In [1]:
```
!dir
#!pip install openpyxl
!python.exe -m pip install --upgrade pip
```

```
Requirement already satisfied: pip in c:\users\amit pc\appdata\local\programs\python\python37\lib\site-packages (23.0.1)
```

Loading data

In [2]:
```
import pandas as pd

train_data = pd.read_csv("Train_data.csv")

train_data.head()


test_data = pd.read_csv("Test_data.csv")

test_data.head()
```

Out[2]:

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score
0	8724	Technology	region_26	Bachelor's	m	sourcing	1	24	NaN	1	1	0	77
1	74430	HR	region_4	Bachelor's	f	other	1	31	3.0	5	0	0	51
2	72255	Sales & Marketing	region_13	Bachelor's	m	other	1	31	1.0	4	0	0	47
3	38562	Procurement	region_2	Bachelor's	f	other	3	31	2.0	9	0	0	65
4	64486	Finance	region_29	Bachelor's	m	sourcing	1	30	4.0	7	0	0	61

Data Info

In [3]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54808 entries, 0 to 54807
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   employee_id           54808 non-null  int64  
 1   department            54808 non-null  object 
 2   region                54808 non-null  object 
 3   education             52399 non-null  object 
 4   gender                54808 non-null  object 
 5   recruitment_channel   54808 non-null  object 
 6   no_of_trainings       54808 non-null  int64  
 7   age                   54808 non-null  int64  
 8   previous_year_rating  50684 non-null  float64
 9   length_of_service     54808 non-null  int64  
 10  KPIs_met >80%         54808 non-null  int64  
 11  awards_won?           54808 non-null  int64  
 12  avg_training_score    54808 non-null  int64  
 13  is_promoted           54808 non-null  int64  
dtypes: float64(1), int64(8), object(5)
memory usage: 5.9+ MB
```

In [4]:
```
train_data.dtypes
```

Out[4]:
```
employee_id              int64
department              object
region                  object
education               object
gender                  object
recruitment_channel     object
no_of_trainings          int64
age                      int64
previous_year_rating   float64
length_of_service        int64
KPIs_met >80%            int64
awards_won?              int64
avg_training_score       int64
is_promoted              int64
dtype: object
```

Data Cleaning

In [5]:
```
#Checking for NULL values
train_data.isna().sum()
```

Out[5]:
```
employee_id              0
department               0
region                   0
education             2409
gender                   0
recruitment_channel      0
no_of_trainings          0
age                      0
previous_year_rating    4124
length_of_service        0
KPIs_met >80%            0
awards_won?              0
avg_training_score       0
is_promoted              0
dtype: int64
```

In [6]:
```
# Replacing NAs in education by 'Other'

train_data['education'] = train_data['education'].fillna('Other')
test_data['education'] = test_data['education'].fillna('Other')
```

In [7]:
```
# Replacing NAs in previous_year_rating by '0'

train_data['previous_year_rating'] = train_data['previous_year_rating'].fillna(0)
test_data['previous_year_rating'] = test_data['previous_year_rating'].fillna(0)
```

In [8]:
```
# All NULL values are gone
train_data.isna().sum()
```

Out[8]:
```
employee_id             0
department              0
region                  0
education               0
gender                  0
recruitment_channel     0
no_of_trainings         0
age                     0
previous_year_rating    0
length_of_service       0
KPIs_met >80%           0
```

```
awards_won?             0
avg_training_score      0
is_promoted             0
dtype: int64
```

In [9]:
```
# Checking for duplicates
train_data.duplicated().sum()
```

Out[9]: 0

In [10]:
```
# Dropping employee_id since it is redundant
train_data = train_data.drop(["employee_id"], axis=1)
test_data = test_data.drop(["employee_id"], axis=1)
```

Checking for column types

In [11]:
```
import numpy as np

numeric_columns = list(train_data.select_dtypes(include=np.number).columns)
categorical_columns = list(train_data.select_dtypes(include="object").columns)
```

Numeric Columns

In [12]:
```
train_data[numeric_columns].describe()
```

Out[12]:

	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score	is_promoted
count	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000
mean	1.253011	34.803915	3.078748	5.865512	0.351974	0.023172	63.386750	0.085170
std	0.609264	7.660169	1.496458	4.265094	0.477590	0.150450	13.371559	0.279137
min	1.000000	20.000000	0.000000	1.000000	0.000000	0.000000	39.000000	0.000000
25%	1.000000	29.000000	2.000000	3.000000	0.000000	0.000000	51.000000	0.000000
50%	1.000000	33.000000	3.000000	5.000000	0.000000	0.000000	60.000000	0.000000
75%	1.000000	39.000000	4.000000	7.000000	1.000000	0.000000	76.000000	0.000000
max	10.000000	60.000000	5.000000	37.000000	1.000000	1.000000	99.000000	1.000000

Categorical Columns

In [13]:
```
train_data[categorical_columns].describe()
```

Out[13]:

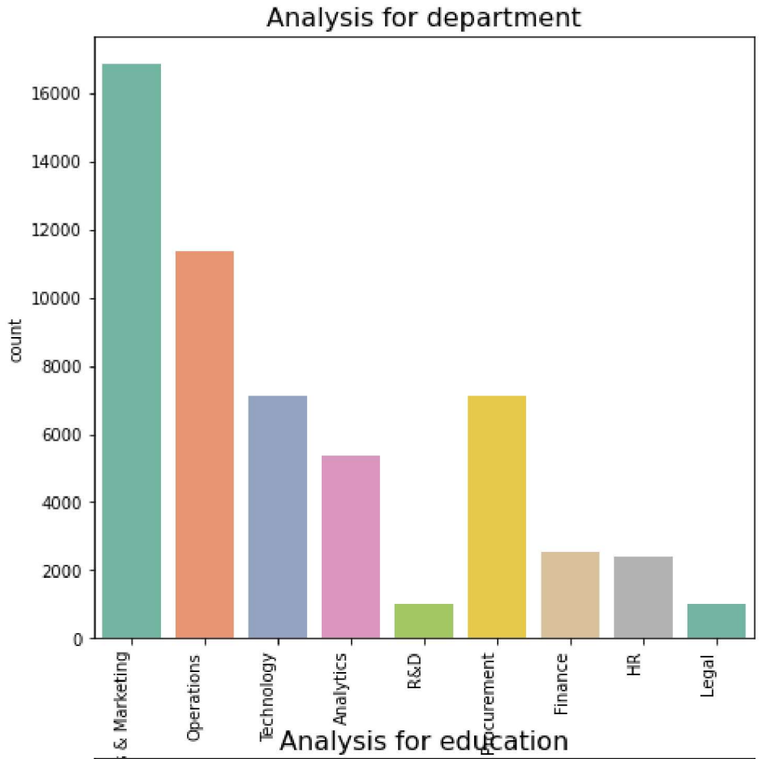
	department	region	education	gender	recruitment_channel
count	54808	54808	54808	54808	54808
unique	9	34	4	2	3
top	Sales & Marketing	region_2	Bachelor's	m	other
freq	16840	12343	36669	38496	30446

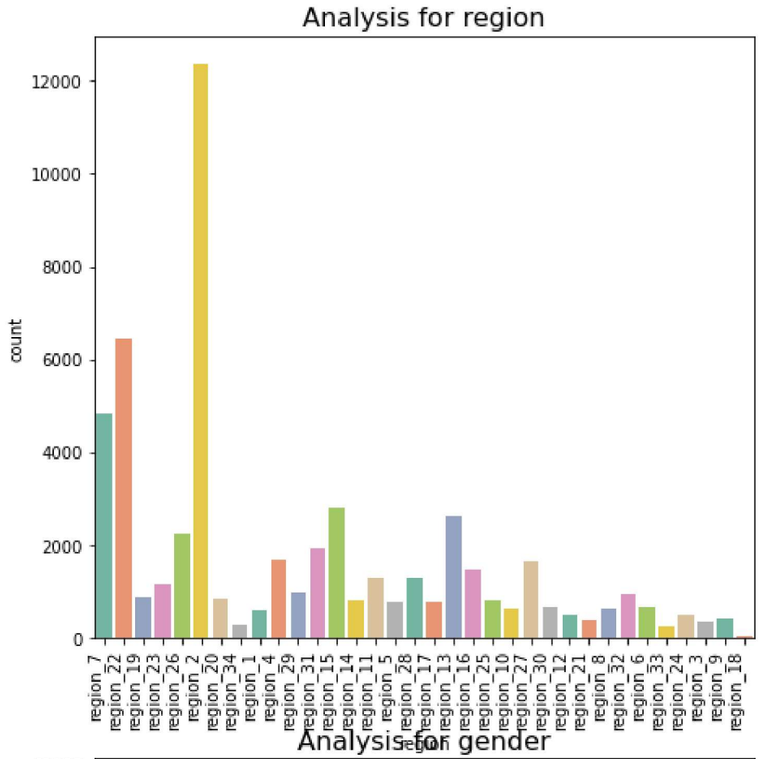
Categorical Column Data Analysis

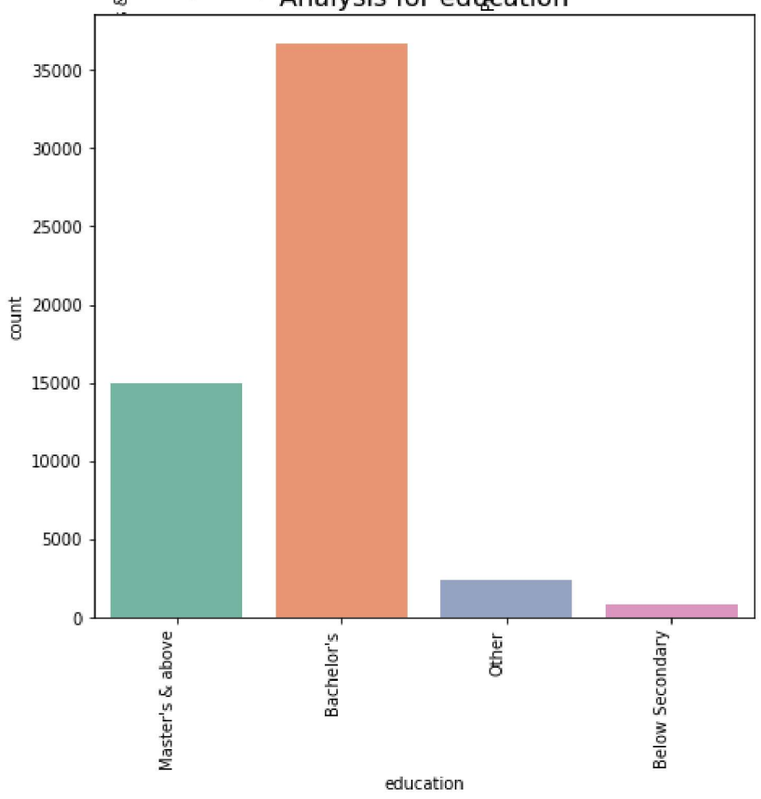
In [14]:

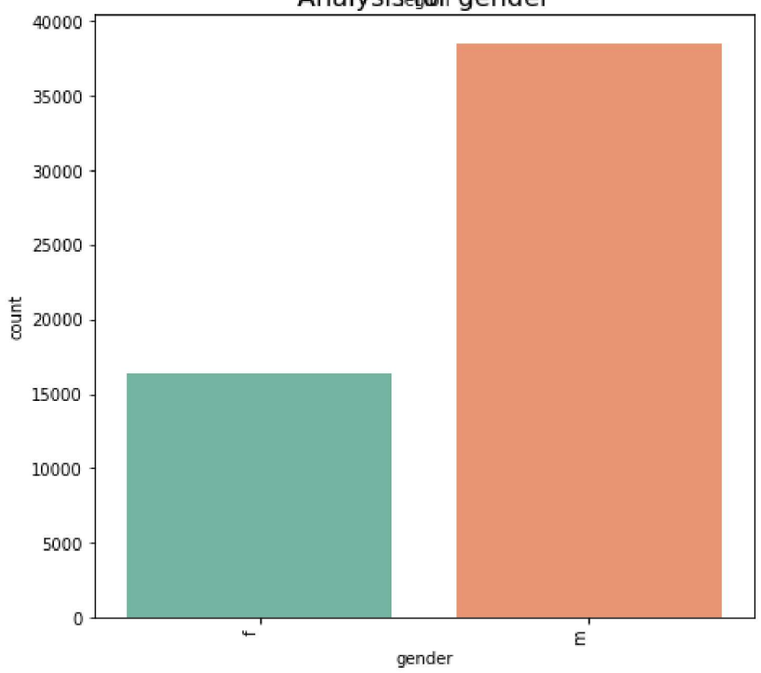
```
import matplotlib.pyplot as plt
import seaborn as sns

fig, ax = plt.subplots(2,2, figsize = (16,15))

ax = np.ravel(ax)
for i in range(len(categorical_columns)):
    sns.countplot(data = train_data, x = categorical_columns[i], ax = ax[i], palette="Set2")
    ax[i].set_xticklabels(labels = train_data[categorical_columns[i]].unique(), rotation=90, ha='right')
    ax[i].set_title(label = "Analysis for "+categorical_columns[i],fontsize=16)
ax = np.reshape(ax, (2, 2))
plt.tight_layout()
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-14-abd4a960abcf> in <module>
      6 ax = np.ravel(ax)
      7 for i in range(len(categorical_columns)):
----> 8     sns.countplot(data = train_data, x = categorical_columns[i], ax = ax[i], palette="Set2")
      9     ax[i].set_xticklabels(labels = train_data[categorical_columns[i]].unique(), rotation=90, ha='right')
     10     ax[i].set_title(label = "Analysis for "+categorical_columns[i],fontsize=16)

IndexError: index 4 is out of bounds for axis 0 with size 4
```









One Hot Encoding for Categorical Data

In [15]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

train_data["department"] = label_encoder.fit_transform(train_data["department"])
train_data["region"] = label_encoder.fit_transform(train_data["region"])
train_data["education"] = label_encoder.fit_transform(train_data["education"])
train_data["gender"] = label_encoder.fit_transform(train_data["gender"])
train_data["recruitment_channel"] = label_encoder.fit_transform(train_data["recruitment_channel"])


test_data["department"] = label_encoder.fit_transform(test_data["department"])
test_data["region"] = label_encoder.fit_transform(test_data["region"])
test_data["education"] = label_encoder.fit_transform(test_data["education"])
test_data["gender"] = label_encoder.fit_transform(test_data["gender"])
test_data["recruitment_channel"] = label_encoder.fit_transform(test_data["recruitment_channel"])
```

In [16]:
```
X = train_data.loc[:, train_data.columns != "is_promoted"].copy()
y = train_data.loc[:, "is_promoted"].copy()

y = label_encoder.fit_transform(y)
y = pd.DataFrame(y, index= train_data.index, columns=["is_promoted"])

categorical_cols_features = list(X.select_dtypes(include="object").columns)
```

In [17]:
```
# Data After Encoding
X.head()
```

Out[17]:

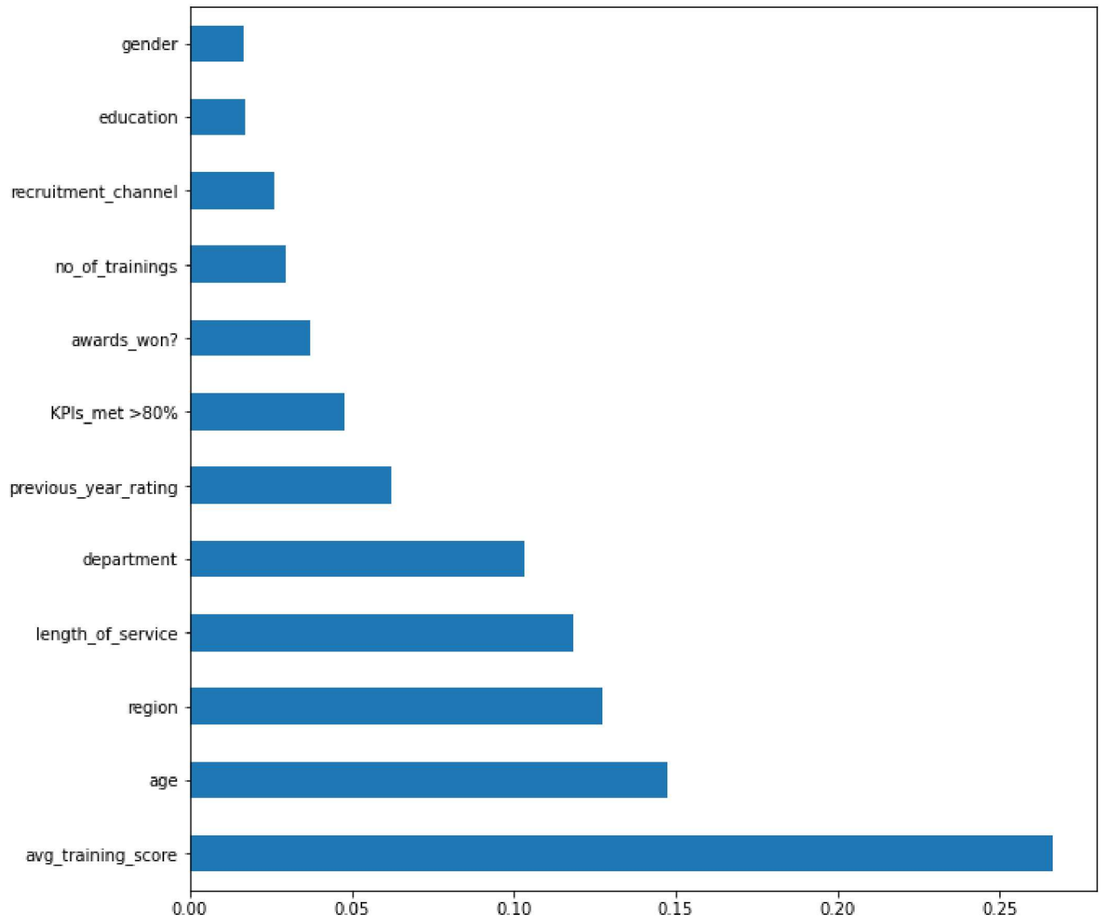
	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score
0	7	31	2	0	2	1	35	5.0	8	1	0	49
1	4	14	0	1	0	1	30	5.0	4	0	0	60
2	7	10	0	1	2	1	34	3.0	7	0	0	50
3	7	15	0	1	0	2	39	1.0	10	0	0	50
4	8	18	0	1	0	1	45	3.0	2	0	0	73

Feature Importance

Getting the feature importance for every column with respect to Target variable

In [18]:
```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt

model = ExtraTreesClassifier()
model.fit(X,np.ravel(y))


f = plt.figure()
f.set_figwidth(10)
f.set_figheight(10)


#plot graph of feature importances for better visualization
```

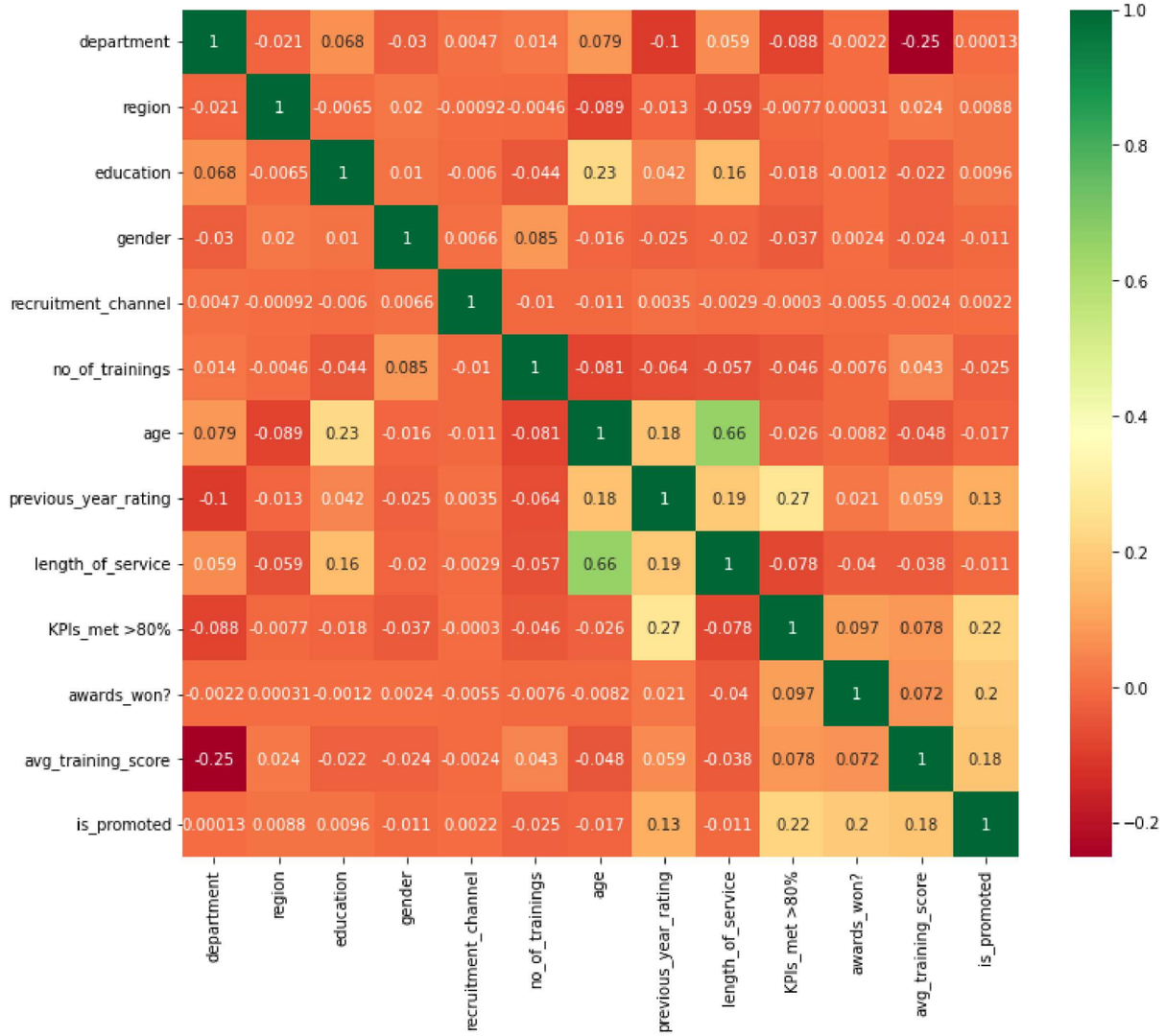
```
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(35).plot(kind='barh')
plt.show()
```



Correlation Matrix

In [19]:

```
#get correlations of each features in dataset
corrmat = train_data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(12,10))
#plot heat map
g=sns.heatmap(train_data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



Inference

- From categorical importance we can infer that there is a hugh class imbalance in all the 4 attributes
- From the feature importance and Correlation Matrix we understand the importance of each attribute among which avg_training_score scores the highest
- All preprocessing steps are performed and after performing EDA in this module are data is ready to train under a classifier

Model Selection

Scaling data

In [23]:
```
X.describe()
```

Out[23]:

	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score
count	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000
mean	4.972796	15.428970	0.691176	0.702379	0.868158	1.253011	34.803915	3.078748	5.865512	0.351974	0.023172	63.386750

	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score
std	2.506046	8.764954	1.010851	0.457216	0.980713	0.609264	7.660169	1.496458	4.265094	0.477590	0.150450	13.371559
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	20.000000	0.000000	1.000000	0.000000	0.000000	39.000000
25%	4.000000	11.000000	0.000000	0.000000	0.000000	1.000000	29.000000	2.000000	3.000000	0.000000	0.000000	51.000000
50%	5.000000	14.000000	0.000000	1.000000	0.000000	1.000000	33.000000	3.000000	5.000000	0.000000	0.000000	60.000000
75%	7.000000	21.000000	2.000000	1.000000	2.000000	1.000000	39.000000	4.000000	7.000000	1.000000	0.000000	76.000000
max	8.000000	33.000000	3.000000	1.000000	2.000000	10.000000	60.000000	5.000000	37.000000	1.000000	1.000000	99.000000

In [35]:
```
from sklearn.preprocessing import MinMaxScaler
MinMaxScaled_X = pd.DataFrame(MinMaxScaler().fit_transform(X),columns=X.columns)
MinMaxScaled_X.describe()
```

Out[35]:

	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score
count	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000
mean	0.621599	0.467545	0.230392	0.702379	0.434079	0.028112	0.370098	0.615750	0.135153	0.351974	0.023172	0.406446
std	0.313256	0.265605	0.336950	0.457216	0.490357	0.067696	0.191504	0.299292	0.118475	0.477590	0.150450	0.222859
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.500000	0.333333	0.000000	0.000000	0.000000	0.000000	0.225000	0.400000	0.055556	0.000000	0.000000	0.200000
50%	0.625000	0.424242	0.000000	1.000000	0.000000	0.000000	0.325000	0.600000	0.111111	0.000000	0.000000	0.350000
75%	0.875000	0.636364	0.666667	1.000000	1.000000	0.000000	0.475000	0.800000	0.166667	1.000000	0.000000	0.616667
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [37]:
```
from sklearn.preprocessing import StandardScaler
StandardScaled_X = pd.DataFrame(StandardScaler().fit_transform(X),columns=X.columns)
StandardScaled_X.describe()
```

Out[37]:

	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score
count	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04
mean	-1.349575e-16	7.687780e-17	7.182176e-17	3.292911e-17	1.685348e-17	5.133829e-17	-3.764808e-16	5.795004e-17	5.477381e-17	1.322350e-17	7.130319e-19	-2.353005e-17
std	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00
min	-1.984338e+00	-1.760319e+00	-6.837634e-01	-1.536223e+00	-8.852391e-01	-4.152762e-01	-1.932601e+00	-2.057375e+00	-1.140785e+00	-7.369860e-01	-1.540178e-01	-1.823794e+00
25%	-3.881832e-01	-5.053091e-01	-6.837634e-01	-1.536223e+00	-8.852391e-01	-4.152762e-01	-7.576815e-01	-7.208737e-01	-6.718582e-01	-7.369860e-01	-1.540178e-01	-9.263589e-01
50%	1.085547e-02	-1.630338e-01	-6.837634e-01	6.509473e-01	-8.852391e-01	-4.152762e-01	-2.354951e-01	-5.262314e-02	-2.029311e-01	-7.369860e-01	-1.540178e-01	-2.532824e-01
75%	8.089329e-01	6.356087e-01	1.294786e+00	6.509473e-01	1.154111e+00	-4.152762e-01	5.477846e-01	6.156274e-01	2.659960e-01	1.356878e+00	-1.540178e-01	9.432980e-01
max	1.207972e+00	2.004710e+00	2.284061e+00	6.509473e-01	1.154111e+00	1.435678e+01	3.289263e+00	1.283878e+00	7.299903e+00	1.356878e+00	6.492758e+00	2.663382e+00

In [31]:
```
from sklearn.preprocessing import StandardScaler
StandardScaled_X = pd.DataFrame(StandardScaler().fit_transform(X),columns=X.columns)
StandardScaled_X.describe()
```

Out[31]:

	0	1	2	3	4	5	6	7	8	9	10	11
count	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04	5.480800e+04
mean	-1.349575e-16	7.687780e-17	7.182176e-17	3.292911e-17	1.685348e-17	5.133829e-17	-3.764808e-16	5.795004e-17	5.477381e-17	1.322350e-17	7.130319e-19	-2.353005e-17
std	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00	1.000009e+00
min	-1.984338e+00	-1.760319e+00	-6.837634e-01	-1.536223e+00	-8.852391e-01	-4.152762e-01	-1.932601e+00	-2.057375e+00	-1.140785e+00	-7.369860e-01	-1.540178e-01	-1.823794e+00
25%	-3.881832e-01	-5.053091e-01	-6.837634e-01	-1.536223e+00	-8.852391e-01	-4.152762e-01	-7.576815e-01	-7.208737e-01	-6.718582e-01	-7.369860e-01	-1.540178e-01	-9.263589e-01
50%	1.085547e-02	-1.630338e-01	-6.837634e-01	6.509473e-01	-8.852391e-01	-4.152762e-01	-2.354951e-01	-5.262314e-02	-2.029311e-01	-7.369860e-01	-1.540178e-01	-2.532824e-01
75%	8.089329e-01	6.356087e-01	1.294786e+00	6.509473e-01	1.154111e+00	-4.152762e-01	5.477846e-01	6.156274e-01	2.659960e-01	1.356878e+00	-1.540178e-01	9.432980e-01
max	1.207972e+00	2.004710e+00	2.284061e+00	6.509473e-01	1.154111e+00	1.435678e+01	3.289263e+00	1.283878e+00	7.299903e+00	1.356878e+00	6.492758e+00	2.663382e+00

In [38]:

```
from sklearn.preprocessing import MaxAbsScaler
MaxAbsScaled_X = pd.DataFrame(MaxAbsScaler().fit_transform(X),columns=X.columns)
MaxAbsScaled_X.describe()
```

Out[38]:

	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score
count	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000
mean	0.621599	0.467545	0.230392	0.702379	0.434079	0.125301	0.580065	0.615750	0.158527	0.351974	0.023172	0.640270
std	0.313256	0.265605	0.336950	0.457216	0.490357	0.060926	0.127669	0.299292	0.115273	0.477590	0.150450	0.135066
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.100000	0.333333	0.000000	0.027027	0.000000	0.000000	0.393939
25%	0.500000	0.333333	0.000000	0.000000	0.000000	0.100000	0.483333	0.400000	0.081081	0.000000	0.000000	0.515152
50%	0.625000	0.424242	0.000000	1.000000	0.000000	0.100000	0.550000	0.600000	0.135135	0.000000	0.000000	0.606061
75%	0.875000	0.636364	0.666667	1.000000	1.000000	0.100000	0.650000	0.800000	0.189189	1.000000	0.000000	0.767677
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [39]:

```
from sklearn.preprocessing import RobustScaler
RobustScaled_X = pd.DataFrame(RobustScaler().fit_transform(X),columns=X.columns)
RobustScaled_X.describe()
```

Out[39]:

	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score
count	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000
mean	-0.009068	0.142897	0.345588	-0.297621	0.434079	0.253011	0.180392	0.039374	0.216378	0.351974	0.023172	0.135470
std	0.835349	0.876495	0.505425	0.457216	0.490357	0.609264	0.766017	0.748229	1.066274	0.477590	0.150450	0.534862
min	-1.666667	-1.400000	0.000000	-1.000000	0.000000	0.000000	-1.300000	-1.500000	-1.000000	0.000000	0.000000	-0.840000
25%	-0.333333	-0.300000	0.000000	-1.000000	0.000000	0.000000	-0.400000	-0.500000	-0.500000	0.000000	0.000000	-0.360000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.666667	0.700000	1.000000	0.000000	1.000000	0.000000	0.600000	0.500000	0.500000	1.000000	0.000000	0.640000
max	1.000000	1.900000	1.500000	0.000000	1.000000	9.000000	2.700000	1.000000	8.000000	1.000000	1.000000	1.560000

In [40]:

```
# Choosing Robust Scaler for future classification
```

```
X = pd.DataFrame(RobustScaler().fit_transform(X),columns=X.columns)
```

In [41]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

print("Shape of X_train",X_train.shape)
print("Shape of y_train",y_train.shape)
print("Shape of X_test",X_test.shape)
print("Shape of y_test",y_test.shape)
```

```
Shape of X_train (36721, 12)
Shape of y_train (36721, 1)
Shape of X_test (18087, 12)
Shape of y_test (18087, 1)
```

XGBoost

In [43]:

```
import xgboost as xgb
from sklearn import metrics
from sklearn.model_selection import GridSearchCV


# Parameter Tuning
model = xgb.XGBClassifier()
param_dist = {"max_depth": [10,30,50],
              "min_child_weight" : [1,3,6],
              "n_estimators": [200],
              "learning_rate": [0.05, 0.1,0.16],}
grid_search = GridSearchCV(model, param_grid=param_dist, cv = 3, verbose=10, n_jobs=-1)
grid_search.fit(X_train, y_train)

grid_search.best_estimator_
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   12.5s
[Parallel(n_jobs=-1)]: Done   8 tasks      | elapsed:   12.8s
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:   30.6s
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:   42.7s
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   58.8s
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done  67 out of  81 | elapsed:  1.6min remaining:   19.4s
[Parallel(n_jobs=-1)]: Done  76 out of  81 | elapsed:  1.8min remaining:    6.9s
[Parallel(n_jobs=-1)]: Done  81 out of  81 | elapsed:  1.8min finished
c:\users\amit pc\appdata\local\programs\python\python37\lib\site-packages\sklearn\utils\validation.py:72: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Pleas
e change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
[20:08:22] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release_1.2.0\src\learner.cc:516:
Parameters: { verbose } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```

Out[43]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.16, max_delta_step=0, max_depth=50,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=200, n_jobs=-1, num_parallel_tree=1, random_state=0,
```

```
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
                tree_method='exact', validate_parameters=1, verbose=1,
                verbosity=None)
```

In [48]:

```
model_xgb = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.16, max_delta_step=0, max_depth=50,
              min_child_weight=1,
              n_estimators=200, n_jobs=-1, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbose=1,
              verbosity=None)
model_xgb.fit(X_train,y_train)

print("XGB Accuracy:",np.round(model_xgb.score(X_test, y_test),2))
```

```
[20:11:46] WARNING: C:\Users\Administrator\workspace\xgboost-win64_release_1.2.0\src\learner.cc:516:
Parameters: { verbose } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


XGB Accuracy: 0.93
```

Light GBM

In [53]:

```
import lightgbm as lgb
from sklearn import metrics



lg = lgb.LGBMClassifier(silent=False)
param_dist = {"max_depth": [25,50, 75],
              "learning_rate" : [0.01,0.05,0.1],
              "num_leaves": [300,900,1200],
              "n_estimators": [200]
             }
grid_search = GridSearchCV(lg, n_jobs=-1, param_grid=param_dist, cv = 3, scoring="roc_auc", verbose=5)
grid_search.fit(X_train ,np.ravel(y_train))
grid_search.best_estimator_
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:   29.0s
[Parallel(n_jobs=-1)]: Done  75 out of  81 | elapsed:   46.1s remaining:    3.6s
[Parallel(n_jobs=-1)]: Done  81 out of  81 | elapsed:   49.4s finished
```

Out[53]:

```
LGBMClassifier(learning_rate=0.01, max_depth=25, n_estimators=200,
               num_leaves=300, silent=False)
```

In [67]:

```
d_train = lgb.Dataset(X_train, label=np.ravel(y_train))
d_test = lgb.Dataset(X_test, label=np.ravel(y_test))
params = {"learning_rate": 0.01, "max_depth":25, "n_estimators":200,"num_leaves":300, "silent":False}



model_lgb = lgb.train(params, d_train)
print("LGBM Accuracy:",np.round(metrics.roc_auc_score(y_test,model_lgb.predict(X_test)),2))
```

```
c:\users\amit pc\appdata\local\programs\python\python37\lib\site-packages\lightgbm\engine.py:148: UserWarning: Found `n_estimators` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argument".format(alias))
c:\users\amit pc\appdata\local\programs\python\python37\lib\site-packages\lightgbm\basic.py:842: UserWarning: silent keyword has been found in `params` and will be ignored.
Please use silent argument of the Dataset constructor to pass this parameter.
  .format(key))
LGBM Accuracy: 0.9
```

CatBoost

In [70]:

```
import catboost as cb


params = {'depth': [4, 7, 10],
          'learning_rate' : [0.03, 0.1, 0.15],
         'l2_leaf_reg': [1,4,9],
         'iterations': [300]}
cb = cb.CatBoostClassifier()
cb_model = GridSearchCV(cb, params, scoring="roc_auc", cv = 3)
cb_model.fit(X_train, y_train)
```

```
0:      learn: 0.6626597        total: 177ms    remaining: 52.8s
1:      learn: 0.6296728        total: 183ms    remaining: 27.3s
2:      learn: 0.6024411        total: 190ms    remaining: 18.8s
3:      learn: 0.5775570        total: 197ms    remaining: 14.6s
4:      learn: 0.5516782        total: 204ms    remaining: 12s
5:      learn: 0.5316971        total: 211ms    remaining: 10.4s
6:      learn: 0.5074128        total: 218ms    remaining: 9.12s
7:      learn: 0.4888483        total: 225ms    remaining: 8.21s
8:      learn: 0.4701638        total: 232ms    remaining: 7.49s
9:      learn: 0.4544760        total: 238ms    remaining: 6.91s
10:     learn: 0.4383534        total: 245ms    remaining: 6.43s
11:     learn: 0.4255390        total: 252ms    remaining: 6.04s
12:     learn: 0.4119026        total: 258ms    remaining: 5.69s
13:     learn: 0.3988480        total: 265ms    remaining: 5.41s
14:     learn: 0.3861647        total: 271ms    remaining: 5.16s
15:     learn: 0.3767283        total: 279ms    remaining: 4.95s
16:     learn: 0.3664709        total: 286ms    remaining: 4.76s
17:     learn: 0.3582252        total: 293ms    remaining: 4.58s
18:     learn: 0.3510102        total: 299ms    remaining: 4.43s
19:     learn: 0.3434335        total: 305ms    remaining: 4.27s
20:     learn: 0.3367717        total: 313ms    remaining: 4.15s
21:     learn: 0.3304430        total: 319ms    remaining: 4.04s
22:     learn: 0.3225893        total: 327ms    remaining: 3.94s
23:     learn: 0.3176251        total: 334ms    remaining: 3.84s
24:     learn: 0.3090463        total: 340ms    remaining: 3.74s
25:     learn: 0.3048049        total: 346ms    remaining: 3.64s
26:     learn: 0.3007645        total: 353ms    remaining: 3.56s
27:     learn: 0.2974581        total: 364ms    remaining: 3.53s
28:     learn: 0.2938049        total: 372ms    remaining: 3.48s
29:     learn: 0.2901896        total: 380ms    remaining: 3.42s
30:     learn: 0.2865920        total: 388ms    remaining: 3.37s
31:     learn: 0.2820886        total: 395ms    remaining: 3.31s
32:     learn: 0.2795114        total: 403ms    remaining: 3.26s
33:     learn: 0.2773040        total: 410ms    remaining: 3.21s
34:     learn: 0.2749820        total: 416ms    remaining: 3.15s
35:     learn: 0.2724542        total: 423ms    remaining: 3.1s
36:     learn: 0.2708260        total: 430ms    remaining: 3.06s
37:     learn: 0.2688410        total: 438ms    remaining: 3.02s
38:     learn: 0.2674357        total: 445ms    remaining: 2.98s
39:     learn: 0.2648287        total: 451ms    remaining: 2.93s
40:     learn: 0.2626902        total: 457ms    remaining: 2.89s
41:     learn: 0.2601445        total: 465ms    remaining: 2.85s
42:     learn: 0.2589561        total: 472ms    remaining: 2.82s
43:     learn: 0.2576519        total: 478ms    remaining: 2.78s
44:     learn: 0.2562807        total: 484ms    remaining: 2.74s
45:     learn: 0.2552270        total: 491ms    remaining: 2.71s
```

```
473:    total: 17.3s    remaining: 948ms
474:    total: 17.3s    remaining: 911ms
475:    total: 17.3s    remaining: 875ms
476:    total: 17.4s    remaining: 838ms
477:    total: 17.4s    remaining: 802ms
478:    total: 17.5s    remaining: 765ms
479:    total: 17.5s    remaining: 729ms
480:    total: 17.5s    remaining: 692ms
481:    total: 17.6s    remaining: 656ms
482:    total: 17.6s    remaining: 621ms
483:    total: 17.7s    remaining: 585ms
484:    total: 17.7s    remaining: 548ms
485:    total: 17.8s    remaining: 512ms
486:    total: 17.8s    remaining: 475ms
487:    total: 17.8s    remaining: 439ms
488:    total: 17.9s    remaining: 402ms
489:    total: 17.9s    remaining: 366ms
490:    total: 17.9s    remaining: 329ms
491:    total: 18s      remaining: 292ms
492:    total: 18s      remaining: 256ms
493:    total: 18s      remaining: 219ms
494:    total: 18.1s    remaining: 183ms
495:    total: 18.1s    remaining: 146ms
496:    total: 18.2s    remaining: 110ms
497:    total: 18.2s    remaining: 73.1ms
498:    total: 18.2s    remaining: 36.5ms
499:    total: 18.3s    remaining: 0us
```

Out[76]:
```
<catboost.core.CatBoostClassifier at 0x187129a1548>
```

In [78]:
```
print("CatBoost Accuracy:",np.round(clf.score(X_test, y_test),2))
```

```
CatBoost Accuracy: 0.94
```

Confusion Matrix

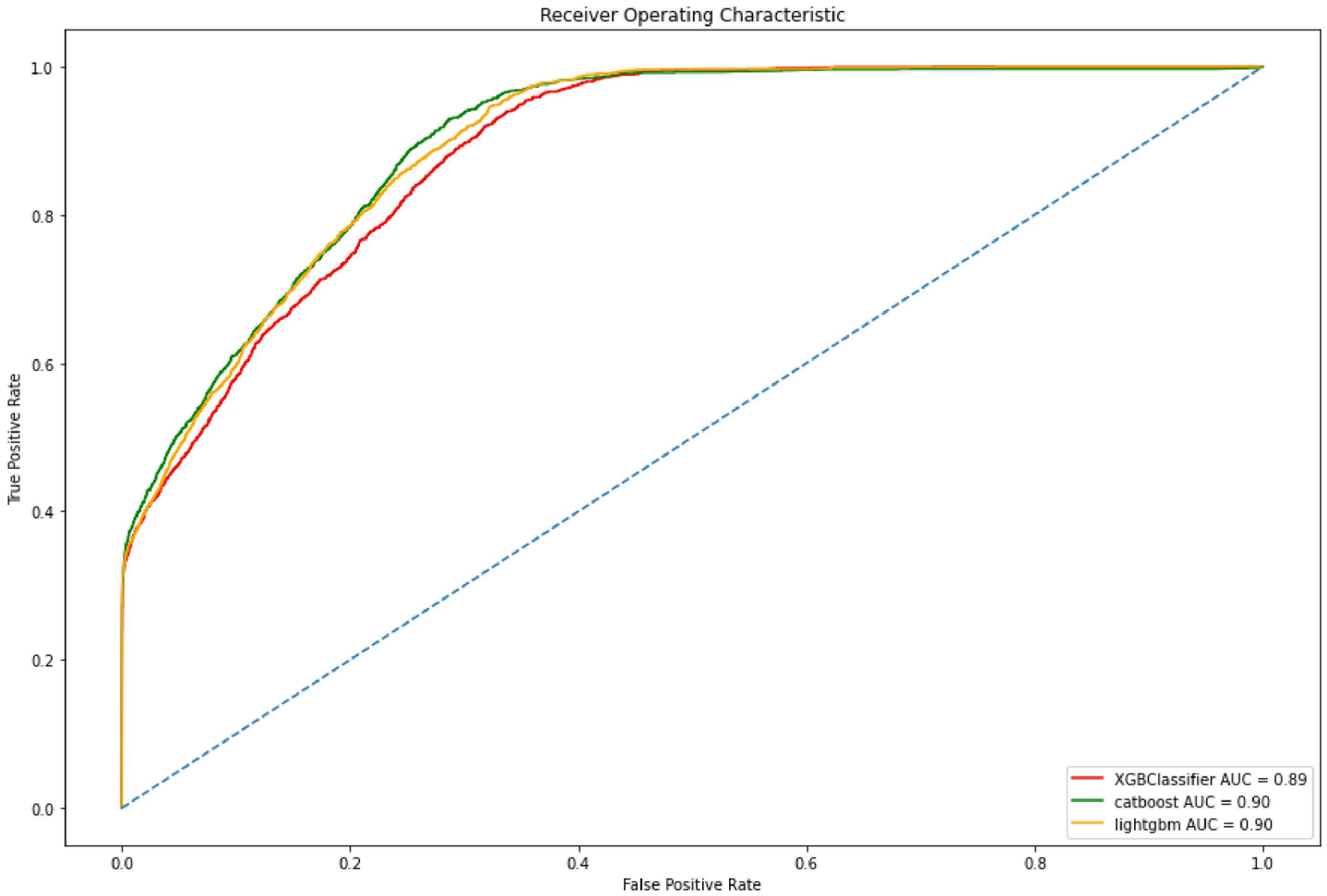
In [85]:
```
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc


fpr=[]
tpr=[]
roc_auc_ls=[]

for model in [model_xgb,model_lgb,clf]:

    if model == model_lgb:
        y_prob =  model_lgb.predict(X_test)
    else:

        y_prob = model.predict_proba(X_test)[:,1] # This will give you positive class prediction probabilities
    y_pred = np.where(y_prob > 0.5, 1, 0) # This will threshold the probabilities to give class predictions.
    confusion_matrix=metrics.confusion_matrix(y_test,y_pred)
    print("Confusion Matrix for",str(model),"\n",confusion_matrix) #Confusion Matrix shows all the rates
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_prob)
    roc_auc = auc(false_positive_rate, true_positive_rate) #Auc score for each model
    fpr.append(false_positive_rate)
    tpr.append(true_positive_rate)
    roc_auc_ls.append(roc_auc)
#Individual scores are captured in the form of lists
```

```
Confusion Matrix for XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.16, max_delta_step=0, max_depth=50,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=200, n_jobs=-1, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbose=1,
              verbosity=None)
 [[16299   314]
 [  899   575]]
Confusion Matrix for <lightgbm.basic.Booster object at 0x0000018701C33A48>
 [[16584    29]
 [  988   486]]
Confusion Matrix for <catboost.core.CatBoostClassifier object at 0x00000187129A1548>
 [[16469   144]
 [  943   531]]
```

Plotting ROC Curve

In [88]:

```
plt.figure(figsize=(15,10))
plt.title('Receiver Operating Characteristic')

#Plotting for Logistic Regression
plt.plot(fpr[0],tpr[0], color='red',label = 'XGBClassifier AUC = %0.2f' % roc_auc_ls[0])

#Plotting for Decision Tree Classifier
plt.plot(fpr[1],tpr[1], color='green',label = 'catboost AUC = %0.2f' % roc_auc_ls[1])

#Plotting for Random Forest Classifier
plt.plot(fpr[2],tpr[2], color='orange',label = 'lightgbm AUC = %0.2f' % roc_auc_ls[2])

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Conclusion:

- RobustScaler gave to most appropriate scaling reducing randomness in data
- From ROC curve we can see that CatBoost has performed more efficiently than other models

In [ ]: