In [156…

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\mohammed kayser\anaconda3\lib\site-packages (1.6.1)
Requirement already satisfied: scipy in c:\users\mohammed kayser\anaconda3\lib\site-packages (from xgboost) (1.7.1)
Requirement already satisfied: numpy in c:\users\mohammed kayser\anaconda3\lib\site-packages (from xgboost) (1.20.3)
Note: you may need to restart the kernel to use updated packages.
```

In [157…

```
import pandas as pd
import numpy as np
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import metrics
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV

import os
```

In [158…

```
df = pd.read_csv('train.csv')
df.columns
```

Out[158…

```
Index(['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', 'is_promoted'],
      dtype='object')
```

In [159…

```
df.head()
```

Out[159…

| | 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 | is_promoted |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 65438 | Sales & Marketing | region_7 | Master's & above | f | sourcing | 1 | 35 | 5.0 | 8 | 1 | 0 | 49 | 0 |
| **1** | 65141 | Operations | region_22 | Bachelor's | m | other | 1 | 30 | 5.0 | 4 | 0 | 0 | 60 | 0 |
| **2** | 7513 | Sales & Marketing | region_19 | Bachelor's | m | sourcing | 1 | 34 | 3.0 | 7 | 0 | 0 | 50 | 0 |
| **3** | 2542 | Sales & Marketing | region_23 | Bachelor's | m | other | 2 | 39 | 1.0 | 10 | 0 | 0 | 50 | 0 |
| **4** | 48945 | Technology | region_26 | Bachelor's | m | other | 1 | 45 | 3.0 | 2 | 0 | 0 | 73 | 0 |

In [160…

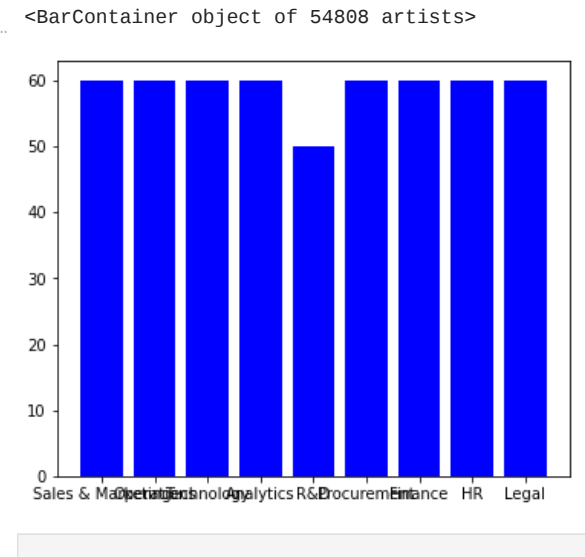
```
eldest_employees = df[['department','gender','age']]
```

In [161…

```
plt.figure(figsize=(6,5))
plt.bar(list(eldest_employees['department']),list(eldest_employees['age']),color="b")
```

Out[161…

```
<BarContainer object of 54808 artists>
```



In [162…

```
df['department'].value_counts()
```

Out[162…

```
Sales & Marketing    16840
Operations           11348
Technology            7138
Procurement           7138
Analytics             5352
Finance               2536
HR                    2418
Legal                 1039
R&D                    999
Name: department, dtype: int64
```

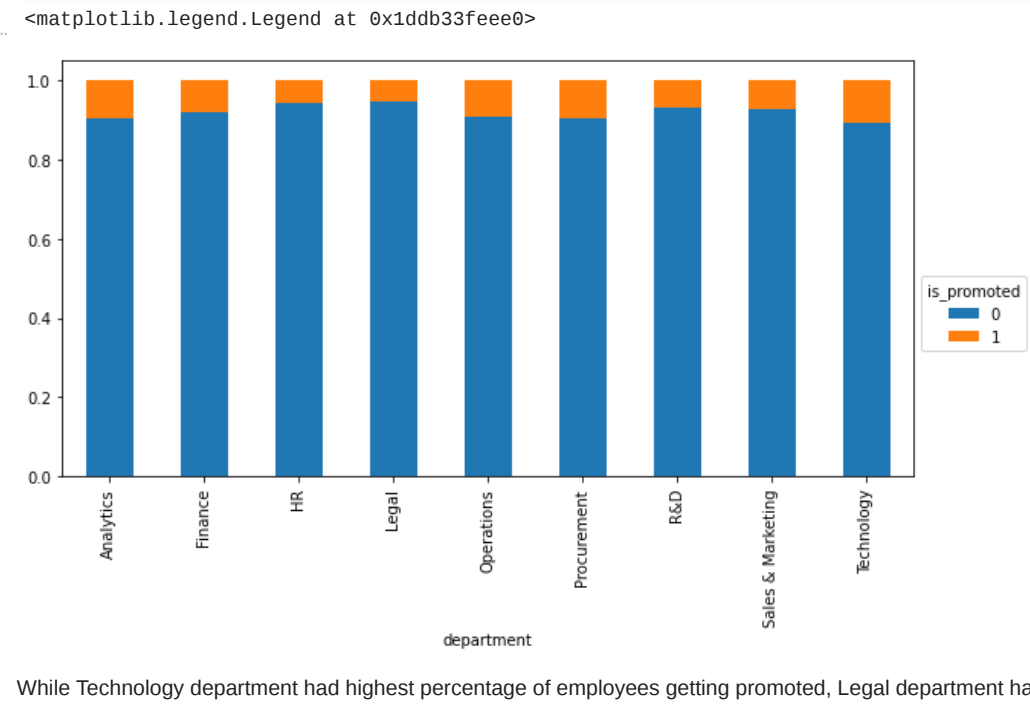
## Percentage of people who got promoted from each department

In [163…

```
plt.rcParams['figure.figsize'] = [10, 5]
ct = pd.crosstab(df.department,df.is_promoted,normalize='index')
ct.plot.bar(stacked=True)
plt.legend(title='is_promoted',bbox_to_anchor=(1,0.5))
```

Out[163…

```
<matplotlib.legend.Legend at 0x1ddb33feee0>
```



While Technology department had highest percentage of employees getting promoted, Legal department has the least number. But we don't see major differences in terms of percentages.

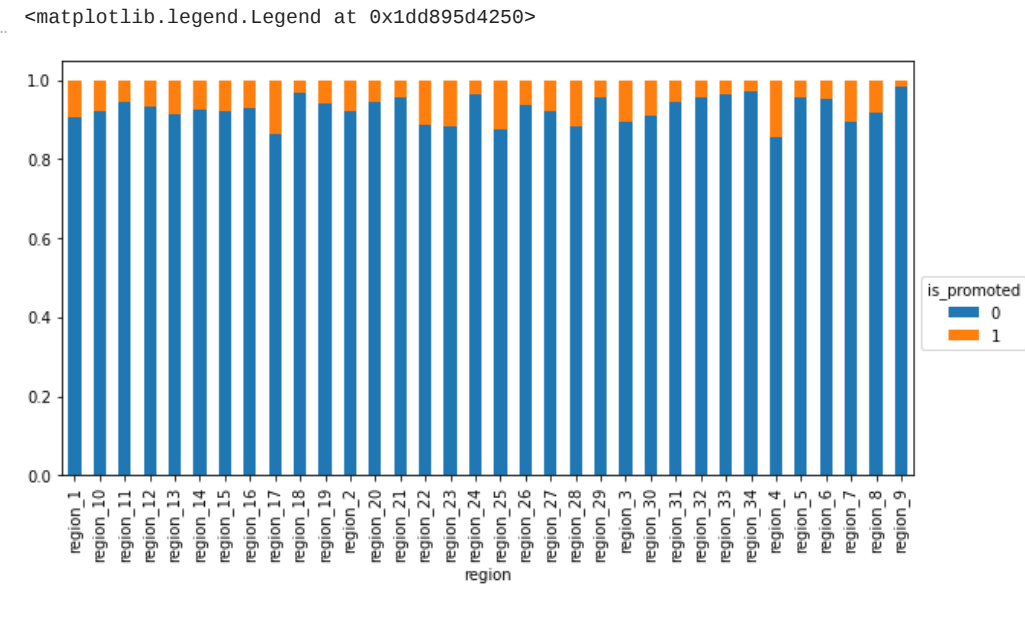
## Percentage of promotions across all the regions

In [164…

```
reg = pd.crosstab(df.region,df.is_promoted,normalize='index')
reg.plot.bar(stacked=True)
plt.legend(title='is_promoted',bbox_to_anchor=(1,0.5))
```

Out[164…

```
<matplotlib.legend.Legend at 0x1dd895d4250>
```



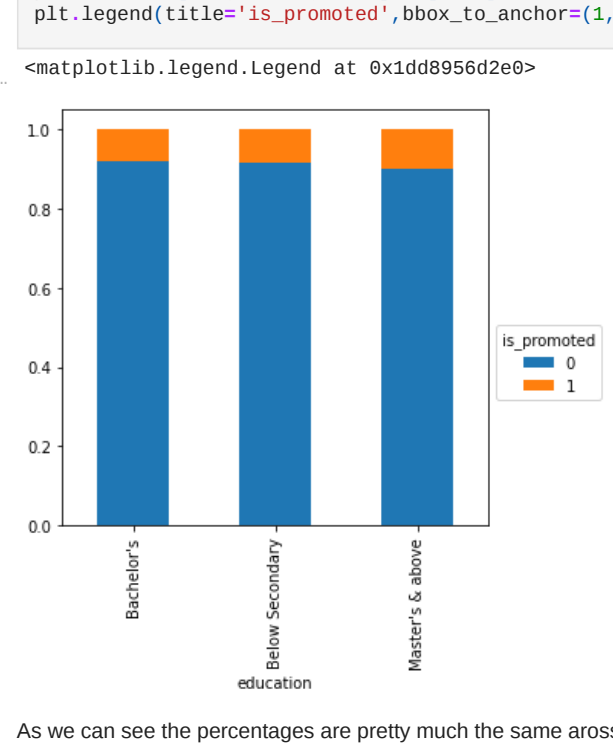
## Distribution of promotions among people with different Educational backgrounds

In [165…

```
plt.rcParams['figure.figsize'] = [5, 5]
edu = pd.crosstab(df.education,df.is_promoted,normalize='index')
edu.plot.bar(stacked=True)
plt.rcParams['figure.figsize'] = [5, 5]
plt.legend(title='is_promoted',bbox_to_anchor=(1,0.5))
```

Out[165…

```
<matplotlib.legend.Legend at 0x1dd8956d2e0>
```



As we can see the percentages are pretty much the same aross different educational backgrounds.

## Variation of promotion percentage with respect to gender

In [166…

```
pd.crosstab(df.gender,df.is_promoted,normalize='index')
```

Out[166…

| is_promoted | 0 | 1 |
|---|---|---|
| **gender** | | |
| **f** | 0.910066 | 0.089934 |
| **m** | 0.916849 | 0.083151 |

There is major differnce in percentages across genders too. Let us check if differnce arises within the departments

## Variation of promotion percentage with recruitment channel that they have come from

In [167…

```
pd.crosstab(df.recruitment_channel,df.is_promoted,normalize='index')
```

Out[167…

| is_promoted | 0 | 1 |
|---|---|---|
| **recruitment_channel** | | |
| **other** | 0.916048 | 0.083952 |
| **referred** | 0.879159 | 0.120841 |
| **sourcing** | 0.914987 | 0.085013 |

According to the data, percentage of promotions is higher among the employees who got recruited through referrals.

Let us check if there is any ratio difference across gender within the departments. For this I chose the top 3 highly populated departments.

In [168…

```
sales = df[df['department']=='Sales & Marketing']
operations = df[df['department']=='Operations']
technology = df[df['department']=='Technology']
hr = df[df['department']=='HR']
fin = df[df['department']=='Finance']
legal = df[df['department']=='Legal']
RnD = df[df['department']=='R&D']
pd.crosstab(sales.gender,sales.is_promoted,normalize='index')
```

Out[168…

| is_promoted | 0 | 1 |
|---|---|---|
| **gender** | | |
| **f** | 0.944198 | 0.055802 |
| **m** | 0.924229 | 0.075771 |

In [169…

```
pd.crosstab(operations.gender,operations.is_promoted,normalize='index')
```

Out[169…

| is_promoted | 0 | 1 |
|---|---|---|
| **gender** | | |
| **f** | 0.905495 | 0.094505 |
| **m** | 0.912907 | 0.087093 |

In [170…

```
pd.crosstab(technology.gender,technology.is_promoted,normalize='index')
```

Out[170…

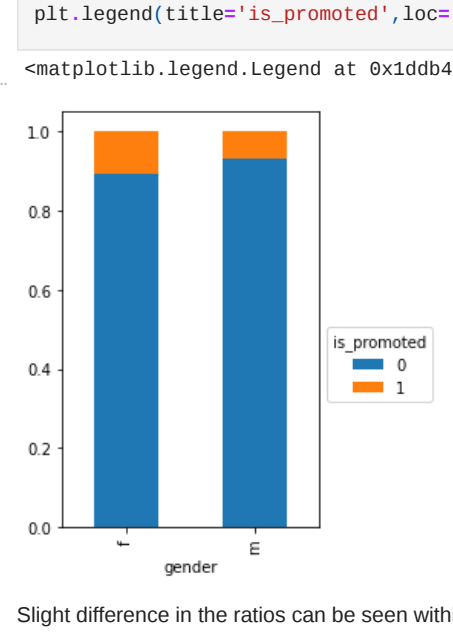
| is_promoted | 0 | 1 |
|---|---|---|
| **gender** | | |
| **f** | 0.900646 | 0.099354 |
| **m** | 0.887126 | 0.112874 |

In [171…

```
plt.rcParams['figure.figsize'] = [3, 5]
gender = pd.crosstab(RnD.gender,RnD.is_promoted,normalize='index')
gender.plot.bar(stacked=True)
plt.legend(title='is_promoted',loc='upper left',bbox_to_anchor=(1, 0.5))
```

Out[171…

```
<matplotlib.legend.Legend at 0x1ddb40eb6d0>
```



Slight difference in the ratios can be seen within the departments unlike the ratios calculated without any department barriers.

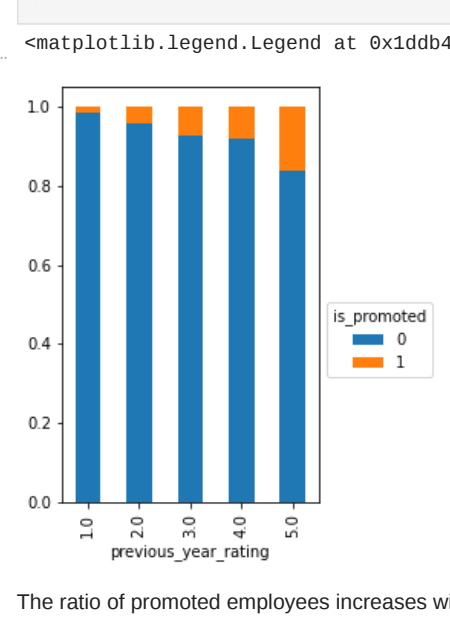
## Difference in the percentage of promoted employees with respect to previous year ratings

In [172…

```
rating = pd.crosstab(df.previous_year_rating,df.is_promoted,normalize='index')
rating.plot.bar(stacked=True)
plt.legend(title='is_promoted',loc='upper left',bbox_to_anchor=(1, 0.5))
```

Out[172…

```
<matplotlib.legend.Legend at 0x1ddb411a160>
```



The ratio of promoted employees increases with previous year rating which is quite obvious.

## Distribution of average training score

In [173…

```
bins = [30,40,50,60,70,80,90,100]
labels = ['30-40','40-50','50-60','60-70','70-80','80-90','90-100']
df['score_binned'] = pd.cut(df['avg_training_score'], bins=bins, labels=labels)
df['score_binned'].value_counts()
```

Out[173…

```
50-60     16020
40-50     11996
60-70      9973
80-90      8739
70-80      7494
90-100      579
30-40         7
Name: score_binned, dtype: int64
```

While most of the employees have score in the range of 50-60, the least score bin has very faint number,

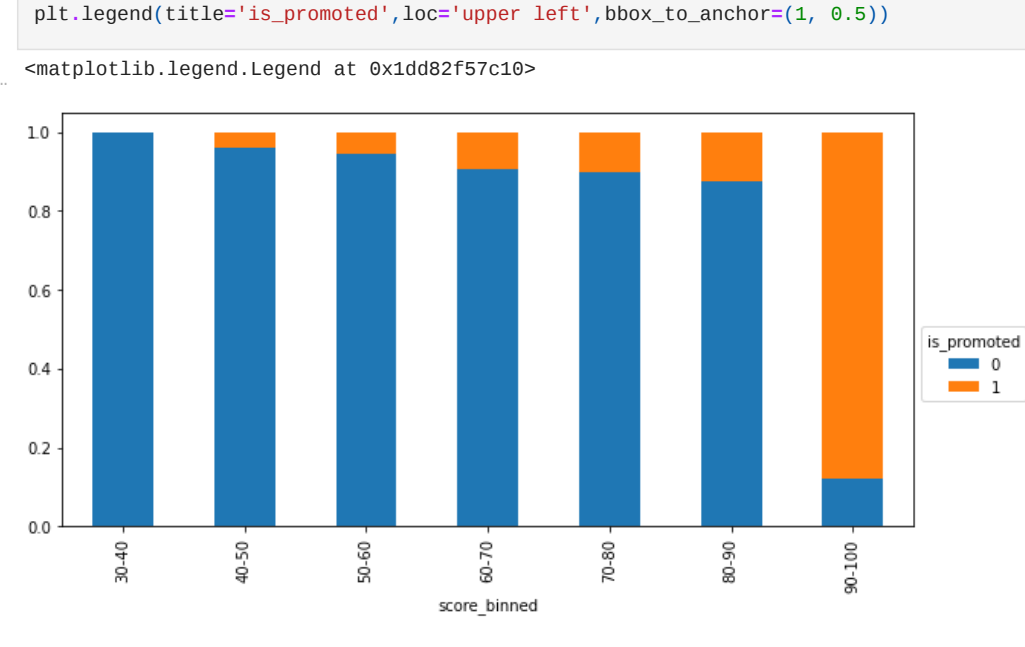
## Distribution of promoted employees ratio across different score ranges

In [174…

```
plt.rcParams['figure.figsize'] = [10, 5]
score_bin = pd.crosstab(df.score_binned,df.is_promoted,normalize='index')
score_bin.plot.bar(stacked=True)
plt.legend(title='is_promoted',loc='upper left',bbox_to_anchor=(1, 0.5))
```

Out[174…

```
<matplotlib.legend.Legend at 0x1dd82f57c10>
```



Promotions ratio increases with the score and the ratio is very high in 90-100 range which means getting promoted is highly dependent on the average score.

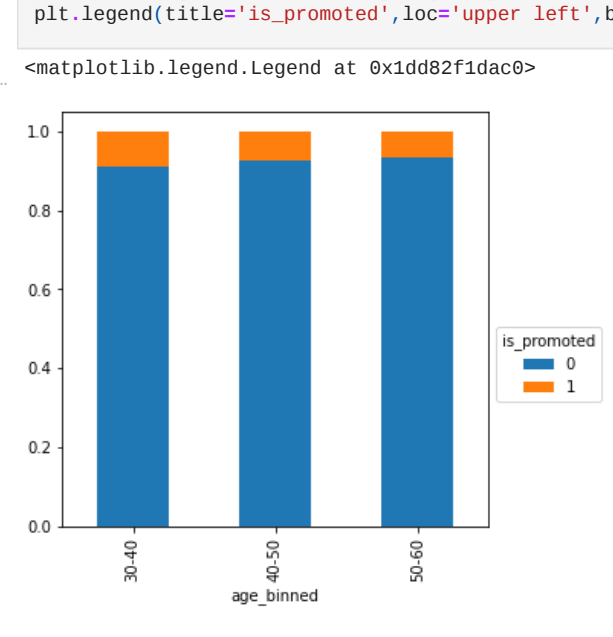
## Distribution of promotion ratios with respect to age

In [175…

```
plt.rcParams['figure.figsize'] = [5, 5]
age_bins = [20,30,40,50,60]
age_labels = ['20-30','30-40','40-50','50-60']
df['age_binned'] = pd.cut(df['age'], bins=bins, labels=labels)
df['age_binned'].value_counts()
age_bin = pd.crosstab(df.age_binned,df.is_promoted,normalize='index')
age_bin.plot.bar(stacked=True)
plt.rcParams['figure.figsize'] = [5, 5]
plt.legend(title='is_promoted',loc='upper left',bbox_to_anchor=(1, 0.5))
```

Out[175…

```
<matplotlib.legend.Legend at 0x1dd82f1dac0>
```



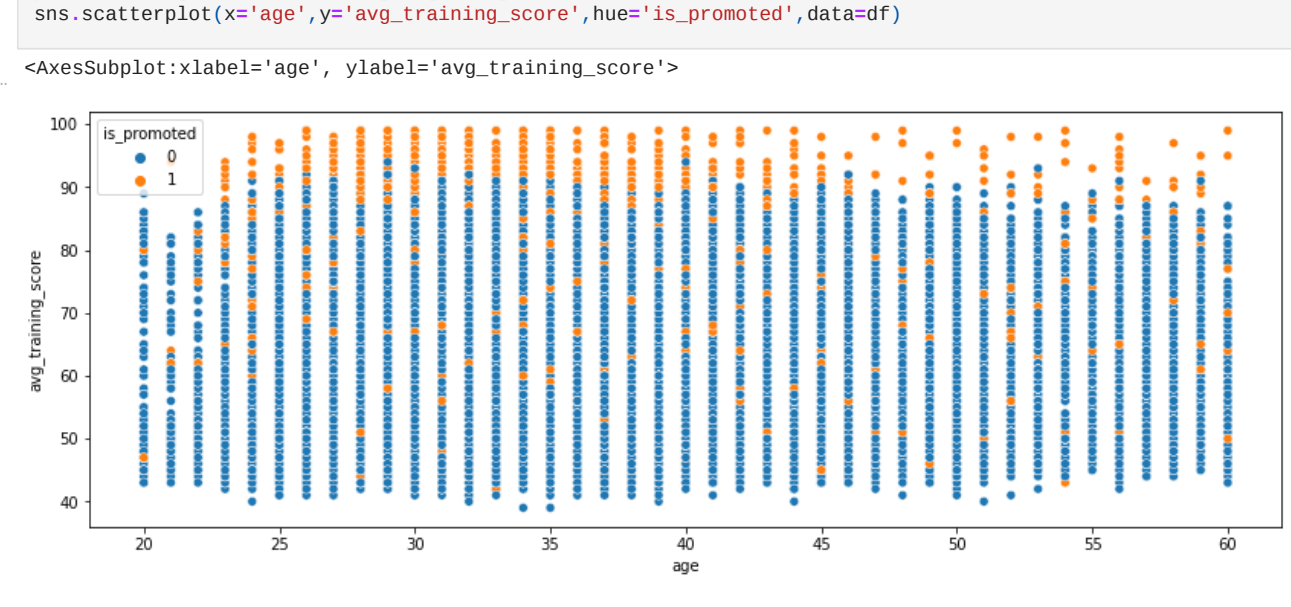
Ratio doesn't vary much with age.

In [176…

```
plt.rcParams['figure.figsize'] = [14, 5]
sns.scatterplot(x='age',y='avg_training_score',hue='is_promoted',data=df)
```

Out[176…

```
<AxesSubplot:xlabel='age', ylabel='avg_training_score'>
```



This graph reinforces the fact the promotions are majorly dependent on the score and not on age.

## Mean score of employees with different educational background

In [177…

```
df.groupby(["education"])['avg_training_score'].mean()
```

Out[177…

```
education
Bachelor's          63.422046
Below Secondary     64.925466
Master's & above    64.061240
Name: avg_training_score, dtype: float64
```

Mean training score doesn't vary with education

## Filling the missing values

In [178…

```
df.isnull().any()
```

Out[178…

```
employee_id             False
department              False
region                  False
education                True
gender                  False
recruitment_channel     False
no_of_trainings         False
age                     False
previous_year_rating     True
length_of_service       False
KPIs_met >80%           False
awards_won?             False
avg_training_score      False
is_promoted             False
score_binned            False
age_binned               True
dtype: bool
```

Fill missing values of 'previous_year_rating' with mean based on 'KPIs_met >80%' and 'education' with median based on 'department'

In [179…

```
df['previous_year_rating'] = df.groupby(["KPIs_met >80%"])["previous_year_rating"].apply(lambda x: x.fillna(x.mean()))
df["education"] = df["education"].astype('object')
df['education'] = df.groupby(["department"])["education"].apply(lambda x: x.fillna(x.value_counts().index[0]))
```

Feature engineering Normalize all the numerical features and encode all the categorical features.

In [180…

```
scaled_features = df.copy()
col_names = ['no_of_trainings', 'age','previous_year_rating','length_of_service','awards_won?','avg_training_score']
label_names = ['department','gender','recruitment_channel','region']
features = scaled_features[col_names]
scaler = preprocessing.StandardScaler().fit(features.values)
features = scaler.transform(features.values)
scaled_features = pd.get_dummies(scaled_features, columns=label_names, drop_first=True)
scaled_features[col_names] = features
scaled_features.drop(columns=['employee_id','age','education','score_binned','age_binned'],inplace=True)
```

The transformed features are fit to a Gradient Boosting Algorithm. Grid Search cross validation is used to find the best hyperparameter('n_estimators')

In [181…

```
X_train, X_test, y_train, y_test = train_test_split(
scaled_features.loc[:, scaled_features.columns != 'is_promoted'], scaled_features['is_promoted'], test_size=0.33, random_state=42)
#forest = RandomForestClassifier(n_jobs=-1, random_state=0,class_weight='balanced',n_estimators=100,bootstrap=True, max_depth=80)
forest = GradientBoostingClassifier(loss='exponential',max_features='auto')
param_grid = {
'n_estimators': [200,500,800]
}
grid_search = GridSearchCV(estimator = forest, param_grid = param_grid,cv = 3, n_jobs = -1, verbose = 2)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
grid_search.best_params_
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
```

Out[181…

```
{'n_estimators': 800}
```

According to the results of grid search, 500 is the optimal number of estimators.

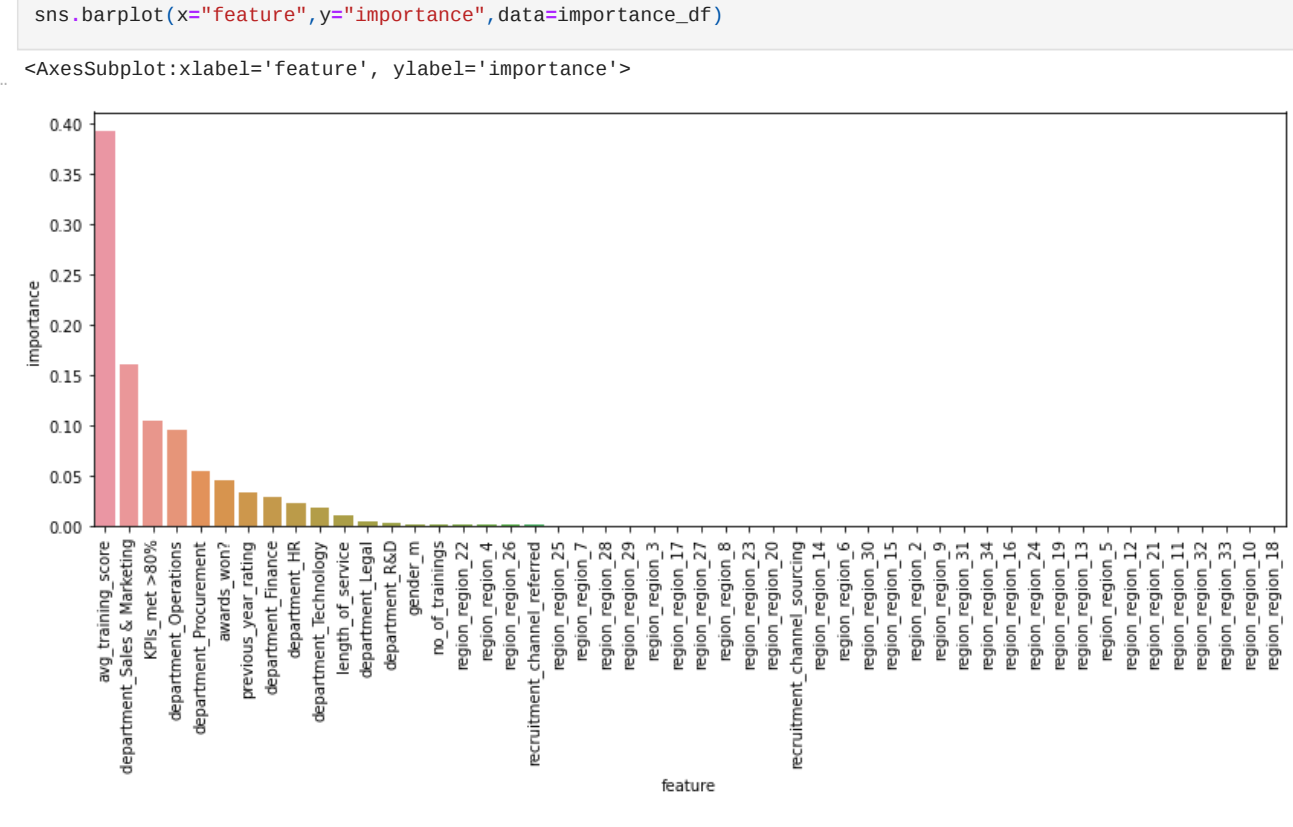
## Finding feature importances

In [182…

```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importance = {}
for i in range(len(X_train.columns)):
    feature_importance[X_train.columns[i]] = feature_importances[i]
importance_df = pd.DataFrame(list(feature_importance.items()),columns=['feature','importance'])
importance_df = importance_df.sort_values('importance',ascending=False)
plt.xticks(rotation='vertical')
plt.rcParams['figure.figsize'] = [18, 10]
sns.barplot(x="feature",y="importance",data=importance_df)
```

Out[182…

```
<AxesSubplot:xlabel='feature', ylabel='importance'>
```



The above graph shows the importance of each feature in building the model. Here also average training score takes first place.

In [183…

```
pred = grid_search.predict(X_test)
accuracy = metrics.accuracy_score(y_test, pred)
'accuracy - '+str(accuracy)
```

Out[183…

```
'accuracy - 0.9450434013379776'
```

In [184…

```
f1 = metrics.f1_score(y_test, pred)
'f1 score - '+str(f1)
```

Out[184…

```
'f1 score - 0.5136986301369862'
```

In [185…

```
recall = metrics.recall_score(y_test,pred)
'recall - '+str(recall)
```

Out[185…

```
'recall - 0.35617367706919945'
```

In [186…

```
precision = metrics.precision_score(y_test,pred)
'precision - '+str(precision)
```

Out[186…

```
'precision - 0.9210526315789473'
```

The model has a high accuracy but F1 score is less because of lesser Recall value. This means that number of False Negatives are higher which might have arised due to unbalanced classes. Oversampling or undersampling might increase F1 score.

In [187…

```
df.head()
```

Out[187…

| | 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 | is_promoted | score_binned | age_binne |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 65438 | Sales & Marketing | region_7 | Master's & above | f | sourcing | 1 | 35 | 5.0 | 8 | 1 | 0 | 49 | 0 | 40-50 | 30-4 |
| **1** | 65141 | Operations | region_22 | Bachelor's | m | other | 1 | 30 | 5.0 | 4 | 0 | 0 | 60 | 0 | 50-60 | NaN |
| **2** | 7513 | Sales & Marketing | region_19 | Bachelor's | m | sourcing | 1 | 34 | 3.0 | 7 | 0 | 0 | 50 | 0 | 40-50 | 30-4 |
| **3** | 2542 | Sales & Marketing | region_23 | Bachelor's | m | other | 2 | 39 | 1.0 | 10 | 0 | 0 | 50 | 0 | 40-50 | 30-4 |
| **4** | 48945 | Technology | region_26 | Bachelor's | m | other | 1 | 45 | 3.0 | 2 | 0 | 0 | 73 | 0 | 70-80 | 40-5 |

In [188…

```
#Employee_Id ,score_binned,age_binned is not going influence the promotion(not relevant)
df=df.drop(columns='score_binned',axis=1)
df=df.drop(columns='age_binned',axis=1)
```

In [189…

```
df.shape
```

Out[189…

```
(54808, 14)
```

In [190…

```
#Keeping Copy of datset for future reference
mainDf=df.copy()
```

In [191…

```
# Percentage of missing values for each column  in Dataset
(df.isnull().mean())*100
```

Out[191…

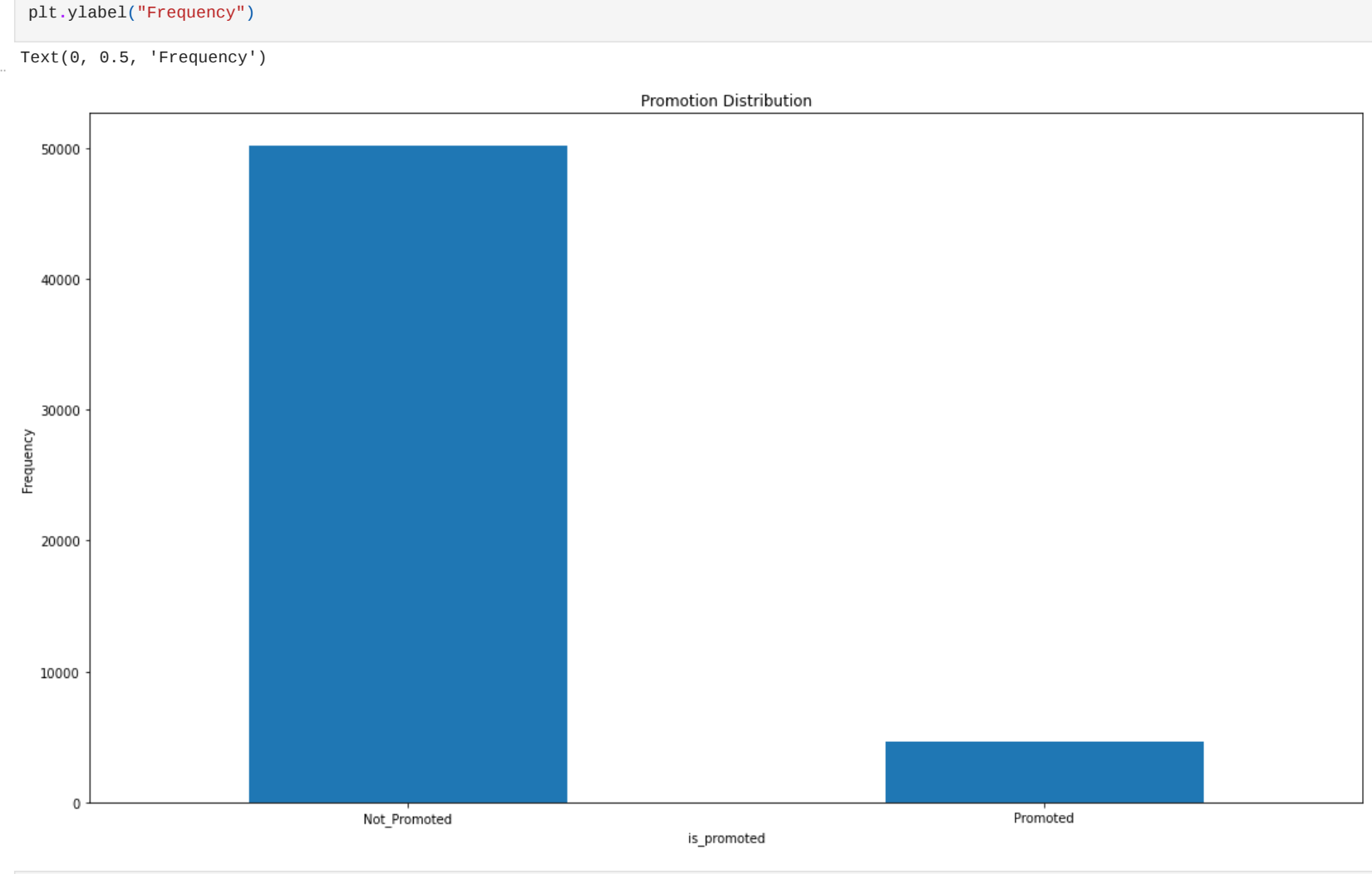
```
employee_id             0.0
department              0.0
region                  0.0
education               0.0
gender                  0.0
recruitment_channel     0.0
no_of_trainings         0.0
age                     0.0
previous_year_rating    0.0
length_of_service       0.0
KPIs_met >80%           0.0
awards_won?             0.0
avg_training_score      0.0
is_promoted             0.0
dtype: float64
```

In [192…

```
#Distribution of Target VAriabel
from matplotlib import pyplot as plt
count_classes = pd.value_counts(df['is_promoted'], sort = True)
LABELS=['Not_Promoted','Promoted',]

count_classes.plot(kind = 'bar', rot=0)

plt.title("Promotion Distribution")

plt.xticks(range(2), LABELS)

plt.xlabel("is_promoted")

plt.ylabel("Frequency")
```

Out[192…

```
Text(0, 0.5, 'Frequency')
```



In [193…

```
#PErcetage promotion in each department
df.groupby(['department'])['is_promoted'].mean()
```

Out[193…

```
department
Analytics            0.095665
Finance              0.081230
HR                   0.056245
Legal                0.051011
Operations           0.090148
Procurement          0.096386
R&D                  0.069069
Sales & Marketing    0.072031
Technology           0.107593
Name: is_promoted, dtype: float64
```

## encoding

In [194…

```
ordinal_labels=df.groupby(['department'])['is_promoted'].mean().sort_values().index
ordinal_labels
```

Out[194…

```
Index(['Legal', 'HR', 'R&D', 'Sales & Marketing', 'Finance', 'Operations',
       'Analytics', 'Procurement', 'Technology'],
      dtype='object', name='department')
```

In [195…

```
enumerate(ordinal_labels,0)
```

Out[195…

```
<enumerate at 0x1dd88ccd1c0>
```

In [196…

```
ordinal_labels2={k:i for i,k in enumerate(ordinal_labels,0)}
ordinal_labels2
```

Out[196…

```
{'Legal': 0,
 'HR': 1,
 'R&D': 2,
 'Sales & Marketing': 3,
 'Finance': 4,
 'Operations': 5,
 'Analytics': 6,
 'Procurement': 7,
 'Technology': 8}
```

In [197…

```
df['department']=df['department'].map(ordinal_labels2)
df.head()
```

Out[197…

| | 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 | is_promoted |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 65438 | 3 | region_7 | Master's & above | f | sourcing | 1 | 35 | 5.0 | 8 | 1 | 0 | 49 | 0 |
| **1** | 65141 | 5 | region_22 | Bachelor's | m | other | 1 | 30 | 5.0 | 4 | 0 | 0 | 60 | 0 |
| **2** | 7513 | 3 | region_19 | Bachelor's | m | sourcing | 1 | 34 | 3.0 | 7 | 0 | 0 | 50 | 0 |
| **3** | 2542 | 3 | region_23 | Bachelor's | m | other | 2 | 39 | 1.0 | 10 | 0 | 0 | 50 | 0 |
| **4** | 48945 | 8 | region_26 | Bachelor's | m | other | 1 | 45 | 3.0 | 2 | 0 | 0 | 73 | 0 |

In [198…

```
df['region'].value_counts().to_dict()
```

Out[198…

```
{'region_2': 12343,
 'region_22': 6428,
 'region_7': 4843,
 'region_15': 2808,
 'region_13': 2648,
 'region_26': 2260,
 'region_31': 1935,
 'region_4': 1703,
 'region_27': 1659,
 'region_16': 1465,
 'region_28': 1318,
 'region_11': 1315,
 'region_23': 1175,
 'region_29': 994,
 'region_32': 945,
 'region_19': 874,
 'region_20': 850,
 'region_14': 827,
 'region_25': 819,
 'region_17': 796,
 'region_5': 766,
 'region_6': 690,
 'region_30': 657,
 'region_8': 655,
 'region_10': 648,
 'region_1': 610,
 'region_24': 508,
 'region_12': 500,
 'region_9': 420,
 'region_21': 411,
 'region_3': 346,
 'region_34': 292,
 'region_33': 269,
 'region_18': 31}
```

In [199…

```
#Count Or Frequency Encoding
region_map=df['region'].value_counts().to_dict()
```

In [200…

```
df['region']=df['region'].map(region_map)
df.head(10)
```

Out[200…

| | 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 | is_promoted |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 65438 | 3 | 4843 | Master's & above | f | sourcing | 1 | 35 | 5.0 | 8 | 1 | 0 | 49 | 0 |
| **1** | 65141 | 5 | 6428 | Bachelor's | m | other | 1 | 30 | 5.0 | 4 | 0 | 0 | 60 | 0 |
| **2** | 7513 | 3 | 874 | Bachelor's | m | sourcing | 1 | 34 | 3.0 | 7 | 0 | 0 | 50 | 0 |
| **3** | 2542 | 3 | 1175 | Bachelor's | m | other | 2 | 39 | 1.0 | 10 | 0 | 0 | 50 | 0 |
| **4** | 48945 | 8 | 2260 | Bachelor's | m | other | 1 | 45 | 3.0 | 2 | 0 | 0 | 73 | 0 |
| **5** | 58896 | 6 | 12343 | Bachelor's | m | sourcing | 2 | 31 | 3.0 | 7 | 0 | 0 | 85 | 0 |
| **6** | 20379 | 5 | 850 | Bachelor's | f | other | 1 | 31 | 3.0 | 5 | 0 | 0 | 59 | 0 |
| **7** | 16290 | 5 | 292 | Master's & above | m | sourcing | 1 | 33 | 3.0 | 6 | 0 | 0 | 63 | 0 |
| **8** | 73202 | 6 | 850 | Bachelor's | m | other | 1 | 28 | 4.0 | 5 | 0 | 0 | 83 | 0 |
| **9** | 28911 | 3 | 610 | Master's & above | m | sourcing | 1 | 32 | 5.0 | 5 | 1 | 0 | 54 | 0 |

In [201…

```
df['education']=df['education'].replace({"Master's & above":3,"Bachelor's":2,'Below Secondary':1})
```

In [202…

```
df['gender']=df['gender'].replace({"m":1,"f":0})
```

In [203…

```
df['recruitment_channel']=df['recruitment_channel'].replace({"other":0,"sourcing":1})
```

In [204…

```
df.head()
```

Out[204…

| | 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 | is_promoted |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 65438 | 3 | 4843 | 3 | 0 | 1 | 1 | 35 | 5.0 | 8 | 1 | 0 | 49 | 0 |
| **1** | 65141 | 5 | 6428 | 2 | 1 | 0 | 1 | 30 | 5.0 | 4 | 0 | 0 | 60 | 0 |
| **2** | 7513 | 3 | 874 | 2 | 1 | 1 | 1 | 34 | 3.0 | 7 | 0 | 0 | 50 | 0 |
| **3** | 2542 | 3 | 1175 | 2 | 1 | 0 | 2 | 39 | 1.0 | 10 | 0 | 0 | 50 | 0 |
| **4** | 48945 | 8 | 2260 | 2 | 1 | 0 | 1 | 45 | 3.0 | 2 | 0 | 0 | 73 | 0 |

In [205…

```
df.shape
```

Out[205…

```
(54808, 14)
```

In [206…

```
#Avoidng Dumy Trap for applying Drop First
df=pd.get_dummies(df,drop_first=True)
```

In [207…

```
df.head()
```

Out[207…

| | employee_id | department | region | education | gender | no_of_trainings | age | previous_year_rating | length_of_service | KPIs_met >80% | awards_won? | avg_training_score | is_promoted | recruitment_channel_1 | recruitment_channel_referr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 65438 | 3 | 4843 | 3 | 0 | 1 | 35 | 5.0 | 8 | 1 | 0 | 49 | 0 | 1 | |
| **1** | 65141 | 5 | 6428 | 2 | 1 | 1 | 30 | 5.0 | 4 | 0 | 0 | 60 | 0 | 0 | |
| **2** | 7513 | 3 | 874 | 2 | 1 | 1 | 34 | 3.0 | 7 | 0 | 0 | 50 | 0 | 1 | |
| **3** | 2542 | 3 | 1175 | 2 | 1 | 2 | 39 | 1.0 | 10 | 0 | 0 | 50 | 0 | 0 | |
| **4** | 48945 | 8 | 2260 | 2 | 1 | 1 | 45 | 3.0 | 2 | 0 | 0 | 73 | 0 | 0 | |

In [208…

```
#As per looking into distribution for frequency in previous_year_rating var 3.0 , 5.0, 4.0 is frequently given to employees.Choosing one of these using mode could create biasness.
# So Adding A separate col for weightage given to NAN values would be help full

df['previous_year_rating_Var']=np.where(df['previous_year_rating'].isnull(),1,0)
```

In [209…

```
df.columns
```

Out[209…

```
Index(['employee_id', 'department', 'region', 'education', 'gender',
       'no_of_trainings', 'age', 'previous_year_rating', 'length_of_service',
       'KPIs_met >80%', 'awards_won?', 'avg_training_score', 'is_promoted',
       'recruitment_channel_1', 'recruitment_channel_referred',
       'previous_year_rating_Var'],
      dtype='object')
```

In [210…

```
df=df[['department', 'region', 'education', 'gender', 'no_of_trainings', 'age',
       'previous_year_rating','previous_year_rating_Var', 'length_of_service', 'KPIs_met >80%',
       'awards_won?', 'avg_training_score','recruitment_channel_referred','recruitment_channel_1','is_promoted']]
```

In [211…

```
df.head()
```

Out[211…

| | department | region | education | gender | no_of_trainings | age | previous_year_rating | previous_year_rating_Var | length_of_service | KPIs_met >80% | awards_won? | avg_training_score | recruitment_channel_referred | recruitment_channel_1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 4843 | 3 | 0 | 1 | 35 | 5.0 | 0 | 8 | 1 | 0 | 49 | 0 | 1 |
| **1** | 5 | 6428 | 2 | 1 | 1 | 30 | 5.0 | 0 | 4 | 0 | 0 | 60 | 0 | 0 |
| **2** | 3 | 874 | 2 | 1 | 1 | 34 | 3.0 | 0 | 7 | 0 | 0 | 50 | 0 | 1 |
| **3** | 3 | 1175 | 2 | 1 | 2 | 39 | 1.0 | 0 | 10 | 0 | 0 | 50 | 0 | 0 |
| **4** | 8 | 2260 | 2 | 1 | 1 | 45 | 3.0 | 0 | 2 | 0 | 0 | 73 | 0 | 0 |

In [212…

```
df.shape
```

Out[212…

```
(54808, 15)
```

## Model Creation

## Splitting the data into Training and Testing sample

In [213…

```
TargetVariable='is_promoted'
Predictors=[i for i in df.columns if i not in TargetVariable]
print(Predictors)
X=df[Predictors].values
y=df[TargetVariable].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=428)
```

```
['department', 'region', 'education', 'gender', 'no_of_trainings', 'age', 'previous_year_rating', 'previous_year_rating_Var', 'length_of_service', 'KPIs_met >80%', 'awards_won?',
'avg_training_score', 'recruitment_channel_referred', 'recruitment_channel_1']
```

In [214…

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(38365, 14) (16443, 14) (38365,) (16443,)
```

## Standardization/Normalization of data

In [215…

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Choose either standardization or Normalization
# On this data Min Max Normalization produced better results

# Choose between standardization and MinMAx normalization
#PredictorScaler=StandardScaler()
PredictorScaler=MinMaxScaler()

# Storing the fit object for later reference
PredictorScalerFit=PredictorScaler.fit(X)

# Generating the standardized values of X
X=PredictorScalerFit.transform(X)

# Split the data into training and testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [216…

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(38365, 14) (16443, 14) (38365,) (16443,)
```

## Logistic Regression

In [217…

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
# choose parameter Penalty='l1' or C=1
# choose different values for solver 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'
clf = LogisticRegression(C=1,penalty='l2', solver='newton-cg')

# Printing all the parameters of logistic regression
# print(clf)

# Creating the model on Training Data
LOG=clf.fit(X_train,y_train)
prediction_log=LOG.predict(X_test)

# Measuring accuracy on Testing Data
from sklearn import metrics
print(metrics.classification_report(y_test, prediction_log))
print(metrics.confusion_matrix(y_test, prediction_log))

# Printing the Overall Accuracy of the model
F1_Score=metrics.f1_score(y_test, prediction_log, average='weighted')
print('Accuracy of the model on Testing Sample Data:', round(F1_Score,2))

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(LOG, X , y, cv=10, scoring='f1_weighted')
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

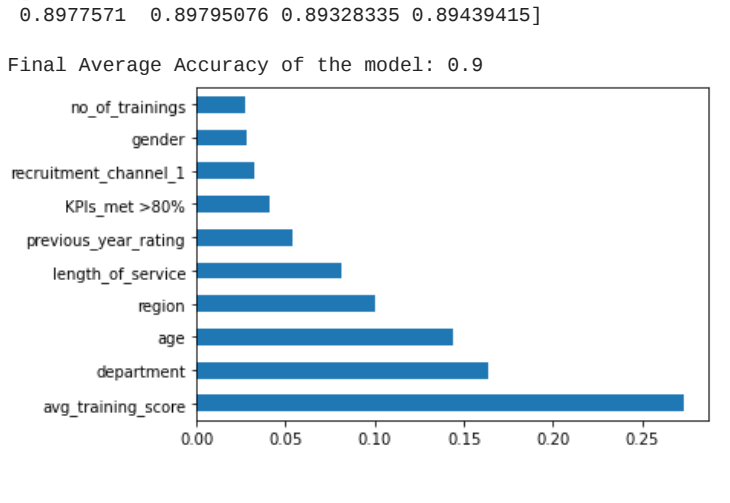
```
              precision    recall  f1-score   support

           0       0.92      0.99      0.96     15110
           1       0.57      0.09      0.15      1333

    accuracy                           0.92     16443
   macro avg       0.75      0.54      0.55     16443
weighted avg       0.90      0.92      0.89     16443

[[15025    85]
 [ 1219   114]]
Accuracy of the model on Testing Sample Data: 0.89

Accuracy values for 10-fold Cross Validation:
 [0.89036144 0.88526115 0.88386885 0.88783788 0.8844949  0.8869337
 0.88926259 0.88625027 0.89016518 0.88858717]

Final Average Accuracy of the model: 0.89
```

## Decision Tree Classifier

In [218…

```
#Decision Trees
from sklearn import tree
#choose from different tunable hyper parameters
clf = tree.DecisionTreeClassifier()

# Printing all the parameters of Decision Trees
print(clf)

# Creating the model on Training Data
DTree=clf.fit(X_train,y_train)
prediction=DTree.predict(X_test)

# Measuring accuracy on Testing Data
from sklearn import metrics
print(metrics.classification_report(y_test, prediction))
print(metrics.confusion_matrix(y_test, prediction))

# Printing the Overall Accuracy of the model
F1_Score=metrics.f1_score(y_test, prediction, average='weighted')
print('Accuracy of the model on Testing Sample Data:', round(F1_Score,2))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(DTree.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(DTree, X , y, cv=10, scoring='f1_weighted')
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
DecisionTreeClassifier()
              precision    recall  f1-score   support

           0       0.95      0.94      0.94     15110
           1       0.38      0.44      0.40      1333

    accuracy                           0.90     16443
   macro avg       0.66      0.69      0.67     16443
weighted avg       0.90      0.90      0.90     16443

[[14145   965]
 [  750   583]]
Accuracy of the model on Testing Sample Data: 0.9

Accuracy values for 10-fold Cross Validation:
 [0.89847065 0.89593016 0.89057039 0.8987841  0.89670184 0.89901917
 0.8977571  0.89795076 0.89328335 0.89439415]

Final Average Accuracy of the model: 0.9
```



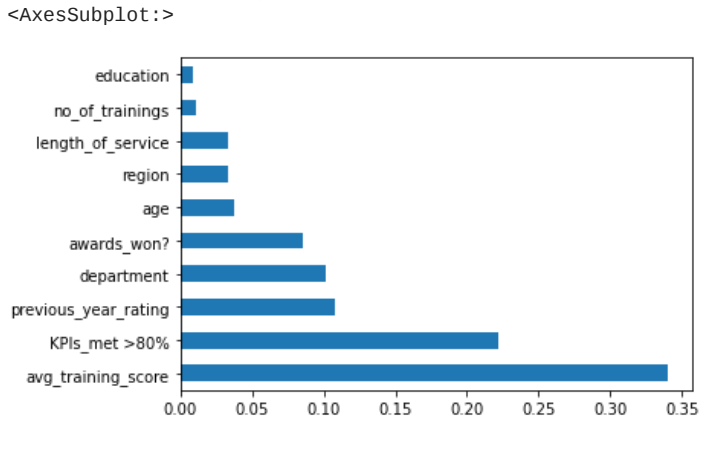
## Random Forest Classifier

In [219…

```
# Random Forest (Bagging of multiple Decision Trees)
from sklearn.ensemble import RandomForestClassifier
# Choose various values of max_depth, n_estimators and criterion for tuning the model
random = RandomForestClassifier(max_depth=10, n_estimators=100,criterion='entropy')


# Printing all the parameters of Random Forest
print(random)

# Creating the model on Training Data
RF=random.fit(X_train,y_train)
prediction_RF=RF.predict(X_test)

# Measuring accuracy on Testing Data
from sklearn import metrics
print(metrics.classification_report(y_test, prediction_RF))
print(metrics.confusion_matrix(y_test, prediction_RF))

# Printing the Overall Accuracy of the model
F1_Score=metrics.f1_score(y_test, prediction_RF, average='weighted')
print('Accuracy of the model on Testing Sample Data:', round(F1_Score,2))

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RF, X , y, cv=10, scoring='f1_weighted')
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))


# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(RF.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')
```

```
RandomForestClassifier(criterion='entropy', max_depth=10)
              precision    recall  f1-score   support

           0       0.93      1.00      0.96     15110
           1       0.94      0.16      0.27      1333

    accuracy                           0.93     16443
   macro avg       0.93      0.58      0.62     16443
weighted avg       0.93      0.93      0.91     16443

[[15096    14]
 [ 1125   208]]
Accuracy of the model on Testing Sample Data: 0.91

Accuracy values for 10-fold Cross Validation:
 [0.90461712 0.90391273 0.89377481 0.90512126 0.89677411 0.9011536
 0.90531769 0.90204051 0.90273471 0.90229311]

Final Average Accuracy of the model: 0.9
```

Out[219…

```
<AxesSubplot:>
```