```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
import statsmodels.formula.api as sfa
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objs as go
import plotly.express as px
from plotly.subplots import make_subplots
import math
from sklearn.metrics import r2_score
import random
from sklearn import metrics
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from collections import Counter
import scipy.stats as ss
import sklearn.preprocessing as sp
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.multiclass import OneVsOneClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
!pip install statsmodels
!pip3 install catboost
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import BaggingRegressor,AdaBoostRegressor,GradientBoostingRegressor
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
cmap = sns.cm.mako_r
%matplotlib inline
from sklearn.svm import SVC
import statsmodels.api as sm
```

```
import statsmodels.api as sm
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scipy>=0.18 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from pa
Requirement already satisfied: catboost in /usr/local/lib/python3.7/dist-packages (0
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from ca
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-packa
```

```
from google.colab import files
file = files.upload()  #upload file into google colab session
df = pd.read_csv("aug_train.csv")
df.head()
```

Choose Files aug_train.csv

- **aug_train.csv**(application/vnd.ms-excel) - 1961145 bytes, last modified: 12/7/2020 - 100% done

```
Saving aug_train.csv to aug_train (1).csv
```

	enrollee_id	city	city_development_index	gender	relevent_experience	enroll
0	8949	city_103	0.920	Male	Has relevent experience	
1	29725	city_40	0.776	Male	No relevent experience	
2	11561	city_21	0.624	NaN	No relevent experience	
3	33241	city_115	0.789	NaN	No relevent experience	
4	666	city_162	0.767	Male	Has relevent experience	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19158 entries, 0 to 19157
Data columns (total 14 columns):
 #   Column                    Non-Null Count  Dtype
```

```
 ---  ------                  --------------  -----
  0   enrollee_id             19158 non-null  int64
  1   city                    19158 non-null  object
  2   city_development_index  19158 non-null  float64
  3   gender                  14650 non-null  object
  4   relevent_experience     19158 non-null  object
  5   enrolled_university     18772 non-null  object
  6   education_level         18698 non-null  object
  7   major_discipline        16345 non-null  object
  8   experience              19093 non-null  object
  9   company_size            13220 non-null  object
  10  company_type            13018 non-null  object
  11  last_new_job            18735 non-null  object
  12  training_hours          19158 non-null  int64
  13  target                  19158 non-null  float64
 dtypes: float64(2), int64(2), object(10)
 memory usage: 2.0+ MB
```

```
df.isnull().sum()
```

```
enrollee_id                 0
city                        0
city_development_index      0
gender                   4508
relevent_experience         0
enrolled_university       386
education_level           460
major_discipline         2813
experience                 65
company_size             5938
company_type             6140
last_new_job              423
training_hours              0
target                      0
dtype: int64
```

```
(df.isnull().sum()/len(df))*100
```

```
enrollee_id                0.000000
city                       0.000000
city_development_index     0.000000
gender                    23.530640
relevent_experience        0.000000
enrolled_university        2.014824
education_level            2.401086
major_discipline          14.683161
experience                 0.339284
company_size              30.994885
company_type              32.049274
last_new_job               2.207955
training_hours             0.000000
target                     0.000000
dtype: float64
```

```
df.shape
```

```
(19158, 14)
```

```
df.size
```

```
268212
```

```
for col in df.columns:
    print(col, df[col].unique())
```

```
enrollee_id [ 8949 29725 11561 ... 24576  5756 23834]
city ['city_103' 'city_40' 'city_21' 'city_115' 'city_162' 'city_176'
 'city_160' 'city_46' 'city_61' 'city_114' 'city_13' 'city_159' 'city_102'
 'city_67' 'city_100' 'city_16' 'city_71' 'city_104' 'city_64' 'city_101'
 'city_83' 'city_105' 'city_73' 'city_75' 'city_41' 'city_11' 'city_93'
 'city_90' 'city_36' 'city_20' 'city_57' 'city_152' 'city_19' 'city_65'
 'city_74' 'city_173' 'city_136' 'city_98' 'city_97' 'city_50' 'city_138'
 'city_82' 'city_157' 'city_89' 'city_150' 'city_70' 'city_175' 'city_94'
 'city_28' 'city_59' 'city_165' 'city_145' 'city_142' 'city_26' 'city_12'
 'city_37' 'city_43' 'city_116' 'city_23' 'city_99' 'city_149' 'city_10'
 'city_45' 'city_80' 'city_128' 'city_158' 'city_123' 'city_7' 'city_72'
 'city_106' 'city_143' 'city_78' 'city_109' 'city_24' 'city_134' 'city_48'
 'city_144' 'city_91' 'city_146' 'city_133' 'city_126' 'city_118' 'city_9'
 'city_167' 'city_27' 'city_84' 'city_54' 'city_39' 'city_79' 'city_76'
 'city_77' 'city_81' 'city_131' 'city_44' 'city_117' 'city_155' 'city_33'
 'city_141' 'city_127' 'city_62' 'city_53' 'city_25' 'city_2' 'city_69'
 'city_120' 'city_111' 'city_30' 'city_1' 'city_140' 'city_179' 'city_55'
 'city_14' 'city_42' 'city_107' 'city_18' 'city_139' 'city_180' 'city_166'
 'city_121' 'city_129' 'city_8' 'city_31' 'city_171']
city_development_index [0.92  0.776 0.624 0.789 0.767 0.764 0.762 0.913 0.926 0.827
 0.855 0.887 0.91  0.884 0.924 0.666 0.558 0.923 0.794 0.754 0.939 0.55
 0.865 0.698 0.893 0.796 0.866 0.682 0.802 0.579 0.878 0.897 0.949 0.925
 0.896 0.836 0.693 0.769 0.775 0.903 0.555 0.727 0.64  0.516 0.743 0.899
 0.915 0.689 0.895 0.89  0.847 0.527 0.766 0.738 0.647 0.795 0.74  0.701
 0.493 0.84  0.691 0.735 0.742 0.479 0.722 0.921 0.848 0.856 0.898 0.83
 0.73  0.68  0.725 0.556 0.448 0.763 0.745 0.645 0.788 0.78  0.512 0.739
 0.563 0.518 0.824 0.487 0.649 0.781 0.625 0.807 0.664]
gender ['Male' nan 'Female' 'Other']
relevent_experience ['Has relevent experience' 'No relevent experience']
enrolled_university ['no_enrollment' 'Full time course' nan 'Part time course']
education_level ['Graduate' 'Masters' 'High School' nan 'Phd' 'Primary School']
major_discipline ['STEM' 'Business Degree' nan 'Arts' 'Humanities' 'No Major' 'Other
experience ['>20' '15' '5' '<1' '11' '13' '7' '17' '2' '16' '1' '4' '10' '14' '18'
 '19' '12' '3' '6' '9' '8' '20' nan]
company_size [nan '50-99' '<10' '10000+' '5000-9999' '1000-4999' '10/49' '100-500'
 '500-999']
company_type [nan 'Pvt Ltd' 'Funded Startup' 'Early Stage Startup' 'Other'
 'Public Sector' 'NGO']
last_new_job ['1' '>4' 'never' '4' '3' '2' nan]
training_hours [ 36  47  83  52   8  24  18  46 123  32 108  23  26 106   7 132  68
  48  65  13  22 148  72  40 141  82 145 206 152  42  14 112  87  20  21
  92 102  43  45  19  90  25  15  98 142  28 228  29  12  17  35   4 136
  27  74  86  75 332 140 182 172  33  34 150 160   3   2 210 101  59 260
 131 109  70  51  60 164 290 133  76 156 120 100  39  55  49   6 125 326
 198  11  41 114 246  81  31  84 105  38 178 104 202  88 218  62  10  80
  77  37 162 190  30  16   5  54  44 110 262 107 134 103  96  57 240  94
 113  56  64 320   9 129  58 126 166  95  97 204 116 161 146 302  53 143
 124 214 288 306 322  67  61 130 220  78 314 226 280  91 234 163 151  85
 256 168 144  66 128  73 122 154  63 292 188  71 135 138 184  89 157 118
 111 192 127 216 139 196  99 167 276 121  69 155 316 242 304 284 278 310
 222 212 250 180 258 330 158 149 165  79 194 176 174 312 200 328 300 153
 232 336 308 147 298 224 254 248 236 170 264 119 117 334 324   1 238 266
```

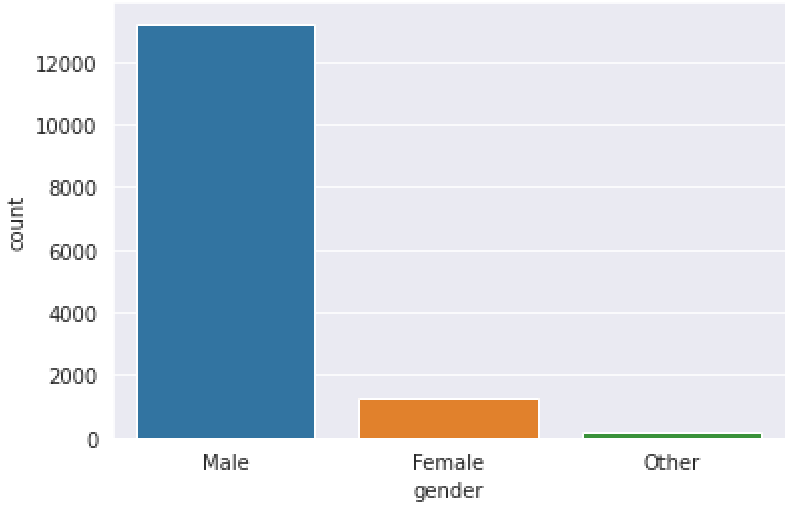
```
 282 268 244 272 294 270 286]
target [1. 0.]
```

Data Visualization

Which gender is more likely to move for a new job?

```
sns.countplot(df.gender)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ed1b01810>
```



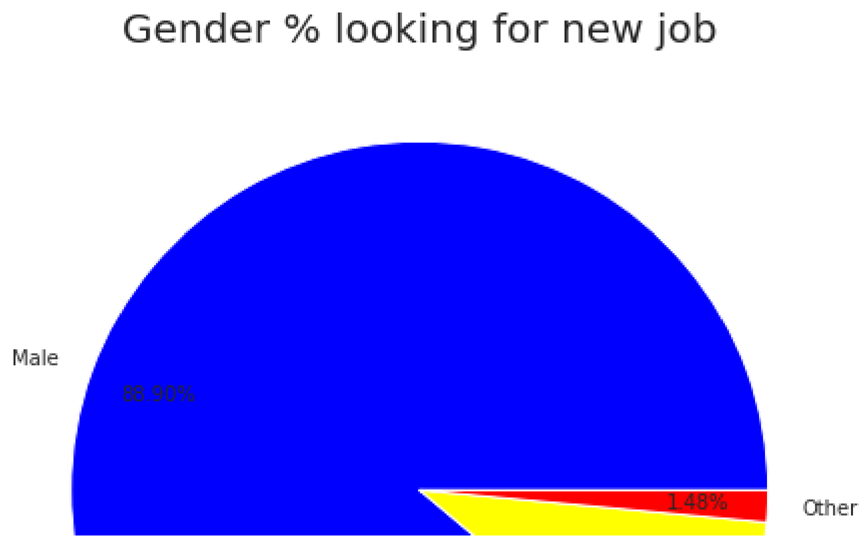

```
genders = df[df['target'] == 1]['gender']
temp_df= (genders.value_counts())/len(genders)*100
```

```
temp_df
```

```
Male      63.052125
Female     6.824367
Other      1.046682
Name: gender, dtype: float64
```

```
plt.figure(figsize=(8,8))
plt.pie(temp_df,labels = temp_df.keys() , colors = ['blue','yellow','red'], autopct="%.2f%
plt.title('Gender % looking for new job', fontsize=20)
```

```
Text(0.5, 1.0, 'Gender % looking for new job')
```

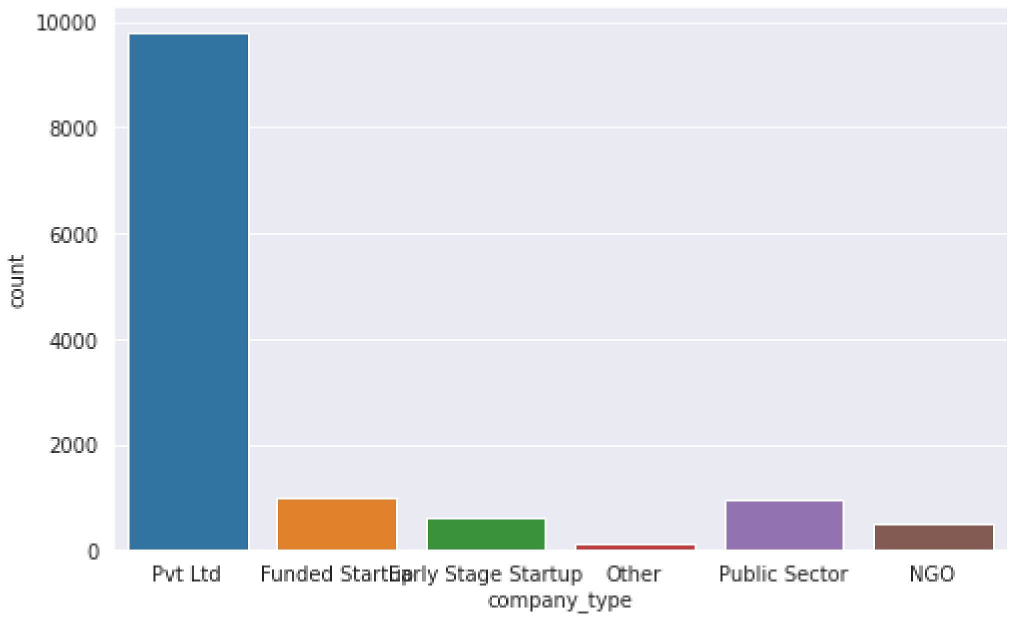



```
male_newjob = df[(df['gender']=='Male') & df['target']==1]
female_newjob = df[(df['gender']=='Female') & df['target']==1]

# print
print('{} % of male who are looking for a new job'.format(len(male_newjob)/len(df['gender'
print('{} % of female who are looking for a new job'.format(len(female_newjob)/len(df['gen
```

```
15.721891637958032 % of male who are looking for a new job
1.7016390019835057 % of female who are looking for a new job
```

From which company type people are looking for new job?

```
plt.figure(figsize=(8,5))
sns.countplot(df['company_type'])
plt.show()
```

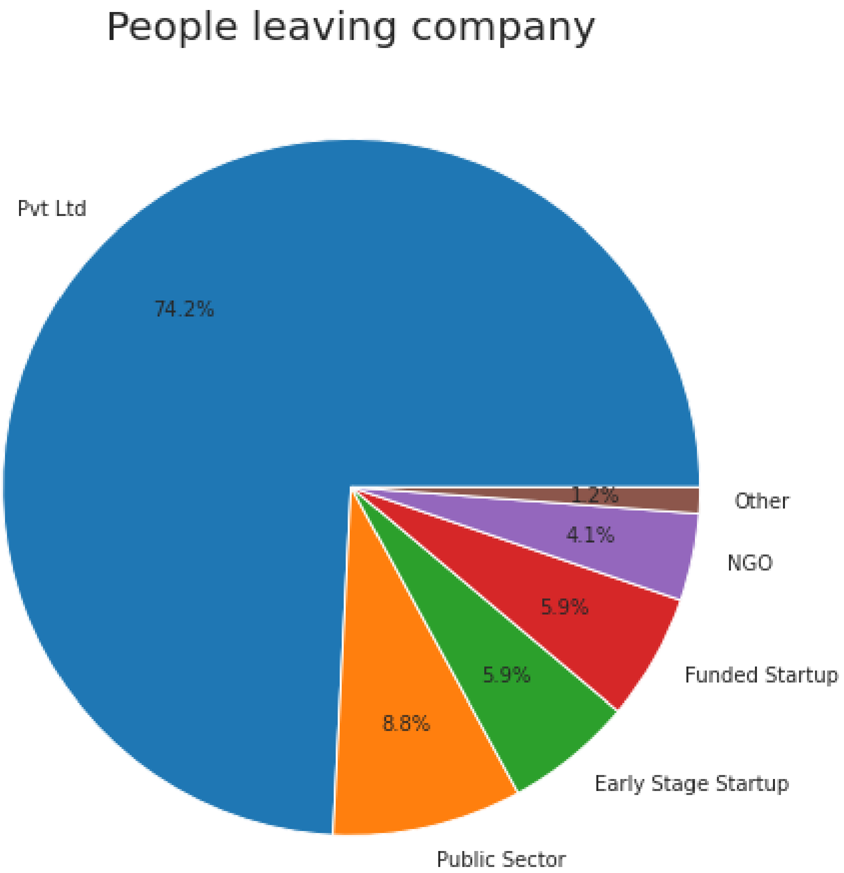


here we can see that most of pvt sector employees are looking for job change

```
company_type = df[df['target'] == 1]['company_type']
```

```
company_type = df[df['target']==1]['company_type']
temp = company_type.value_counts()
labels = temp.keys()
bar,ax = plt.subplots(figsize=(8,8))
plt.pie(x = temp, labels = labels, autopct="%.1f%%",pctdistance=0.7)
plt.title('People leaving company', fontsize=20)
```

```
Text(0.5, 1.0, 'People leaving company')
```

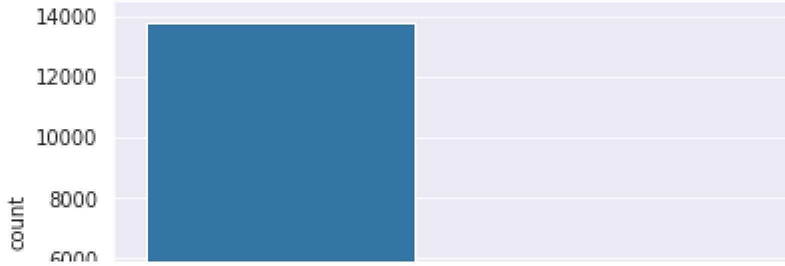



```
for i in df['company_type'].unique():
    company_newjob = df[(df['company_type']==i) & df['target']==1]
    print('{} % of {} who are looking for a new job'.format(len(company_newjob)/len(df['co
```

```
0.0 % of nan who are looking for a new job
9.2650589831924 % of Pvt Ltd who are looking for a new job
0.7307652155757386 % of Funded Startup who are looking for a new job
0.741204718655392 % of Early Stage Startup who are looking for a new job
0.15137279465497444 % of Other who are looking for a new job
1.096147823363608 % of Public Sector who are looking for a new job
0.5063158993631903 % of NGO who are looking for a new job
```

```
sns.countplot(df['relevent_experience'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ed19bea50>
```



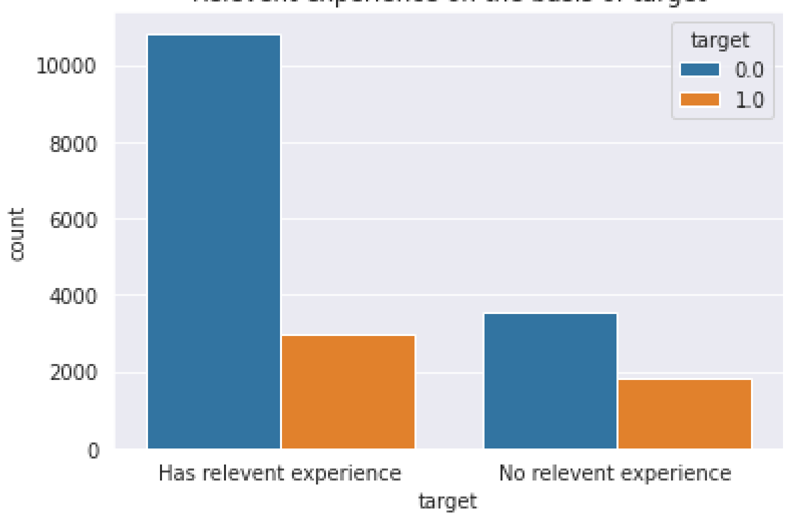

```
(df['relevent_experience']).value_counts()
```

```
Has relevent experience    13792
No relevent experience      5366
Name: relevent_experience, dtype: int64
```

relevent_experience

```
sns.countplot(df['relevent_experience'],hue=df['target'])
plt.xlabel('target')
plt.ylabel('count')
plt.title('Relevent experience on the basis of target')
```

```
Text(0.5, 1.0, 'Relevent experience on the basis of target')
```



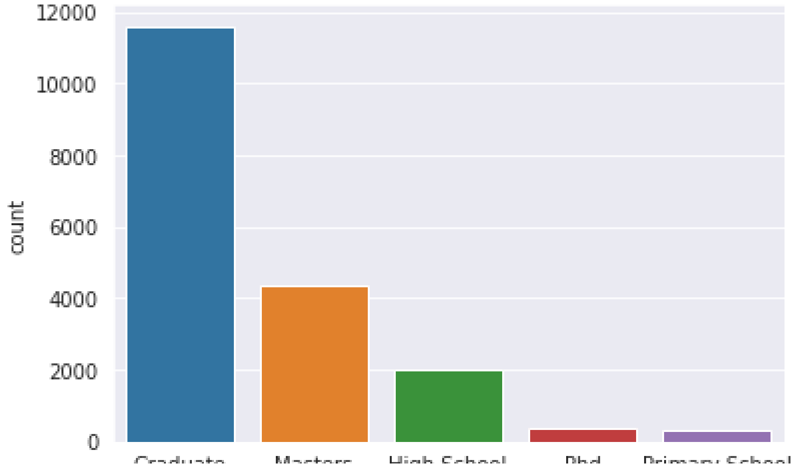

```
yes_newjob = df[(df['relevent_experience']=='Has relevent experience') & df['target']==1]
no_newjob = df[(df['relevent_experience']=='No relevent experience') & df['target']==1]

# print
print('{} % of having relevant experience who are looking for a new job'.format(len(yes_ne
print('{} % of not havinf relevant experience who are looking for a new job'.format(len(no
```

```
15.45568430942687 % of having relevant experience who are looking for a new job
9.479068796325295 % of not havinf relevant experience who are looking for a new job
```

Did any people got into data science field without having graduation degree?

```
sns.countplot(df['education_level'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ed18ebe50>
```




```
df.education_level.value_counts()
```

```
Graduate          11598
Masters            4361
High School        2017
Phd                 414
Primary School      308
Name: education_level, dtype: int64
```

```
people_withoutdegree = df[(df['education_level'] == 'Primary School')& (df['education_leve
print("People who have got into the data science world without graduation are", len(people
```

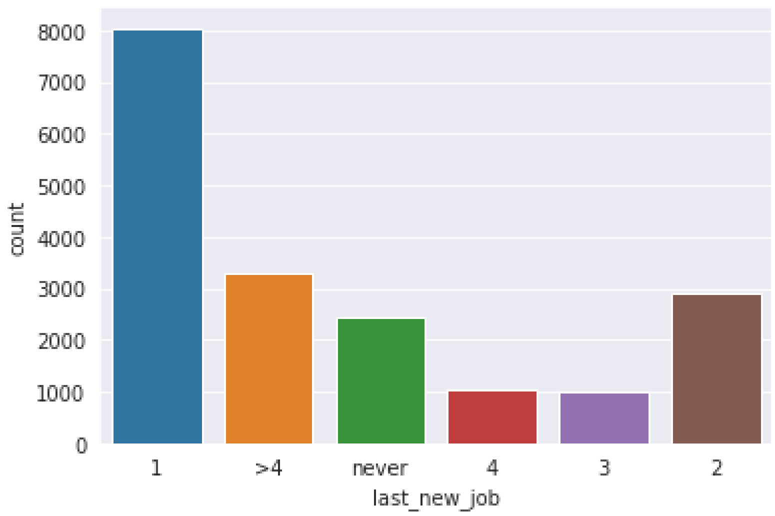
```
People who have got into the data science world without graduation are 0
```

So their is not a single person who get into this field without graduation.

Years between last and current job

```
sns.countplot(df['last_new_job'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ed1d16890>
```



DATA PREPROCESSING

First of all we are going to drop unnecessary columns,so we don't require enrollee_id and city column

```
df.drop(columns=["city","enrollee_id"],inplace=True)
```

```
df
```

	city_development_index	gender	relevent_experience	enrolled_university	edu
0	0.920	Male	Has relevent experience	no_enrollment	
1	0.776	Male	No relevent experience	no_enrollment	
2	0.624	NaN	No relevent experience	Full time course	
3	0.789	NaN	No relevent experience	NaN	
4	0.767	Male	Has relevent experience	no_enrollment	
...	...	...	...	...	
19153	0.878	Male	No relevent experience	no_enrollment	
19154	0.920	Male	Has relevent experience	no_enrollment	
19155	0.920	Male	Has relevent experience	no_enrollment	

Countplot for some categorical feature

We already have seen countplot for various features.Now,we are going to see countplot for the features we haven't seen yet

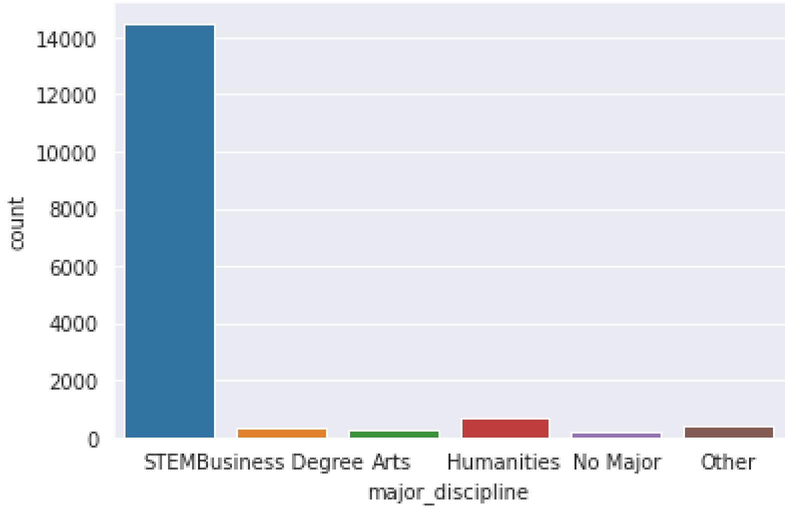
```
sns.countplot(df['enrolled_university'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ed2446310>
```

So it's seems like most people who are currently doing job haven't enrolled in any university

```
sns.countplot(df['major_discipline'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ed1a71710>
```

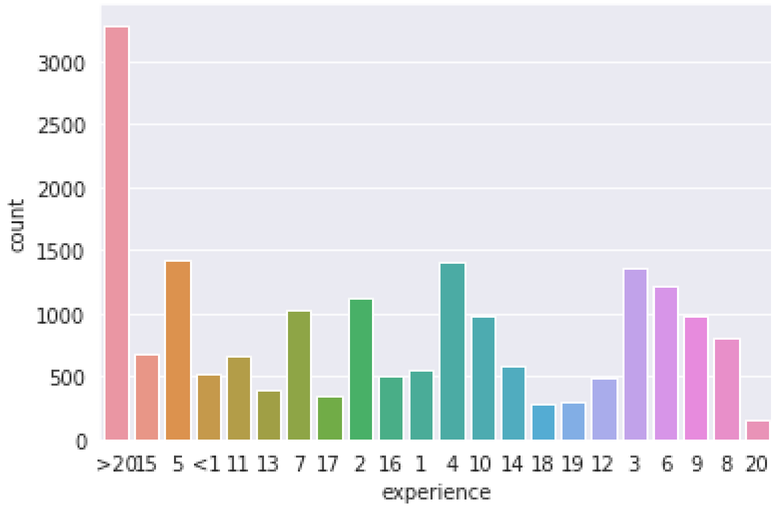


Most of the Candidates are from STEM. That is their major discipline was in one of the Following:

Science Technology Engineering Mathematics

```
sns.countplot(df['experience'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ed17ab890>
```



dealing with null values

```
# null value
percent_null = df.isnull().mean()*100
print(percent_null)
```

```
city_development_index     0.000000
gender                    23.530640
relevent_experience        0.000000
enrolled_university        2.014824
education_level            2.401086
major_discipline          14.683161
experience                 0.339284
company_size              30.994885
company_type              32.049274
last_new_job               2.207955
training_hours             0.000000
target                     0.000000
dtype: float64
```

Columns in which we have 2% or less than 2% null values we can drop those null values

```
df.dropna(subset=['enrolled_university','education_level','experience','last_new_job'], ax
```

```
# after dropping those null values
df.shape
```

```
(18014, 12)
```

Now we are going to fill null values with their mode as all the columns left have dtype as 'object'

```
df.dtypes
```

```
city_development_index    float64
gender                     object
relevent_experience        object
enrolled_university        object
education_level            object
major_discipline           object
experience                 object
company_size               object
company_type               object
last_new_job               object
training_hours              int64
target                    float64
dtype: object
```

```
col_mode = ['gender','company_size','major_discipline','company_type','relevent_experience
for col in col_mode:
    df[col].fillna(df[col].mode()[0],inplace=True)
```

Let's change the dtype of experience and last_new_job column

```
df.replace(to_replace = 'Has relevent experience',value = 'Yes',inplace = True)
df.replace(to_replace = 'No relevent experience',value='No',inplace = True )

df.replace(to_replace = '<1',value = '0',inplace = True)
df.replace(to_replace = '>20',value = '21',inplace=True)
```

```
df.replace(to_replace = 'never',value = '0',inplace=True)
df.replace(to_replace = '>4',value = '5',inplace=True)

df.replace(to_replace = '<10',value = 'around_10',inplace=True)
df.replace(to_replace = '10/49',value = 'around_50',inplace=True)
df.replace(to_replace = '50-99',value = 'around_100',inplace=True)
df.replace(to_replace = '100-500',value = 'around_500',inplace=True)
df.replace(to_replace = '500-999',value = 'around_1000',inplace=True)
df.replace(to_replace = '1000-4999',value = 'around_5000',inplace=True)
df.replace(to_replace = '5000-9999',value = 'around_10000',inplace=True)
df.replace(to_replace = '10000+',value = 'more_than_10000',inplace=True)

df.replace(to_replace = 'Full time course',value = 'Full_time_course',inplace=True)
df.replace(to_replace = 'Part time course',value = 'Part_time_course',inplace=True)

df.replace(to_replace = 'Primary School',value = 'Primary_School',inplace=True)
df.replace(to_replace = 'High School',value = 'High_School',inplace=True)

df.replace(to_replace = 'Business Degree',value = 'Business_Degree',inplace=True)
df.replace(to_replace = 'No Major',value = 'No_Major',inplace=True)

df.replace(to_replace = 'Pvt Ltd',value = 'Pvt_Ltd',inplace=True)
df.replace(to_replace = 'Funded Startup',value = 'Funded_Startup',inplace=True)
df.replace(to_replace = 'Public Sector',value = 'Public_Sector',inplace=True)
df.replace(to_replace = 'Early Stage Startup',value = 'Early_Stage_Startup',inplace=True)

df['major_discipline'].replace('Other','Other_major',inplace=True)
df['company_type'].replace('Other','Other_type',inplace=True)


df = df.astype({'experience':int,'last_new_job':int})
```

Handling Categorical Values

```
# get dummies

education_df = pd.get_dummies(df[['education_level']],drop_first=True,prefix=[None])
company_size_df = pd.get_dummies(df[['company_size']],drop_first=True,prefix=[None])
company_type_df = pd.get_dummies(df[['company_type']],drop_first=True,prefix=[None])
major_df = pd.get_dummies(df[['major_discipline']],drop_first=True,prefix=[None])
university_df = pd.get_dummies(df[['enrolled_university']],drop_first=True,prefix=[None])
experience_df = pd.get_dummies(df[['relevent_experience']],drop_first=True,prefix=[None])
gender_df = pd.get_dummies(df[['gender']],drop_first=True,prefix=[None])


# drop original columns
df.drop(['education_level','company_size','company_type','major_discipline','enrolled_univ


final_df = pd.concat([df,education_df,company_size_df,company_type_df,major_df,university_


final_df.head(5)
```

	city_development_index	experience	last_new_job	training_hours	target	High_Sc
0	0.920	21	1	36	1.0	
1	0.776	15	5	47	0.0	
2	0.624	5	0	83	0.0	
4	0.767	21	4	8	0.0	
5	0.764	11	1	24	1.0	

```
final_df.to_csv('final_df.csv')
```

```
X = final_df.drop(['target'], axis = 1)
Y = final_df['target']
```

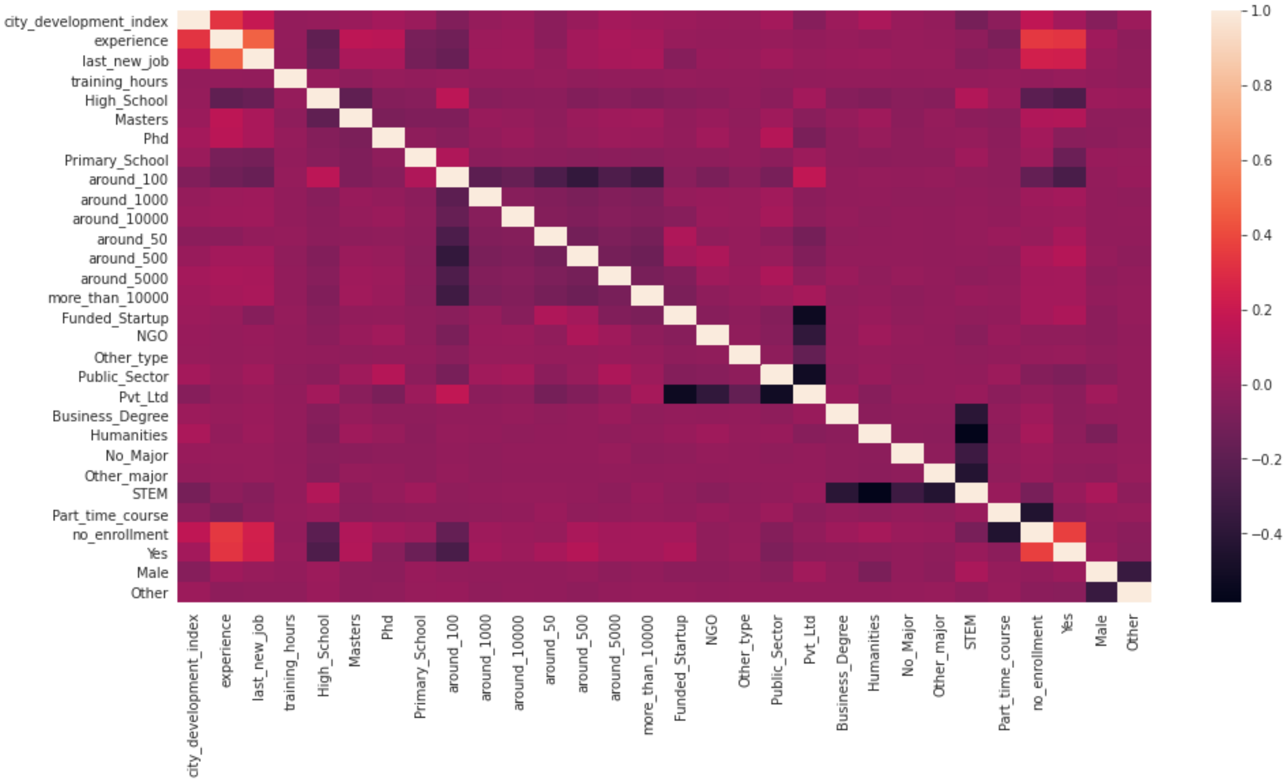
correlation

```
X.corr()
```

	city_development_index	experience	last_new_job	training_ho
city_development_index	1.000000	0.330516	0.188412	0.002
experience	0.330516	1.000000	0.475681	0.001
last_new_job	0.188412	0.475681	1.000000	-0.003
training_hours	0.002648	0.001900	-0.003823	1.000
High_School	0.009820	-0.186970	-0.158639	0.010
Masters	0.027950	0.154977	0.084206	-0.018
Phd	0.066337	0.142408	0.079211	0.007
Primary_School	0.026310	-0.095175	-0.107812	-0.005
around_100	-0.066454	-0.125080	-0.157063	0.009
around_1000	0.010563	0.033003	0.035970	-0.002

```
plt.figure(figsize=(16,8))
sns.heatmap(X.corr())
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f6ed17325d0>



check multicolineartiy

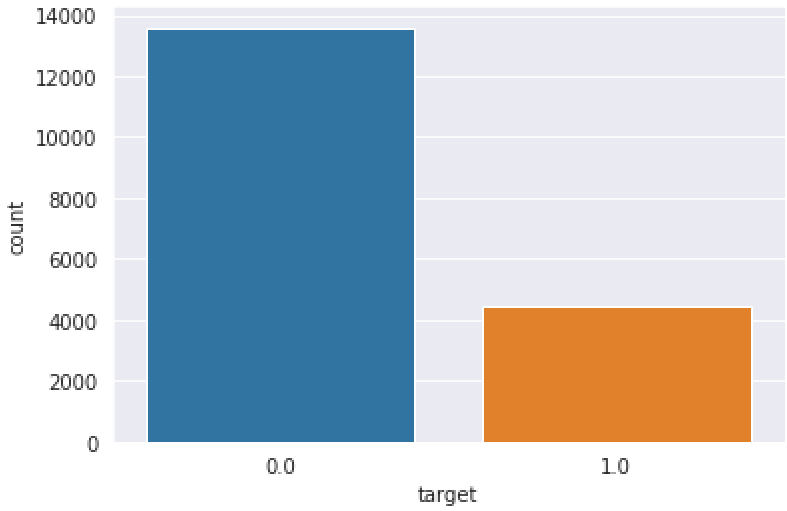
```
def calculate_vif_(X, thresh=5.0):
    variables = list(range(X.shape[1]))
    dropped = True
    while dropped:
        dropped = False
        vif = [variance_inflation_factor(X.iloc[:, variables].values, ix)
               for ix in range(X.iloc[:, variables].shape[1])]

        maxloc = vif.index(max(vif))
        if max(vif) > thresh:
            print('dropping \'' + X.iloc[:, variables].columns[maxloc] +
                  '\' at index: ' + str(maxloc))
            del variables[maxloc]
            dropped = True

    print('Remaining variables:')
    print(X.columns[variables])
    return X.iloc[:, variables]


calculate_vif_(X)
```

```
dropping 'city_development_index' at index: 0
dropping 'STEM' at index: 23
dropping 'Pvt_Ltd' at index: 18
dropping 'Male' at index: 25
```

lets check data is balanced or not

```
sns.countplot(df['target'])
```

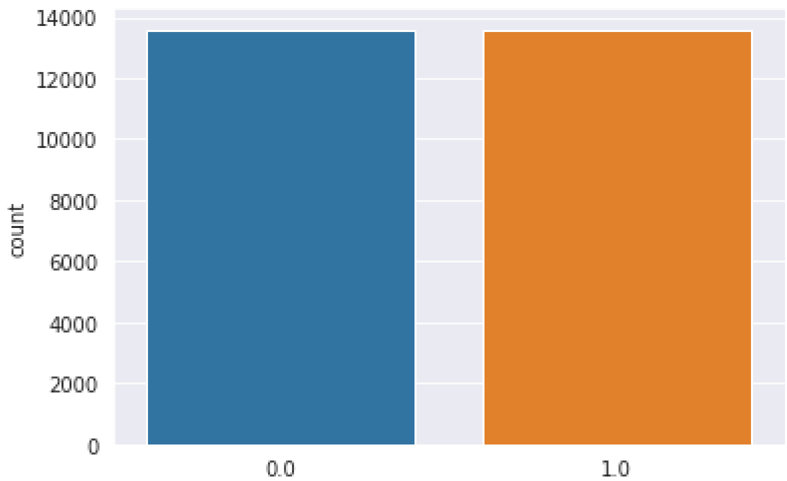
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ec8d6cbd0>
```



data is imblanced.

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state = 402)
X_smote, Y_smote = smote.fit_resample(X,Y)
sns.countplot(Y_smote)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ec746b410>
```




```
X_train, X_test, Y_train, Y_test = train_test_split(X_smote, Y_smote, test_size = 0.2 ,ran
```

standard scaler

```
scaler=StandardScaler()
scaler.fit(X_train)
#scaler.fit_transform(X_train)
X_train = scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

KNN CLASSIFIER

```
#predicting using the KNeighbors_Classifier
model_KNN=KNeighborsClassifier(n_neighbors=int(np.sqrt(len(X_train))),
                               metric='minkowski')
#euclidean,manhattan,minkowski
#fit the model on the data and predict the values
model_KNN.fit(X_train,Y_train)

Y_pred=model_KNN.predict(X_test)
print(list(zip(Y_test,Y_pred)))
```

```
[(1.0, 1.0), (1.0, 1.0), (0.0, 0.0), (0.0, 0.0), (0.0, 0.0), (1.0, 1.0), (1.0, 1.0),
```

```
cfm=confusion_matrix(Y_test,Y_pred)
print(cfm)
print()

print("Classification report: ")

print(classification_report(Y_test,Y_pred))

acc=accuracy_score(Y_test, Y_pred)
print("Accuracy of the model: ",acc)
```

```
[[2104  652]
 [ 731 1951]]

Classification report: 
              precision    recall  f1-score   support

         0.0       0.74      0.76      0.75      2756
         1.0       0.75      0.73      0.74      2682

    accuracy                           0.75      5438
   macro avg       0.75      0.75      0.75      5438
weighted avg       0.75      0.75      0.75      5438

Accuracy of the model:  0.7456785582934903
```

we will try to improve accuracy

```
from sklearn.metrics import accuracy_score
my_dict={}
for K in range(1,30):
    model_KNN = KNeighborsClassifier(n_neighbors=K,metric="euclidean")
```

```
    model_KNN.fit(X_train, Y_train)
    Y_pred = model_KNN.predict(X_test)
    print ("Accuracy is ", accuracy_score(Y_test,Y_pred), "for K-Value:",K)
    my_dict[K]=accuracy_score(Y_test,Y_pred)
```

```
Accuracy is  0.8148216255976461 for K-Value: 1
Accuracy is  0.7795145273997793 for K-Value: 2
Accuracy is  0.7888929753585877 for K-Value: 3
Accuracy is  0.7721588819418904 for K-Value: 4
Accuracy is  0.7763883780801766 for K-Value: 5
Accuracy is  0.768481059212946 for K-Value: 6
Accuracy is  0.7751011401250459 for K-Value: 7
Accuracy is  0.7705038617138654 for K-Value: 8
Accuracy is  0.7767561603530709 for K-Value: 9
Accuracy is  0.7670099301213682 for K-Value: 10
Accuracy is  0.7697682971680765 for K-Value: 11
Accuracy is  0.7690327326222876 for K-Value: 12
Accuracy is  0.7682971680764987 for K-Value: 13
Accuracy is  0.7703199705774182 for K-Value: 14
Accuracy is  0.7716072085325487 for K-Value: 15
Accuracy is  0.7699521883045237 for K-Value: 16
Accuracy is  0.7695844060316293 for K-Value: 17
Accuracy is  0.7699521883045237 for K-Value: 18
Accuracy is  0.7695844060316293 for K-Value: 19
Accuracy is  0.7699521883045237 for K-Value: 20
Accuracy is  0.7712394262596542 for K-Value: 21
Accuracy is  0.7705038617138654 for K-Value: 22
Accuracy is  0.7697682971680765 for K-Value: 23
Accuracy is  0.7692166237587348 for K-Value: 24
Accuracy is  0.7671938212578153 for K-Value: 25
Accuracy is  0.7649871276204487 for K-Value: 26
Accuracy is  0.7666421478484737 for K-Value: 27
Accuracy is  0.7666421478484737 for K-Value: 28
Accuracy is  0.7653549098933431 for K-Value: 29
```

```
my_dict
```

```
{1: 0.8148216255976461,
 2: 0.7795145273997793,
 3: 0.7888929753585877,
 4: 0.7721588819418904,
 5: 0.7763883780801766,
 6: 0.768481059212946,
 7: 0.7751011401250459,
 8: 0.7705038617138654,
 9: 0.7767561603530709,
 10: 0.7670099301213682,
 11: 0.7697682971680765,
 12: 0.7690327326222876,
 13: 0.7682971680764987,
 14: 0.7703199705774182,
 15: 0.7716072085325487,
 16: 0.7699521883045237,
 17: 0.7695844060316293,
 18: 0.7699521883045237,
 19: 0.7695844060316293,
 20: 0.7699521883045237,
 21: 0.7712394262596542,
 22: 0.7705038617138654,
```

```
     23: 0.7697682971680765,
     24: 0.7692166237587348,
     25: 0.7671938212578153,
     26: 0.7649871276204487,
     27: 0.7666421478484737,
     28: 0.7666421478484737,
     29: 0.7653549098933431}
```

```
for k in my_dict:
    if my_dict[k]==max(my_dict.values()):
        print("KNN CLASSIFIER MAX ACCURACY IS : ",k,":",my_dict[k])
```

```
KNN CLASSIFIER MAX ACCURACY IS :  1 : 0.8148216255976461
```

LOGISTIC REGRESSION

```
log_reg=sm.Logit(Y_smote, X_smote).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.578978
         Iterations 6
```

```
print(log_reg.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                27186
Model:                          Logit   Df Residuals:                    27156
Method:                           MLE   Df Model:                           29
Date:                Sat, 24 Apr 2021   Pseudo R-squ.:                  0.1647
Time:                        00:45:17   Log-Likelihood:                -15740.
converged:                       True   LL-Null:                       -18844.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -5.0071      0.115    -43.542      0.000      -5.233      -4.782
x2            -0.0223      0.003     -8.420      0.000      -0.027      -0.017
x3             0.0446      0.010      4.531      0.000       0.025       0.064
x4            -0.0005      0.000     -2.380      0.017      -0.001   -9.68e-05
x5            -1.1549      0.053    -21.617      0.000      -1.260      -1.050
x6            -0.2783      0.036     -7.814      0.000      -0.348      -0.208
x7            -0.5264      0.116     -4.533      0.000      -0.754      -0.299
x8            -1.8967      0.141    -13.489      0.000      -2.172      -1.621
x9             1.4127      0.068     20.698      0.000       1.279       1.547
x10            0.1838      0.097      1.887      0.059      -0.007       0.375
x11            0.4013      0.108      3.710      0.000       0.189       0.613
x12            0.6820      0.082      8.361      0.000       0.522       0.842
x13            0.2010      0.077      2.599      0.009       0.049       0.353
x14            0.2170      0.089      2.446      0.014       0.043       0.391
x15            0.4441      0.079      5.609      0.000       0.289       0.599
x16            0.2641      0.118      2.229      0.026       0.032       0.496
x17            0.9033      0.134      6.719      0.000       0.640       1.167
x18            1.2499      0.209      5.966      0.000       0.839       1.660
x19            1.1017      0.118      9.305      0.000       0.870       1.334
x20            1.1256      0.096     11.737      0.000       0.938       1.314
```

```
x21           2.5896      0.166     15.599      0.000       2.264       2.915
x22           2.4161      0.149     16.180      0.000       2.123       2.709
x23           2.2993      0.186     12.341      0.000       1.934       2.664
x24           2.2954      0.160     14.318      0.000       1.981       2.610
x25           2.6041      0.124     21.029      0.000       2.361       2.847
x26          -0.4173      0.068     -6.172      0.000      -0.550      -0.285
x27          -0.2174      0.040     -5.415      0.000      -0.296      -0.139
x28          -0.3175      0.038     -8.354      0.000      -0.392      -0.243
x29           0.3026      0.059      5.155      0.000       0.188       0.418
x30           0.3639      0.164      2.216      0.027       0.042       0.686
==============================================================================
```

```
#create a model
classifier=LogisticRegression()
#fitting training data to the model
classifier.fit(X_train,Y_train)

Y_pred=classifier.predict(X_test)
print(list(zip(Y_test,Y_pred)))
```

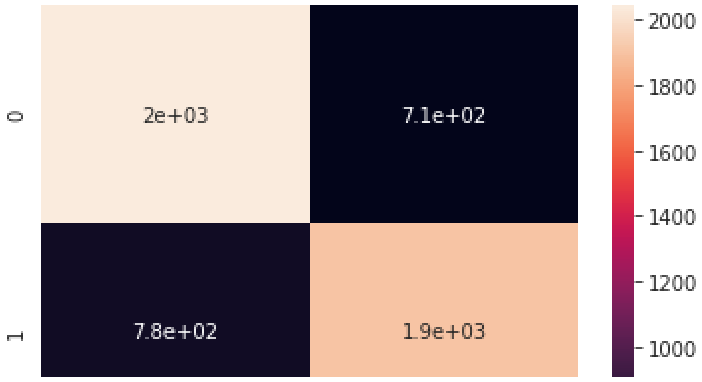
```
[(1.0, 1.0), (1.0, 0.0), (0.0, 0.0), (0.0, 0.0), (0.0, 0.0), (1.0, 1.0), (1.0, 1.0),
```

```
from sklearn.metrics import confusion_matrix, accuracy_score,classification_report

confusion_matrix=confusion_matrix(Y_test,Y_pred)
print(confusion_matrix)
print()
print("Classification report: ")

print(classification_report(Y_test,Y_pred))

accuracy_score=accuracy_score(Y_test, Y_pred)
print("Accuracy of the model: ",accuracy_score)
```

```
[[2042  714]
 [ 776 1906]]

Classification report: 
              precision    recall  f1-score   support

         0.0       0.72      0.74      0.73      2756
         1.0       0.73      0.71      0.72      2682

    accuracy                           0.73      5438
   macro avg       0.73      0.73      0.73      5438
weighted avg       0.73      0.73      0.73      5438

Accuracy of the model:  0.7260022066936374
```

```
sns.heatmap(confusion_matrix, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ec735c690>
```



Decision Trees

```
#predicting using the Decision_Tree_Classifier
model_DecisionTree=DecisionTreeClassifier(criterion="gini",random_state=10,splitter="best"
#fit the model on the data and predict the values
model_DecisionTree.fit(X_train,Y_train)
Y_pred=model_DecisionTree.predict(X_test)
print(Y_pred)
#print(list(zip(Y_test,Y_pred)))
```

```
[1. 1. 0. ... 1. 1. 0.]
```

```
from sklearn.metrics import confusion_matrix, accuracy_score,classification_report
#confusion matrix
print(confusion_matrix(Y_test,Y_pred))
print(accuracy_score(Y_test,Y_pred))
print(classification_report(Y_test,Y_pred))
```

```
[[2182  574]
 [ 513 2169]]
0.8001103346818683
              precision    recall  f1-score   support

         0.0       0.81      0.79      0.80      2756
         1.0       0.79      0.81      0.80      2682

    accuracy                           0.80      5438
   macro avg       0.80      0.80      0.80      5438
weighted avg       0.80      0.80      0.80      5438
```

```
model_DecisionTree.score(X_train,Y_train)
```

```
0.9982067316534854
```

Random_Forest_Classifier

```
#predicting using the Random_Forest_Classifier
model_RandomForest=RandomForestClassifier(n_estimators=50, random_state=10)

#fit the model on the data and predict the values
```

```
model_RandomForest.fit(X_train,Y_train)

Y_pred=model_RandomForest.predict(X_test)


#confusion matrix
print(confusion_matrix(Y_test,Y_pred))
print(accuracy_score(Y_test,Y_pred))
print(classification_report(Y_test,Y_pred))
```

```
[[2411  345]
 [ 510 2172]]
0.8427730783376242
              precision    recall  f1-score   support

         0.0       0.83      0.87      0.85      2756
         1.0       0.86      0.81      0.84      2682

    accuracy                           0.84      5438
   macro avg       0.84      0.84      0.84      5438
weighted avg       0.84      0.84      0.84      5438
```

Random forrest accuracy is 82.83

✓ 0s completed at 8:49 PM