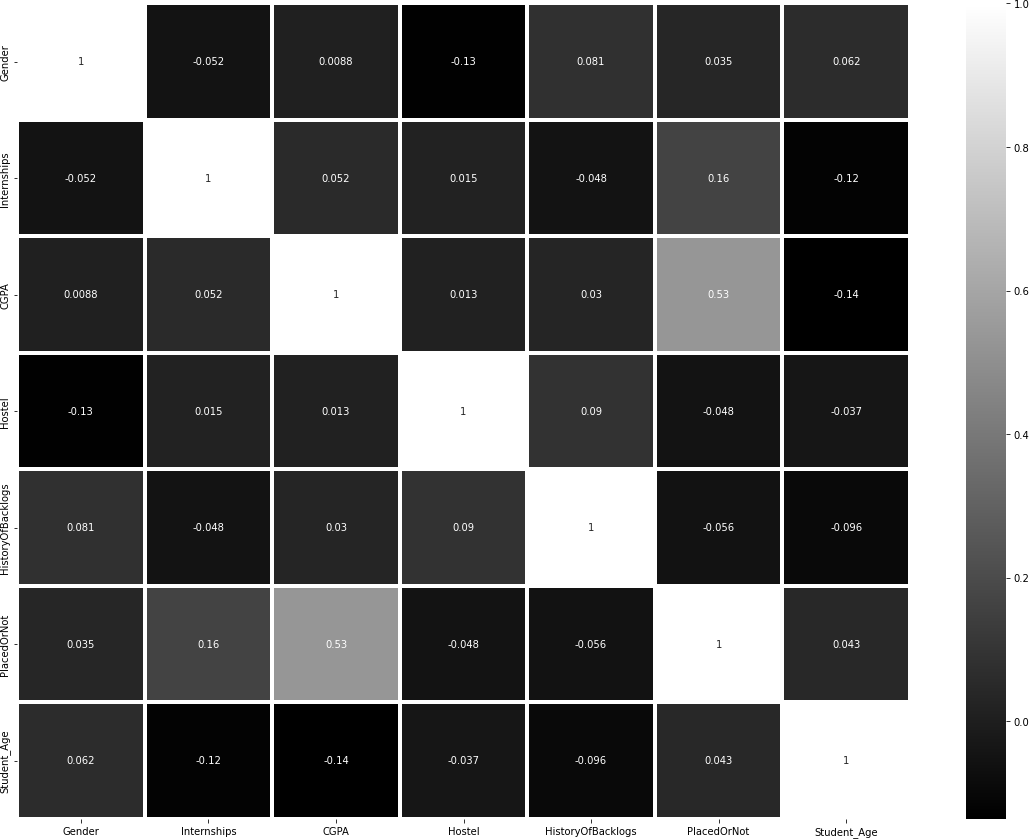
plt.figure(figsize=(20,15))

*#plot heat map*

g=sns.heatmap(df[top\_corr\_features].

↪corr(),annot=**True**,cmap="gist\_yarg\_r",linewidths=3.5,

linecolor='white')



[59]:

**import matplotlib**

background\_color = "#f6f6f6"

fig = plt.figure(figsize=(14,8), facecolor=background\_color) gs = fig.add\_gridspec(1, 1)

ax0 = fig.add\_subplot(gs[0, 0]) colors = ["#c6ff1a"]

colormap = matplotlib.colors.LinearSegmentedColormap.from\_list("", colors)

ax0.set\_facecolor(background\_color)

ax0.text(-1.1, 1.25, 'Correlation of Numerical Features with Target',␣

↪fontsize=20, fontweight='bold')

chart\_df = pd.DataFrame(df.corrwith(df['PlacedOrNot'])) chart\_df.columns = ['corr']

sns.barplot(x=chart\_df.index, y=chart\_df['corr'], ax=ax0, color='#d2ff4d',␣

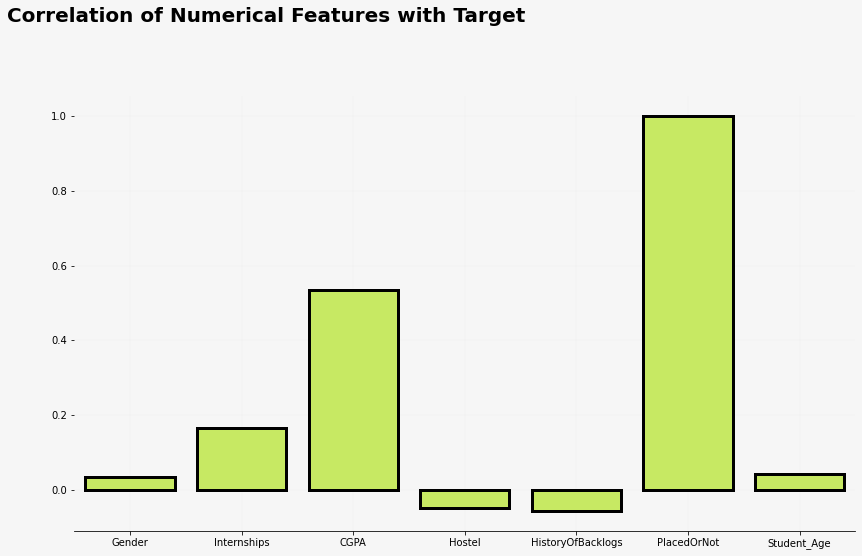
↪zorder=3, edgecolor='black', linewidth=3)

ax0.grid(which='major', axis='x', zorder=0, color='#EEEEEE', linewidth=0.4)

ax0.grid(which='major', axis='y', zorder=0, color='#EEEEEE', linewidth=0.4) ax0.set\_ylabel('')

**for** s **in** ["top","right", 'left']: ax0.spines[s].set\_visible(**False**)

plt.show()



[60]:

*#Converting categorical into numeric.*

dummy\_stream = pd.get\_dummies(df['Stream'])

[61]:

df = pd.concat([df.drop(["Stream"], axis = 1),dummy\_stream], axis = 1)

df.head(5).style.set\_properties(\*\*{"background-color": "black","color":␣

↪"white", "border-color": "black","font-size":"11.5pt",'width': 200})

1. : <pandas.io.formats.style.Styler at 0x73b6118a4f50> [62]:

*#Rearrange columns*

df = df[['Student\_Age','Gender','Civil', 'Computer Science', 'Electrical', 'Electronics And Communication', 'Information Technology', 'Mechanical', 'Internships', 'CGPA', 'Hostel',␣

↪'HistoryOfBacklogs','PlacedOrNot']]

df.head(5).style.set\_properties(\*\*{"background-color": "black","color":␣

↪"white", "border-color": "black","font-size":"11.5pt",'width': 200})

1. : <pandas.io.formats.style.Styler at 0x73b6059d2190> [63]:

scaler = StandardScaler()

scaler.fit(df.drop('PlacedOrNot',axis=1))

scaled\_features = scaler.transform(df.drop('PlacedOrNot',axis=1))

[64]:

scaled\_features = pd.DataFrame(scaled\_features, columns = df.columns[:-1]) scaled\_features.head().style.set\_properties(\*\*{"background-color":␣

↪"black","color": "white", "border-color": "black","font-size":"11.

↪5pt",'width': 200})

[64]: <pandas.io.formats.style.Styler at 0x73b6059e8fd0> [65]:

X = scaled\_features

y = df['PlacedOrNot']

[66]:

**from sklearn.model\_selection import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size = 0.

↪30,random\_state = 0)

[67]:

[68]:

*# =============================================================================*

*# Cross validation on differnet set of algorithm!!!*

*# ============================================================================= ################################################################*

kfold = StratifiedKFold(n\_splits=8,shuffle=**True**, random\_state=42)

rs = 15 clrs = []

clrs.append(AdaBoostClassifier(random\_state=rs))

**from sklearn.ensemble import** RandomForestClassifier␣

↪,AdaBoostClassifier,BaggingClassifier,ExtraTreesClassifier,GradientBoostingClassifier

**from sklearn.model\_selection import** GridSearchCV, cross\_val\_score,␣

↪StratifiedKFold, learning\_curve ,KFold

**from sklearn.metrics import**␣

↪roc\_curve,accuracy\_score,f1\_score,auc,confusion\_matrix,roc\_auc\_score,plot\_confusion\_matrix

**from xgboost.sklearn import** XGBClassifier

**from sklearn.metrics import**␣

↪classification\_report,confusion\_matrix,accuracy\_score

**from sklearn.tree import** DecisionTreeClassifier

clrs.append(GradientBoostingClassifier(random\_state=rs)) clrs.append(RandomForestClassifier(random\_state=rs)) clrs.append(ExtraTreesClassifier(random\_state = rs)) clrs.append(DecisionTreeClassifier(random\_state = rs))

cv\_results = []

**for** clr **in** clrs :

cv\_results.append(cross\_val\_score(clr, X\_train, y\_train , scoring =␣

↪'accuracy', cv = kfold, n\_jobs=-1))

cv\_means = [] cv\_std = []

**for** cv\_result **in** cv\_results: cv\_means.append(cv\_result.mean()) cv\_std.append(cv\_result.std())

cv\_df = pd.DataFrame({"CrossVal\_Score\_Means":cv\_means,"CrossValerrors":␣

↪cv\_std,"Algo":["RandomForestClassifier","AdaBoostClassifier","Gradient␣

↪Boosting",'ExtraTreesClassifier','DecisionTreeClassifier']})

[69]:

g = sns.barplot("CrossVal\_Score\_Means","Algo",data = cv\_df,orient =␣

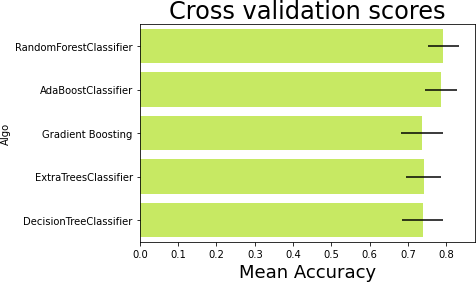
↪"h",\*\*{'xerr':cv\_std},color = '#d2ff4d') g.set\_xlabel("Mean Accuracy",fontsize = 18)

g = g.set\_title("Cross validation scores",fontsize = 24) plt.figure(figsize = (12,8))

print(cv\_df)

CrossVal\_Score\_Means CrossValerrors Algo

* 1. 0.792386 0.039934 RandomForestClassifier
  2. 0.786111 0.042556 AdaBoostClassifier
  3. 0.737184 0.055406 Gradient Boosting
  4. 0.740896 0.045384 ExtraTreesClassifier
  5. 0.738359 0.053756 DecisionTreeClassifier



<Figure size 864x576 with 0 Axes>

[70]:

**from sklearn.tree import** DecisionTreeClassifier dtc = DecisionTreeClassifier()

dtc.fit(X\_train, y\_train) y\_pred = dtc.predict(X\_test)

[71]:

confusion\_matrix(y\_test, y\_pred)

[71]: array([[120, 33],

[ 53, 136]])

[72]:

print(classification\_report(y\_test, y\_pred))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.69 | 0.78 | 0.74 | 153 |
| 1 | 0.80 | 0.72 | 0.76 | 189 |
| accuracy |  |  | 0.75 | 342 |
| macro avg | 0.75 | 0.75 | 0.75 | 342 |
| weighted avg | 0.76 | 0.75 | 0.75 | 342 |

[73]:

print(accuracy\_score(y\_test, y\_pred))

0.7485380116959064

[79]:

print(grid\_search.best\_params\_) print(grid\_search.best\_score\_)

{'algorithm': 'SAMME.R', 'learning\_rate': 0.001, 'n\_estimators': 180}

0.7850632911392406