End-to-End Machine Learning Example

August 29, 2022

0.1 Build a Machine Learning Classification model to identify potential employees who deserve a promotion at the company.

0.1.1 The main objective of the excercise is to predict the emplyee promotion possibility/potential for the next year based on the employee data.

Identify who is a potential candidate for promotion based on past traning history and perfromance score ?

1 Building the Test and Development Environment -Step1

[129]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from imblearn.over_sampling import SMOTE
from prettytable import PrettyTable
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import time
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

2 Getting the Dataset -Step2

[10]:
```
data=pd.read_csv(r'C:\Users\chala\Desktop\OPM\employee_promotion.csv')
```

[11]:
```
data.head()
```

[11]:
```
   employee_id         department     region         education gender  \
0        65438  Sales & Marketing   region_7  Master's & above      f
1        65141         Operations  region_22        Bachelor's      m
2         7513  Sales & Marketing  region_19        Bachelor's      m
3         2542  Sales & Marketing  region_23        Bachelor's      m
4        48945         Technology  region_26        Bachelor's      m

  recruitment_channel  no_of_trainings  age  previous_year_rating  \
0            sourcing                1   35                   5.0
1               other                1   30                   5.0
2            sourcing                1   34                   3.0
3               other                2   39                   1.0
4               other                1   45                   3.0

   length_of_service  awards_won  avg_training_score  is_promoted
0                  8           0                49.0            0
1                  4           0                60.0            0
2                  7           0                50.0            0
3                 10           0                50.0            0
4                  2           0                73.0            0
```

[12]:
```
data.tail()
```

[12]:
```
       employee_id         department     region         education gender  \
54803         3030         Technology  region_14        Bachelor's      m
54804        74592         Operations  region_27  Master's & above      f
54805        13918          Analytics   region_1        Bachelor's      m
54806        13614  Sales & Marketing   region_9               NaN      m
54807        51526                 HR  region_22        Bachelor's      m

      recruitment_channel  no_of_trainings  age  previous_year_rating  \
54803            sourcing                1   48                   3.0
54804               other                1   37                   2.0
54805               other                1   27                   5.0
54806            sourcing                1   29                   1.0
54807               other                1   27                   1.0

       length_of_service  awards_won  avg_training_score  is_promoted
54803                 17           0                78.0            0
54804                  6           0                56.0            0
54805                  3           0                79.0            0
```

```
54806                    2            0                  NaN             0
54807                    5            0                 49.0             0
```

[13]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54808 entries, 0 to 54807
Data columns (total 13 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  awards_won            54808 non-null  int64
 11  avg_training_score    52248 non-null  float64
 12  is_promoted           54808 non-null  int64
dtypes: float64(2), int64(6), object(5)
memory usage: 5.4+ MB
```

[14]:
```
data.columns
```

[14]:
```
Index(['employee_id', 'department', 'region', 'education', 'gender',
       'recruitment_channel', 'no_of_trainings', 'age', 'previous_year_rating',
       'length_of_service', 'awards_won', 'avg_training_score', 'is_promoted'],
      dtype='object')
```

[15]:
```
print('length of the data is' , len(data))
```

```
length of the data is 54808
```

[16]:
```
data.shape
```

[16]:
```
(54808, 13)
```

[17]:
```
#Checking  Null values/missing values
np.sum(data.isnull().any (axis=1))
```

[17]:
```
8428
```

3 Data Exploration -Step3

[18]:
```
# Count of missing values in each column
data.isnull().sum()
```

[18]:
```
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
awards_won                 0
avg_training_score      2560
is_promoted                0
dtype: int64
```

[19]:
```
#Rows and Columns
print('Count of Columns in the data is:  ', len(data.columns))
```

```
Count of Columns in the data is:   13
```

[20]:
```
# Data Description
print('Count of rows in the data is:', len(data))
```

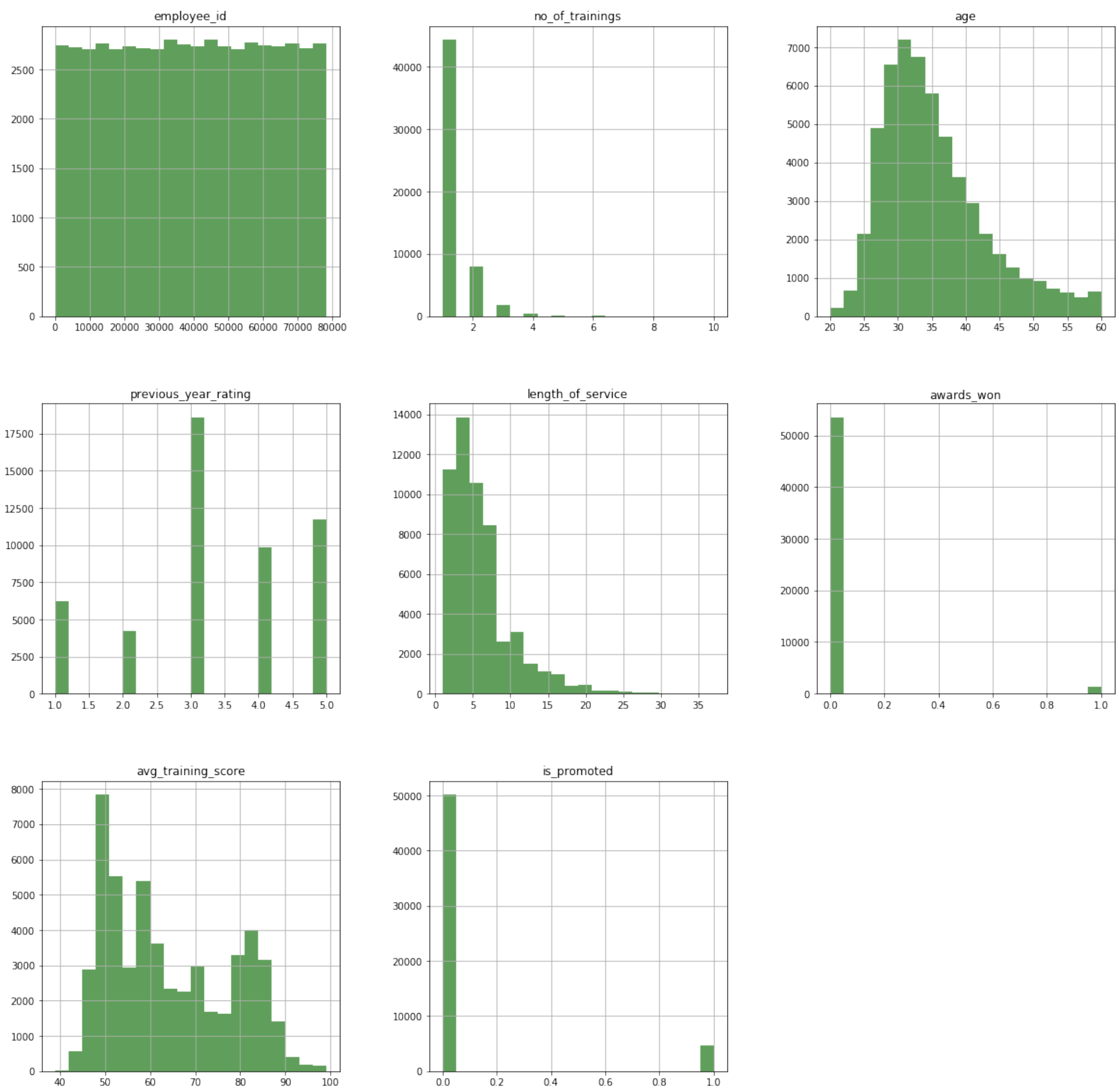
```
Count of rows in the data is: 54808
```

[21]:
```
data.describe()
```

[21]:
```
        employee_id  no_of_trainings           age  previous_year_rating  \
count  54808.000000     54808.000000  54808.000000          50684.000000
mean   39195.830627         1.253011     34.803915              3.329256
std    22586.581449         0.609264      7.660169              1.259993
min        1.000000         1.000000     20.000000              1.000000
25%    19669.750000         1.000000     29.000000              3.000000
50%    39225.500000         1.000000     33.000000              3.000000
75%    58730.500000         1.000000     39.000000              4.000000
max    78298.000000        10.000000     60.000000              5.000000

       length_of_service    awards_won  avg_training_score   is_promoted
count       54808.000000  54808.000000        52248.000000  54808.000000
mean            5.865512      0.023172           63.712238      0.085170
std             4.265094      0.150450           13.521910      0.279137
min             1.000000      0.000000           39.000000      0.000000
25%             3.000000      0.000000           51.000000      0.000000
```

50%	5.000000	0.000000	60.000000	0.000000
75%	7.000000	0.000000	77.000000	0.000000
max	37.000000	1.000000	99.000000	1.000000

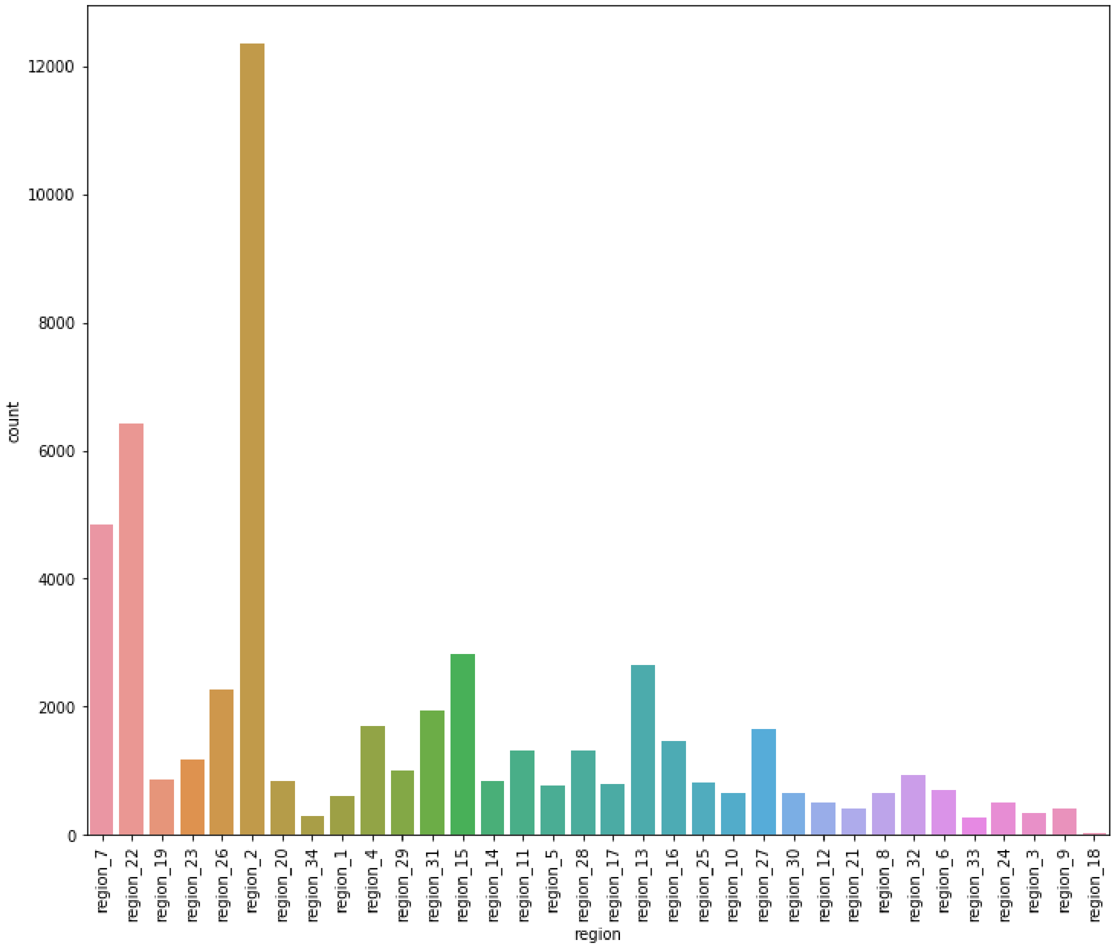
[22]:
```python
# Univeriate Analysis
# Numeric features distribution

data.hist(figsize=(20,20),bins = 20, color="#107009AA")
plt.title("Numeric Features Distribution")
plt.show()
```



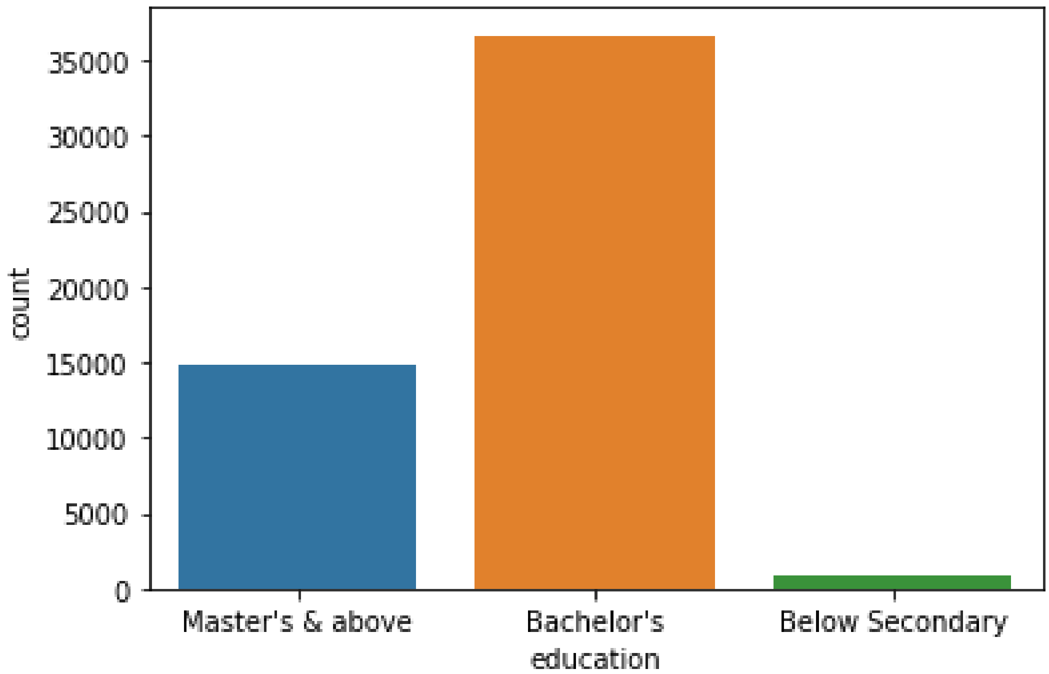
[25]:
```python
# Region Distribution
plt.figure(figsize=(12,10))
sns.countplot(data.region)
```

```
plt.xticks(rotation=90)
```

[25]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]),
 <a list of 34 Text xticklabel objects>)
```

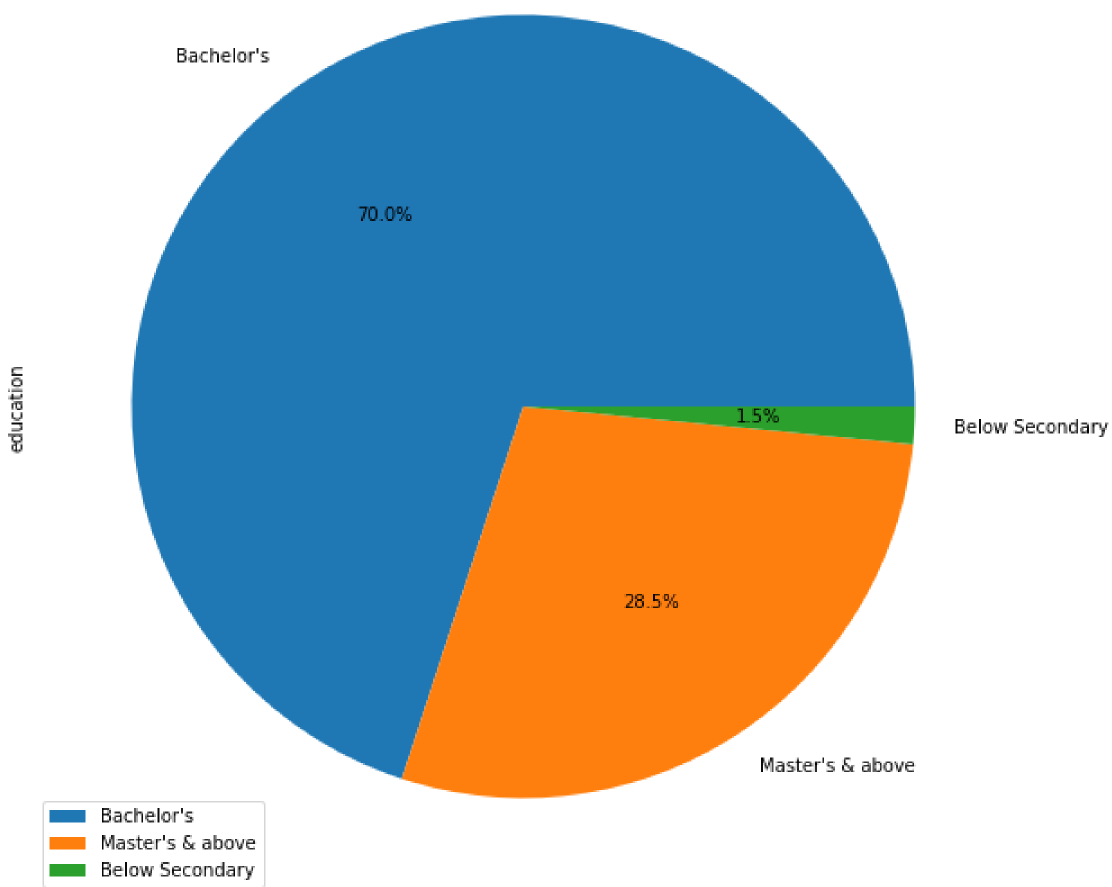


[26]:
```
# Education
sns.countplot(data= data, x = "education")
plt.show()
```



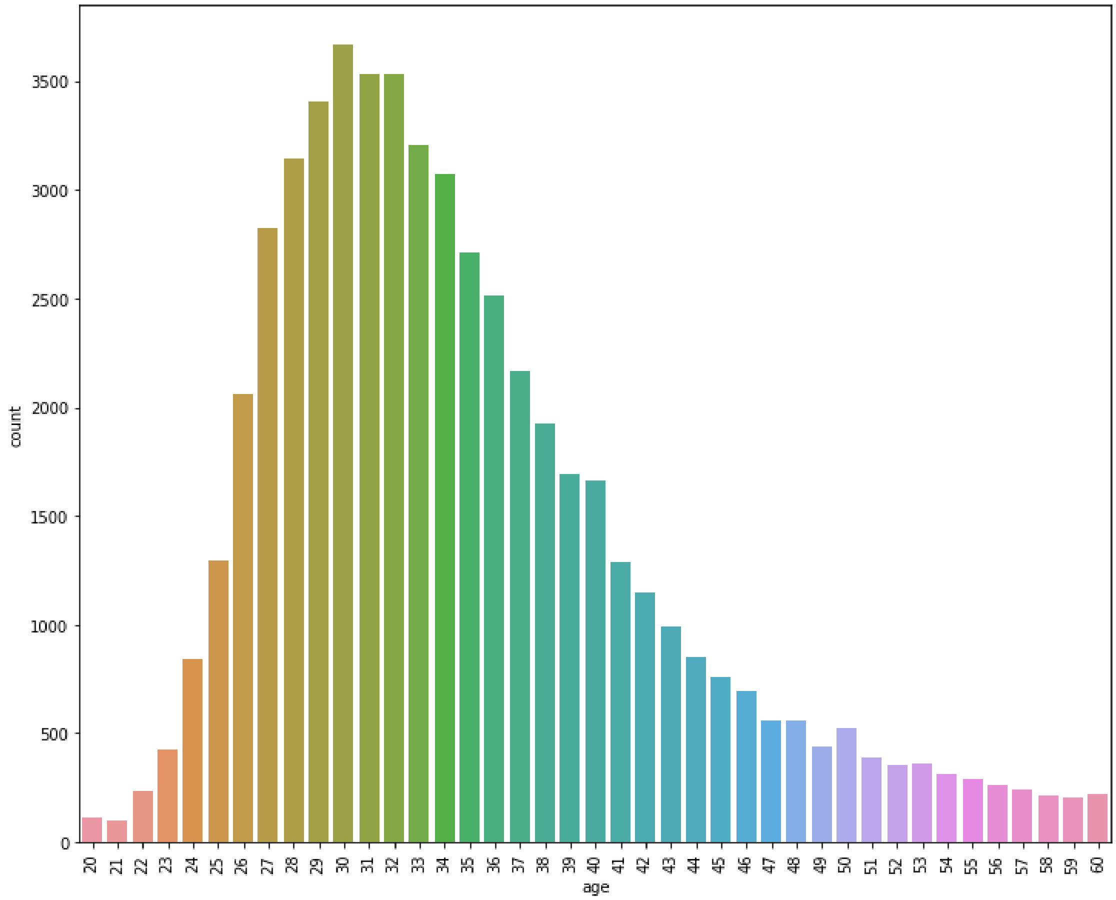
[27]:
```
data["education"].value_counts().head(7).plot(kind = 'pie', autopct='%1.1f%%',␣
↪figsize=(10, 10), startangle=0).legend()
```

[27]: <matplotlib.legend.Legend at 0x223a6c91c50>



[34]:
```
# Age distribution

plt.figure(figsize=(12,10))
sns.countplot(data.age)
plt.xticks(rotation=90)
```

[34]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40]), <a list of 41 Text xticklabel objects>)
```



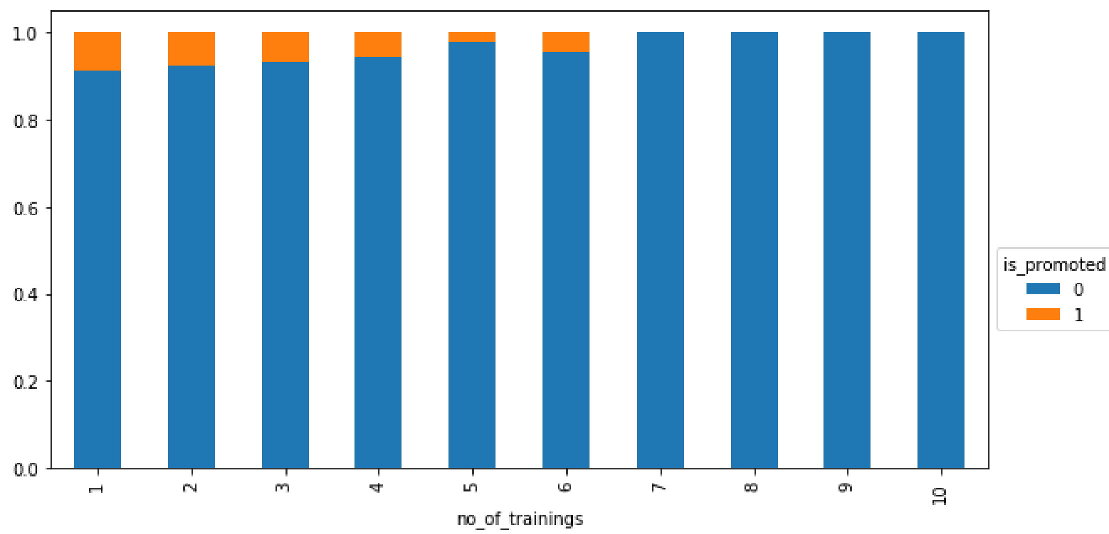
[45]:
```
data["is_promoted"].value_counts()
```

[45]:
```
0    50140
1     4668
Name: is_promoted, dtype: int64
```

4 We can clearly see that, the data is not balanced. The promoted employees are only 4668 and not promoted employees are 50140. 91% and 9% ratio is very unbalanced.

[46]:
```
plt.rcParams['figure.figsize'] = [10, 5]
score_bin = pd.crosstab(data.no_of_trainings,data.is_promoted,normalize='index')
score_bin.plot.bar(stacked=True)
plt.legend(title='is_promoted',loc='upper left',bbox_to_anchor=(1, 0.5))
```

[46]:
```
<matplotlib.legend.Legend at 0x223a7c19f60>
```



5 Feature Engineering: Step4

[51]:
```
#Checking categorical type columns in the data
data.select_dtypes(include='object')
```

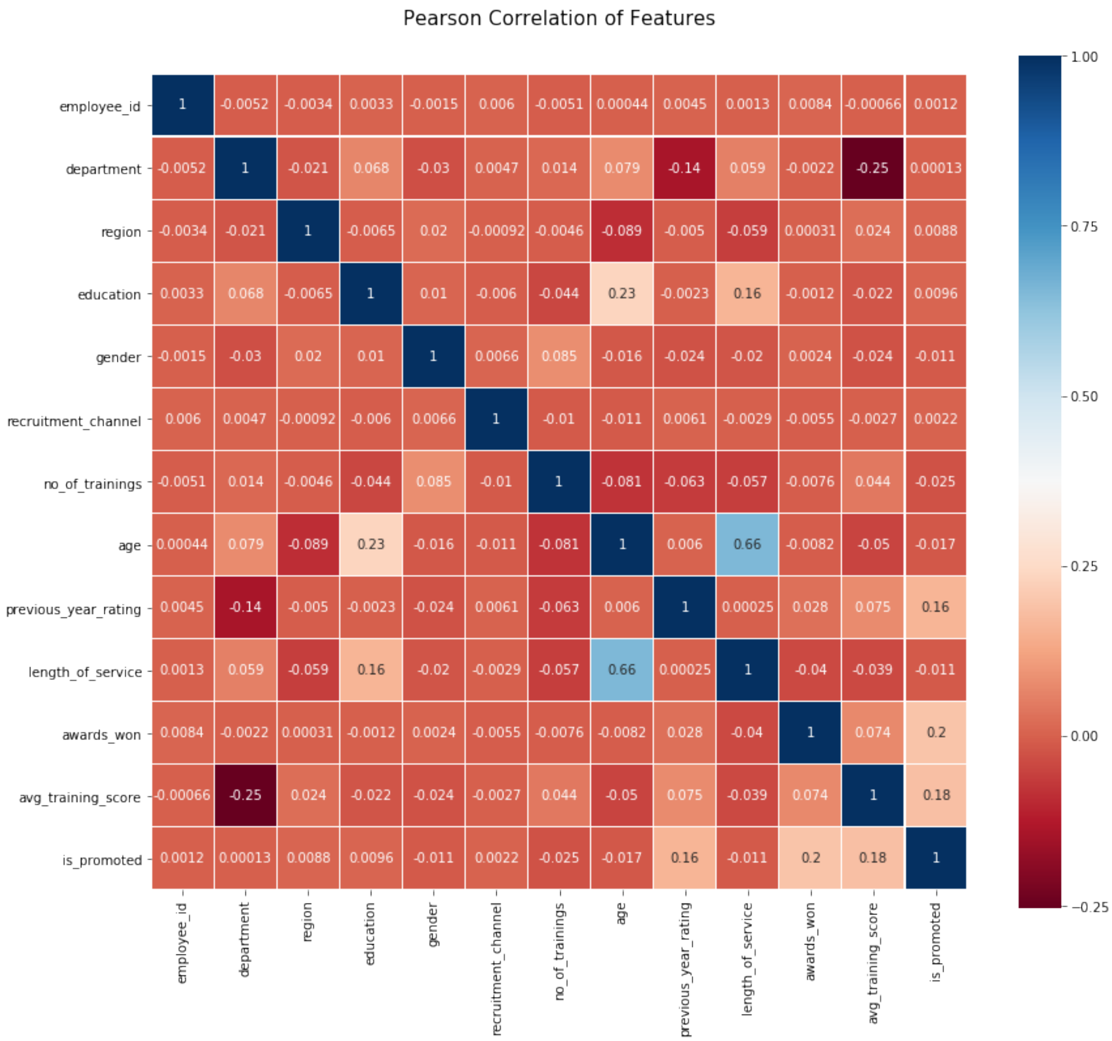
[51]:
```
                department     region         education gender  \
0        Sales & Marketing   region_7  Master's & above      f
1               Operations  region_22        Bachelor's      m
2        Sales & Marketing  region_19        Bachelor's      m
3        Sales & Marketing  region_23        Bachelor's      m
4               Technology  region_26        Bachelor's      m
…                      …          …                 …    …
54803           Technology  region_14        Bachelor's      m
54804           Operations  region_27  Master's & above      f
54805            Analytics   region_1        Bachelor's      m
54806    Sales & Marketing   region_9               NaN      m
54807                   HR  region_22        Bachelor's      m

      recruitment_channel
0                sourcing
1                   other
2                sourcing
3                   other
4                   other
…                     …
54803            sourcing
54804               other
54805               other
```

```
54806            sourcing
54807               other

[54808 rows x 5 columns]
```

[52]:
```
#Encoding these categorical features into numeric type

pro= preprocessing.LabelEncoder()
encpro=pro.fit_transform(data['department'])
data['department'] = encpro

pro= preprocessing.LabelEncoder()
encpro=pro.fit_transform(data['region'])
data['region'] = encpro

pro= preprocessing.LabelEncoder()
encpro=pro.fit_transform(data['education'].astype(str))
data['education'] = encpro

pro= preprocessing.LabelEncoder()
encpro=pro.fit_transform(data['gender'])
data['gender'] = encpro

pro= preprocessing.LabelEncoder()
encpro=pro.fit_transform(data['recruitment_channel'].astype(str))
data['recruitment_channel'] = encpro

pro= preprocessing.LabelEncoder()
encpro=pro.fit_transform(data['recruitment_channel'])
data['recruitment_channel'] = encpro
```

[53]:
```
#Bivariate Analysis
colormap = plt.cm.RdBu
plt.figure(figsize=(14,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(data.corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap=colormap, linecolor='white', annot=True)
```

[53]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x223aa2e7a90>
```


Pearson Correlation of Features
employee_id
department
region
education
gender
recruitment_channel
no_of_trainings
age
previous_year_rating
length_of_service
awards_won
avg_training_score
is_promoted
1.00
0.75
0.50
0.25
0.00
−0.25

5.0.1 promoted target is good correlated with the following features:

6 employee Id

7 deparment

8 region

9 education

10 recruitement_channel

11 previous year rating

12 awards won

13 avg training score

14 promoted target is not good correlated with the following features:¶

15 gender

16 no of training

17 age

18 length of service

[64]:
```
#Deleting the duplicate rows

current=len(data)
print('Rows of data before Delecting ', current)
```

```
Rows of data before Delecting  54808
```

[65]:
```
data=data.drop_duplicates()
```

[66]:
```
now=len(data)
print('Rows of data before Delecting ', now)
```

```
Rows of data before Delecting  54808
```

[68]:
```
diff=current-now
print('Duplicated rows deleted ', diff)
```

```
Duplicated rows deleted  0
```

[69]:
```
data=data.drop(columns=['employee_id'])
```

[70]:
```
#Missing value Treatment
data.isnull().sum()
```

[70]:
```
department                0
region                    0
education                 0
gender                    0
recruitment_channel       0
no_of_trainings           0
age                       0
previous_year_rating   4124
length_of_service         0
awards_won                0
avg_training_score     2560
is_promoted               0
dtype: int64
```

[71]:
```
data.isnull().sum().sum()/len(data)
```

[71]:
```
0.12195299956210773
```

[72]:
```
#lets calculate the total missing values in the each column
data_total = data.isnull().sum()
data_percent = ((data.isnull().sum()/data.shape[0])*100).round(2)
missing_data = pd.concat([data_total, data_percent],
                                axis=1,
                                keys=['Train_Total', 'Train_Percent␣
 ↪%','Test_Total', 'Test_Percent %'],
                                sort = True)
missing_data.style.bar(color = ['gold'])
```

[72]:
```
<pandas.io.formats.style.Styler at 0x223bda309e8>
```

[73]:
```
#working on the previous_year_rating
py=data[data['previous_year_rating'].isnull()]
py.head()
```

[73]:
```
    department  region  education  gender  recruitment_channel  \
10           8      15          3       1                    2
23           5      29          0       1                    0
29           7      28          0       1                    2
56           7      24          0       0                    0
58           7       7          0       1                    0

    no_of_trainings  age  previous_year_rating  length_of_service  awards_won  \
10                1   30                   NaN                  1           0
```

```
23                  1   27                    NaN                  1            0
29                  1   26                    NaN                  1            0
56                  1   29                    NaN                  1            0
58                  2   27                    NaN                  1            0

    avg_training_score  is_promoted
10                77.0            0
23                70.0            0
29                44.0            0
56                49.0            0
58                47.0            0
```

[ ]:
```
py['length_of_service'].value_counts()
```

Since the length of service is 1 for all the employees with previous year rating as null.,which means they are the new recruits with 1 year experience. So they may not be having the previous year rating.We impute 0 for the null values.

[74]:
```
data['previous_year_rating'].fillna(value=0,inplace=True)
```

[75]:
```
#Working on the Education and Previous_Year_rating
data['education'] = data['education'].fillna(data['education'].mode()[0])
data['avg_training_score'] = data['avg_training_score'].
 ↪fillna(data['avg_training_score'].mode()[0])
```

[76]:
```
data.isnull().sum() #Now we dont have any any missing values in the features.
```

[76]:
```
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
awards_won              0
avg_training_score      0
is_promoted             0
dtype: int64
```

18.0.1 Target : Target are the Results like in this project 1 and 0 are Target.

18.0.2 Inputs :Inputs are the data features that we feed into model like in this project department, region,education,gender are the inputs.

18.0.3 Training Data We use training data when we train the models. We feed train data to tensorflow model so that model can learn from the data.

18.0.4 Testing Data

We use testing data after training the model. We use this data to evalaute the performance that how the model perform after training. So in this way first we get predictions from the trained model without giving the Target and then we compare the true Target with predictions and get the performance of the model.

18.1 Separating input feature and label

[77]:
```
X=data.drop(columns=['is_promoted'])
y=data['is_promoted']
```

[78]:
```
X.head()
y.head()
```

[78]:
```
0    0
1    0
2    0
3    0
4    0
Name: is_promoted, dtype: int64
```

18.2 Model building - Logistic Regression(Random Forest Clssfier Model)-Step5

18.2.1 Separating the 70% data for training data and 30% for testing data

18.2.2 As we prepared all the data, now we are separating/splitting the all data into training data and testing data.

70% data will be used in the training

30% data will be used to test the performance of the model.

[82]:
```
X_train, X_test, y_train, y_test = train_test_split(X_up, y_up, test_size=0.3,␣
 ↪random_state=2)
```

[92]:
```
X_train, X_test, y_train, y_test = train_test_split(downsample.
 ↪drop(columns=['is_promoted']), downsample['is_promoted'], test_size=0.3,␣
 ↪random_state=2)
```

[93]:
```
X_train.shape
```

[93]:
```
(6535, 11)
```

18.2.3 Training the Random Forest Model ———————1

[94]:
```
LR=RandomForestClassifier()
LR= LR.fit(X_train , y_train)
LR
```

[94]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

18.2.4 Evaluation of Trained model on test data

18.2.5 Accuracy

Accuracy is the number of correctly classify promoted or not promoted. Accuracy= Total number of correct predictions/Total number of predictions

[95]:
```
print('Test set\n  Accuracy: {:0.2f}'.format(LR.score(X_test, y_test))) #the␣
↪accuracy of the model on test data is given below
```

```
Test set
  Accuracy: 0.68
```

18.3 Getting prediction of the test data and then we will compare the true Target/classes of the data with predictions

[96]:
```
y_pred = LR.predict(X_test) #getting predictions on the trained model
```

18.4 Precision Score on test data

Precision measure the number of positive class predictions that actually belong to the positive class

[97]:
```
print('Precision',round(f1_score(y_test, y_pred, average='micro'),3),'%')
```

```
Precision 0.677 %
```

18.4.1 Recall Score on test data¶

Recall measures the number of positive class predictions made out of all positive records in the dataset

[98]:
```
print('Recall',round(recall_score(y_test, y_pred, average='micro'),4),'%')
```

```
Recall 0.6769 %
```

18.5 F1 Measure Score on test data¶

F-Measure is the average of the precision and recall.

[99]:
```
print('F1',round(f1_score(y_test, y_pred, average='micro'),2),'%')
```

```
F1 0.68 %
```

18.5.1 Bagging and Boosting——————————1

[100]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=2)
```

[101]:
```
DT=DecisionTreeClassifier()
DT= DT.fit(X_train , y_train)
DT
```

[101]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

[102]:
```
dt=DT.score(X_test, y_test)
print('Test set\n  Accuracy: {:0.2f}'.format(DT.score(X_test, y_test))) #the␣
↪accuracy of the model on test data is given below
```

```
Test set
  Accuracy: 0.88
```

18.5.2 Random Forest Classifier Model——————————2

[103]:
```
RN=RandomForestClassifier()
RN= RN.fit(X_train , y_train)
RN
```

[103]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[104]:
```
RandomForestClassifier()
```

[104]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators='warn',
                       n_jobs=None, oob_score=False, random_state=None,
```

```
                    verbose=0, warm_start=False)
```

[105]:
```
rn=RN.score(X_test, y_test)
print('Test set\n  Accuracy: {:0.2f}'.format(RN.score(X_test, y_test))) #the␣
 ↪accuracy of the model on test data is given below
```

```
Test set
  Accuracy: 0.93
```

18.5.3 Bagging Classifier Model ——————————————3

[106]:
```
BC=BaggingClassifier()
BC= BC.fit(X_train , y_train)
BC
```

[106]:
```
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=False,
                  max_features=1.0, max_samples=1.0, n_estimators=10,
                  n_jobs=None, oob_score=False, random_state=None, verbose=0,
                  warm_start=False)
```

[107]:
```
bc=BC.score(X_test, y_test)
print('Test set\n  Accuracy: {:0.2f}'.format(BC.score(X_test, y_test))) #the␣
 ↪accuracy of the model on test data is given below
```

```
Test set
  Accuracy: 0.93
```

18.5.4 XGB Classifierr Model———————————————4

[108]:
```
XG=XGBClassifier(verbosity = 0)
XG= XG.fit(X_train , y_train)
XG
```

[108]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=4,
              num_parallel_tree=1, objective='binary:logistic',
              predictor='auto', random_state=0, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              use_label_encoder=True, validate_parameters=1, verbosity=0)
```

[109]:
```
xg=XG.score(X_test, y_test)
print('Test set\n  Accuracy: {:0.2f}'.format(XG.score(X_test, y_test))) #the␣
 ↪accuracy of the model on test data is given below
```

```
Test set
  Accuracy: 0.94
```

18.5.5 AdaBoost Classifier Model——————————————-5

[110]:
```
AD=AdaBoostClassifier()
AD= AD.fit(X_train , y_train)
AD
```

[110]:
```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=None)
```

[111]:
```
ad=AD.score(X_test, y_test)
print('Test set\n  Accuracy: {:0.2f}'.format(AD.score(X_test, y_test))) #the␣
 ↪accuracy of the model on test data is given below
```

```
Test set
  Accuracy: 0.92
```

18.5.6 Gradient Boosting Classifier Model——————————6

[112]:
```
GB=GradientBoostingClassifier()
GB= GB.fit(X_train , y_train)
GB
```

[112]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

[113]:
```
gb=GB.score(X_test, y_test)
print('Test set\n  Accuracy: {:0.2f}'.format(GB.score(X_test, y_test))) #the␣
 ↪accuracy of the model on test data is given below
```

```
Test set
  Accuracy: 0.94
```

18.5.7 Comparison of All Models

[114]:
```
x = PrettyTable()
print('\n')
x.field_names = ["Model", "Accuracy"]
x.add_row(["Decision Tree Model", round(dt,2)])
```

```
x.add_row(["Random Forest Classifier Model", round(rn,2)])
x.add_row(["Bagging Classifier Model", round(bc,2)])
x.add_row(["XGB Classifierr Model", round(xg,2)])
x.add_row(["AdaBoost Classifier Model", round(ad,2)])
x.add_row(["Gradient Boosting Classifier Model", round(gb,2)])

print(x)
print('\n')
```

```
+------------------------------------+----------+
|               Model                | Accuracy |
+------------------------------------+----------+
|        Decision Tree Model         |   0.88   |
|   Random Forest Classifier Model   |   0.93   |
|      Bagging Classifier Model      |   0.93   |
|       XGB Classifierr Model        |   0.94   |
|     AdaBoost Classifier Model      |   0.92   |
| Gradient Boosting Classifier Model |   0.94   |
+------------------------------------+----------+
```

18.6 XGBoost and Gradient Decent models are giving highest accuracy with 94% which is good. But Decision tree model is lower than others with only 88% accuracy. Random forest,Bagging and Adaboost also performed well with 93% accuracy.

18.6.1 Hyperparameter Tuning using Grid Search

Random Forest Model

XGB Classifierr Model

Gradient Boosting Classifier Model