CAPSTONE PROJECT

Cricket Win Prediction

By

C S Vaidhyeesswaran

Table of contents:

Tables and figures

1) INTRODUCTION:

Almost every match prediction that we make begins by providing match details. In other words, you'll be able to know who is playing, when the match will be played, and where it will take place. This is very important if you're going to research weather conditions, which could also be beneficial for making the right bet when wagering on cricket.

Problem Statement:

BCCI has hired an external analytics consulting firm for data analytics. The major objective of this tie up is to extract actionable insights from the historical match data and make strategic changes to make India win. Primary objective is to create Machine Learning models which correctly predicts a win for the Indian Cricket Team. Once a model is developed then you have to extract actionable insights and recommendation.

Need of the Study:

Also, below are the details of the next 10 matches, India is going to play. You have to predict the result of the matches and if you are getting prediction as a Loss then suggest some changes and re-run your model again until you are getting Win as a prediction. You cannot use the same strategy in the entire series, because opponent will get to know your strategy and they can come with counter strategy. Hence for all the below 5 matches you have to suggest unique strategies to make India win. The suggestions should be in-line with the variables that have been mentioned in the given data set. Do consider the feasibility of the suggestions very carefully as well.

1. One Test match with England in England. All the match are day matches. In England, it will be rainy season at the time to match.

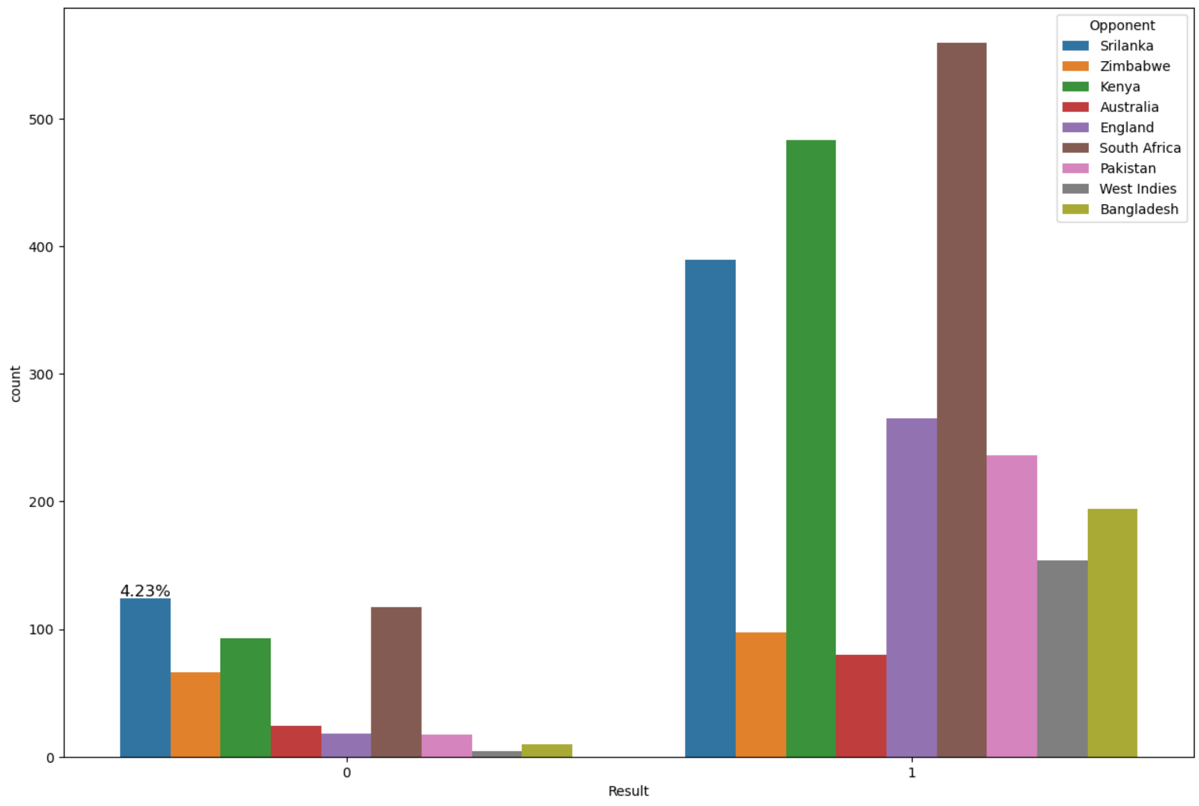
2. Two T20 match with Australia in India. All the match are Day and Night matches. In India, it will be winter season at the time to match.

3. Two ODI match with Sri Lanka in India. All the match are Day and Night matches. In India, it will be winter season at the time to match.

3) Exploratory Data Analysis:

a) Univariate Analysis:

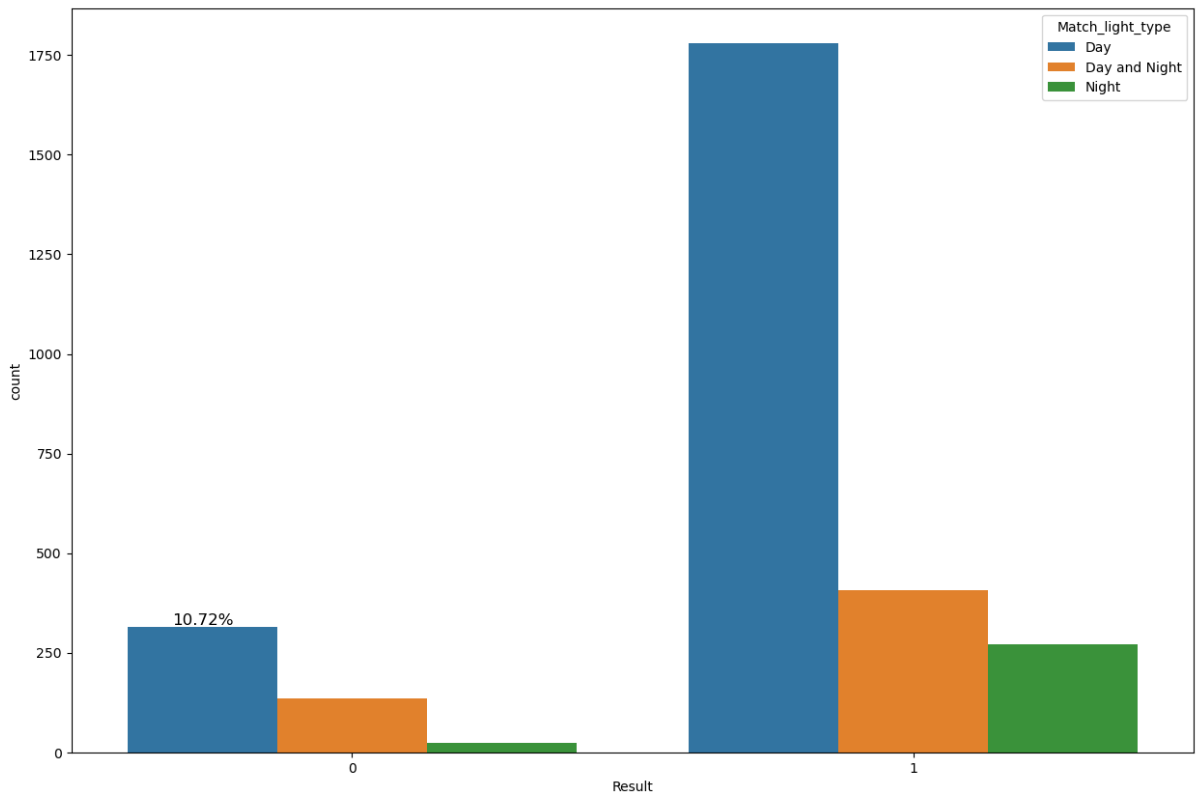
Win/Loss distribution against different opponents



This distribution shows that the data contains major proportion of wins over lose. But if we just consider the data

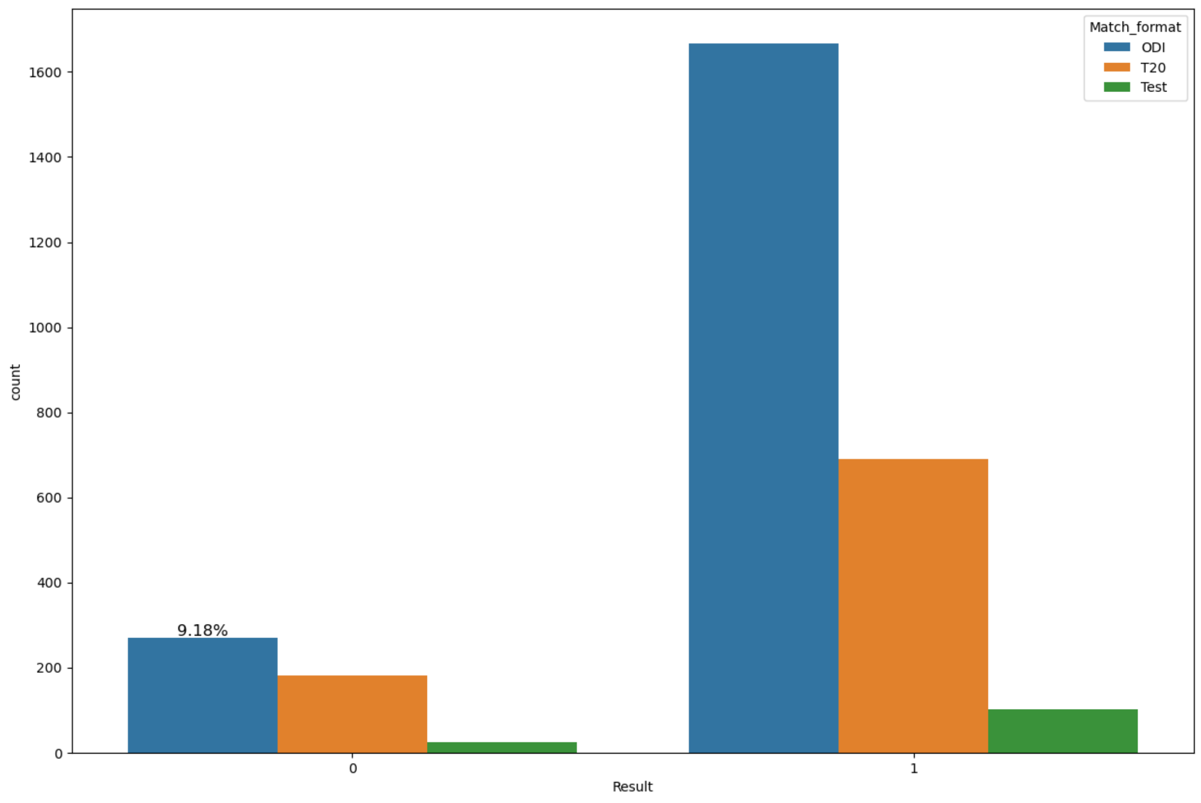
Top three teams against India have won is South Africa-19% of times, Kenya-16.5% of times and Srilanka-13.3% of times.

Win/Loss Distribution against on basis of Match Light type



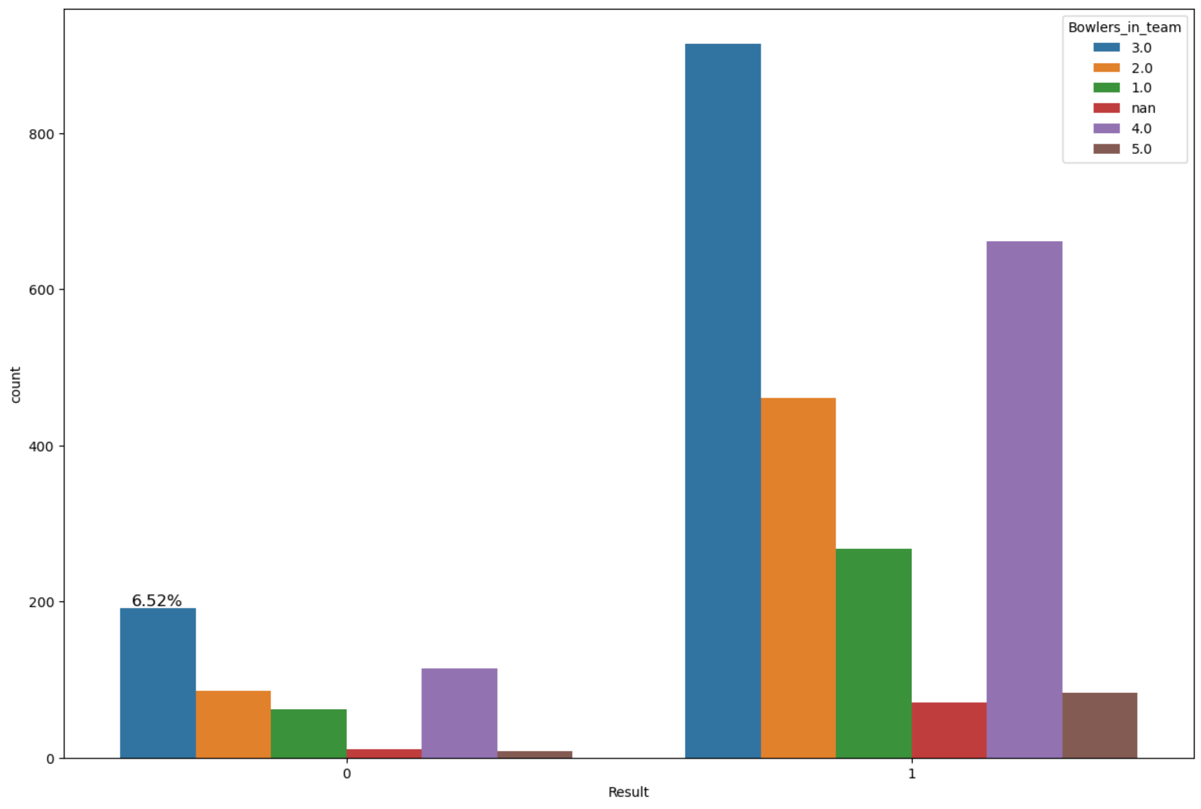
India has won maximum matches in Day time-60% of time, Day and Night matches-14% and Night matches 9.3% of times.

Win/Loss distribution against Match type



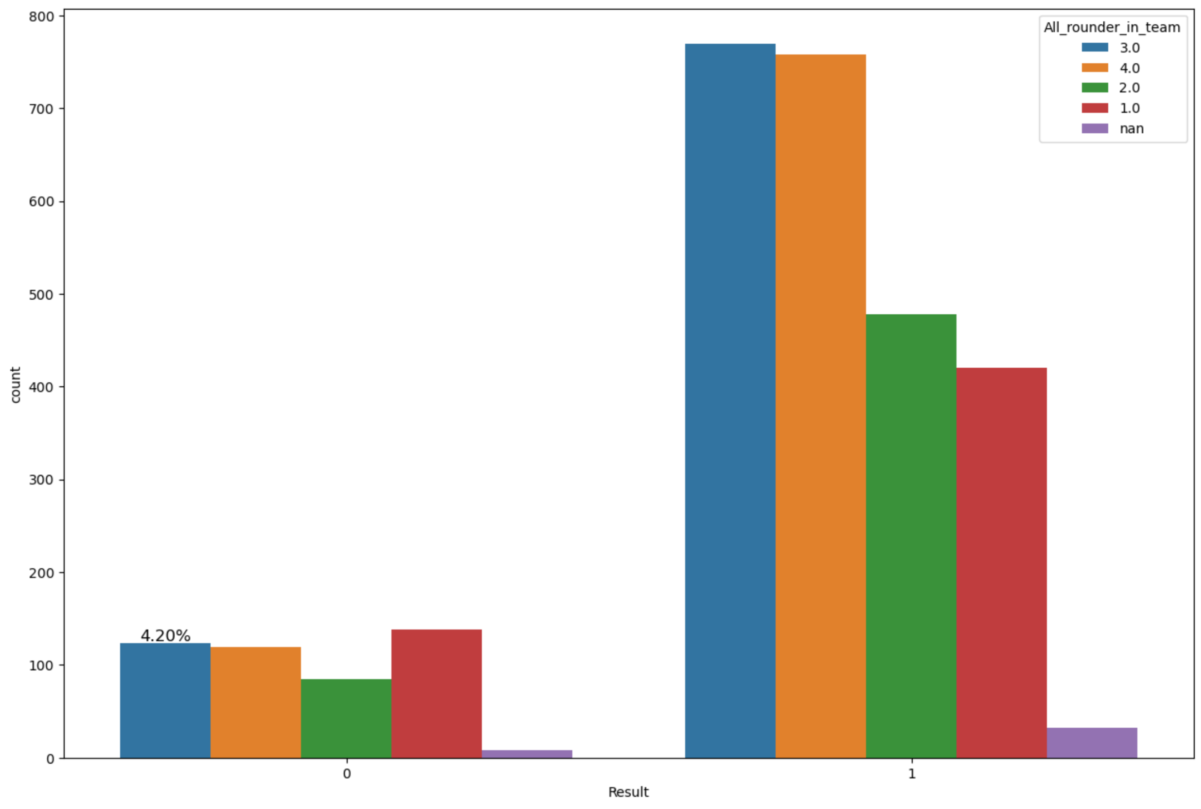
India has maximum proportion of wins in ODI that is 57%, then T20 13.5% and Test i.e., 3.5%.

Win/Loss distribution against Bowlers in Team



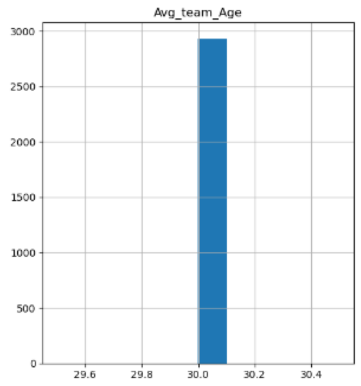
India has won 31.2% of times with 3 number of bowlers in team, then 4 number of bowlers i.e., 23% of times.

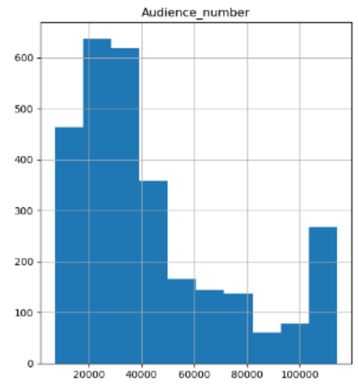
Win/Loss Distribution against All rounders

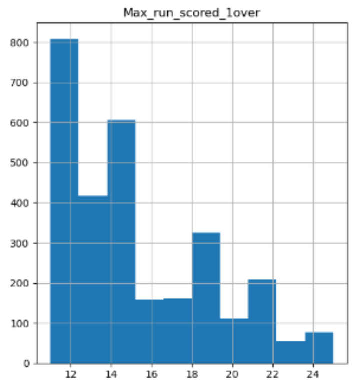


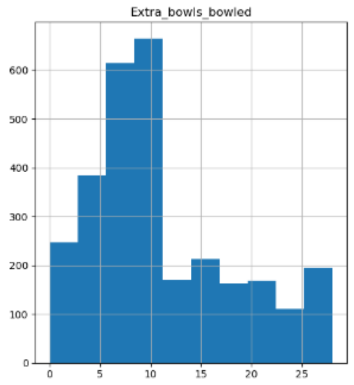
India has won major matches with 3 number of all rounders in team 26.3% of times, 25.9% with 4 all rounders.

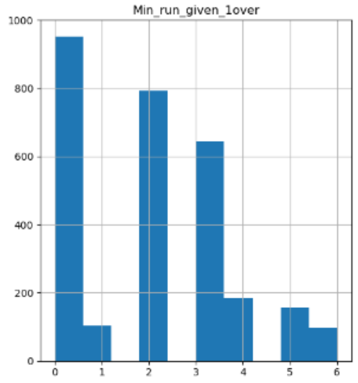
Histogram Distribution:

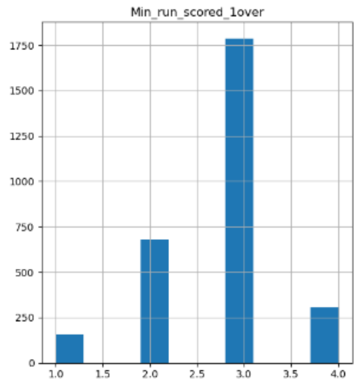


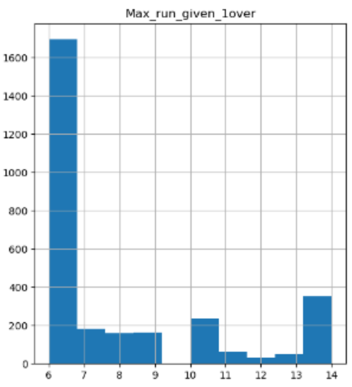


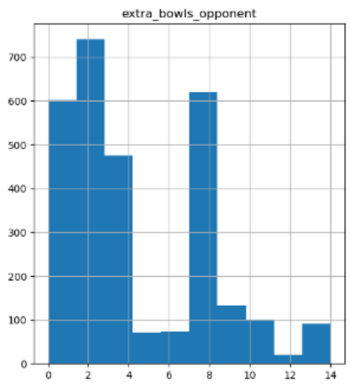


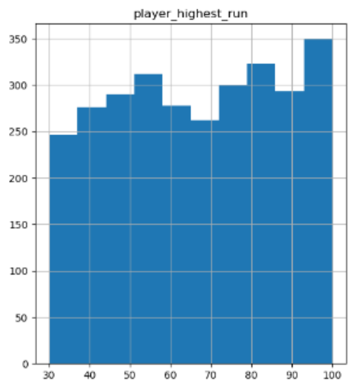


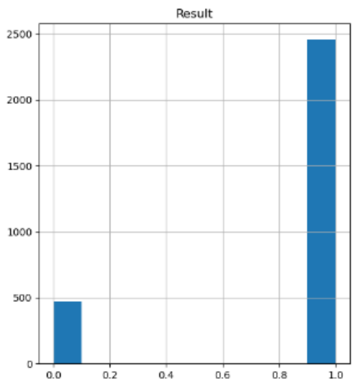








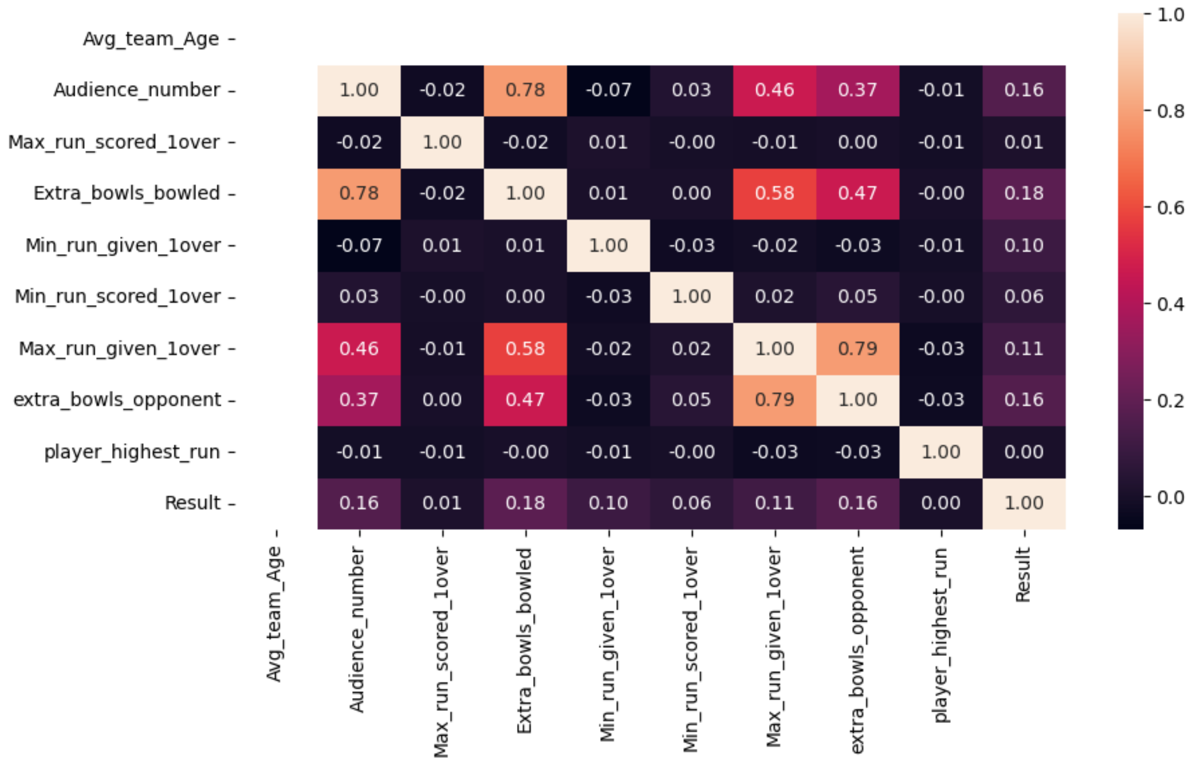




Out of all the histograms available above Player Highest run and Audience number are continuous and have a distribution on histogram. For the features other than the mentioned are showing discrete nature. Player highest run is a uniform distribution whereas audience number is a skewed distribution to the right.

By seeing the Bivariate analysis we will have a better understanding

b) Bivariate Analysis

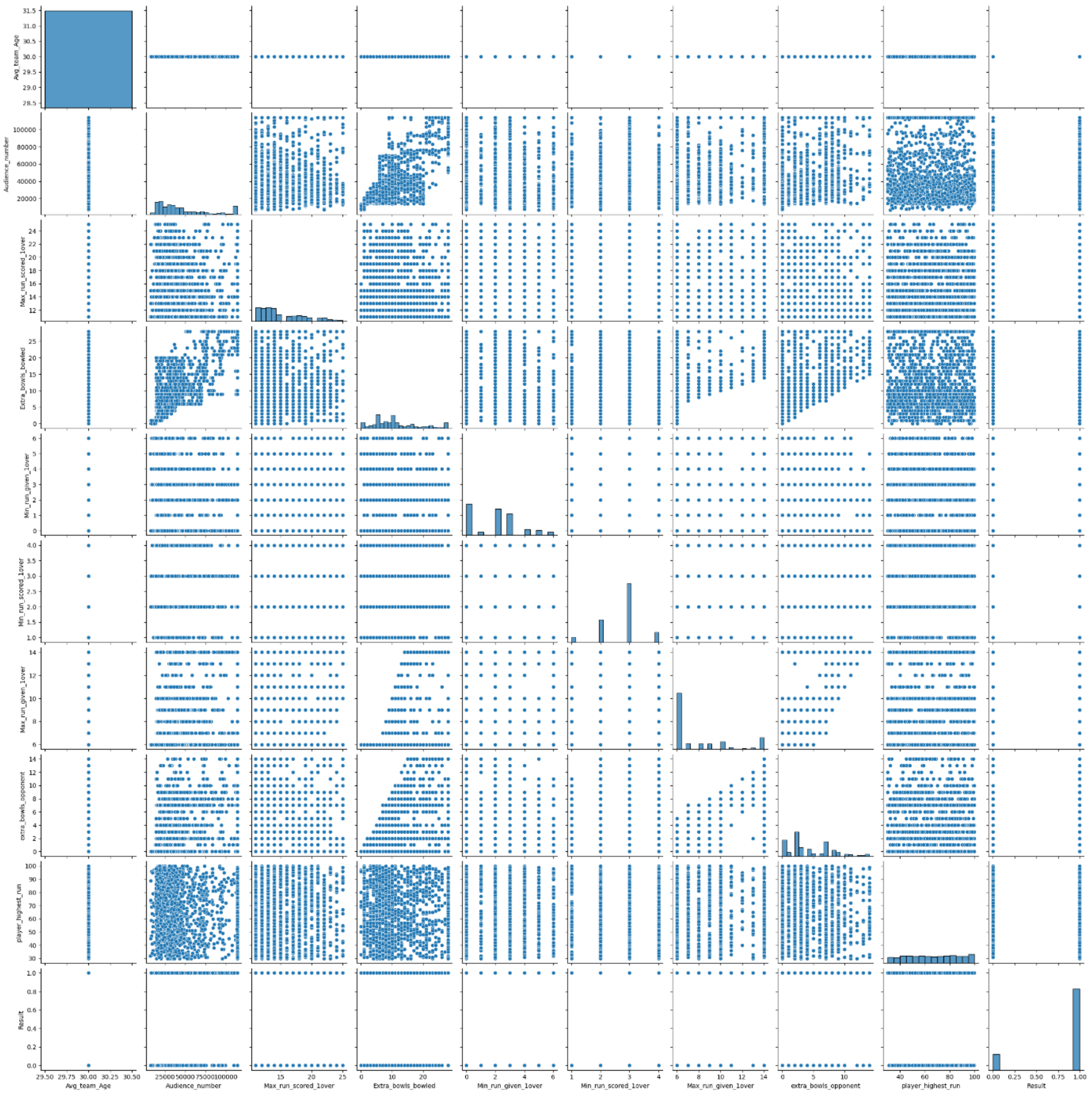


From the heat map we cannot see any significant correlation of any feature with our target variable that is the result. Although we can see some significant correlation between the features.

1. Audience number and player highest wicket-94%
2. Player highest wicket and extra bowls bowled-79%

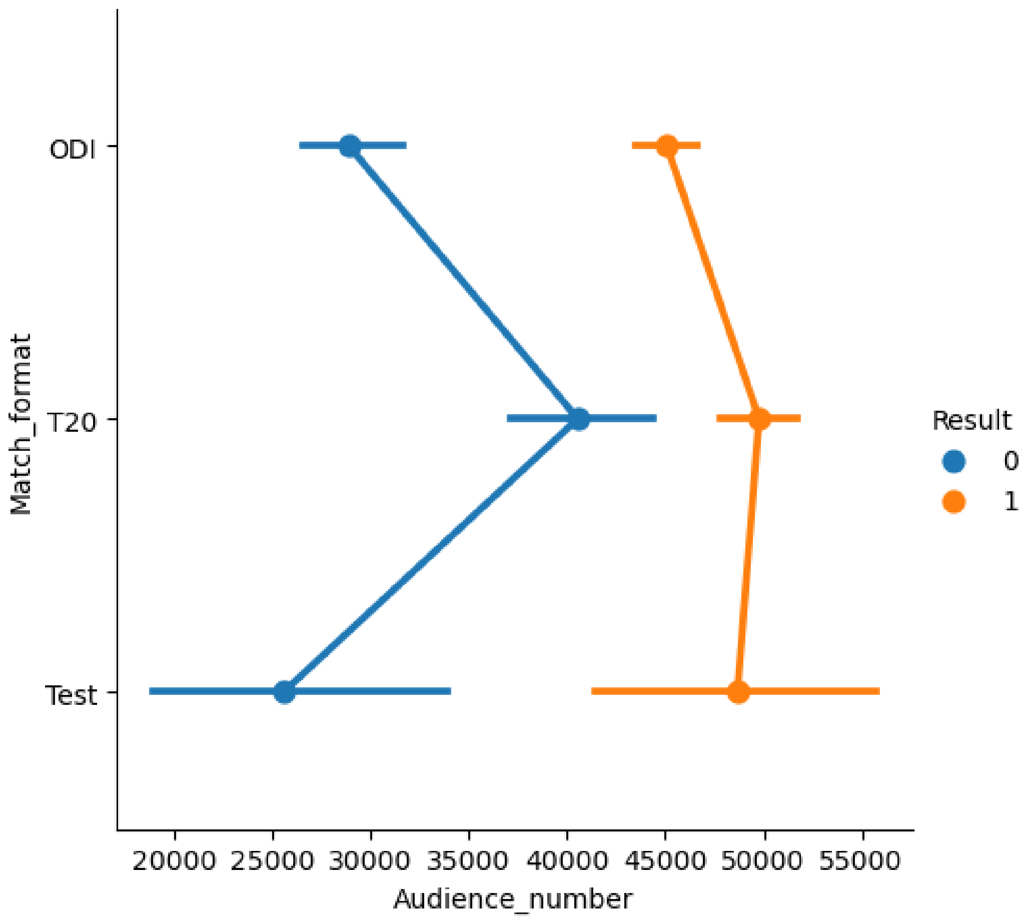
But both these correlations are for discrete in nature as we can see in the scatterplot.

From the pairplot we can see that the features don’t have a significant linear relationship between variables.



Multivariate Analysis:

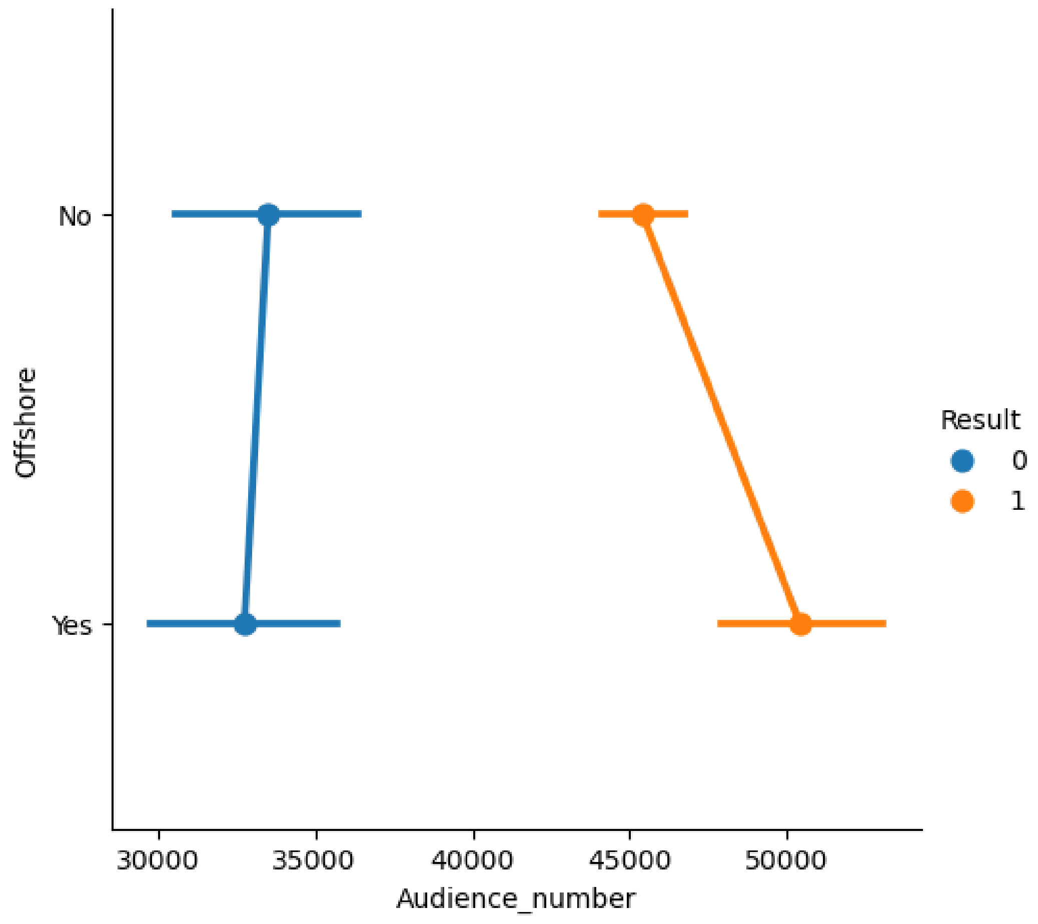
Relationship between Audience number match win/loss and match_format



From this we can interpret quite a few things

1. India won Test, T20 match when its audience approximately 42,000 to 55,000.
2. For ODI it is between 42,000 to 47,000.

So from this we can assume that large audience really effects team performance and morale and may boost their chances of win. Let's confirm it with Offshore as well.



Here we can see that when India is playing at home with audience is around 47,000 to 55,000, which is quite acceptable as Cricket has huge fanbase in India, so India has won matches at that level of audience.Although we didn't find any correlation with these features but we can bet on these features to boost Indian team performance.

Business Inisghts from EDA:(How your analysis is impacting the business

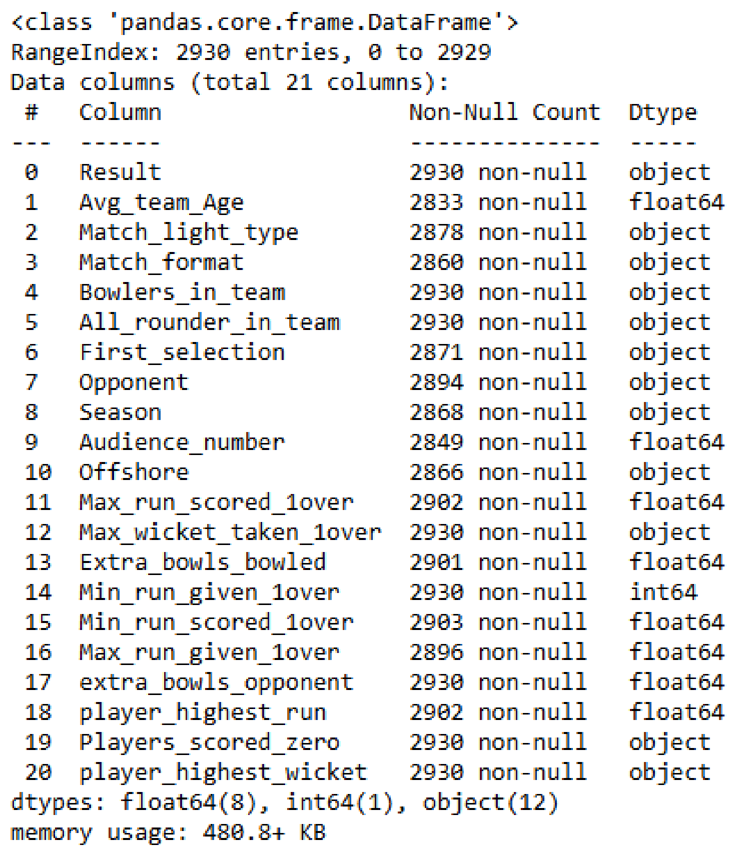
As far for our target column that is the result column, we cannot find any variable effecting with high correlation.

Although with univariate, bivariate and multivariate analysis we can take some insights which we can use to improve our odds of winning for different matches. Although feature impact and their insights will reflect differently when we will see how machine learning models finding these features important.

- India always performs best in when playing home from every opponent.
- India performs best when playing day matches.
- With audience of more than 40,000, India performs best when playing offshore as well as for different match formats.

2) DATA CLEANING AND PRE-PROCESSING

Visual Inspection of data:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Result                 2930 non-null   object 
 1   Avg_team_Age           2833 non-null   float64
 2   Match_light_type       2878 non-null   object 
 3   Match_format           2860 non-null   object 
 4   Bowlers_in_team        2930 non-null   object 
 5   All_rounder_in_team    2930 non-null   object 
 6   First_selection        2871 non-null   object 
 7   Opponent               2894 non-null   object 
 8   Season                 2868 non-null   object 
 9   Audience_number        2849 non-null   float64
 10  Offshore               2866 non-null   object 
 11  Max_run_scored_1over   2902 non-null   float64
 12  Max_wicket_taken_1over 2930 non-null   object 
 13  Extra_bowls_bowled     2901 non-null   float64
 14  Min_run_given_1over    2930 non-null   int64  
 15  Min_run_scored_1over   2903 non-null   float64
 16  Max_run_given_1over    2896 non-null   float64
 17  extra_bowls_opponent   2930 non-null   float64
 18  player_highest_run     2902 non-null   float64
 19  Players_scored_zero    2930 non-null   object 
 20  player_highest_wicket  2930 non-null   object 
dtypes: float64(8), int64(1), object(12)
memory usage: 480.8+ KB
```

On observing the data, we can see the data contains 2930 rows and 21 columns i.e., Data attributes out of which 12 variables are of object type, 1 variable of integer type and 8 variable of float type.

Understanding of attributes:

Initially, data contained 617 rows with missing values. First, we started with checking the values in categorical columns and seeing how many unique entries are present and any of them is wrongly written or repeated in other form. For example- ‘Match format’ columns contain entries like,

1. T20
2. Test
3. ODI

But we found that it has also one entry ‘20-20’ which is same for T20 so first replaced ‘20-20’ with ‘T20’.

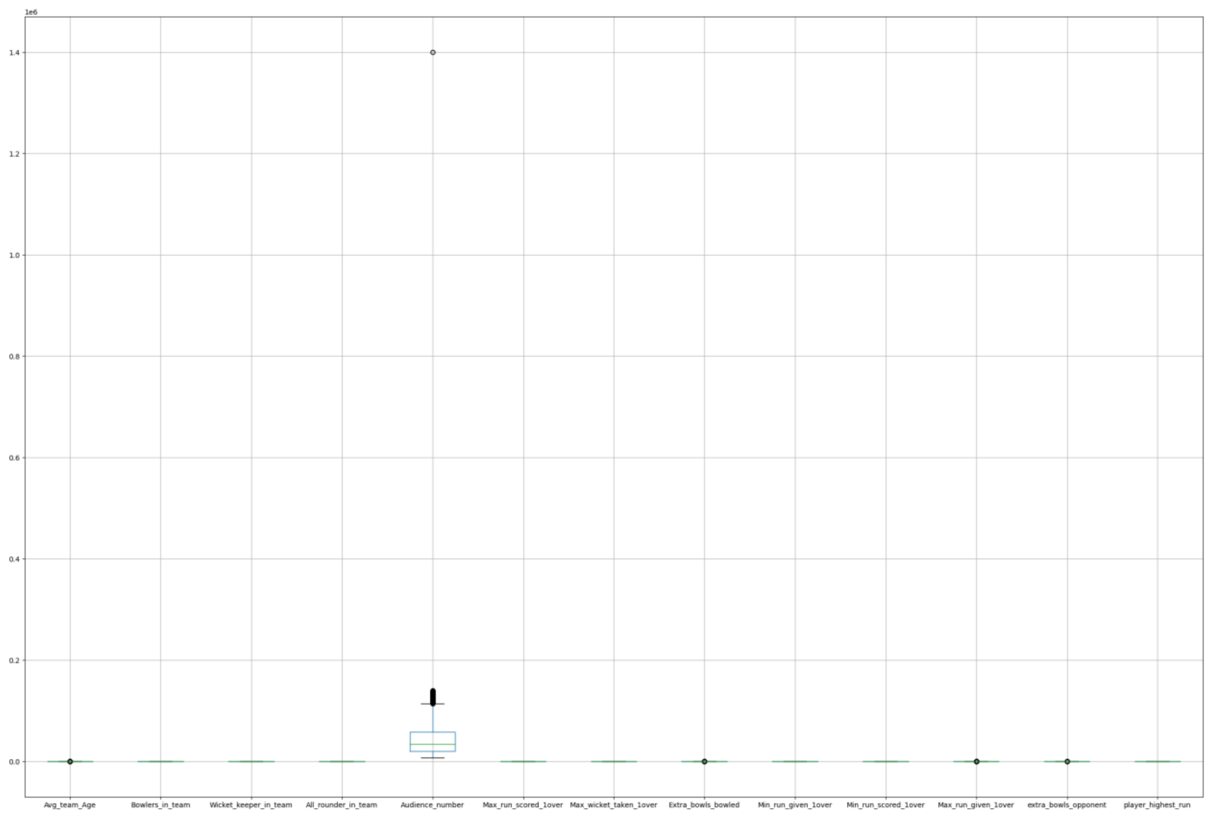
Similarly, we checked for categorical columns and replaced the ambiguous values with the values relevant with data.

Missing Values Treatment:

We checked for rows with missing values. If rows have missing values less than 3% then we tried to drop those rows. But our result showed that there were not enough rows with 1,2,3 missing values to drop less than 3%.

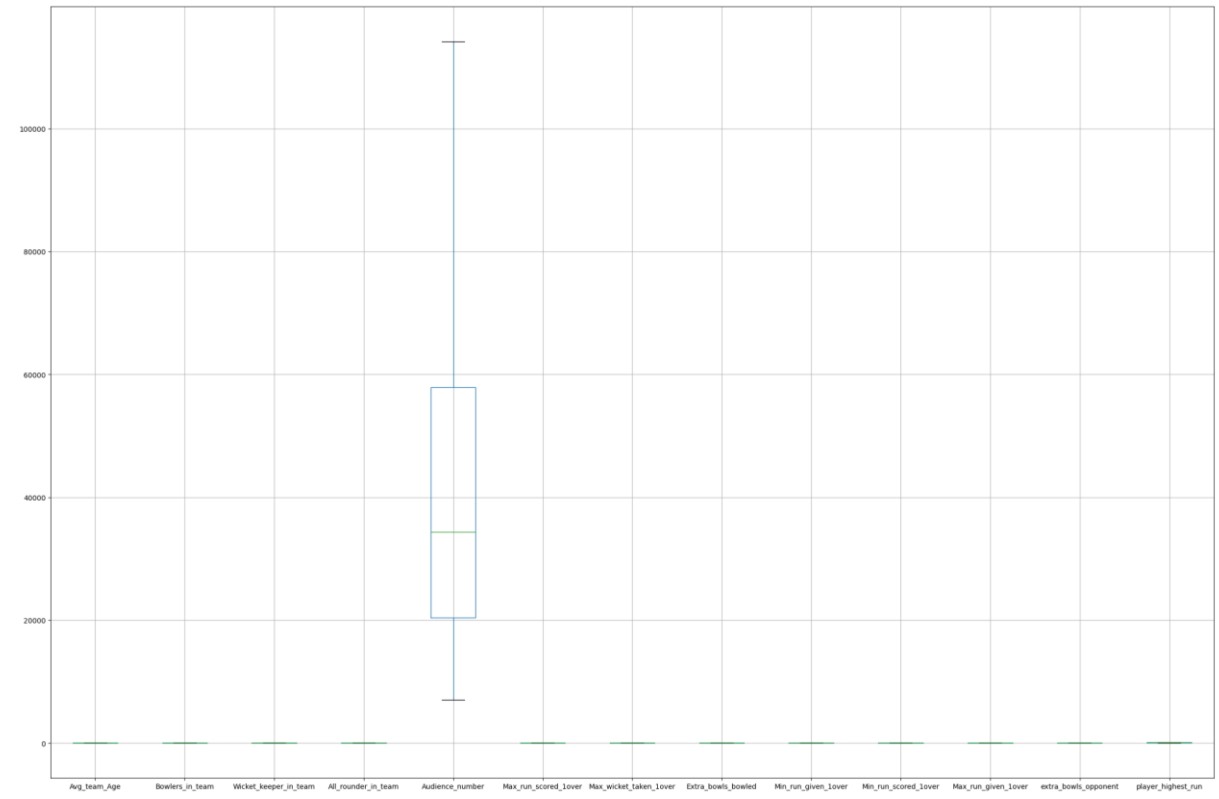
So, for imputing missing values we used different approach for categorical and continuous columns.

Outlier Treatment:



From the boxplot we can see that there are various outliers in continuous variables so we replaced these outliers using IQR method i.e., to replace extreme values with the whiskers value of boxplot.

After treatment below is the outlier free data



There are some features in the data which earlier present as integer data. But as those columns didn't contain continuous data we converted to categorical data.

1. Bowlers in team
2. All rounders in team
3. Max wicket taken in 1 over

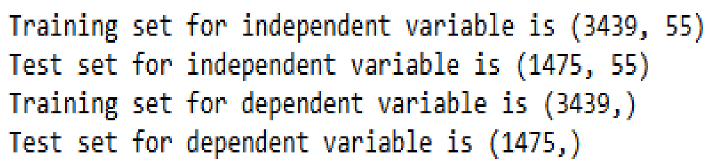
We also dropped 'Wicket Keeper in team' column as it contained only one value throughout data i.e., one. We also dropped 'Game Number' which was just sort of index column.

We checked the data for duplicate values. The data didn't contain duplicate entries.

Splitting the data into test and train dataset:

- Our target variable is Result. As we can see that there is a data imbalance in the variable.
- So here, we will be using the oversampling technique (i.e., SMOTE) and check whether it improves our model performance or not.
- We have stored all the predictor in x and target variable in y.

Let's look at the shape of the data after splitting our dataset.

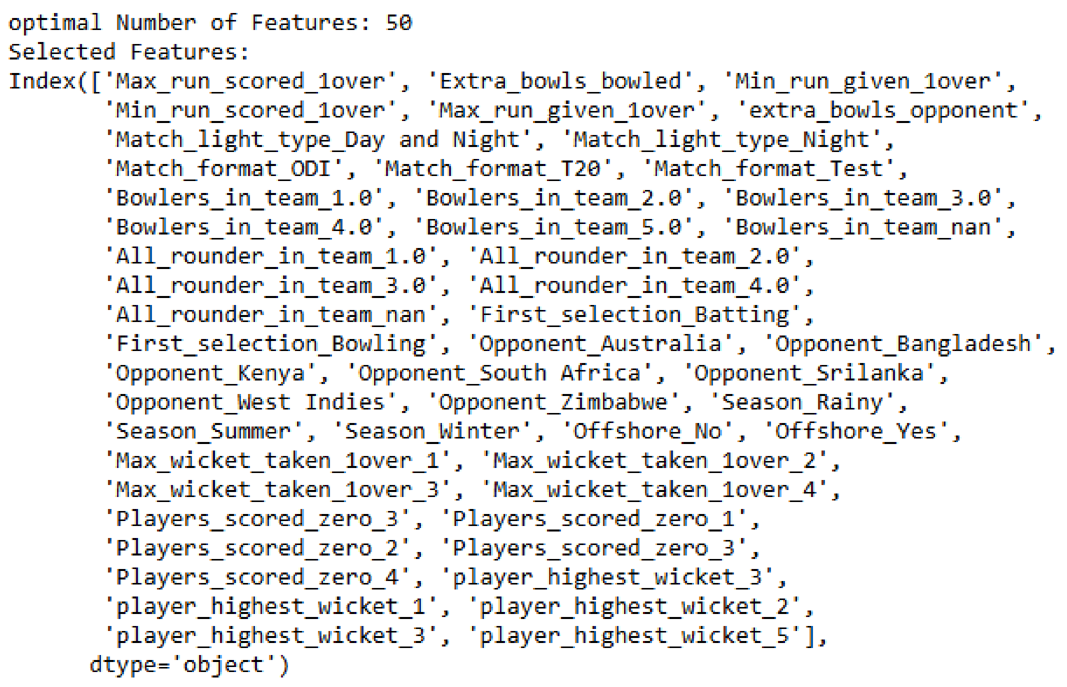


```
Training set for independent variable is (3439, 55)
Test set for independent variable is (1475, 55)
Training set for dependent variable is (3439,)
Test set for dependent variable is (1475,)
```

- X train - denotes 70% training dataset (except the target column called "Result")
- X test- denotes 30% test dataset (except the target column called "Result").
- y train- denotes the 70% training dataset with only the target column called "Result".
- y test- denotes 30% test dataset with only the target column called "Result".

Feature Selection:

- Chi-square test is used to determine the relationship between the predictor and target variable.
- In feature selection, we aim to select the features which are highly dependent on the target variable.
- Higher the Chi-square value indicate that the feature is more dependent on the target variable and can be select for model training.
- Chi-square score for Game number is null. So, we eliminate non-significant variable game number.
- After second iteration we find wicket keeper as non-significant variable as per chi-square test.



```
optimal Number of Features: 50
Selected Features:
Index(['Max_run_scored_1over', 'Extra_bowls_bowled', 'Min_run_given_1over',
       'Min_run_scored_1over', 'Max_run_given_1over', 'extra_bowls_opponent',
       'Match_light_type_Day and Night', 'Match_light_type_Night',
       'Match_format_ODI', 'Match_format_T20', 'Match_format_Test',
       'Bowlers_in_team_1.0', 'Bowlers_in_team_2.0', 'Bowlers_in_team_3.0',
       'Bowlers_in_team_4.0', 'Bowlers_in_team_5.0', 'Bowlers_in_team_nan',
       'All_rounder_in_team_1.0', 'All_rounder_in_team_2.0',
       'All_rounder_in_team_3.0', 'All_rounder_in_team_4.0',
       'All_rounder_in_team_nan', 'First_selection_Batting',
       'First_selection_Bowling', 'Opponent_Australia', 'Opponent_Bangladesh',
       'Opponent_Kenya', 'Opponent_South Africa', 'Opponent_Srilanka',
       'Opponent_West Indies', 'Opponent_Zimbabwe', 'Season_Rainy',
       'Season_Summer', 'Season_Winter', 'Offshore_No', 'Offshore_Yes',
       'Max_wicket_taken_1over_1', 'Max_wicket_taken_1over_2',
       'Max_wicket_taken_1over_3', 'Max_wicket_taken_1over_4',
       'Players_scored_zero_3', 'Players_scored_zero_1',
       'Players_scored_zero_2', 'Players_scored_zero_3',
       'Players_scored_zero_4', 'player_highest_wicket_3',
       'player_highest_wicket_1', 'player_highest_wicket_2',
       'player_highest_wicket_3', 'player_highest_wicket_5'],
      dtype='object')
```

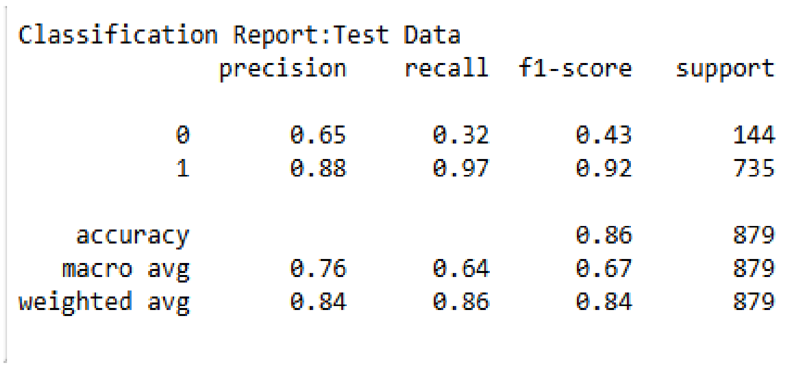
Model Building:

The model's used in the problem are as follows:

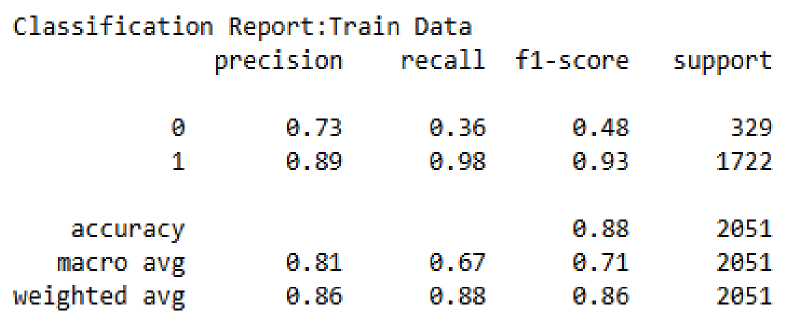
1. Logistic Regression
2. Linear Discriminant Analysis
3. Naïve Bayes
4. KNN
5. Decision Tree Classifier
6. Bagging
7. Ada Boosting
8. Gradient Boosting
9. MLPClassifier
10. Random Forest Classifier

Logistic Regression

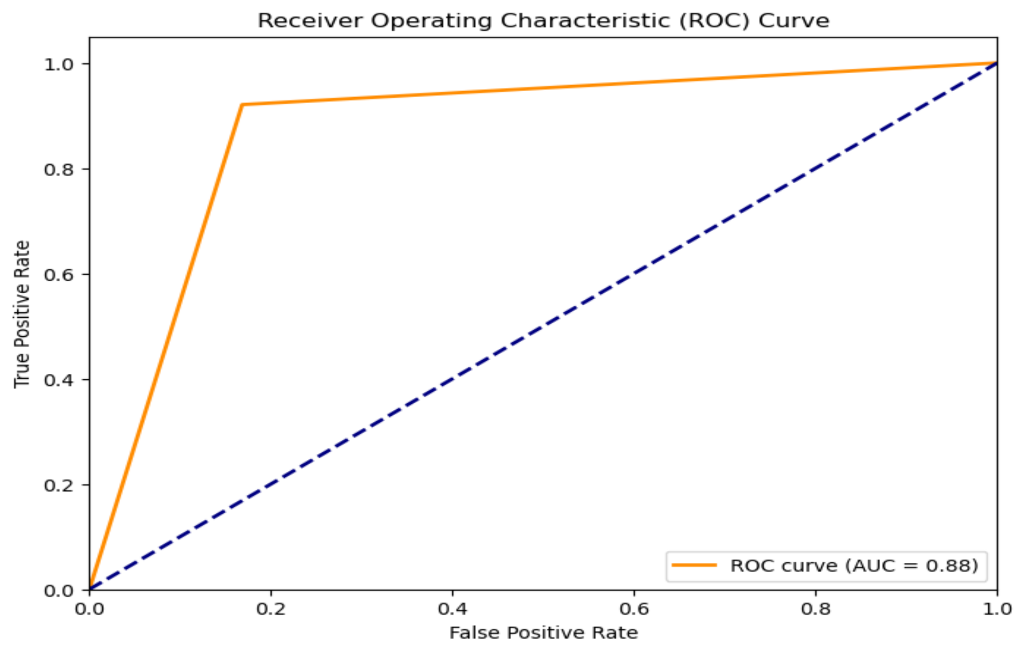
Performance matrix on test data


```
Classification Report:Test Data
              precision    recall  f1-score   support

           0       0.65      0.32      0.43       144
           1       0.88      0.97      0.92       735

    accuracy                           0.86       879
   macro avg       0.76      0.64      0.67       879
weighted avg       0.84      0.86      0.84       879
```

Performance matrix on train data


```
Classification Report:Train Data
              precision    recall  f1-score   support

           0       0.73      0.36      0.48       329
           1       0.89      0.98      0.93      1722

    accuracy                           0.88      2051
   macro avg       0.81      0.67      0.71      2051
weighted avg       0.86      0.88      0.86      2051
```

- We can observe very low recall and F1 score for the zero class.
- For the training set we got accuracy=88%, Precision=76%, recall=64%, F1-score=67%.
- For the test set we got accuracy=88%, Precision=81%, recall=67%, F1-score=71%.



ROC Curve for Logistic Regression

Linear Discriminant Analysis:

Perfromance matrix on test data

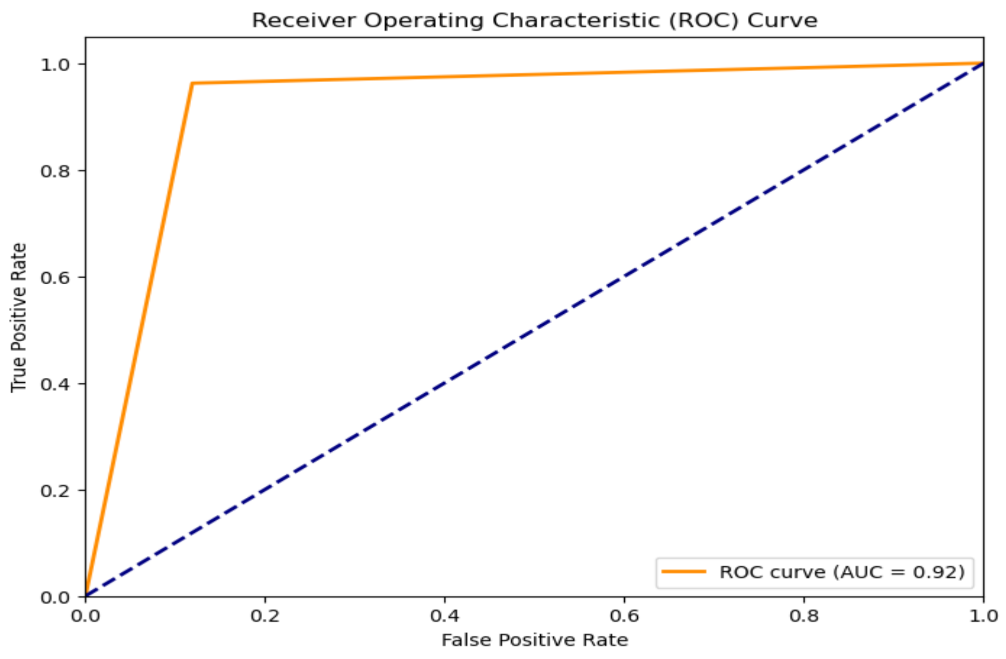
```
              precision    recall  f1-score   support

           0       0.95      0.81      0.87       731
           1       0.83      0.96      0.89       744

    accuracy                           0.88      1475
   macro avg       0.89      0.88      0.88      1475
weighted avg       0.89      0.88      0.88      1475


array([[589, 142],
       [ 33, 711]], dtype=int64)
```

Performance matrix on train data

```
              precision    recall  f1-score   support

           0       0.94      0.82      0.87      1726
           1       0.84      0.95      0.89      1713

    accuracy                           0.88      3439
   macro avg       0.89      0.88      0.88      3439
weighted avg       0.89      0.88      0.88      3439


array([[1408,  318],
       [  90, 1623]], dtype=int64)
```

- LDA is performing well as compared to Logistic model.
- For the training set we got accuracy=88%, Precision=89%, recall=88%, F1-score=88%.
- For the test set we got accuracy=88%, Precision=89%, recall=88%, F1-score=88%.

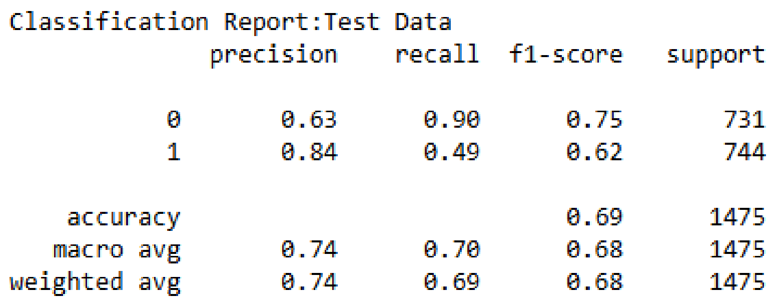
ROC Curve for Linear Discriminant Analysis



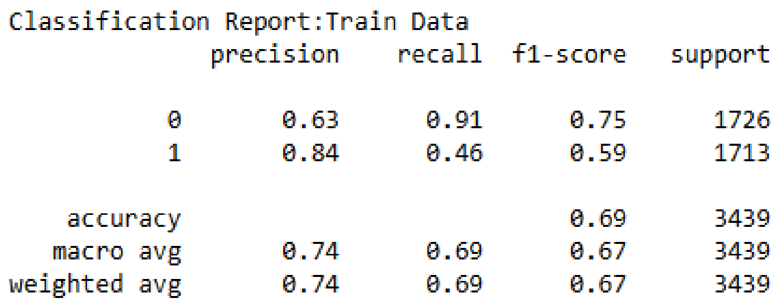
Naïve Bayes

- Navie Baye’s classifiers is a model based on applying Baye’s theorem with strong independent assumption between the features.
- Here the method that we are going to use is the GaussianNB() method, also know as BernoulliNB().A general assumption in this method is the data is following a normal distribution.

Performance matrix on Training and Test dataset

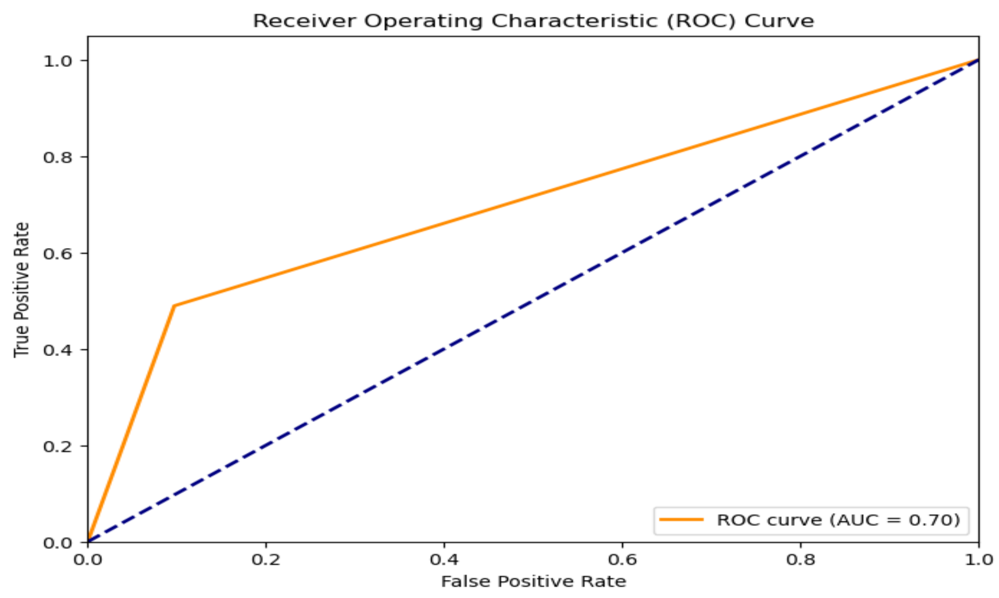


```
Classification Report:Test Data
              precision    recall  f1-score   support

           0       0.63      0.90      0.75       731
           1       0.84      0.49      0.62       744

    accuracy                           0.69      1475
   macro avg       0.74      0.70      0.68      1475
weighted avg       0.74      0.69      0.68      1475
```



```
Classification Report:Train Data
              precision    recall  f1-score   support

           0       0.63      0.91      0.75      1726
           1       0.84      0.46      0.59      1713

    accuracy                           0.69      3439
   macro avg       0.74      0.69      0.67      3439
weighted avg       0.74      0.69      0.67      3439
```

- Naïve Bayes model performs well.
- For the training set we got accuracy=69%, Precision=74%, recall=69%, F1-score=67%.
- For the test set we got accuracy=69%, Precision=74%, recall=70%, F1-score=68%.
- Suprisingly our recall and precision have increased in the test set.



ROC Curve for Naïve Bayes

KNN

- KNN is a distance based supervised machine learning algorithm that can be use to solve both classification and regression problems. It becomes very slow when we deal with large amount of data.
- For this classifier scaling data is necessary. KNN is a distance base algorithm, so we have scaled our data.
- Here, I have use z-score for scaling our data.

Performance matrix on scaled training and test dataset:

```
Classification Report:Test Data
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       731
           1       1.00      0.90      0.95       744

    accuracy                           0.95      1475
   macro avg       0.95      0.95      0.95      1475
weighted avg       0.96      0.95      0.95      1475
```

```
Classification Report:Train Data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1726
           1       1.00      1.00      1.00      1713

    accuracy                           1.00      3439
   macro avg       1.00      1.00      1.00      3439
weighted avg       1.00      1.00      1.00      3439
```

- KNN model performs well.

Decision Tree Classifier:

- Cart is a classification and regression predictive model machine learning technique.

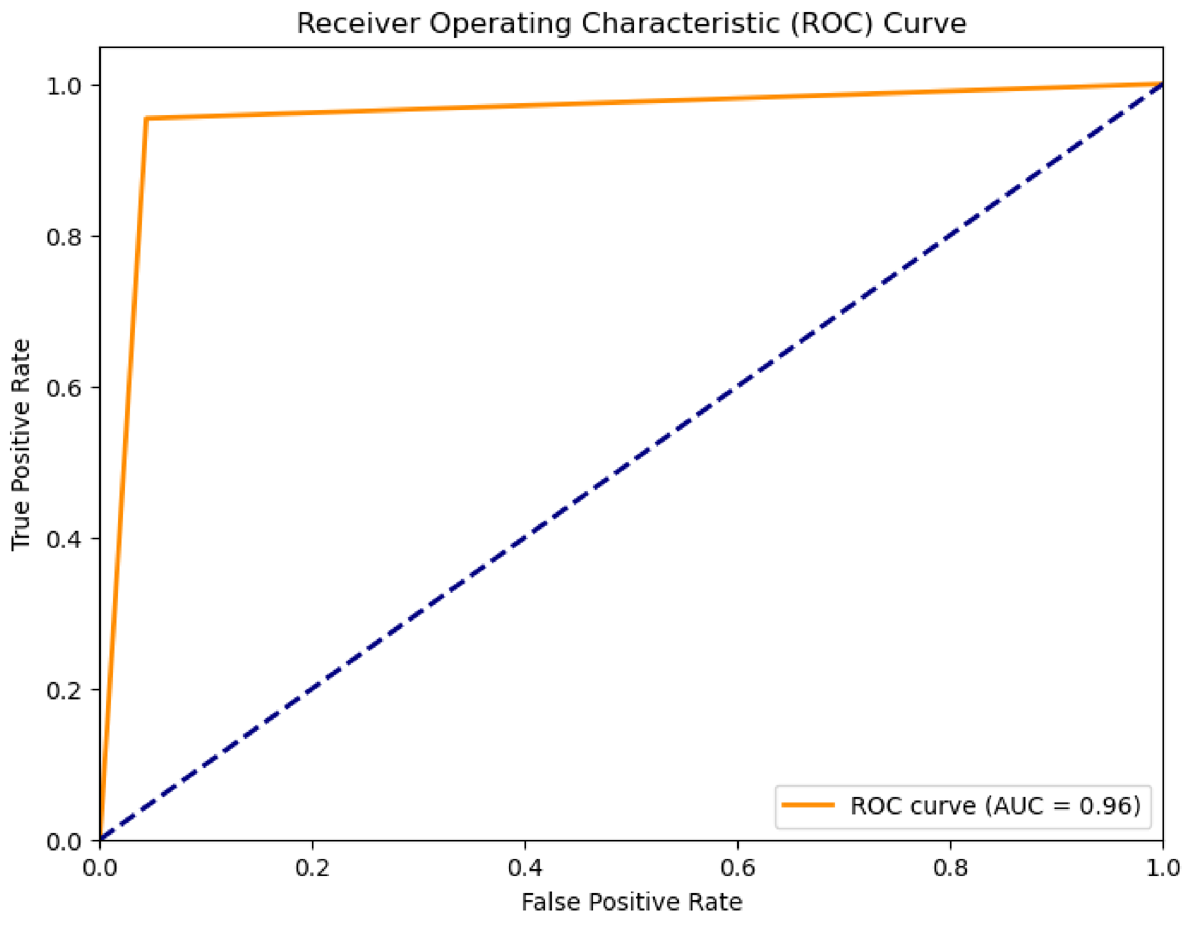
Performance matrix on training and testing data

```
Classification Report: Test Data
              precision    recall  f1-score   support

           0       0.95      0.96      0.95       731
           1       0.96      0.95      0.96       744

    accuracy                           0.96      1475
   macro avg       0.96      0.96      0.96      1475
weighted avg       0.96      0.96      0.96      1475
```

```
Classification Report:Train Data
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1726
           1       0.99      0.98      0.99      1713

    accuracy                           0.99      3439
   macro avg       0.99      0.99      0.99      3439
weighted avg       0.99      0.99      0.99      3439
```

- We can clearly see that our train set is overfitted. We can solve this issue by using pruning technique or by selecting important variable.

- For the training set we got accuracy=99%, Precision=99%, recall=99%, F1-score=99%.
- For the test set we got accuracy=96%, Precision=96%, recall=96%, F1-score=96%.

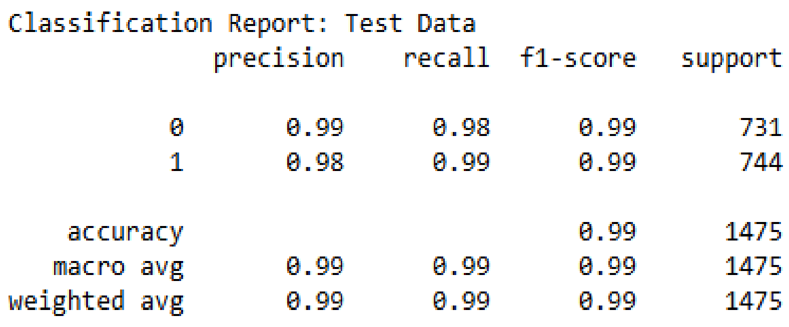


ROC Curve for Decision Tree Classifier

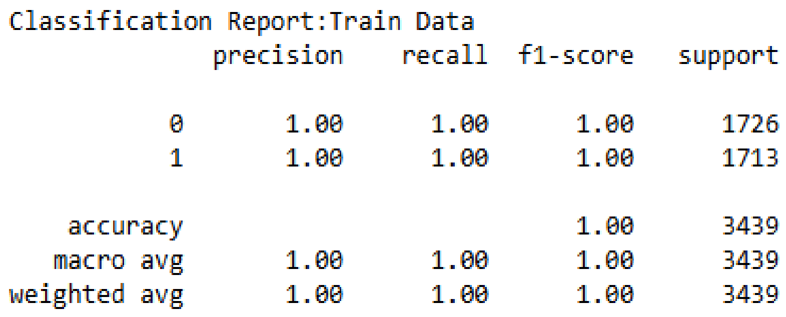
Bagging (Ensemble Method)

- Bagging is an ensemble technique. Ensemble techniques are the machine learning techniques that combine several base models to get an optimal model.
- Bagging is desined to improve the performance of existing machine learning algorithms used in statistical classification or regression.
- Here, we will use random forest as the base classifier. We use Hyper-parameters that we have obtain from Grid Search.

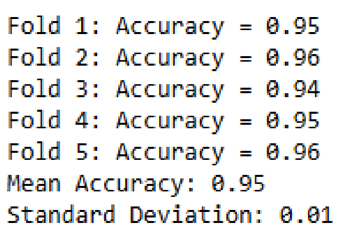
Performance matrix on training and testing data



```
Classification Report: Test Data
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       731
           1       0.98      0.99      0.99       744

    accuracy                           0.99      1475
   macro avg       0.99      0.99      0.99      1475
weighted avg       0.99      0.99      0.99      1475
```



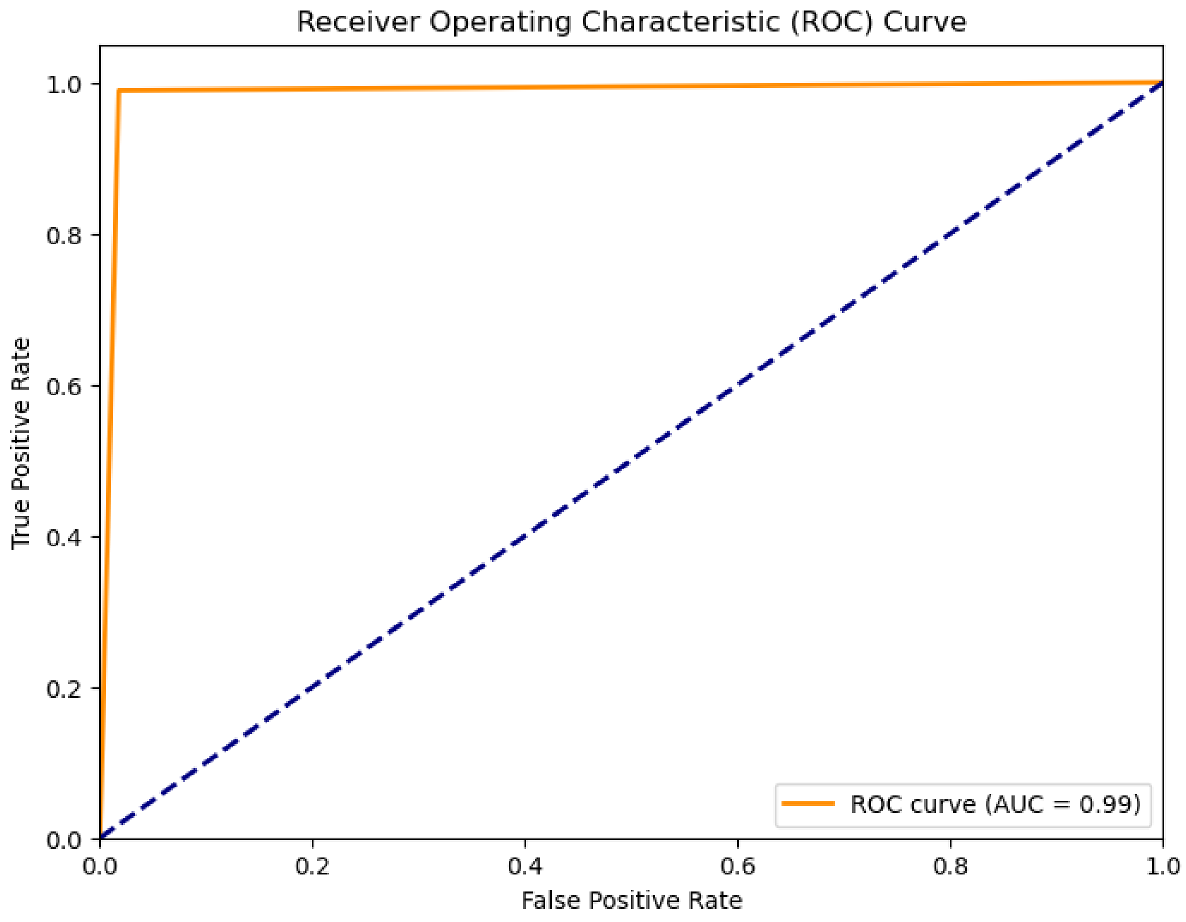
```
Classification Report:Train Data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1726
           1       1.00      1.00      1.00      1713

    accuracy                           1.00      3439
   macro avg       1.00      1.00      1.00      3439
weighted avg       1.00      1.00      1.00      3439
```



```
Fold 1: Accuracy = 0.95
Fold 2: Accuracy = 0.96
Fold 3: Accuracy = 0.94
Fold 4: Accuracy = 0.95
Fold 5: Accuracy = 0.96
Mean Accuracy: 0.95
Standard Deviation: 0.01
```

- Bagging model performs good but not as good as compared to simple random forest model.
- For the training set we got accuracy=100%, Precision=100%, recall=100%, F1-score=100%.

- For the test set we got accuracy=99%, Precision=99%, recall=99%, F1-score=99%.

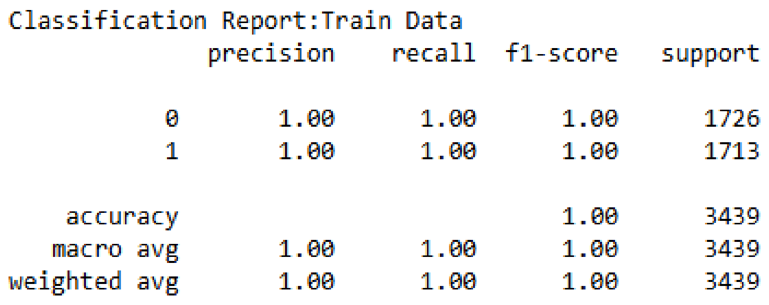


ROC Curve for Bagging

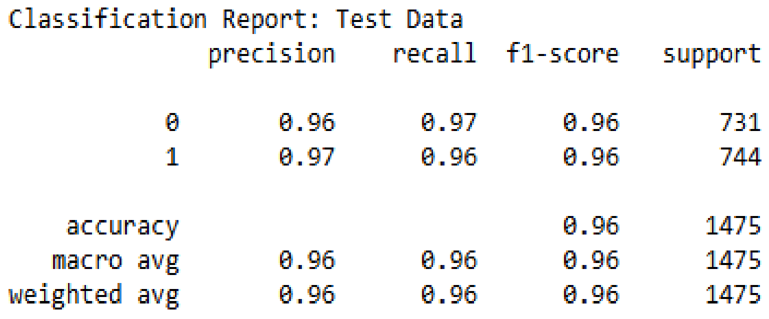
Ada Boosting

- This model is used to increase the efficiency of binary classifiers, but now used to improve multiclass classifiers as well.

Performance matrix on training and testing data

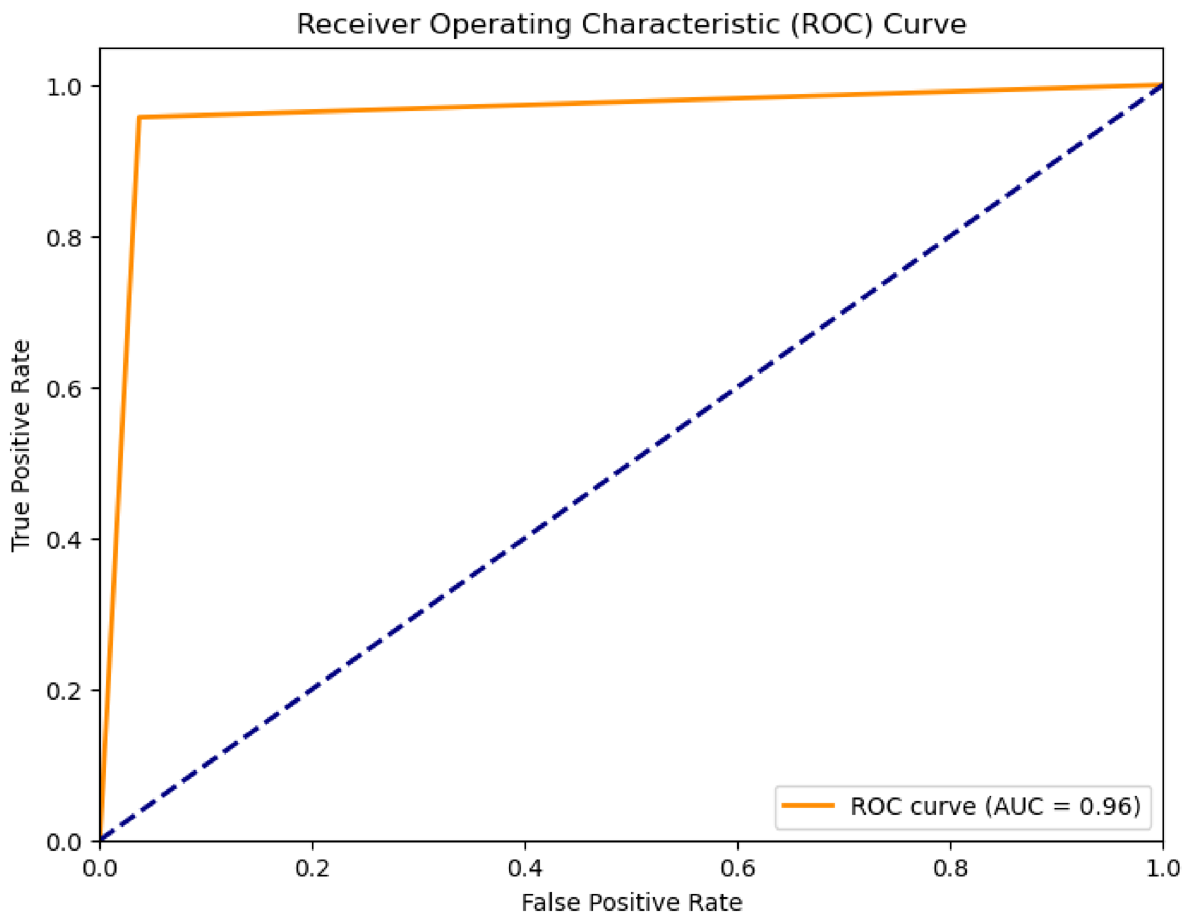


```
Classification Report:Train Data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1726
           1       1.00      1.00      1.00      1713

    accuracy                           1.00      3439
   macro avg       1.00      1.00      1.00      3439
weighted avg       1.00      1.00      1.00      3439
```



```
Classification Report: Test Data
              precision    recall  f1-score   support

           0       0.96      0.97      0.96       731
           1       0.97      0.96      0.96       744

    accuracy                           0.96      1475
   macro avg       0.96      0.96      0.96      1475
weighted avg       0.96      0.96      0.96      1475
```

- Ada boosting is performing good as compared to other models.
- For the training set we got accuracy=100%, Precision=100%, recall=100%, F1-score=100%.
- For the test set we got accuracy=96%, Precision=96%, recall=96%, F1-score=96%.

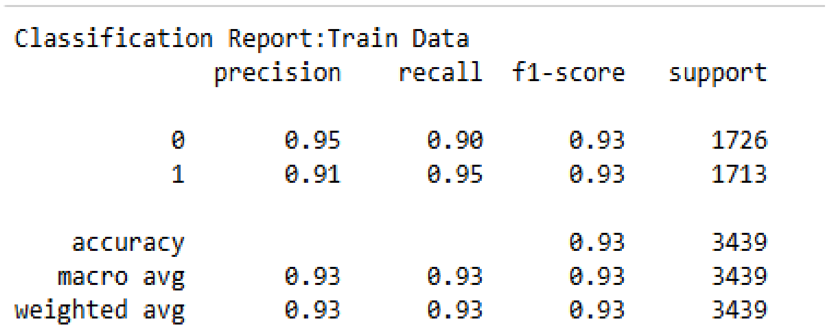


ROC Curve for Ada Boosting

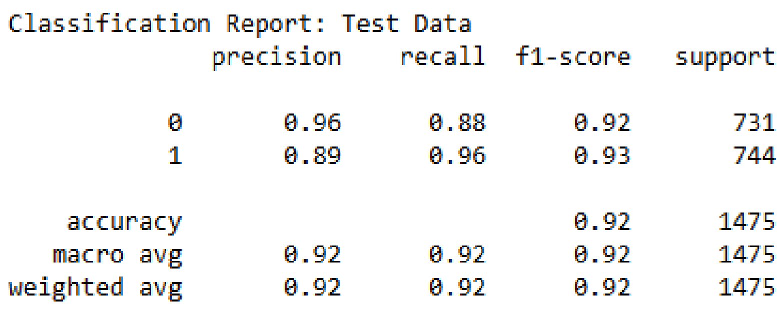
Gradient Boosting method

- This model is just like the Ada Boosting model. Gradient Boosting works by sequentially adding the misidentified predictors and under-fitted predictions to the ensemble, ensuring the errors identified previously are corrected.
- This method tries to fit the new predictor to the residual errors made by the previous one.

Performance matrix on training and testing dataset

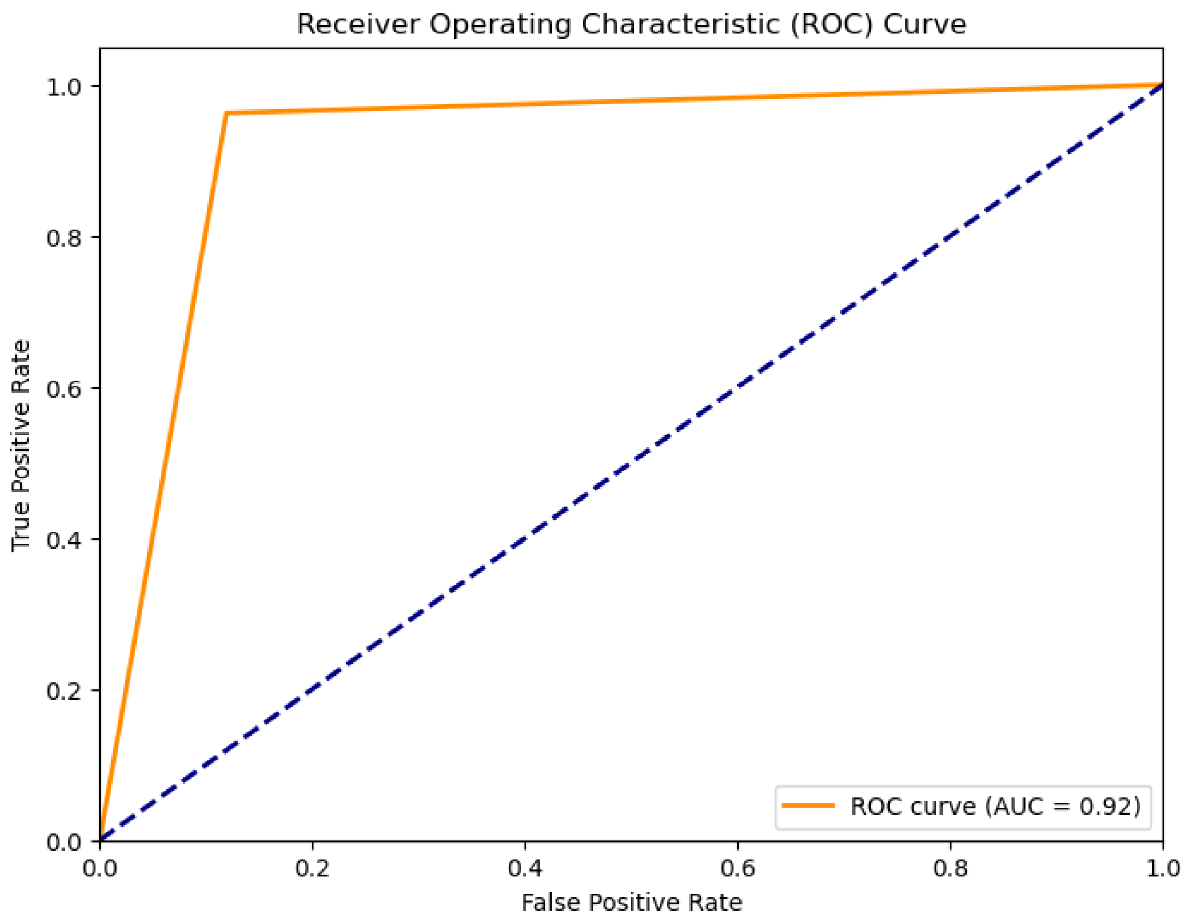


```
Classification Report:Train Data
              precision    recall  f1-score   support

           0       0.95      0.90      0.93      1726
           1       0.91      0.95      0.93      1713

    accuracy                           0.93      3439
   macro avg       0.93      0.93      0.93      3439
weighted avg       0.93      0.93      0.93      3439
```



```
Classification Report: Test Data
              precision    recall  f1-score   support

           0       0.96      0.88      0.92       731
           1       0.89      0.96      0.93       744

    accuracy                           0.92      1475
   macro avg       0.92      0.92      0.92      1475
weighted avg       0.92      0.92      0.92      1475
```

- Gradient Boosting model is performing good for this classification problem.
- For the training set we got accuracy=93%, Precision=93%, recall=93%, F1-score=93%.
- For the test set we got accuracy=92%, Precision=92%, recall=92%, F1-score=92%.



ROC Curve for Gradient Boosting Method

Random Forest model

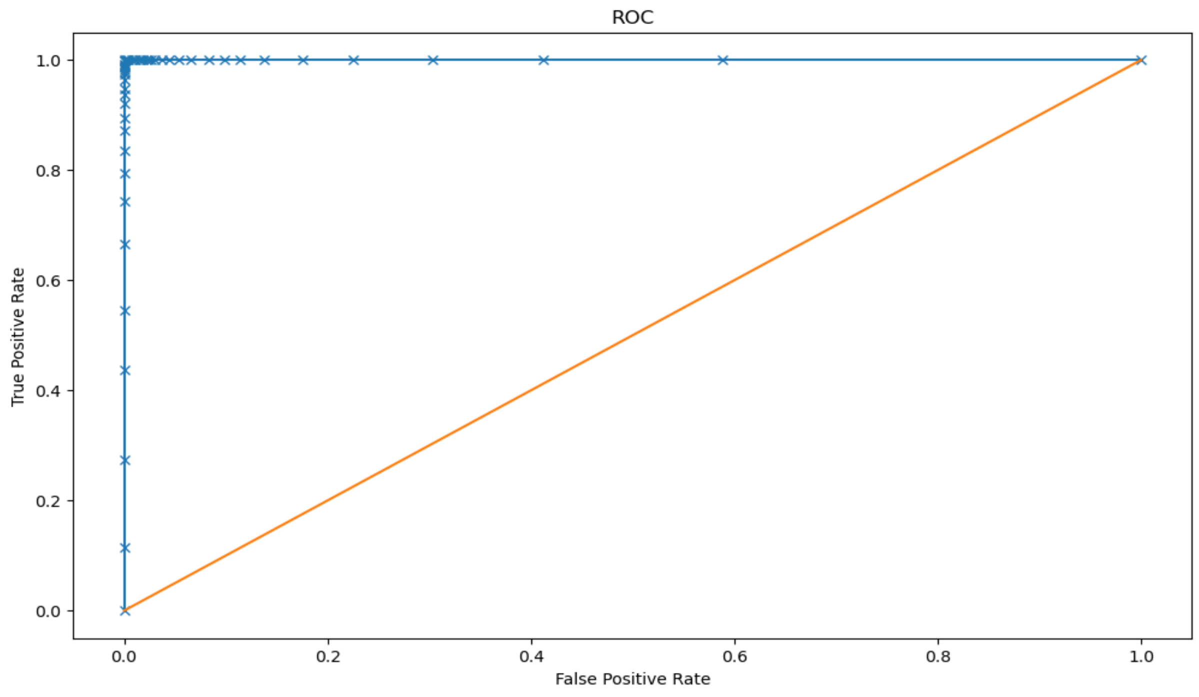
- Random Forest an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time.

Performance matrix on training and testing data

```
              precision    recall  f1-score   support

           0       0.95      0.90      0.93      1726
           1       0.91      0.95      0.93      1713

    accuracy                           0.93      3439
   macro avg       0.93      0.93      0.93      3439
weighted avg       0.93      0.93      0.93      3439
```

```
              precision    recall  f1-score   support

           0       0.96      0.88      0.92       731
           1       0.89      0.96      0.93       744

    accuracy                           0.92      1475
   macro avg       0.92      0.92      0.92      1475
weighted avg       0.92      0.92      0.92      1475
```

- We can clearly see that our train set is over fitted. We can solve this issue by using pruning technique or by selecting important variable that I will do in further model tunning part.
- For the training set we got accuracy=93%, Precision=93%, recall=93%, F1-score=93%.
- For the test set we got accuracy=92%, Precision=92%, recall=92%, F1-score=92%.



ROC Curve for Random Forest Model

Model chosen and why and effort to improve model performance:

- In this classification problem the most important measurement matrix we see is Recall, precision, accuracy, and F1-Score.
- In this case, precision is the total predicted win and loss. Recall is total Actually win and loss.
- F1- score is the harmonic mean of precision and recall.
- In this case our most important matrix is Recall because we must predict winning for the Indian team and must reduce the false positive rate.
- Comparing all models, I am going with '**Gradient Boosting Model'** for this Case study.

Gradient Boost Model have less False '+ve' and False '-ve' for both win and loss Classes. Compare to other model it has Higher Precision, Recall and Accuracy for both Train and Test.

Efforts to improve model performance:

- Try to collect more some more predictor, like total score, bowling style etc. for better Model.
- Try to add more than 3 all-rounders in the team that will improve the team performance

Gradient Boosting:

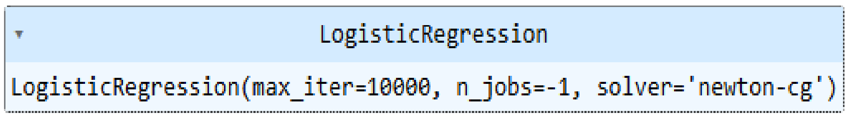
Gradient Boosting is a powerful boosting algorithm that combines several weak learners into strong learners, in which each new model is trained to minimize the loss function such as mean squared error or cross-entropy of the previous model using gradient descent. In each iteration, the algorithm computes the gradient of the loss function with respect to the predictions of the current ensemble and then trains a new weak model to minimize this gradient. The predictions of the new model are then added to the ensemble, and the process is repeated until a stopping criterion is met.

Model Tuning:

- Tuning is the process of maximizing a model's performance without overfitting or creating too high of a variance. In machine learning, this is accomplished by selecting appropriate "hyper-parameters"
- Grid Search is one of the most common methods of optimizing the parameters.

Model Validation-how was the model validated? Just accuracy, or anything else too?

Apply Logistic Regression with hyper parameters



```
▾                    LogisticRegression
LogisticRegression(max_iter=10000, n_jobs=-1, solver='newton-cg')
```

Performance matrix on training and testing data with tuning

```
              precision    recall  f1-score   support

           0       0.93      0.85      0.89      1726
           1       0.86      0.93      0.90      1713

    accuracy                           0.89      3439
   macro avg       0.89      0.89      0.89      3439
weighted avg       0.89      0.89      0.89      3439


array([[1463,  263],
       [ 112, 1601]], dtype=int64)
```

```
              precision    recall  f1-score   support

           0       0.94      0.84      0.88       731
           1       0.86      0.94      0.90       744

    accuracy                           0.89      1475
   macro avg       0.90      0.89      0.89      1475
weighted avg       0.90      0.89      0.89      1475


array([[613, 118],
       [ 42, 702]], dtype=int64)
```

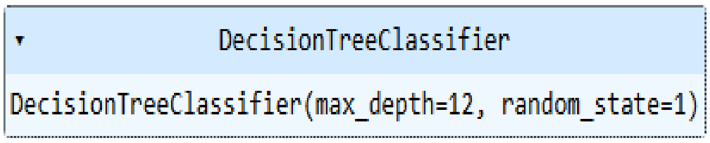
- This model performs well for both train and test set with hyper parameter as compared to without tuning.

- For the training set we got accuracy=89%, Precision=89%, recall=89%, F1-score=89%.
- For the test set we got accuracy=89%, Precision=90%, recall=89%, F1-score=89%.

Apply Decision Tree with hyperparameters

- Before fitting the model, it is important to know about the hyper parameters that is involved in CART model building.

- criterion = ‘gini, max_depth = ‘12’



```
▾                DecisionTreeClassifier
DecisionTreeClassifier(max_depth=12, random_state=1)
```

Performance matrix on training and testing data with tuning

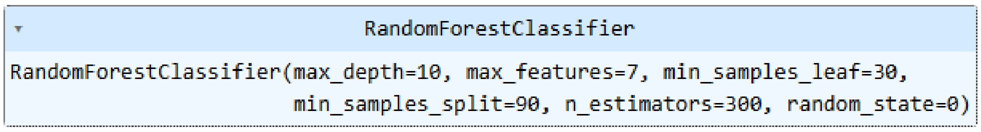
```
Classification Report:Train Data
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1726
           1       0.99      0.98      0.99      1713

    accuracy                           0.99      3439
   macro avg       0.99      0.99      0.99      3439
weighted avg       0.99      0.99      0.99      3439
```

```
Classification Report: Test Data
              precision    recall  f1-score   support

           0       0.95      0.96      0.95       731
           1       0.96      0.95      0.96       744

    accuracy                           0.96      1475
   macro avg       0.96      0.96      0.96      1475
weighted avg       0.96      0.96      0.96      1475
```

- For the training set we got accuracy=99%, Precision=99%, recall=99%, F1-score=99%.

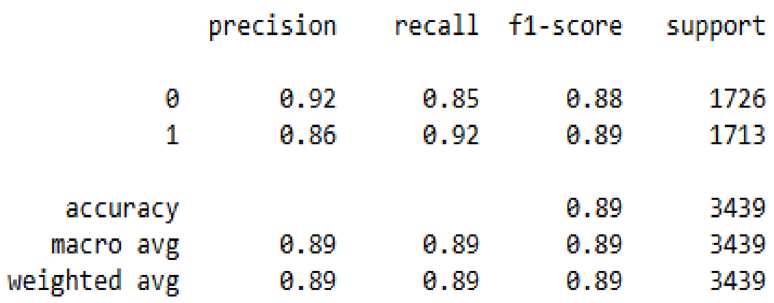
- For the test set we got accuracy=96%, Precision=96%, recall=96%, F1-score=96%.

Applying Random Forest with hyperparameters

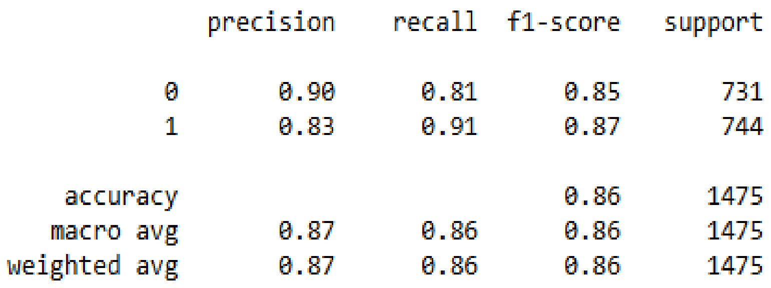


```
RandomForestClassifier
RandomForestClassifier(max_depth=10, max_features=7, min_samples_leaf=30,
                       min_samples_split=90, n_estimators=300, random_state=0)
```

Performance matrix on training and testing data

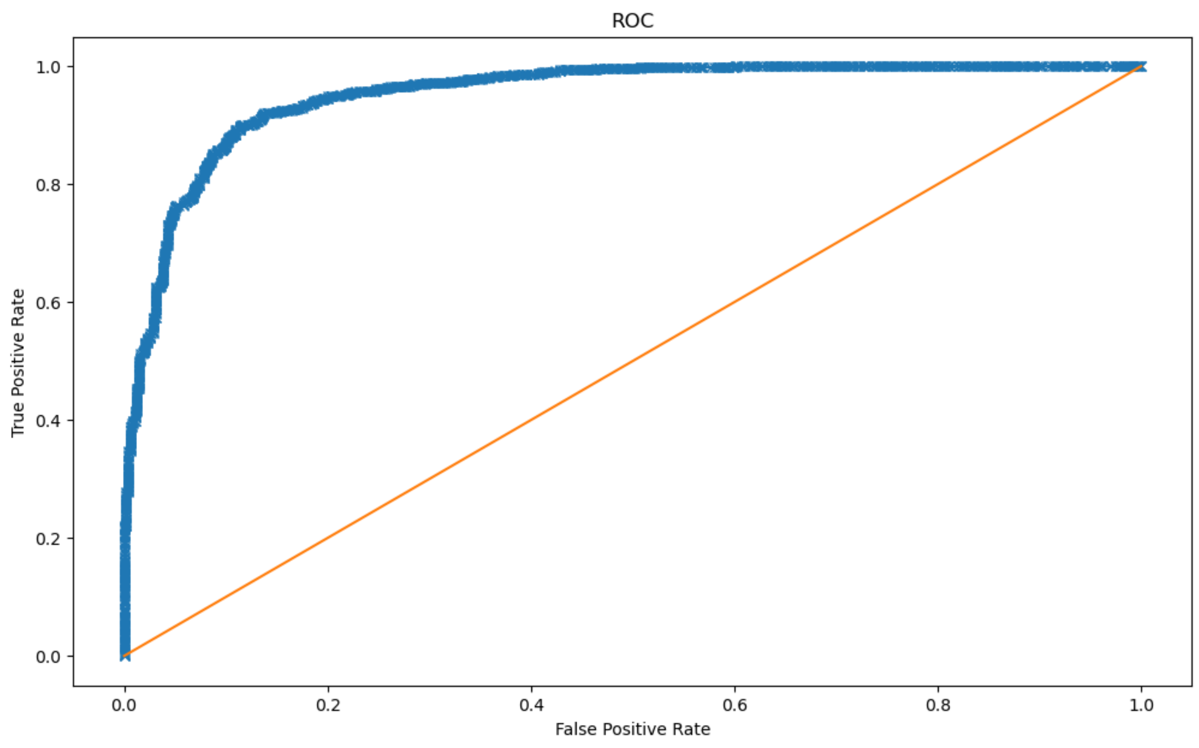


```
              precision    recall  f1-score   support

           0       0.92      0.85      0.88      1726
           1       0.86      0.92      0.89      1713

    accuracy                           0.89      3439
   macro avg       0.89      0.89      0.89      3439
weighted avg       0.89      0.89      0.89      3439
```



```
              precision    recall  f1-score   support

           0       0.90      0.81      0.85       731
           1       0.83      0.91      0.87       744

    accuracy                           0.86      1475
   macro avg       0.87      0.86      0.86      1475
weighted avg       0.87      0.86      0.86      1475
```

- After tunning the Random Forest model, we succeed to come up with overfitting issue.
- For the Training set we got Accuracy= 89%, Precision= 89%, recall= 89 %, f1-score= 89 %.
- For the Testing set we got Accuracy= 86%, Precision= 87%, recall= 86%, f1-score= 86%.


ROC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

Comparing all models

	LR-TRAIN	LR-TEST	LR(Tune)-TRAIN	LR(Tune)-TEST	LDA-Train	LDA-Test	LDA(Tune)-Train	LDA(Tune)-Test	KNN-Train	KNN-Test	NB-Train	NB-Test
Accuracy	**89**	**88**	**89**	**89**	**88**	**88**	**88**	**88**	**100**	**93**	**69**	**69**
F1 Score	**89**	**88**	**89**	**89**	**88**	**88**	**88**	**88**	**100**	**93**	**67**	**68**
Recall	**89**	**88**	**89**	**89**	**88**	**88**	**88**	**88**	**100**	**93**	**69**	**70**
Precision	**89**	**88**	**89**	**90**	**89**	**89**	**89**	**89**	**100**	**93**	**74**	**64**

	CART-TR	CART-TEST	RF-TRAIN	RF-TEST	Bagging-Train	Bagging-Test	Ada-train	Ada-Test	GB-Train	GB-Test
Accuracy	100	96	93	92	100	99	100	99	93	92
F1 Score	100	96	93	92	100	99	100	99	93	92
Recall	100	96	93	92	100	99	100	99	93	92
Precision	100	96	93	92	100	99	100	99	93	92

- In this classification problem the most important measurement matrix we see is Recall, precision, accuracy, and F1-Score.
- In this case, precision is the total predicted win and loss. Recall is total Actually win and loss.
- F1- score is the harmonic mean of precision and recall.

- In this case our most important matrix is Recall because we must predict winning for the Indian team and must reduce the false positive rate.
- Comparing all models, I am going with '**Gradient Boosting Model'** for this Case study.
- Gradient Boost Model have less False '+ve' and False '-ve' for both win and loss Classes. Compare to other model it has Higher Precision, Recall and Accuracy for both Train and Test.

Recommendation:

- Try to collect more some more predictor, like total score, bowling style etc. for better Model.
- Try to add more than 3 all-rounders in the team that will improve the team performance.
- If team opt for bowling first with an Avg team age of 30, with 4 bowlers in the team has higher chance to win against England in test match in Rainy season in England
- If team opt for bowling first with an Avg team age of 30, minimum 3 bowlers in the team, scoring average 15 runs per over has higher chance to win against Australia in T20 match in Winter season in India.
- If team opt for Batting first with an Avg team age of 30, with 3 bowlers in the team and at least one player should score century has higher chance to win against Sri Lanka in ODI match in Winter season in India.