HR ANALYTICS

PROJECT SUMMARY REPORT

Batch Details	
Team Members	1. JAGADEESAN M 2. KISHORE M 3. DHANISH ASWIN S 4. MATHESWARAN P 5. DHRUV DAVEY
Domain of Project	HR Analytics
Proposed Project Title	Prediction of employee promotion eligibility
Group Number	5
Team Leader	Jagadeesan M
Mentor Name	Mr. Jayveer Nanda

TABLE OF CONTENTS

Abstract:

With the emergence of HR Analytics in organizations; gathering, interpreting, and measuring of HR data has become easy. HR Analytics act as a tool which is a combination of statistical techniques that enable collection, interpretation, measurement, and forecasting of data. HR analytics enlightens solution to the organizational problems and make accurate decisions. HR analytics hence aligns HR strategy with overall business strategy to obtain a competitive advantage.

Introduction:

HR analytics is revolutionizing the way human resources department operate, leading to higher efficiency and better results overall. Human resources has been using analytics for years. However, the collection, processing and analysis of data has been largely manual, and given the nature of human resources dynamics and HR KPIs, i.e. Key Performance Indicators, the approach has been constraining HR. Therefore, it is surprising that HR departments woke up to the utility of machine learning so late in the game. In this opportunity, we're going to do predictive analytics in identifying the employees most likely to get promoted.Our proposed model will try to deal with such problem.

Need for HR Analytics:

HR analytics can provide the insights companies need to tackle difficult challenges such as lack of diversity, high turnover rate and so on. Data eliminates the guesswork out of HR, enabling HR to become subject matter experts trusted by the C–suite. HR analytics help to identify those traits that are predictors of success within an organization or a team, so we can focus our search for talent and avoid making mistakes that can be expensive and time-consuming to correct.

Data Dictionary:

The dataset which we are going to deal with is releated to HR Analytics.Using this dataset,we are able to create a predictive model that helps the client in identifying the right people to promote. This model will increase productivity and reduce the attrition of employees which builds strong integrity between employees and the company.

Dataset Domain : HR

No. of Observations in Train data : 54808

No. of Observations in Test data : 23490

No. of features in Train data : 14(including Target)

No. of features in Test data 13

No. of numerical features 6

No. of categorical features 8

Employee_id	Unique ID for employee
Department	Department of employee
Region	Region of employment (unordered)
Education	Education Level of employees
Gender	Gender of employees
Recruitment_channel	Channel of recruitment for employee
No_of_trainings	Number of other trainings completed in previous year on soft skills, technical skills etc.
Age	Age of Employee
Previous_year_rating	Employee Rating for the previous year
Length_of_service	Length of service in years
KPIs_met>80%	if Percent of KPIs(Key performance Indicators) >80% then 1 else 0
Awards_won?	if awards won during previous year then 1 else 0
Avg_training_score	Average score in current training evaluations
is_promoted (Target)	Recommended for promotion

Missing values:

	Null count	Null percent
Employee id	0	0.000000
department	0	0.000000
region	0	0.000000
education	2409	4.395344
gender	0	0.000000
Recruitment channel	0	0.000000
No of trainings	0	0.000000
age	0	0.000000
Previous year rating	4124	7.524449
Length of service	0	0.000000
KPIs met >80%	0	0.000000
Awards won?	0	0.000000
Avg training score	0	0.000000
Is promoted	0	0.000000

- We could observe about 4.3% of values in education and 7.5% of null values in previous_year_rating.
- We have used Rule based imputation to 'education' and mode imputation to 'previous_year_rating' .

NULL Imputation

```
ind=hr[hr['education'].isna()].index
```

```
for i in ind:
    if hr.loc[i,'length_of_service']<=3:
        hr.loc[i,'education']='Below Secondary'
    if hr.loc[i,'length_of_service']>3 and hr.loc[i,'length_of_service']<7:
        hr.loc[i,'education']="Bachelor's"
    else:
        hr.loc[i,'education']="Master's & above"
```

-
```
hr['previous_year_rating']=hr['previous_year_rating'].replace({np.nan:3.0})#mode imputation
```

Project Justification :

The client is a large MNC and they have 9 broad verticals across the organization, one of the challenges for the client is identifying the right people for promotion without bias for enhanced organizational efficiency.

Project Statement :

The key objective is to create a predictive model that helps the client in identifying the right people for promotion. This model will increase productivity and reduce the attrition of employees which builds strong integrity between employees and the company.

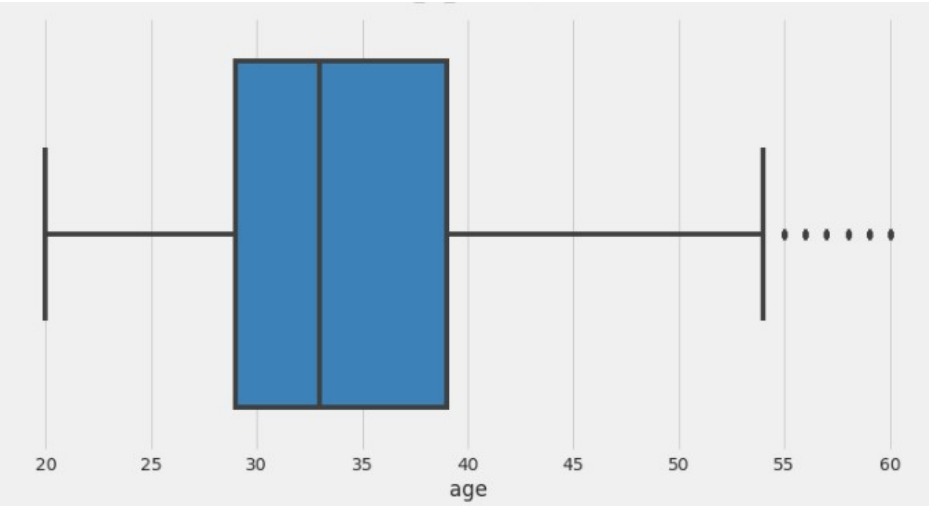
Complexity involved :

Since none of the predictor features are not much correlated with the target feature, every predictor variable is important to predict the promotion eligibility (target variable).

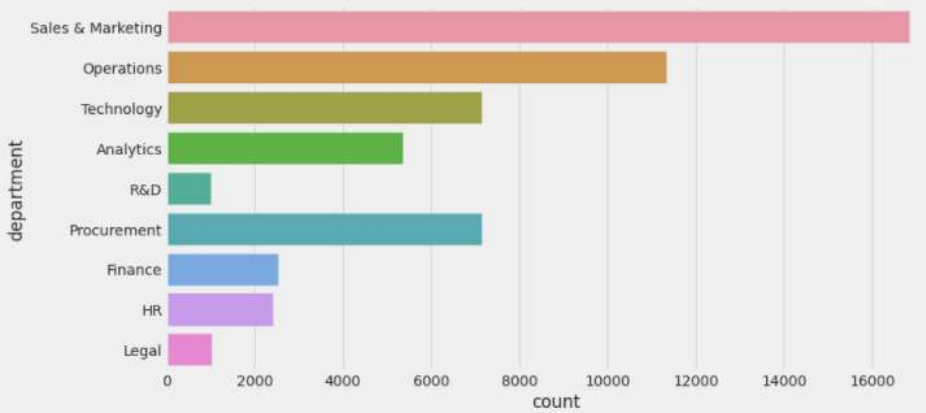
Project outcome :

Promotion is one of the key aspects of talent development that can affect both employees and the organization as well. An AI-based, data-driven approach can be used hand in hand with the rule-based, human-centric approach to get more intuitive results and better insights of employee promotion eligibility.

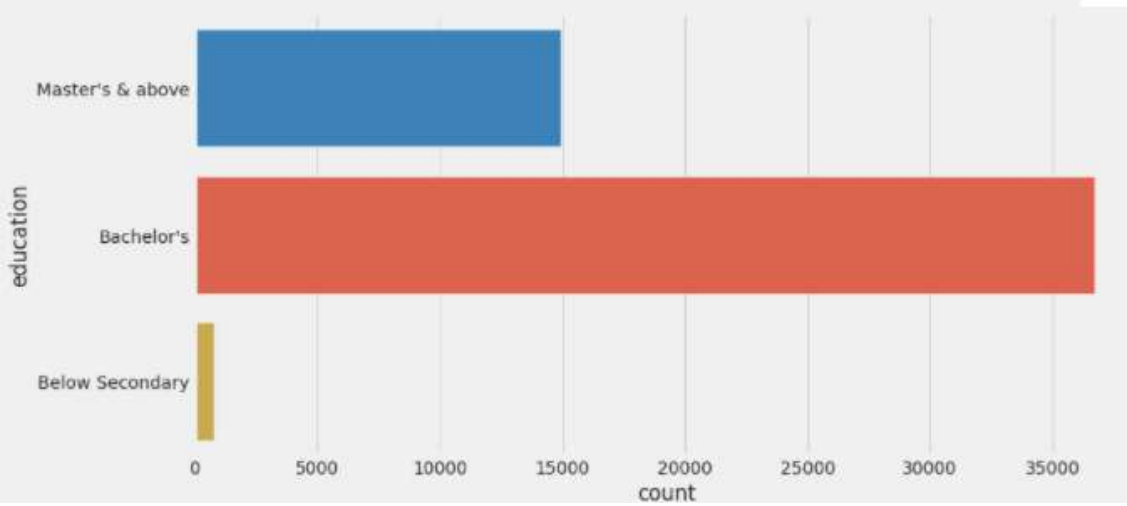
Uni-variate Analysis



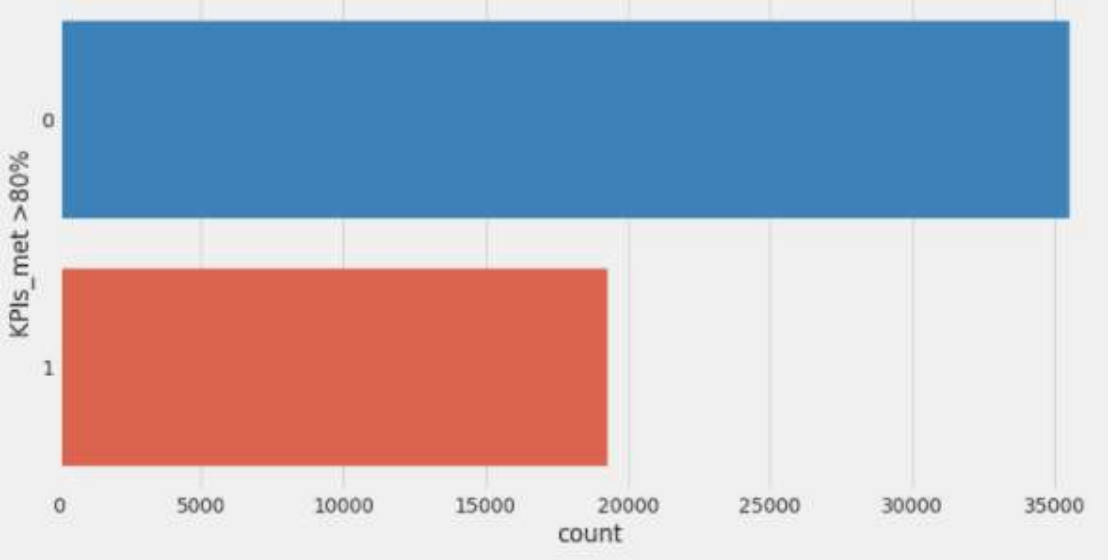
- 50% of employees age is between 28 and 40



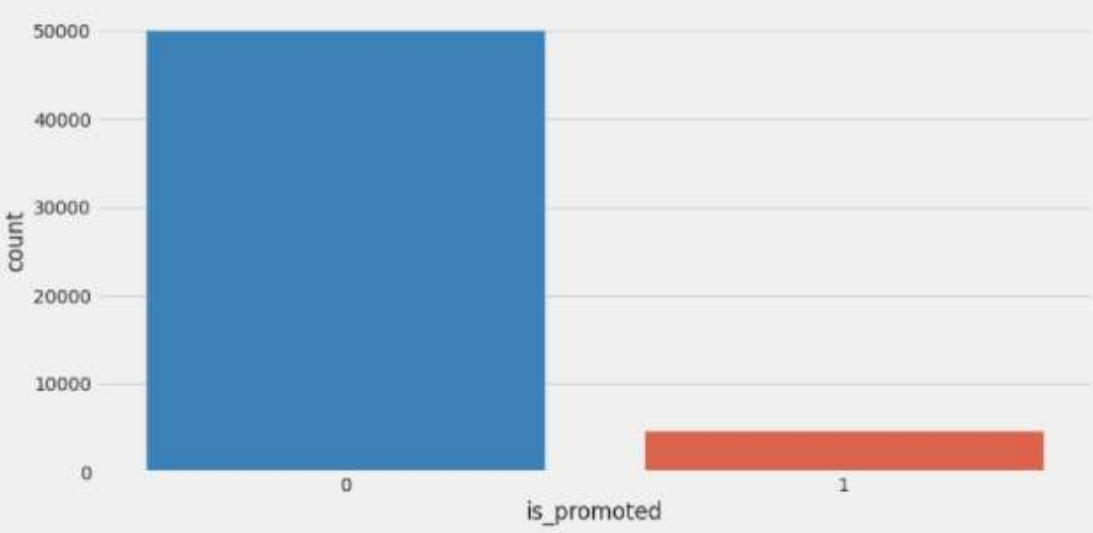
- Majority of employees are in sales& marketing department followed by operations, Technology etc



- Most of the employee's education level is bachelor's



- 50% of employees have achieved KPI>80% milestone.



- From the above plot of the target variable, we could sense that there was a huge data imbalance between the employees promoted and not promoted.

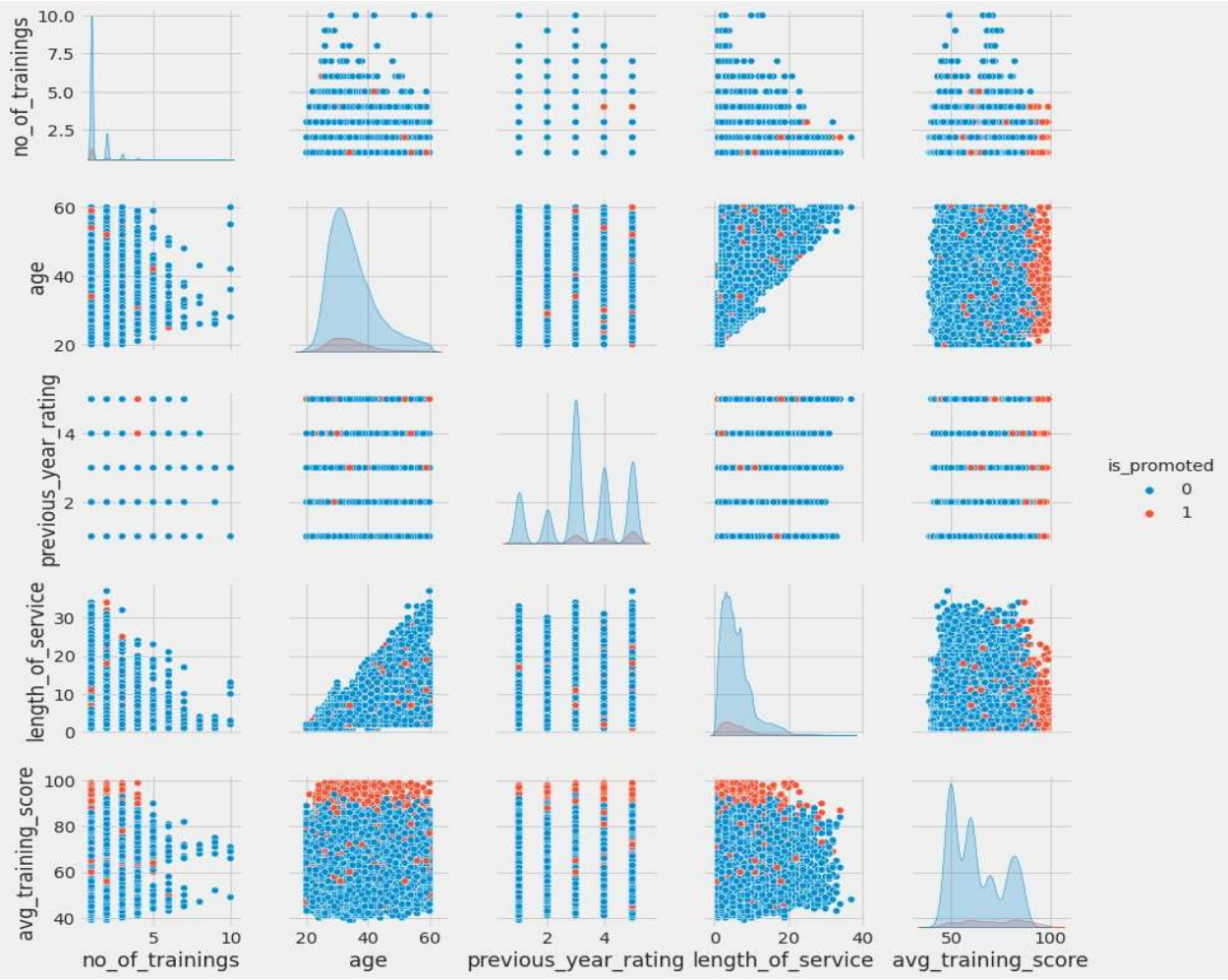
```
df['is_promoted'].value_counts(normalize=True)
```

```
0    0.91483
1    0.08517
```

- The quantitative representation interprets that only 8.5% of employees are promoted, the remaining employees are not promoted, employees.

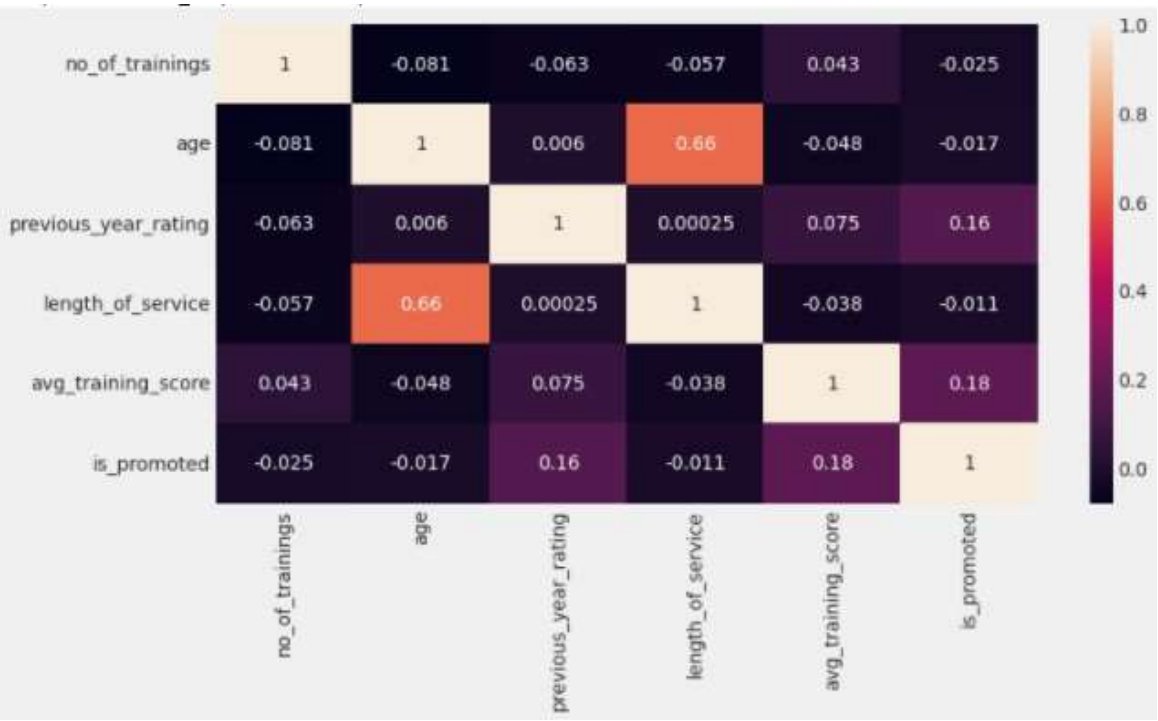
Numeric vs Numeric

Pair plot:



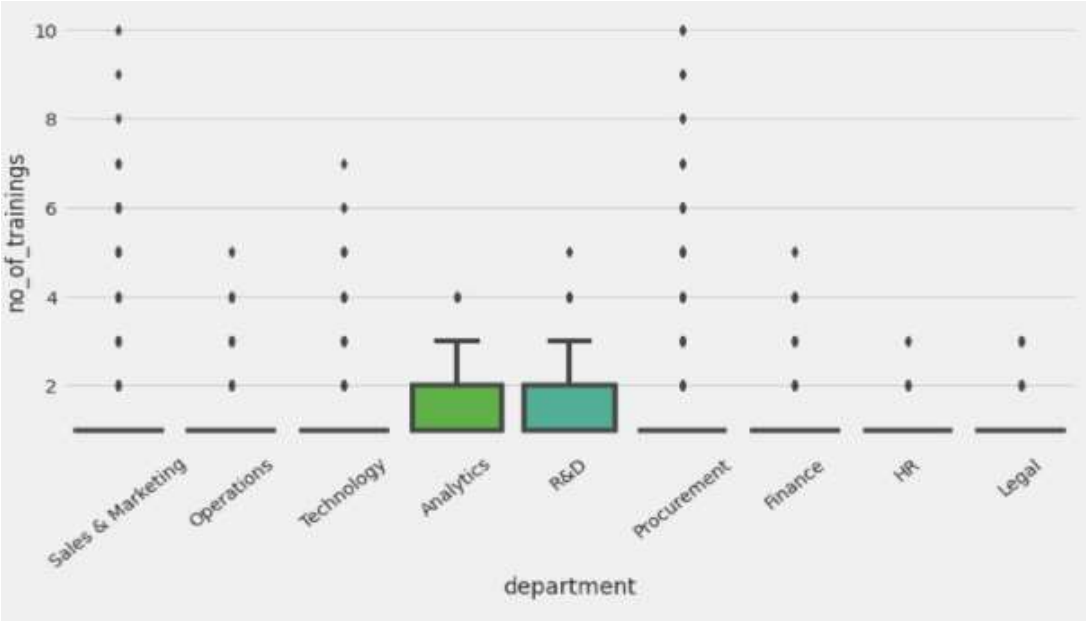
- Visually there is no significant difference has been observed between the promoted and not promoted employees in all diagonal plots. The distrubition looks overlapping, however, we can verify it with statistical testing.

- The predominant employees who scored above 80 in avg_training_score are promoted. This is a strong indication of avg_training_score feature importance.

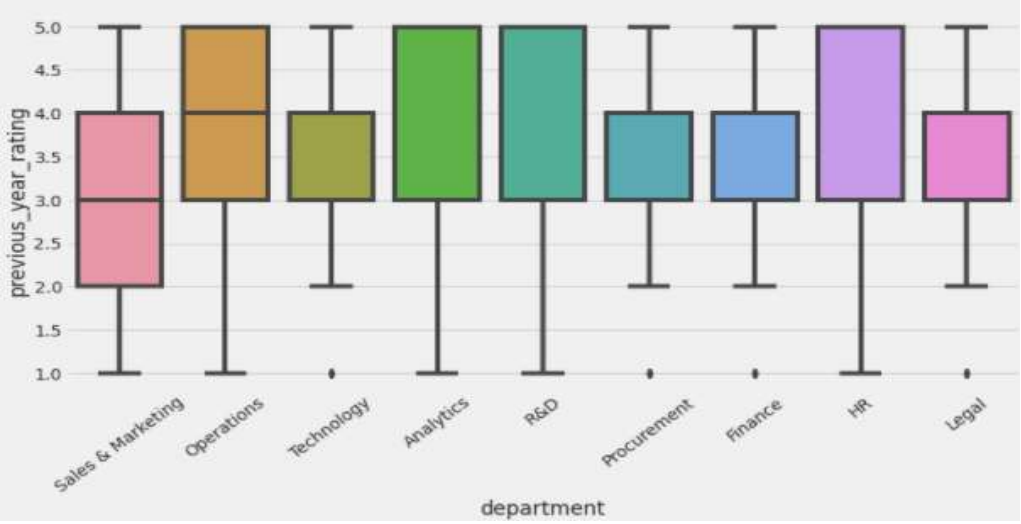


- The average_training_score and previous_year_rating express a comparatively high correlation against the target feature.
- There was slight multicollinearity has been observed between age and length_of_service features.

Department Vs No_of_trainings :

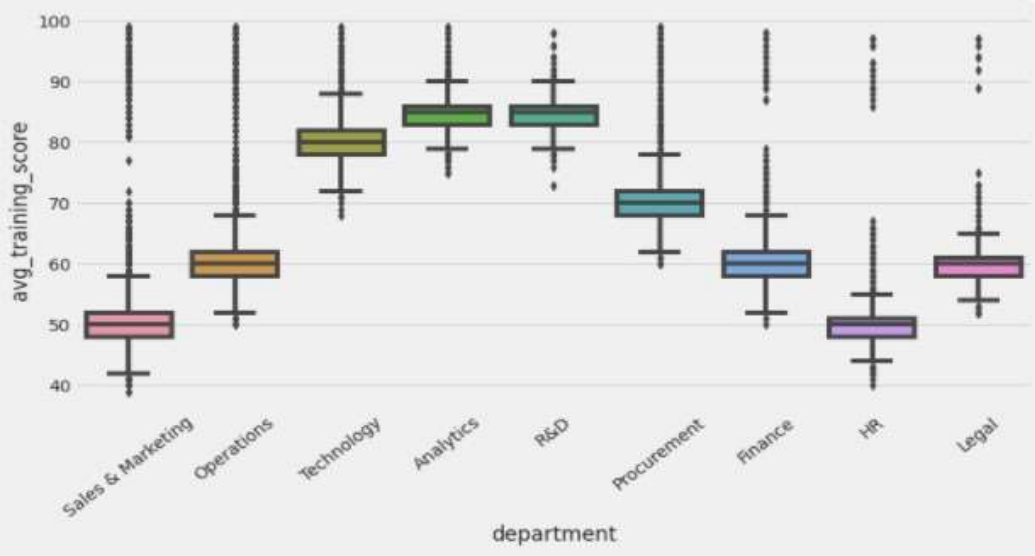


- Majority of employees in analytics and R&D are attending more training.

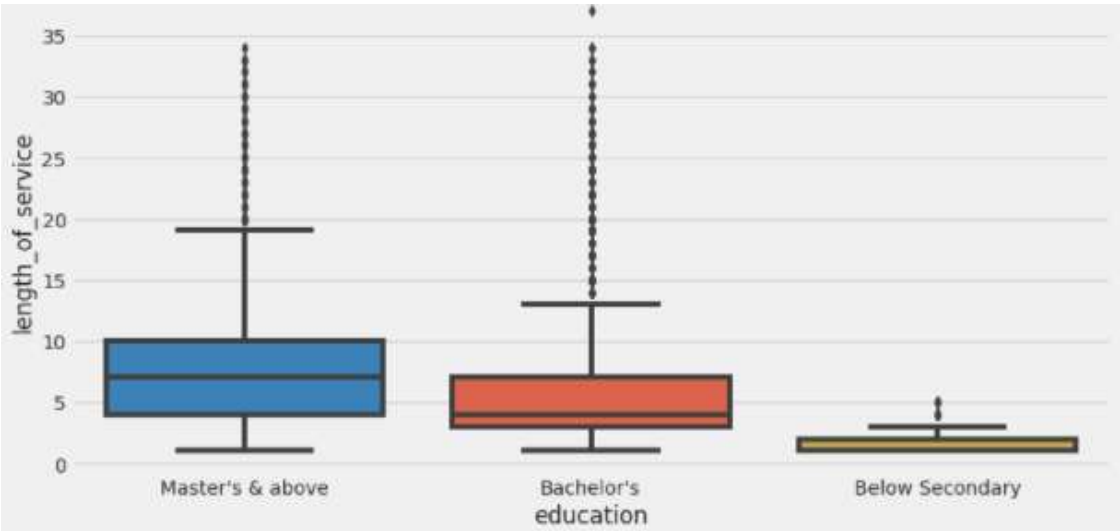


- The 50% of employees in sales and marketing scored a rating less than 3.0.

Department vs avg_training_score:

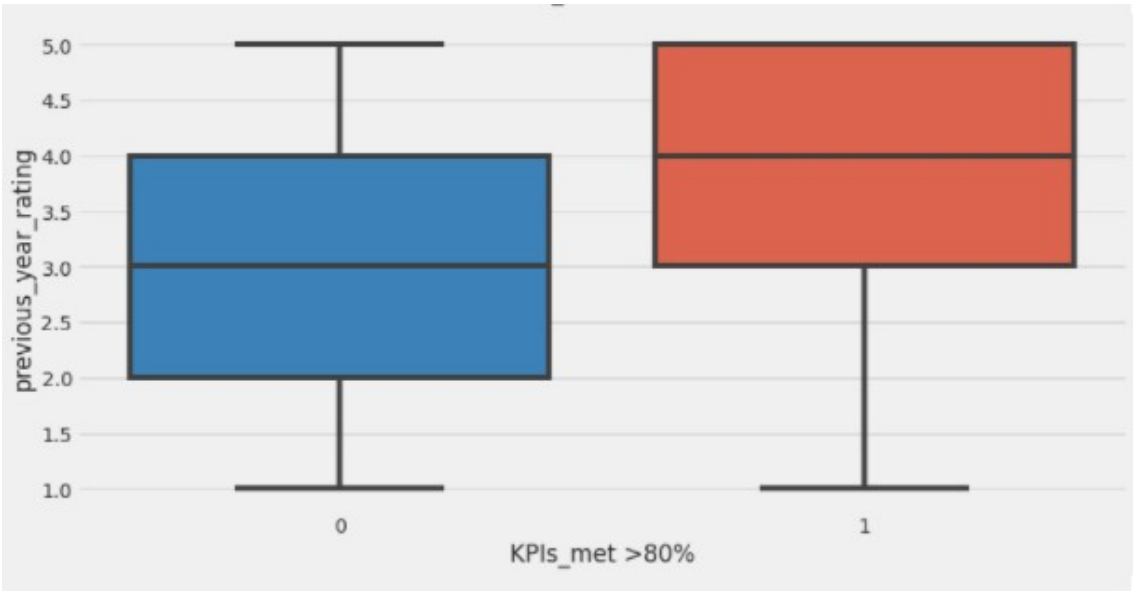


- The employees in Technology, Analytics and R&D have better average_training_score than other departments.

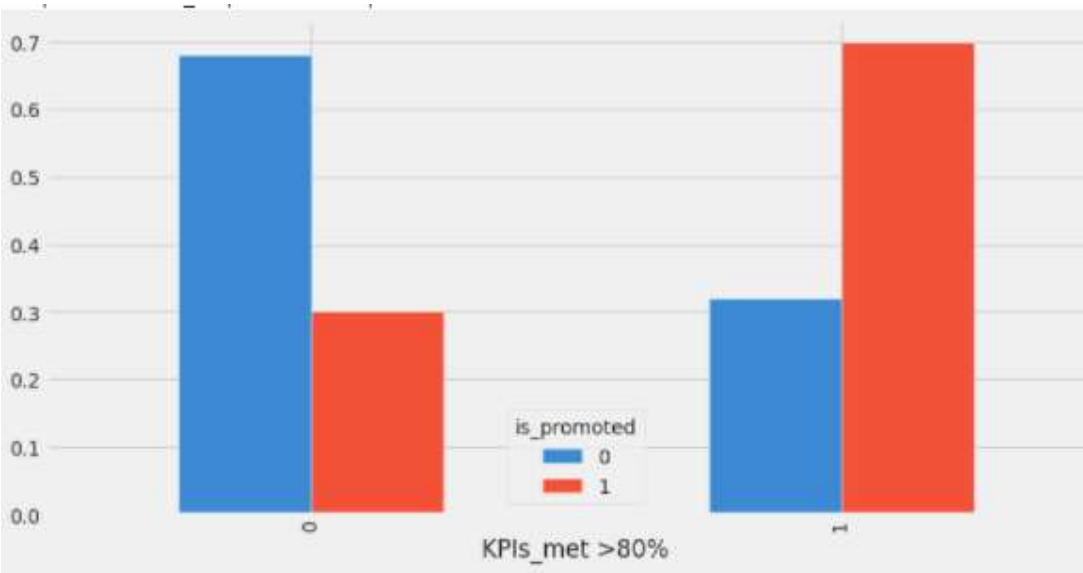


- The employee education and length of service have an association.
- The employees with masters and above are the long run employees than the employees with other educational qualifications.

KPIs_met>80% vs previous_year_rating:

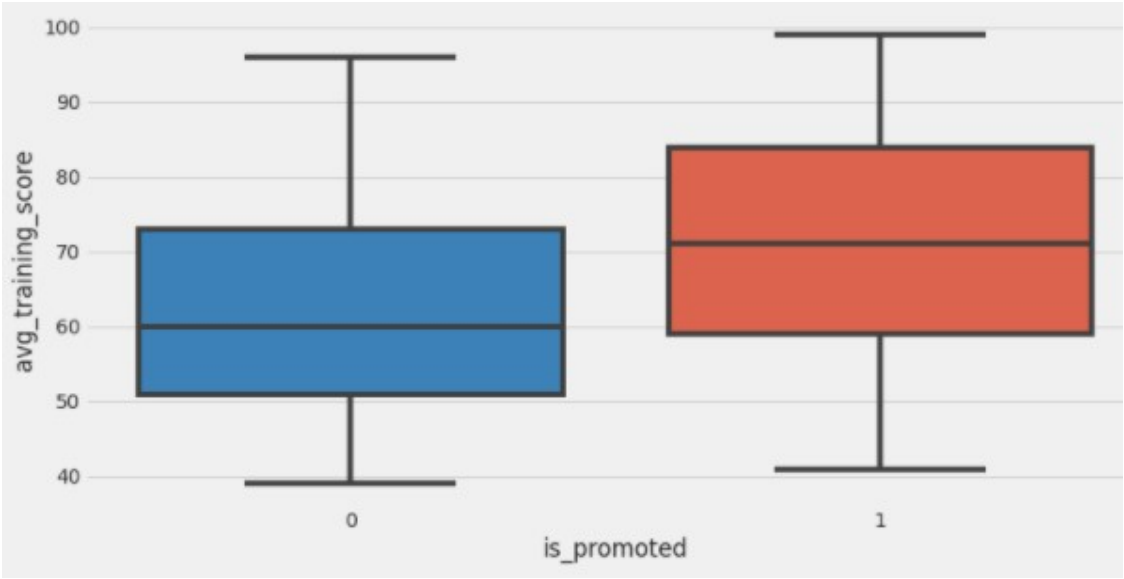


- The employees who achieved the KPI_met>80% milestone received the maximum rating(3.0 to 5.0).

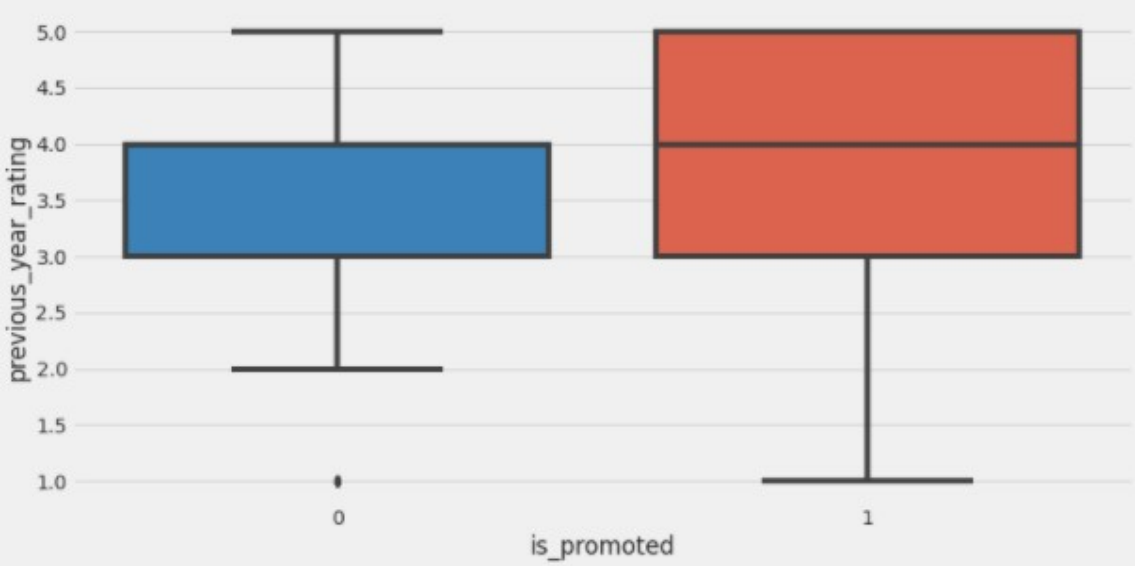


- 69% of promoted employees have achieved the KPI>80% milestone. This indicates the significance of KPIs_met >80% feature in predicting the target.

Is_promoted vs avg_training_score :

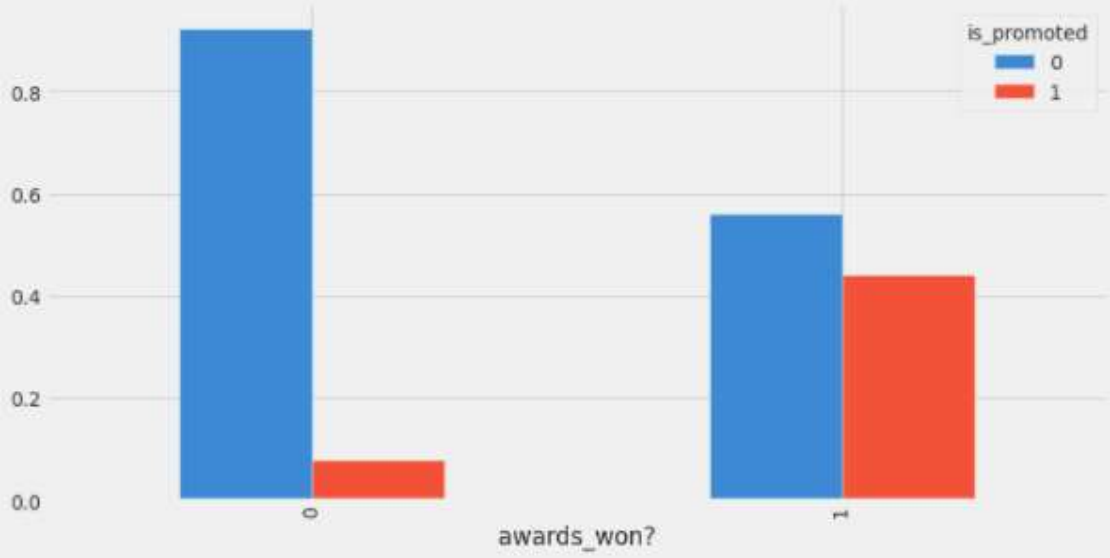


- The median training score of promoted employees is higher than the not promoted employees. This infers that training score is also one of the powerful feature in target prediction



- The median rating of promoted employees is better than the not promoted employees. This infers that rating is also one of the significant features in target prediction.

Awards_won? Vs is_promoted:



- 44% of award winners are promoted, this indicates there is some relationship between awards won and the target variable.

	Columns	P-val	Significance
0	department	9.882497e-29	Significant
1	region	7.275378e-80	Significant
2	education	1.071652e-08	Significant
3	gender	9.301396e-03	Significant
4	recruitment_channel	6.650938e-05	Significant
5	no_of_trainings	5.570014e-09	Significant
6	age	5.846305e-05	Significant
7	previous_year_rating	0.000000e+00	Significant
8	length_of_service	1.249145e-02	Significant
9	KPIs_met >80%	0.000000e+00	Significant
10	awards_won?	0.000000e+00	Significant
11	avg_training_score	0.000000e+00	Significant
12	is_promoted	0.000000e+00	Significant

- Statistical test confirms that all the features are significant

Feature Engineering:

- Transformations are not required for now.
- Encoding is required for the categorical column.
- Statistical test confirms that all the features are significant, so no further feature engineering is required as of now.

ENCODING:

- We have done a one-hot encoding for all the columns except column region. Because it has 35 levels of categories.
- Here we done frequency encoding for column region

Encoding

In [24]:
```
hr=pd.get_dummies(data=hr,columns=['department','education','gender','recruitment_channel']
                  ,drop_first=True)
```

In [25]:
```
m=hr['region'].value_counts()
hr['region']=hr['region'].map(m)
```

Train test split:

Before building the model, each data set is separated into train and test data for performing evaluation test on the model.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =train_test_split(x.drop(['education_Below Secondary','gender_m','recruitment_channel_referred',
                                                           'recruitment_channel_sourcing'],axis=1),
                                                   y, test_size=0.30,random_state=10)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(38365, 17)
(38365,)
(16443, 17)
(16443,)
```

MODEL BUILDING

Base model Building:

Logistic Regression Using Logit Function.

Logistic Regression is one of the simple and commonly used Machine Learning algorithms for two-class classification. It is easy to implement and can be used as the baseline for any binary classification problem. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.

The equation of a logistic regression equation is defined as,

$$Ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k$$

Performance Metrics of Logistic Regression Using Logit Function

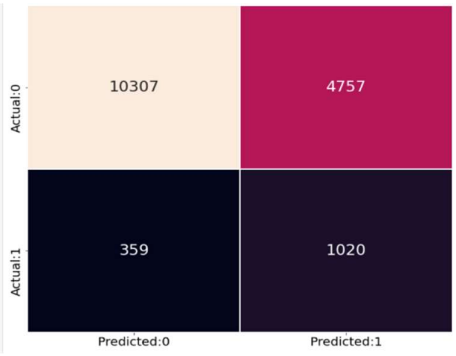
	coef	std err	z	P>|z|	[0.025	0.975]
const	-29.9348	0.466	-64.245	0.000	-30.848	-29.022
region	1.256e-05	4.41e-06	2.850	0.004	3.92e-06	2.12e-05
no_of_trainings	-0.1079	0.035	-3.118	0.002	-0.176	-0.040
age	-0.0286	0.004	-7.996	0.000	-0.036	-0.022
previous_year_rating	0.2591	0.017	15.026	0.000	0.225	0.293
length_of_service	0.0282	0.006	4.752	0.000	0.017	0.040
KPIs_met >80%	1.8790	0.044	42.659	0.000	1.793	1.965
awards_won?	1.4752	0.079	18.761	0.000	1.321	1.629
avg_training_score	0.3081	0.005	60.302	0.000	0.298	0.318
department_Finance	7.1472	0.159	44.996	0.000	6.836	7.459
department_HR	10.0516	0.210	47.879	0.000	9.640	10.463
department_Legal	6.9818	0.211	33.138	0.000	6.569	7.395
department_Operations	7.3102	0.140	52.116	0.000	7.035	7.585
department_Procurement	4.4387	0.104	42.821	0.000	4.236	4.642
department_R&D	-0.4592	0.146	-3.142	0.002	-0.746	-0.173
department_Sales & Marketing	10.4856	0.187	56.130	0.000	10.119	10.852
department_Technology	1.7452	0.074	23.491	0.000	1.600	1.891
education_Below Secondary	-0.1407	0.154	-0.916	0.360	-0.442	0.161
education_Master's & above	0.1919	0.042	4.609	0.000	0.110	0.273
gender_m	0.0194	0.041	0.469	0.639	-0.062	0.101
recruitment_channel_referred	-0.0994	0.113	-0.881	0.378	-0.320	0.122
recruitment_channel_sourcing	-0.0097	0.037	-0.260	0.795	-0.082	0.063

- We can see that there are some insignificant features after encoding.

Logistic Regression Using Sklearn.

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

Performance Metrics of Logistic Regression Using Logit Function.




```
Training Accuracy
0.6965723967157565
Testing Accuracy
0.6888645624277808
Confusion Matrix
[[10307  4757]
 [  359  1020]]
              precision    recall  f1-score   support

           0       0.97      0.68      0.80     15064
           1       0.18      0.74      0.29      1379

    accuracy                           0.69     16443
   macro avg       0.57      0.71      0.54     16443
weighted avg       0.90      0.69      0.76     16443

cross_val_f1_train 0.5361173911637599
cross_val_varience_train 0.052547808295940225

cross_val_f1_test 0.5089011308449205
cross_val_varience_test 0.046609830595935084
```

Models which we going to use are:

Decision Tree with Hyper Parameter Tuning.

Decision tree is a machine learning algorithm which can be used for classification as well as regression problems. The name itself suggests that it uses a flowchart like a tree structure to show the predictions that result from a series of feature-based splits. It starts with a root node and ends with a decision made by leaves. They are powerful analytical models that have the ability to comprehend data with minimal pre-processing time. Initially we tried training the model with decision tree with default parameters and observed that the model was over fit. Hence in order to overcome this problem, we will be doing Hyperparameter tuning using GridsearchCV.

Random Forest with Hyper Parameter Tuning.

Random forests or random decision forests are an ensemble learning method
for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees.

Gradient Boosting with Hyper Parameter Tuning.

Gradient boosting is also an ensemble learning method for classification. Gradient boosting algorithm is one of the most powerful algorithms in the field of machine learning. As we know that the errors in machine learning algorithms are broadly classified into two categories i.e. Bias Error and Variance Error. As gradient boosting is one of the boosting algorithms it is used to minimize bias error of the model**.**

XGBoost With Hyper Parameter Tuning.

XGBoost or Extreme Gradient Boosting, is a decision-tree based ensemble learner using the boosting technique, which is designed to achieve great speed and optimal performance. XGBoost works by building decision trees in sequences, one after another, where the next built decision tree focuses on the errors and weaknesses from the previous one and improves the prediction performance with those weaknesses in consideration. In each iteration or sequence the tree structure learns from the previous tree's outcome and residuals. Residuals is the difference between the real value and the predictive value. By combining these sets of weak learners, starting with one base learner, XGBoost creates a strong classifier in order to make more accurate predictions. XGBoost's great cache optimization comes with both pros and cons. The advantage is that XGBoost can predict outputs with a high accuracy. Unlike most tree learning algorithms, the XGBoost classifier finds the best split amongst trees and is able to handle datasets with missing values accurately. This is why it is considered a good model when it comes to performance in terms of its high prediction accuracy.

- Below results are obtained in a way that model doesn't overfit using multiple hyper tuned parameters.

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	Kappa Score	f1_macro
0	Logistic	0.722924	0.184215	0.749819	0.700541	0.186186	0.515430
1	Decisiontree	0.807952	0.226689	0.895577	0.735024	0.263174	0.593178
2	RandomForest	0.798906	0.210606	0.910080	0.706380	0.238305	0.588661
3	Gradient	0.671131	0.935039	0.344453	0.943015	0.479962	0.720957
4	XGBoost	0.673603	0.932302	0.349529	0.943319	0.484879	0.730644

- Below results are obtained in a way that model doesn't overfit using multiple hyper tuned parameters with outlier capping

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	Kappa Score	f1_macro
0	Logistic	0.717218	0.178936	0.749094	0.690689	0.177514	0.515430
1	Decisiontree	0.810148	0.226011	0.903553	0.732409	0.262648	0.592648
2	RandomForest	0.800412	0.217056	0.897027	0.720002	0.247971	0.601092
3	Gradient	0.672745	0.924855	0.348078	0.942954	0.482039	0.720957
4	XGBoost	0.677558	0.931947	0.357505	0.943928	0.493203	0.730644
5	Stacking	0.807107	0.220000	0.909355	0.722009	0.253462	0.630581

- There is no much difference with outlier treatment.

SMOTE:

SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie togeth

SMOTE

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 2)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train.ravel())
```

```
print(sum(y_train_res == 1))
print(sum(y_train_res == 0))
```

```
35076
35076
```

- Smote synthesis the minority group records to match the majority group records

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	Kappa Score	f1_macro
0	Logistic	0.679586	0.166911	0.661349	0.694764	0.153128	0.515430
1	Decisiontree	0.692743	0.239757	0.543147	0.817247	0.244787	0.573626
2	Randomforest	0.744627	0.305486	0.617839	0.850149	0.334090	0.614655
3	Gradient	0.701850	0.452639	0.453952	0.908168	0.403169	0.718985
4	XGBoost	0.677124	0.715092	0.367658	0.934683	0.454567	0.730644
5	Stacking	0.725739	0.304684	0.570703	0.854771	0.323273	0.619360

- Problem with smote is it overfits the model.

```
train_smote accuracy : 0.9289685254875129
train accuracy : 0.9108823146096703
test accuracy : 0.9081676093170346

synthasized data
[[33372  1704]
 [ 3279 31797]]
              precision    recall  f1-score   support

           0       0.91      0.95      0.93     35076
           1       0.95      0.91      0.93     35076

    accuracy                           0.93     70152
   macro avg       0.93      0.93      0.93     70152
weighted avg       0.93      0.93      0.93     70152

train data without smote
[[33372  1704]
 [ 1715  1574]]
              precision    recall  f1-score   support

           0       0.95      0.95      0.95     35076
           1       0.48      0.48      0.48      3289

    accuracy                           0.91     38365
   macro avg       0.72      0.71      0.72     38365
weighted avg       0.91      0.91      0.91     38365

test data without smote
[[14307   757]
 [  753   626]]
              precision    recall  f1-score   support

           0       0.95      0.95      0.95     15064
           1       0.45      0.45      0.45      1379

    accuracy                           0.91     16443
   macro avg       0.70      0.70      0.70     16443
weighted avg       0.91      0.91      0.91     16443

train cohen score 0.4306372918538671
test cohen score 0.40316868348096835
Train_roc_auc 0.7149923437760533
Test roc auc 0.7018499410973417
```

Synthesized data macro f1score is 0.93 and test data's is 0.70. this shows that the use of smote is overfit.

Below results are obtained using smote and outlier capped

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	Kappa Score	f1_macro
0	Logistic	0.670802	0.150713	0.705584	0.641854	0.127831	0.515430
1	Decisiontree	0.713687	0.331196	0.524293	0.871313	0.337890	0.589313
2	RandomForest	0.732976	0.317460	0.580131	0.860184	0.338668	0.627356
3	Gradient	0.696033	0.469626	0.437273	0.911391	0.404731	0.720957
4	XGBoost	0.719970	0.369476	0.521392	0.885240	0.370705	0.704693
5	Stacking	0.721071	0.337881	0.538796	0.872773	0.348116	0.704693

Model building by applying PCA:

Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance**.**

```
train accuracy : 0.9142708197575915
tesr accuracy : 0.9161345253299277
[[35076     0]
 [ 3289     0]]
              precision    recall  f1-score   support

           0       0.91      1.00      0.96     35076
           1       0.00      0.00      0.00      3289

    accuracy                           0.91     38365
   macro avg       0.46      0.50      0.48     38365
weighted avg       0.84      0.91      0.87     38365

[[15064     0]
 [ 1379     0]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     15064
           1       0.00      0.00      0.00      1379

    accuracy                           0.92     16443
   macro avg       0.46      0.50      0.48     16443
weighted avg       0.84      0.92      0.88     16443
```

- PCA completely avoids classifying the minority class, this happens because PCA removes minority class as noise

Model evaluation:

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	Kappa Score	f1_macro	Description
0	Logistic	0.711940	0.176562	0.739666	0.688865	0.173107	0.508901	without Smote
1	Decisiontree	0.808973	0.227031	0.897752	0.735085	0.263861	0.593102	without Smote
2	RandomForest	0.803209	0.210622	0.923133	0.703400	0.239067	0.588480	without Smote
3	Gradient	0.674032	0.936047	0.350254	0.943502	0.486301	0.720957	without Smote
4	XGBoost	0.673603	0.932302	0.349529	0.943319	0.484879	0.730644	without Smote
0	Logistic	0.717218	0.178936	0.749094	0.690689	0.177514	0.515430	without Smote&OutlierTreated
1	Decisiontree	0.811235	0.226432	0.905729	0.732591	0.263459	0.592938	without Smote&OutlierTreated
2	RandomForest	0.799298	0.221213	0.883249	0.729429	0.253708	0.613304	without Smote&OutlierTreated
3	Gradient	0.672745	0.924855	0.348078	0.942954	0.482039	0.720957	without Smote&OutlierTreated
4	XGBoost	0.677558	0.931947	0.357505	0.943928	0.493203	0.730644	without Smote&OutlierTreated
5	Stacking	0.792765	0.235078	0.833938	0.758499	0.271438	0.638129	without Smote&OutlierTreated
0	Logistic	0.679586	0.166911	0.661349	0.694764	0.153128	0.515430	with Smote
1	Decisiontree	0.692776	0.239833	0.543147	0.817308	0.244880	0.573526	with Smote
2	Randomforest	0.744742	0.311717	0.613488	0.853980	0.339982	0.614830	with Smote
3	Gradient	0.701850	0.452639	0.453952	0.908168	0.403169	0.718985	with Smote
4	XGBoost	0.677124	0.715092	0.367658	0.934683	0.454567	0.730644	with Smote
5	Stacking	0.690622	0.507476	0.418419	0.917168	0.414267	0.611892	with Smote
0	Logistic	0.670802	0.150713	0.705584	0.641854	0.127831	0.515430	with Smote&OutlierTreated
1	Decisiontree	0.712995	0.330734	0.522843	0.871252	0.337059	0.589092	with Smote&OutlierTreated
2	RandomForest	0.732799	0.312476	0.583031	0.857447	0.334172	0.627807	with Smote&OutlierTreated
3	Gradient	0.696033	0.469626	0.437273	0.911391	0.404731	0.720957	with Smote&OutlierTreated
4	XGBoost	0.719970	0.369476	0.521392	0.885240	0.370705	0.704693	with Smote&OutlierTreated
5	Stacking	0.720135	0.235422	0.626541	0.798030	0.250918	0.704693	with Smote&OutlierTreated

- We didn't consider results obtained using SMOTE. Because it overfits the model
- If we try to avoid the overfit, it lowers the model's performance metrics score.
- From the above result table, we can conclude that model Random forest with outlier capping without using smote gives best performance considering with our constraints.
- Constraints are predicting "1" is important because predicting eligible employees for promotion is more important than predicting employees not eligible for promotion.So considering recall is very important.

5.Comparison to benchmark

- Our accuracy score in the base model(Logistic Regression) is 0.70 and F1_score is 0.51
- In our case predicting “1” is important because predicting eligible employees for promotion is more important than predicting employees not eligible for promotion.
- So in this case, we have to set our recall score high.
- Recall score for base model(Logistic Regression) is 0.70.

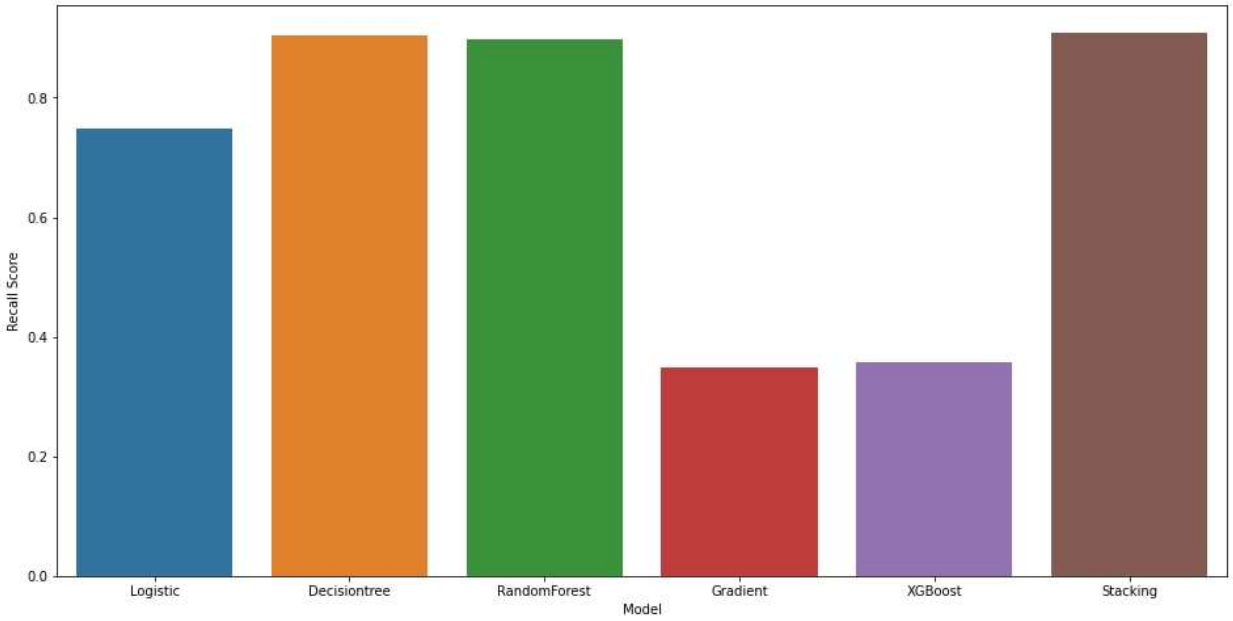
```
predictions=[1 if i>0.15 else 0 for i in y_pred_prob[:,1]]
predictions_train=[1 if i>0.15 else 0 for i in y_pred_prob1[:,1]]
```

```
print('train accuracy :', accuracy_score(y_train,predictions_train))
print('test accuracy :', accuracy_score(y_test,predictions))

print(confusion_matrix(y_train, predictions_train))
print(classification_report(y_train, predictions_train))

print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))

print('Train_roc_auc',roc_auc_score(y_train, predictions_train))
print('Test_roc_auc',roc_auc_score(y_test, predictions))
```

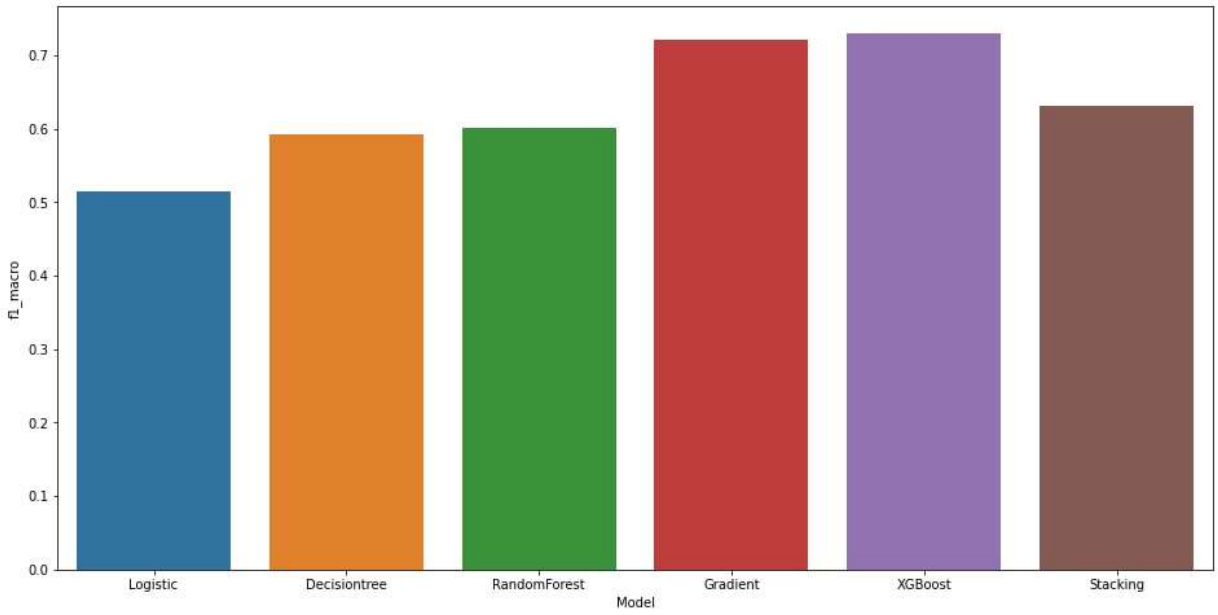
```
train accuracy : 0.847673660888831
test accuracy : 0.8414522897281518
[[30232  4844]
 [ 1000  2289]]
              precision    recall  f1-score   support

           0       0.97      0.86      0.91     35076
           1       0.32      0.70      0.44      3289

    accuracy                           0.85     38365
   macro avg       0.64      0.78      0.68     38365
weighted avg       0.91      0.85      0.87     38365

[[12886  2178]
 [  429   950]]
              precision    recall  f1-score   support

           0       0.97      0.86      0.91     15064
           1       0.30      0.69      0.42      1379

    accuracy                           0.84     16443
   macro avg       0.64      0.77      0.66     16443
weighted avg       0.91      0.84      0.87     16443
```

- Above results are obtained by Gradient-Boosting model with hyper-tuned parameters and adjusted threshold of 0.15.
- We can clearly see that recall scores are higher than precision scores and they also have good accuracy score and F1_score.
- But by adjusting threshold too much, it creates bias in this model. So, we’re not considering this approach.
- From the above result table, we can conclude that model Random forest with outlier capping without using smote gives best performance considering with our constraints.
- Constraints are predicting “1” is important because predicting eligible employees for promotion is more important than predicting employees not eligible for promotion.So considering recall is very important.

Recall Score plot:



F1_Score plot:



From the above plots we can conclude that model Random Forest gives the balanced F1_Score and recall score. So we chose our final model as Random Forest with outlier capped data.

Implications:

- Predicting the employee promotion eligibility in HR analytics help to identify those traits that are predictors of success within an organization or a team, so we can focus our search for talent and avoid making mistakes that can be expensive and time- consuming to correct.

Limitations:

- Since our model didn't linearly separate both the classes much we can 72% rely on this model results.
- Since we have only limited data points regarding eligible employee promotion, model cannot learn much about the eligible employee promotion data points. So that we are not able obtain high performance in the model.

Closing Reflections:

- There is huge learning in terms of understanding the domain and how important is the understanding the dataset is imperative. As the lack of understanding of the dataset can lead to catastrophic results.
- From our results we learnt that how important is to classify the important class over the other class.
- Ensemble techniques helped to improve the performance of our employee promotion eligibility.
- Understood how to handling imbalanced dataset and different imbalance handling techniques.
- Outlier treatment helped us to improve the performance of our model slightly.

References:

- https://www.analyticsvidhya.com/blog/2021/05/logistic-regression-supervised-learning-algorithm-for-classification/
- https://www.analyticsvidhya.com/blog/2021/05/dealing-with-missing-values-in-python-a- complete-guide/
- https://www.geeksforgeeks.org/grouping-categorical-variables-in-pandas-dataframe/
- https://www.analyticsvidhya.com/blog/2021/09/a-complete-guide-to-understand-classification-in-machine-learning/
- https://pythonbasics.org/split-train-test/
- https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd- one-out/
- https://towardsdatascience.com/churn-prediction-with-machine-learning-ca955d52bd8c
- https://towardsdatascience.com/3-ways-to-improve-your-machine-learning-results-without-more-data-f2f0fe78976e
- https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/
- https://www.analyticsvidhya.com/blog/2021/04/a-profound-comprehension-of-bias-and-variance/
- https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/