M505_Indiviual_Final_Project(Jupyter Notebook_code)_GH1019253

March 22, 2022

1 Final Assessment for M505(Intro A.I and Machine Learing) Group B

[1]:
```
from IPython.display import Image
Image("GISMA_LOGO.png",width = 200, height = 200)
```

[1]:

1.0.1 IBM Human Resource Analytics Employees Attrition Data Set

2 Table of the Content

5.2) Getting all the Column names and shape of dataframe

5.3) Checking the dataset for null values and data types

5.4) Looking for Unique values in each Columns

5.5) Categorizing the data according to its type

5.6) Encoding the target feature

5.7) Plotting the data to visualize relations between the columns

2.0.6 6) Data Preprocessing and Feature Engineering

6.1) Looking for outliers in DataFrame

6.2)Encoding Categorical columns and Dropping unwanted columns

2.0.7 7) Model Building on the Original data

7.1) Splitting the Data into Train and Test Set

7.2) Logistic Regression Classifier

7.3) Navie Bayes

7.4) Decision Tree Classifier

2.0.8 8) Hyperparameter Tuning

8.1) Random Forest

8.2) Gradient Boosting

2.0.9 9) Feature Engineering on Imbalanced DataSet

9.1) Logistic Regression after Feature Engineering

9.2) Naive Bayes after Feature Engineering

9.3) Decision Tree after Feature Engineering

9.4) Random Forest after Feature Engineering

2.0.10 10) Backward Elimination on Imbalanced Data

10.1) Logistic Regression after backward elimination

10.2) Naive Bayes after Backward Elimination

10.3) Decision Tree after Backward Elimination

10.4) Random Forest after Backward Elimination

2.0.11 11) Balancing the Dataset using Smote and StandardScaler

11.1) Logistic Regression after Balacing the Data

11.2) Naive Bayes after Balacing the Data

11.3) Decision Tree after Balacing the Data

11.4) Random Forest after Balacing the Data

11.5) Gradient Boosting after Balacing the Data

2.0.12 12) Feature Selection on Balanced Dataset

2.0.13 13) Evaluation of the Models we have created So Far

2.0.14 14) Conclusion & Actionable Insights

3 1) Introduction

Attrition in the industry is getting critical these days specially after pandemic, most of the Industries had to cut down there staff in half, back in the days due to covid, and the impact of that lasted long and still getting critical as now the employees are more leaning towards leaving the company if anything bad happens or they don't get what they thougth before joining the company in other words we can say that loyality of the employees is getting hard to achive and due to that integrity of the company is oh high risk."Attrition is said to be the gradual reduction in the numberof employees through retirement, resignation or death. It can also be said as Employee Turnover or Employee Defection".A thoroughly prepared and welladapted worker leaves the association, it makes a vacuum. Along these lines, the association loses key abilities, information and business connections.Present day managers are incredibly keen on decreasing Attrition rate in the association, so that it will add to the greatest adequacy,development, and progress of the organization. Subsequently, we want a strategies, calculations to forecast of representative wearing down utilizing different data mining procedures.

4 2) Problem Statement

My Clients organization is worried about elevated degree of Attrition rate and because of which their trustworthiness can be on the high gamble, They have employed me as a Data Scientist to sort out what are the realities that influences the whittling down rate and to carry the experiences of information to the company directors so they can attempt to settle on choice which can help their company.The Dataset we will utilize is given by IBM on Kaggle(www.kaggle.com).

In the event that we examine about the kind of Data we have, it depends on mathematical and categorical data, we have 1500 entries roughly. We have 35 unique columns and 1470 rows.This data is delevelope by IBM itself by doing survey. We will be chipping away at this dataset to figure out the bits of knowledge we will attempt Multiple Data Science techinques to find out and predict causes of attrition in future so that company can counter this hurdles in coming future.

My job is to build a Machine Learning pipeline and find out the best algorithm which can predict the Attrition in future and handover that pipeline to my Senior Data Scientist to give the answers to the management of the company.

Note in previous pipeline we have answered the business questions efficiently and now we are looking for the best classification Technique.

5 3) Methodology and Approach

This Attrition dataset is the internal data of the organization, which is hard to get, and a few data has a specific level of secrecy, in this manner my Machine Learning pipeline will utilized the dataset provided by IBM on kaggle. The sample size of this given dataset is 1471, there are 35 columns in total which are unique, Mainly divided into two kinds , one is categorical data which we will convert into Numerical data using "OneHotEncoder" and other is numerical one.We we build a ML pipeline in a manner that first we will visualize data and explore the data in depth that what kind of data we have(I.e it's datatype , missing or duplicate value etc.), then we will look for corelations in the data, later on we will do some feature engineering ad filter out the important and relevent features and then we will apply some machine learning models and compare them that which one will work the best.

6 4) Importing Libraries

[2]:
```
import sys

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import missingno

import warnings
warnings.filterwarnings('ignore')
import os
```

7 5) Data Exploration

7.1 5.1) Reading the data set using pandas csv read function

[3]:
```
HR_Emp_df = pd.read_csv("data/HR_Employee_data.csv")
HR_Emp_df.head()
```

[3]:
```
   Age Attrition     BusinessTravel  DailyRate              Department  \
0   41       Yes      Travel_Rarely       1102                   Sales
1   49        No  Travel_Frequently        279  Research & Development
2   37       Yes      Travel_Rarely       1373  Research & Development
3   33        No  Travel_Frequently       1392  Research & Development
4   27        No      Travel_Rarely        591  Research & Development

   DistanceFromHome  Education EducationField  EmployeeCount  EmployeeNumber  \
0                 1          2  Life Sciences              1               1
1                 8          1  Life Sciences              1               2
2                 2          2          Other              1               4
3                 3          4  Life Sciences              1               5
4                 2          1        Medical              1               7

   …  RelationshipSatisfaction StandardHours  StockOptionLevel  \
0  …                         1            80                 0
1  …                         4            80                 1
2  …                         2            80                 0
3  …                         3            80                 0
4  …                         4            80                 1

   TotalWorkingYears  TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
0                  8                      0               1               6
1                 10                      3               3              10
2                  7                      3               3               0
3                  8                      3               3               8
4                  6                      3               3               2

  YearsInCurrentRole  YearsSinceLastPromotion  YearsWithCurrManager
0                  4                        0                     5
1                  7                        1                     7
2                  0                        0                     0
3                  7                        3                     0
4                  2                        2                     2

[5 rows x 35 columns]
```

7.2 5.2) Getting all the Column names and shape of dataframe

[4]:
```
HR_Emp_df.columns
```

[4]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
```

[5]:
```
HR_Emp_df.shape
```

[5]:
```
(1470, 35)
```

7.3 5.3) Checking the dataset for null values and data types

[6]:
```
HR_Emp_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EmployeeNumber            1470 non-null   int64
 10  EnvironmentSatisfaction   1470 non-null   int64
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64
 13  JobInvolvement            1470 non-null   int64
 14  JobLevel                  1470 non-null   int64
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64
 17  MaritalStatus             1470 non-null   object
```

```
 18  MonthlyIncome             1470 non-null   int64
 19  MonthlyRate               1470 non-null   int64
 20  NumCompaniesWorked        1470 non-null   int64
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64
 24  PerformanceRating         1470 non-null   int64
 25  RelationshipSatisfaction  1470 non-null   int64
 26  StandardHours             1470 non-null   int64
 27  StockOptionLevel          1470 non-null   int64
 28  TotalWorkingYears         1470 non-null   int64
 29  TrainingTimesLastYear     1470 non-null   int64
 30  WorkLifeBalance           1470 non-null   int64
 31  YearsAtCompany            1470 non-null   int64
 32  YearsInCurrentRole        1470 non-null   int64
 33  YearsSinceLastPromotion   1470 non-null   int64
 34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

[7]:
```
HR_Emp_df.describe()
```

[7]:
```
               Age    DailyRate  DistanceFromHome    Education  EmployeeCount  \
count  1470.000000  1470.000000       1470.000000  1470.000000         1470.0
mean     36.923810   802.485714          9.192517     2.912925            1.0
std       9.135373   403.509100          8.106864     1.024165            0.0
min      18.000000   102.000000          1.000000     1.000000            1.0
25%      30.000000   465.000000          2.000000     2.000000            1.0
50%      36.000000   802.000000          7.000000     3.000000            1.0
75%      43.000000  1157.000000         14.000000     4.000000            1.0
max      60.000000  1499.000000         29.000000     5.000000            1.0

       EmployeeNumber  EnvironmentSatisfaction   HourlyRate  JobInvolvement  \
count     1470.000000              1470.000000  1470.000000     1470.000000
mean      1024.865306                 2.721769    65.891156        2.729932
std        602.024335                 1.093082    20.329428        0.711561
min          1.000000                 1.000000    30.000000        1.000000
25%        491.250000                 2.000000    48.000000        2.000000
50%       1020.500000                 3.000000    66.000000        3.000000
75%       1555.750000                 4.000000    83.750000        3.000000
max       2068.000000                 4.000000   100.000000        4.000000

          JobLevel  …  RelationshipSatisfaction  StandardHours  \
count  1470.000000  …               1470.000000         1470.0
mean      2.063946  …                  2.712245           80.0
std       1.106940  …                  1.081209            0.0
min       1.000000  …                  1.000000           80.0
```

```
25%        1.000000  ...                    2.000000           80.0
50%        2.000000  ...                    3.000000           80.0
75%        3.000000  ...                    4.000000           80.0
max        5.000000  ...                    4.000000           80.0

       StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  \
count       1470.000000        1470.000000            1470.000000
mean           0.793878          11.279592               2.799320
std            0.852077           7.780782               1.289271
min            0.000000           0.000000               0.000000
25%            0.000000           6.000000               2.000000
50%            1.000000          10.000000               3.000000
75%            1.000000          15.000000               3.000000
max            3.000000          40.000000               6.000000

       WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
count      1470.000000     1470.000000         1470.000000
mean          2.761224        7.008163            4.229252
std           0.706476        6.126525            3.623137
min           1.000000        0.000000            0.000000
25%           2.000000        3.000000            2.000000
50%           3.000000        5.000000            3.000000
75%           3.000000        9.000000            7.000000
max           4.000000       40.000000           18.000000

       YearsSinceLastPromotion  YearsWithCurrManager
count              1470.000000           1470.000000
mean                  2.187755              4.123129
std                   3.222430              3.568136
min                   0.000000              0.000000
25%                   0.000000              2.000000
50%                   1.000000              3.000000
75%                   3.000000              7.000000
max                  15.000000             17.000000

[8 rows x 26 columns]
```

7.4 5.4) Looking for Unique values in each Columns

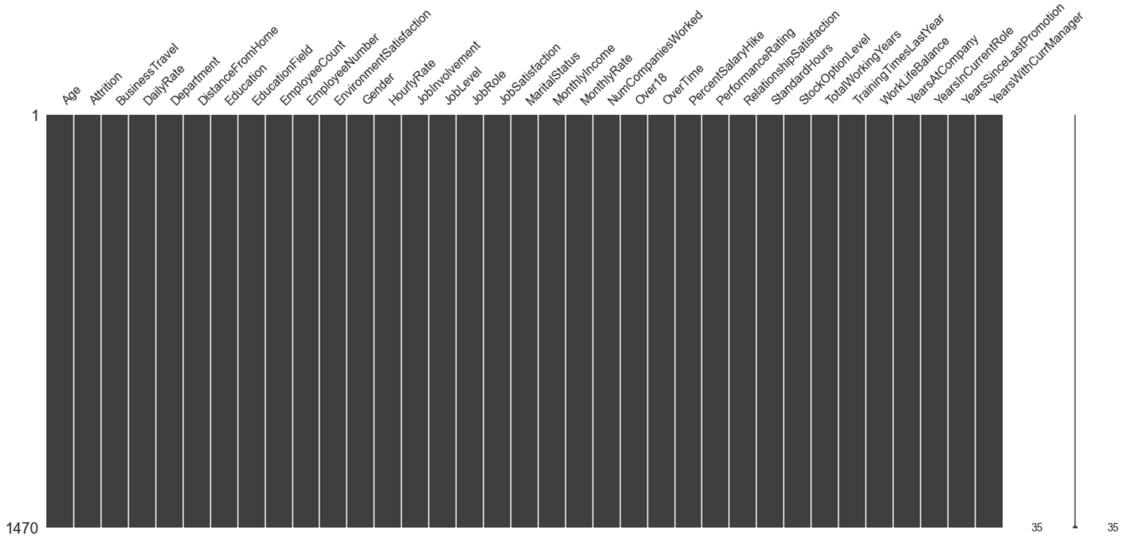
[8]:
```
for column in HR_Emp_df.columns:
    print(f"{column}:Unique values in each column No = {HR_Emp_df[column].
 ↪nunique()}")
```

```
Age:Unique values in each column No = 43
Attrition:Unique values in each column No = 2
BusinessTravel:Unique values in each column No = 3
DailyRate:Unique values in each column No = 886
```

```
Department:Unique values in each column No = 3
DistanceFromHome:Unique values in each column No = 29
Education:Unique values in each column No = 5
EducationField:Unique values in each column No = 6
EmployeeCount:Unique values in each column No = 1
EmployeeNumber:Unique values in each column No = 1470
EnvironmentSatisfaction:Unique values in each column No = 4
Gender:Unique values in each column No = 2
HourlyRate:Unique values in each column No = 71
JobInvolvement:Unique values in each column No = 4
JobLevel:Unique values in each column No = 5
JobRole:Unique values in each column No = 9
JobSatisfaction:Unique values in each column No = 4
MaritalStatus:Unique values in each column No = 3
MonthlyIncome:Unique values in each column No = 1349
MonthlyRate:Unique values in each column No = 1427
NumCompaniesWorked:Unique values in each column No = 10
Over18:Unique values in each column No = 1
OverTime:Unique values in each column No = 2
PercentSalaryHike:Unique values in each column No = 15
PerformanceRating:Unique values in each column No = 2
RelationshipSatisfaction:Unique values in each column No = 4
StandardHours:Unique values in each column No = 1
StockOptionLevel:Unique values in each column No = 4
TotalWorkingYears:Unique values in each column No = 40
TrainingTimesLastYear:Unique values in each column No = 7
WorkLifeBalance:Unique values in each column No = 4
YearsAtCompany:Unique values in each column No = 37
YearsInCurrentRole:Unique values in each column No = 19
YearsSinceLastPromotion:Unique values in each column No = 16
YearsWithCurrManager:Unique values in each column No = 18
```

[9]:
```
missingno.matrix(HR_Emp_df)
```

[9]:
```
<AxesSubplot:>
```



7.5 5.5) Categorizing the data according to its type

[10]:
```
# Classifying Categorical data and numerical data features with threshold of 9
HR_Emp_cat_df = pd.DataFrame()
HR_Emp_Num_df = pd.DataFrame()
def count_categorical(HR_Emp_df):
    for i in HR_Emp_df.columns:
        th = 9
        if len(HR_Emp_df[i].unique()) > th:
            HR_Emp_Num_df[i] = HR_Emp_df[i]
        elif len(HR_Emp_df[i].unique()) == 1:
            continue
        else:
            HR_Emp_cat_df[i] = HR_Emp_df[i]

count_categorical(HR_Emp_df)
```

[11]:
```
HR_Emp_Num_df.describe()
```

[11]:
```
               Age    DailyRate  DistanceFromHome  EmployeeNumber  \
count  1470.000000  1470.000000       1470.000000     1470.000000
mean     36.923810   802.485714          9.192517     1024.865306
std       9.135373   403.509100          8.106864      602.024335
min      18.000000   102.000000          1.000000        1.000000
25%      30.000000   465.000000          2.000000      491.250000
50%      36.000000   802.000000          7.000000     1020.500000
75%      43.000000  1157.000000         14.000000     1555.750000
max      60.000000  1499.000000         29.000000     2068.000000
```

```
       HourlyRate  MonthlyIncome   MonthlyRate  NumCompaniesWorked  \
count 1470.000000    1470.000000   1470.000000         1470.000000
mean    65.891156    6502.931293  14313.103401            2.693197
std     20.329428    4707.956783   7117.786044            2.498009
min     30.000000    1009.000000   2094.000000            0.000000
25%     48.000000    2911.000000   8047.000000            1.000000
50%     66.000000    4919.000000  14235.500000            2.000000
75%     83.750000    8379.000000  20461.500000            4.000000
max    100.000000   19999.000000  26999.000000            9.000000

      PercentSalaryHike  TotalWorkingYears  YearsAtCompany  \
count       1470.000000        1470.000000     1470.000000
mean          15.209524          11.279592        7.008163
std            3.659938           7.780782        6.126525
min           11.000000           0.000000        0.000000
25%           12.000000           6.000000        3.000000
50%           14.000000          10.000000        5.000000
75%           18.000000          15.000000        9.000000
max           25.000000          40.000000       40.000000

      YearsInCurrentRole  YearsSinceLastPromotion  YearsWithCurrManager
count        1470.000000              1470.000000           1470.000000
mean            4.229252                 2.187755              4.123129
std             3.623137                 3.222430              3.568136
min             0.000000                 0.000000              0.000000
25%             2.000000                 0.000000              2.000000
50%             3.000000                 1.000000              3.000000
75%             7.000000                 3.000000              7.000000
max            18.000000                15.000000             17.000000
```

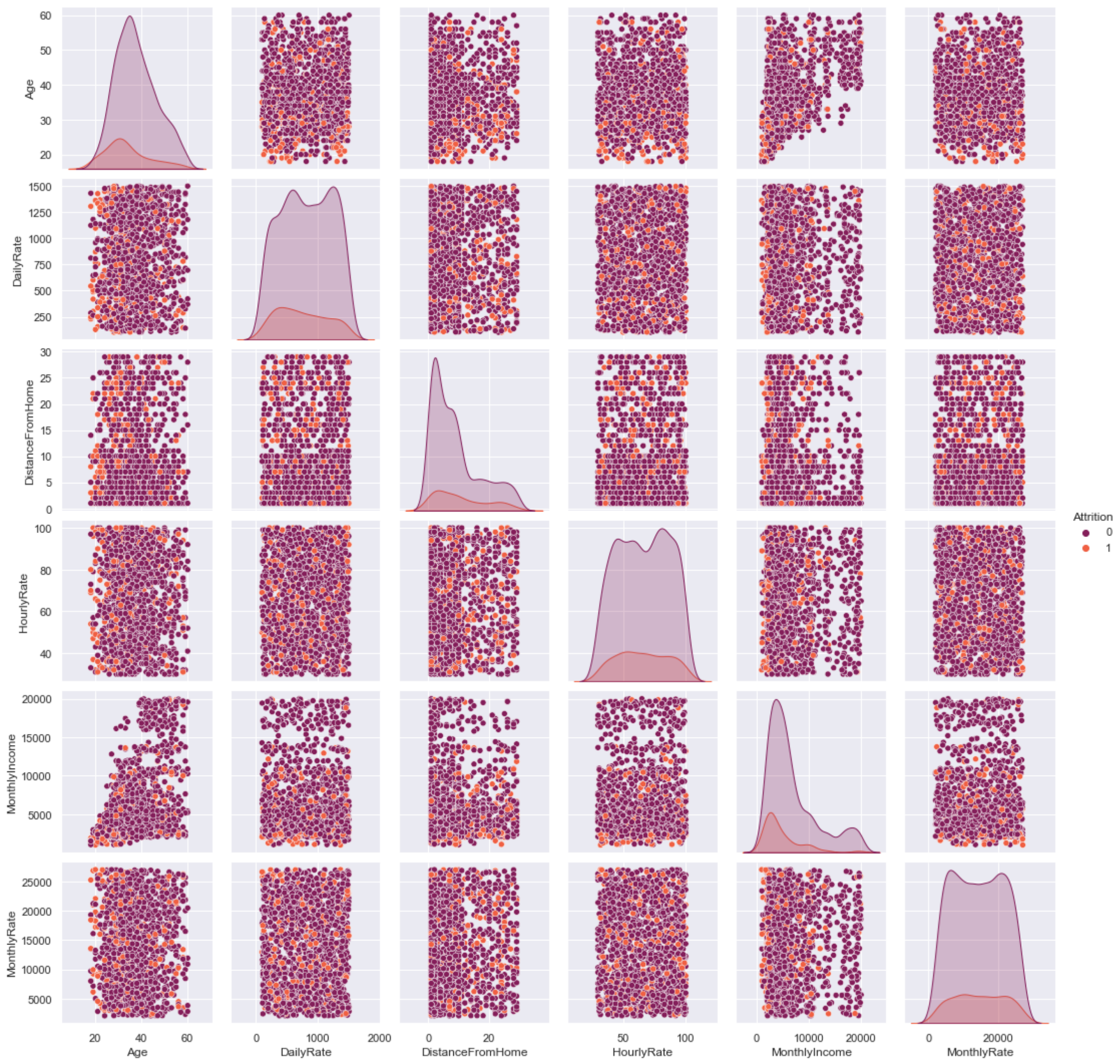
7.6 5.6) Encoding the target feature

[12]:
```
#Encode target feature
target_map = {'Yes':1, 'No':0}
HR_Emp_df['Attrition'] = HR_Emp_df["Attrition"].apply(lambda x: target_map[x])
```

- So here we are converting "Attrition" column into boolean which means that actually the "Attrition" column contain "Yes" for those employees who quit and "No" for those who stayed at the company, we have used lambda funtion to keep the code concise and converted yes and no into "1" and "0"

7.7 5.7) Plotting the data to visualize relations between the columns

[13]:
```
cols_pair = ['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate',␣
 ↪'MonthlyIncome', 'MonthlyRate', 'Attrition']
sns.pairplot(HR_Emp_df[cols_pair], diag_kind = "kde", hue='Attrition',␣
 ↪palette='rocket')
plt.show()
```



Our a large portion of the Employees lies between 27 to 40 age bunch. - A large portion of our Employees is located(lives close by) from workplace. - The Majority of our workers have level 3 of Education. - The vast majority of the workers are the part of our organization for under 10 years

[14]:
```
col = ['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate', 'MonthlyIncome',␣
 ↪'MonthlyRate']
plt.figure(figsize=(25,15))
```

```
sns.heatmap(HR_Emp_Num_df[col].corr(),cmap="BuPu", annot=True)
plt.show()
```

8 6) Data Preprocessing and Feature Engineering

- Here I am going to use multiple techniques to preprocess the Data and extarct the features out of it which are important to us.

8.1 6.1) Looking for outliers in DataFrame

[15]:
```
## Selecting columns with outlier with quantiles method

outliers = []
def search_features_with_outliers(HR_Emp_df):
    for i in HR_Emp_df.columns:
        q1 = HR_Emp_df[i].quantile(0.25)
        q3 = HR_Emp_df[i].quantile(0.75)
        for j in HR_Emp_df[i]:
            if j > q3+1.5*(q3-q1):
                outliers.append(i)
                break
            else:
                continue
```

```
    pass
    return outliers

search_features_with_outliers(HR_Emp_Num_df)
```

[15]:
```
['MonthlyIncome',
 'NumCompaniesWorked',
 'TotalWorkingYears',
 'YearsAtCompany',
 'YearsInCurrentRole',
 'YearsSinceLastPromotion',
 'YearsWithCurrManager']
```

- Here above we got the outlier we have in our data

8.2 6.2)Encoding Categorical columns and Dropping unwanted columns

[16]:
```
HR_Emp_df = pd.get_dummies(HR_Emp_df,␣
 ↪columns=['BusinessTravel','Department','EducationField','Gender','JobRole','MaritalStatus',

HR_Emp_df = HR_Emp_df.drop(['Over18','StandardHours','EmployeeCount'], axis=1)

HR_Emp_df['Attrition'].head()
```

[16]:
```
0    1
1    0
2    1
3    0
4    0
Name: Attrition, dtype: int64
```

[17]:
```
pd.set_option('display.max_columns',None)
HR_Emp_df.head()
```

[17]:
```
   Age  Attrition  DailyRate  DistanceFromHome  Education  EmployeeNumber  \
0   41          1       1102                 1          2               1
1   49          0        279                 8          1               2
2   37          1       1373                 2          2               4
3   33          0       1392                 3          4               5
4   27          0        591                 2          1               7

   EnvironmentSatisfaction  HourlyRate  JobInvolvement  JobLevel  \
0                        2          94               3         2
1                        3          61               2         2
2                        4          92               2         1
3                        4          56               3         1
4                        1          40               3         1
```

```
   JobSatisfaction  MonthlyIncome  MonthlyRate  NumCompaniesWorked  \
0                4           5993        19479                   8
1                2           5130        24907                   1
2                3           2090         2396                   6
3                3           2909        23159                   1
4                2           3468        16632                   9

   PercentSalaryHike  PerformanceRating  RelationshipSatisfaction  \
0                 11                  3                         1
1                 23                  4                         4
2                 15                  3                         2
3                 11                  3                         3
4                 12                  3                         4

   StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear  \
0                 0                  8                      0
1                 1                 10                      3
2                 0                  7                      3
3                 0                  8                      3
4                 1                  6                      3

   WorkLifeBalance  YearsAtCompany  YearsInCurrentRole  \
0                1               6                   4
1                3              10                   7
2                3               0                   0
3                3               8                   7
4                3               2                   2

   YearsSinceLastPromotion  YearsWithCurrManager  BusinessTravel_Non-Travel  \
0                        0                     5                          0
1                        1                     7                          0
2                        0                     0                          0
3                        3                     0                          0
4                        2                     2                          0

   BusinessTravel_Travel_Frequently  BusinessTravel_Travel_Rarely  \
0                                 0                             1
1                                 1                             0
2                                 0                             1
3                                 1                             0
4                                 0                             1

   Department_Human Resources  Department_Research & Development  \
0                           0                                  0
1                           0                                  1
2                           0                                  1
```

```
3                          0                                  1
4                          0                                  1

   Department_Sales  EducationField_Human Resources  \
0                 1                               0
1                 0                               0
2                 0                               0
3                 0                               0
4                 0                               0

   EducationField_Life Sciences  EducationField_Marketing  \
0                             1                         0
1                             1                         0
2                             0                         0
3                             1                         0
4                             0                         0

   EducationField_Medical  EducationField_Other  \
0                       0                     0
1                       0                     0
2                       0                     1
3                       0                     0
4                       1                     0

   EducationField_Technical Degree  Gender_Female  Gender_Male  \
0                                0              1            0
1                                0              0            1
2                                0              0            1
3                                0              1            0
4                                0              0            1

   JobRole_Healthcare Representative  JobRole_Human Resources  \
0                                  0                        0
1                                  0                        0
2                                  0                        0
3                                  0                        0
4                                  0                        0

   JobRole_Laboratory Technician  JobRole_Manager  \
0                              0                0
1                              0                0
2                              1                0
3                              0                0
4                              1                0

   JobRole_Manufacturing Director  JobRole_Research Director  \
0                               0                          0
```

```
1                                          0                                  0
2                                          0                                  0
3                                          0                                  0
4                                          0                                  0

   JobRole_Research Scientist  JobRole_Sales Executive  \
0                           0                        1
1                           1                        0
2                           0                        0
3                           1                        0
4                           0                        0

   JobRole_Sales Representative  MaritalStatus_Divorced  \
0                             0                       0
1                             0                       0
2                             0                       0
3                             0                       0
4                             0                       0

   MaritalStatus_Married  MaritalStatus_Single  OverTime_No  OverTime_Yes
0                      0                     1            0             1
1                      1                     0            1             0
2                      0                     1            0             1
3                      1                     0            0             1
4                      1                     0            1             0
```

9 7) Model Building on the Original data

- I am going to use Classification algorithms as the data is of the type where claffication algorithms can work better and I will use F1 score to predict the TP , TN , FN and FP because we are using classification algorithms and f1 score works the best for these algorithms

[18]:
```python
HR_Emp_df['Attrition'].value_counts(normalize=True)
```

[18]:
```
0    0.838776
1    0.161224
Name: Attrition, dtype: float64
```

[19]:
```python
HR_Emp_df



x = HR_Emp_df.drop(['Attrition'], axis=1)
y = HR_Emp_df['Attrition']
```

[20]:
```python
#Importing the Algorithms we are going to use

from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, classification_report, accuracy_score,␣
↪roc_auc_score, roc_curve, confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import BaggingClassifier,RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.ensemble import AdaBoostClassifier
```

9.1 7.1) Splitting the Data into Train and Test Set

[21]:
```
#Spliting data Set in training set and testing set

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30,␣
↪random_state=1)
```

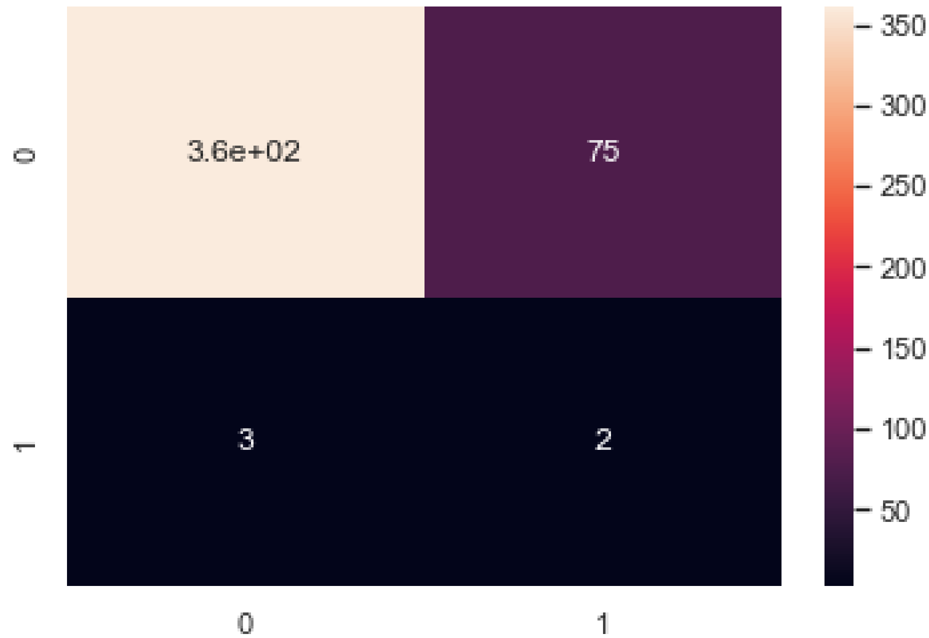
[22]:
```
Model = []
Accuracy = []
F1Score = []
```

9.2 7.2) Logistic Regression Classifier

[23]:
```
lr = LogisticRegression()
lr.fit(x_train, y_train)
lr_pred = lr.predict(x_test)
lr_pred1 = lr.predict(x_train)
print("Accuracy: {}%".format( 100 * accuracy_score(y_train, lr_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(y_test, lr_pred, average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(y_test, lr_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(y_test,lr_pred)))
```

```
Accuracy: 85.03401360544217%

f1 score [90.25        4.87804878]%

accuracy score 82.31292517006803%

roc auc score 50.88661338661339%
```

[24]:
```python
# Comparing the results using Confusion Matrix
# Testing Set Performance

con_max = confusion_matrix(lr_pred, y_test)
sns.heatmap(con_max, annot=True);
```



[25]:
```python
# Analyzing the KPI (Key Performance Indicator)

print(classification_report(y_test, lr_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.99      0.90       364
           1       0.40      0.03      0.05        77

    accuracy                           0.82       441
   macro avg       0.61      0.51      0.48       441
weighted avg       0.75      0.82      0.75       441
```

- Well after applying Logistic regression classifier on Imbalanced dataset we have got the Training score of 85% approx and testing accuracy score of 82% which means that there is no minor errors between both
- We got a very good F1 score which means our model predicted correctly.

```
[26]: Model.append('LR on Imbalanced Data')
F1Score.append(f1_score(y_test, lr_pred, average=None))
Accuracy.append(accuracy_score(y_test, lr_pred))
```

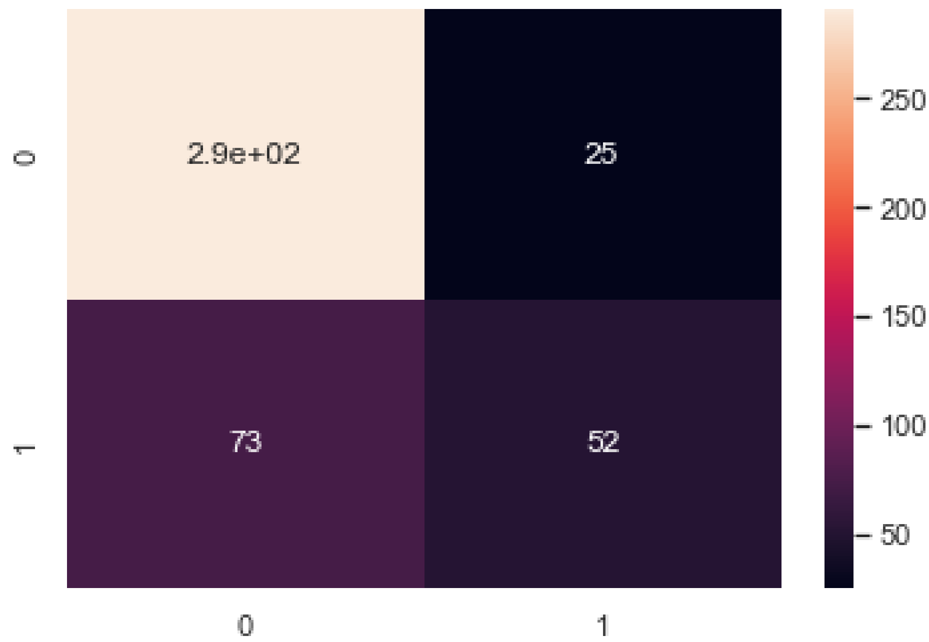
- Here we are appending this model to do comparision between other results we will get.

9.3 7.3) Navie Bayes

```
[27]: nav_by = GaussianNB()
nav_by.fit(x_train,y_train)
nav_by_pred = nav_by.predict(x_test)
nav_by_pred1 = nav_by.predict(x_train)


print("Train score: {}%".format( 100 * accuracy_score(y_train, nav_by_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(y_test, nav_by_pred, average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(y_test, nav_by_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(y_test, nav_by_pred)))
```

```
Train score: 81.5354713313897%

f1 score [85.58823529 51.48514851]%

accuracy score 77.77777777777779%

roc auc score 73.73876123876124%
```

```
[28]: # Comparing the results using Confusion Matrix
# Testing Set Performance

con_max = confusion_matrix(nav_by_pred, y_test)
sns.heatmap(con_max, annot=True);
```



[29]:
```
# Analyzing the KPI (Key Performance Indicator)

print(classification_report(y_test, nav_by_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.80      0.86       364
           1       0.42      0.68      0.51        77

    accuracy                           0.78       441
   macro avg       0.67      0.74      0.69       441
weighted avg       0.83      0.78      0.80       441
```

- Naive bayes classifier on Imbalanced dataset given us the training accuracy of 81% and testing accuracy is 77% which is less than Logistic regression
- We didn't got a better F1 score if we compare to Logistic Regression

[30]:
```
Model.append('NB on Imbalanced Data')
F1Score.append(f1_score(y_test, nav_by_pred, average=None))
Accuracy.append(accuracy_score(y_test, nav_by_pred))
```

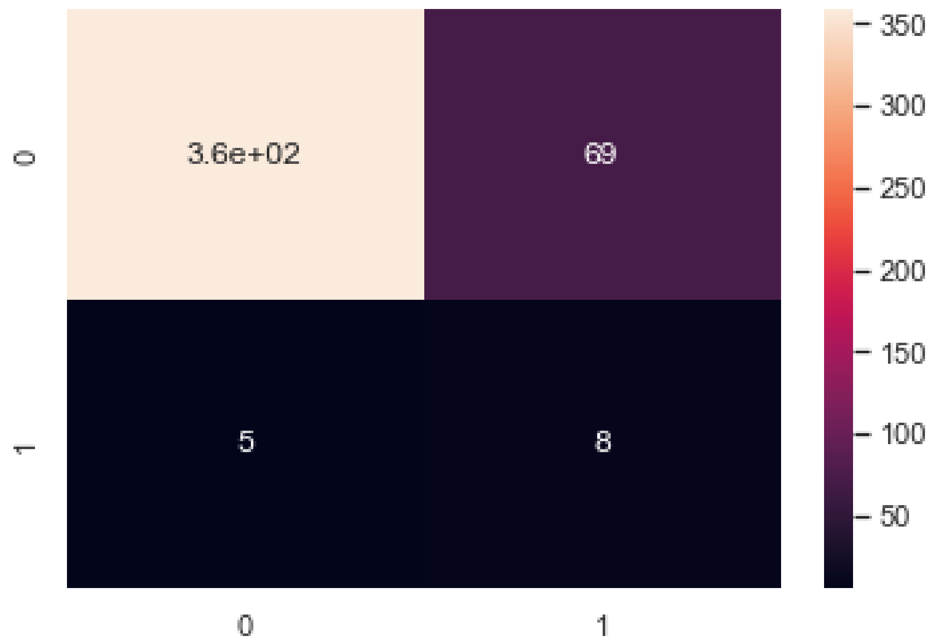
9.4 7.4) Decision Tree Classifier

- Using Gini as Criterion

[31]:
```
#Here I am using Gini as Criterion

gini = DecisionTreeClassifier(criterion='gini', random_state=100, max_depth=3,␣
 ↪min_samples_leaf=5)
gini.fit(x_train, y_train)
gini_pred = gini.predict(x_test)
gini_pred1= gini.predict(x_train)

print("Train score: {}%".format( 100 * accuracy_score(y_train, gini_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(y_test, gini_pred, average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(y_test, gini_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(y_test, gini_pred)))
```

```
Train score: 86.10301263362487%

f1 score [90.65656566 17.77777778]%

accuracy score 83.21995464852607%

roc auc score 54.50799200799202%
```

[32]:
```
# Comparing the results using Confusion Matrix
# Testing Set Performance

con_max = confusion_matrix(gini_pred, y_test)
sns.heatmap(con_max, annot=True);
```



[33]:
```
print(classification_report(y_test, gini_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.99      0.91       364
           1       0.62      0.10      0.18        77

    accuracy                           0.83       441
   macro avg       0.73      0.55      0.54       441
weighted avg       0.80      0.83      0.78       441
```

- We have noticed that decision tree worked better than above 2 algorithms and given us the best Training acurracy score on Imbalaced dataset so far which is 86% but still we have 3% difference while comparing to Testing Accuracy
- We got a very good F1 score which means our model predicted correctly if we compare to above 2 algorithms

[34]:
```
Model.append('Decision Tree on Imbalanced Data')
F1Score.append(f1_score(y_test, gini_pred, average=None))
Accuracy.append(accuracy_score(y_test, gini_pred))
```

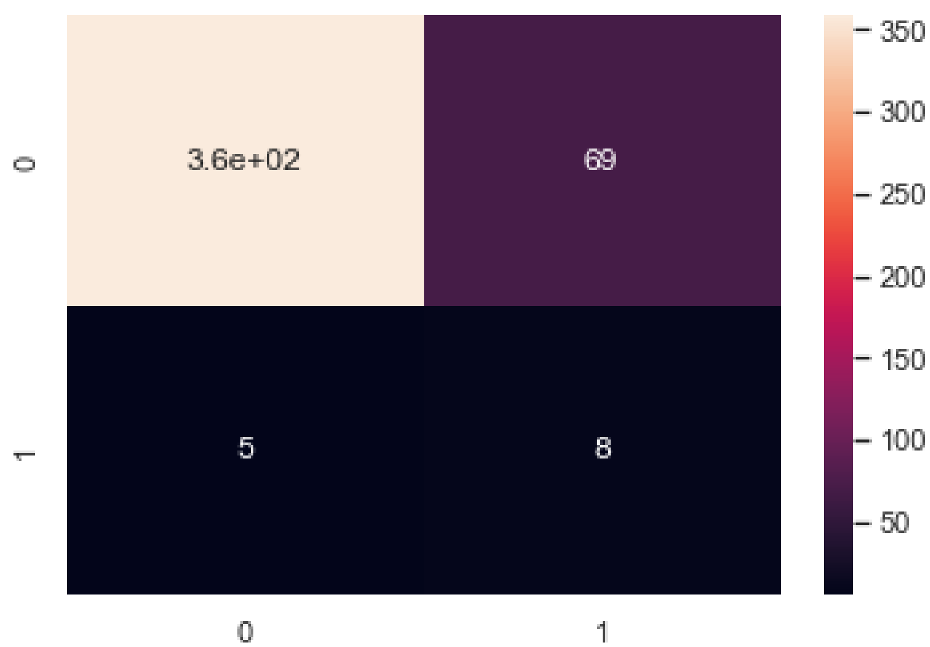
- Using Entropy as Criterion to see what difference will it make

[35]:
```
#Here I am trying Entropy as Criterion to see what difference will it make
```

```
entro = DecisionTreeClassifier(criterion='entropy', random_state=100,␣
↪max_depth=3, min_samples_leaf=5)
entro.fit(x_train, y_train)
entro_pred = entro.predict(x_test)
entro_pred1 = entro.predict(x_train)
print("Train score: {}%".format( 100 * accuracy_score(y_train, entro_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(y_test, entro_pred, average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(y_test, entro_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(y_test, entro_pred)))
```

```
Train score: 86.10301263362487%

f1 score [90.65656566 17.77777778]%

accuracy score 83.21995464852607%

roc auc score 54.50799200799202%
```

[36]:
```
# Comparing the results using Confusion Matrix
# Testing Set Performance

con_max = confusion_matrix(entro_pred, y_test)
sns.heatmap(con_max, annot=True);
```



[37]:
```
print(classification_report(y_test, entro_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.99      0.91       364
           1       0.62      0.10      0.18        77

    accuracy                           0.83       441
   macro avg       0.73      0.55      0.54       441
weighted avg       0.80      0.83      0.78       441
```

- Well after applying Entropy we haven't notice any difference.

10 8) Hyperparameter Tuning

- Let's try tuning the model to see if we can further improve the accuracy

[38]:
```
classifiers = { "LogisiticRegression": LogisticRegression()}
```

[39]:
```
from sklearn.model_selection import cross_val_score


for key, classifier in classifiers.items():
    classifier.fit(x_train, y_train)
    training_score = cross_val_score(classifier, x_train, y_train, cv=5)
    print("Classifiers: ", classifier.__class__.__name__, "Has a training score␣
 ↪of", round(training_score.mean(), 2) * 100, "% accuracy score")
training_score.mean()
print("ROC_AUC_Accuracy: {}%".format( 100 *training_score.mean()))
```

```
Classifiers:  LogisticRegression Has a training score of 85.0 % accuracy score
ROC_AUC_Accuracy: 84.93724840161023%
```

Hence we can see that after tuning the logistic regression model, we got the accuracy of 85% which is same before tuning so we will try to implement some other classification models and try to get better results.

Here we are going to use Grid Search and try to find which Parameters are working better.

10.1 8.1) Random Forest

[40]:
```
params = {

    'n_estimators':[50,100,150,200],
    'criterion':['gini','entropy'],
    #'splitter':['best','random'],
    'max_depth':[5,10,15],
    'max_leaf_nodes':range(2,10,1),
    'max_features':['auto','log2']

}

ran_for = RandomForestClassifier()

gs = GridSearchCV(estimator=ran_for, param_grid=params, cv=3,scoring='recall',␣
 ↪n_jobs=-1)
gs.fit(x,y)
```

[40]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [5, 10, 15],
                         'max_features': ['auto', 'log2'],
                         'max_leaf_nodes': range(2, 10),
                         'n_estimators': [50, 100, 150, 200]},
             scoring='recall')
```

- Here i have provided some parameters to RandomForestClassifier to check which will work the best among them and will it improve over accurracy score.

[41]:
```
gs.best_params_
```

[41]:
```
{'criterion': 'gini',
 'max_depth': 10,
 'max_features': 'auto',
 'max_leaf_nodes': 8,
 'n_estimators': 50}
```

- From the given parameters these are the best which can be used to build our model.

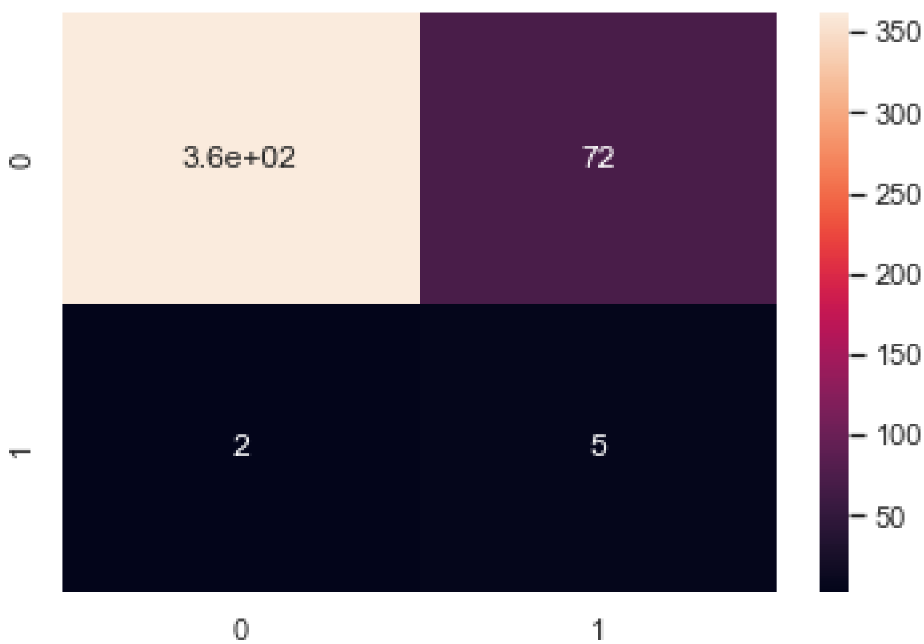
Now we are running the RandomForestClassifier again on the best suitable parameters we got

[42]:
```
ran_for = RandomForestClassifier(**gs.best_params_)
ran_for.fit(x_train,y_train)
y_pred_rf = ran_for.predict(x_test)
y_pred_rf1 = ran_for.predict(x_train)
```

```
[43]: print("Train score: {}%".format( 100 * accuracy_score(y_train, y_pred_rf1)))
print()
print("f1 score {}%".format( 100 * f1_score(y_test, y_pred_rf, average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(y_test, y_pred_rf)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(y_test,y_pred_rf)))
```

```
Train score: 86.29737609329446%

f1 score [90.72681704 11.9047619 ]%

accuracy score 83.21995464852607%

roc auc score 52.972027972027966%
```

```
[44]: # Comparing the results using Confusion Matrix
# Testing Set Performance

con_max = confusion_matrix(y_pred_rf, y_test)
sns.heatmap(con_max, annot=True);
```




```
[45]: print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support
```

```
              0       0.83      0.99      0.91       364
              1       0.71      0.06      0.12        77

       accuracy                           0.83       441
      macro avg       0.77      0.53      0.51       441
   weighted avg       0.81      0.83      0.77       441
```

- So what we have noticed that even after selecting the best parameters we didn't got any better result on the Imbalaced data we provided to the model

[46]:
```
Model.append('Random Forest on Imbalanced Data')
F1Score.append(f1_score(y_test, y_pred_rf, average=None))
Accuracy.append(accuracy_score(y_test, y_pred_rf))
```

10.2 8.2) Gradient Boosting

- Now we are applying Gradient Boosting algorithm to see what result we can get from this classification Algorithm

[47]:
```
gb_params ={
    'n_estimators': 1500,
    'max_features': 0.9,
    'learning_rate' : 0.25,
    'max_depth': 4,
    'min_samples_leaf': 2,
    'subsample': 1,
    'max_features' : 'sqrt',
    'verbose': 0
}
```

[48]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

[49]:
```
grad_boost = GradientBoostingClassifier(**gb_params)
grad_boost.fit(x_train, y_train)
grad_boost_pred = grad_boost.predict(x_test)
grad_boost_pred1 = grad_boost.predict(x_train)
```

[50]:
```
print("Train score: {}%".format( 100 * accuracy_score(y_train,␣
 ↪grad_boost_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(y_test, grad_boost_pred,␣
 ↪average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(y_test,␣
 ↪grad_boost_pred)))
```

```
print()
print("roc auc score {}%".format( 100 *roc_auc_score(y_test, grad_boost_pred)))
```

```
Train score: 100.0%

f1 score [90.90909091 37.5       ]%

accuracy score 84.12698412698413%

roc auc score 61.713286713286706%
```

[51]:
```
print(classification_report(y_test, grad_boost_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.91       364
           1       0.60      0.27      0.37        77

    accuracy                           0.84       441
   macro avg       0.73      0.62      0.64       441
weighted avg       0.82      0.84      0.82       441
```

Note:

- Well after applying Gradient Boosting classifier on Imbalanced dataset we have got the Training score of 100% and testing accuracy score of 84% which means that there is a Huge difference in both and this model seems to be overfitted. Which means that it learned rules specifically for the train set, those rules do not generalize well beyond the train set.

[52]:
```
Model.append('Gradient Boosting on Imbalanced Data')
F1Score.append(f1_score(y_test, grad_boost_pred, average=None))
Accuracy.append(accuracy_score(y_test, grad_boost_pred))
```

11 9) Feature Engineering on Imbalanced DataSet

[53]:
```
emp_hr_df_new = HR_Emp_df.copy()
```

[54]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()
Scaled_X = min_max_scaler.fit_transform(emp_hr_df_new.drop('Attrition',axis=1))
Y=emp_hr_df_new.Attrition.values
X_new = SelectKBest(chi2, k=2).fit_transform(Scaled_X, Y)
```

```
x_train_f,x_test_f,y_train_f,y_test_f=train_test_split(X_new,Y,test_size=0.
↪30,random_state=1)
```

- Here we are going SelectKBest and chi2 to extract the best features out of the given dataset.

11.1 9.1) Logistic Regression after Feature Engineering

[55]:
```
LRC = LogisticRegression()
LRC.fit(x_train_f, y_train_f)
LRC_pred = LRC.predict(x_test_f)
LRC_pred1 = LRC.predict(x_train_f)

print("Train Acurracy score: {}%".format( 100 * accuracy_score(y_train_f,␣
↪LRC_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(y_test_f, LRC_pred, average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(y_test_f, LRC_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(y_test_f, LRC_pred)))
```

```
Train Acurracy score: 85.03401360544217%

f1 score [90.58971142 11.76470588]%

accuracy score 82.99319727891157%

roc auc score 52.83466533466533%
```

[56]:
```
print(classification_report(y_test, LRC_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.99      0.91       364
           1       0.62      0.06      0.12        77

    accuracy                           0.83       441
   macro avg       0.73      0.53      0.51       441
weighted avg       0.80      0.83      0.77       441
```

- Well after applying Logistic Regression on Imbalanced dataset after Feature Engineering we have got the Training score of 85% and testing accuracy score of 83% approx which mean we have improved the difference between training and testing set

11.2 9.2) Naive Bayes after Feature Engineering

[57]:
```
nav_bay = GaussianNB()
nav_bay.fit(x_train_f,y_train_f)
nav_bay_pred = nav_bay.predict(x_test_f)
nav_bay_pred1 = nav_bay.predict(x_train_f)

print("Train score: {}%".format( 100 * accuracy_score(y_train_f,␣
↪nav_bay_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(y_test_f, nav_bay_pred,␣
↪average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(y_test_f,␣
↪nav_bay_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(y_test_f,nav_bay_pred)))
```

```
Train score: 83.38192419825073%

f1 score [89.31788932 20.95238095]%

accuracy score 81.17913832199547%

roc auc score 54.8076923076923%
```

[58]:
```
print(classification_report(y_test, nav_bay_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.95      0.89       364
           1       0.39      0.14      0.21        77

    accuracy                           0.81       441
   macro avg       0.62      0.55      0.55       441
weighted avg       0.76      0.81      0.77       441
```

- Well after applying Naive Bayes on Imbalanced dataset after Feature Engineering we have got the Training score of 83% and testing accuracy score of 81% approx which mean we have improved the difference between training and testing set after applying feature engineering.

11.3 9.3) Decision Tree after Feature Engineering

[59]:
```
DT_gn = DecisionTreeClassifier(criterion='gini', random_state=100, max_depth=3,␣
↪min_samples_leaf=5)
DT_gn.fit(x_train_f, y_train_f)
DT_gn_pred = DT_gn.predict(x_test_f)
```

```
DT_gn_pred1= DT_gn.predict(x_train_f)


print("Train score: {}%".format( 100 * accuracy_score(y_train_f, DT_gn_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(y_test_f, DT_gn_pred,␣
↪average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(y_test_f, DT_gn_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(y_test_f,DT_gn_pred)))
```

```
Train score: 85.03401360544217%

f1 score [90.58971142 11.76470588]%

accuracy score 82.99319727891157%

roc auc score 52.83466533466533%
```

[60]:
```
print(classification_report(y_test, DT_gn_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.99      0.91       364
           1       0.62      0.06      0.12        77

    accuracy                           0.83       441
   macro avg       0.73      0.53      0.51       441
weighted avg       0.80      0.83      0.77       441
```

- Well after applying Feature Engineering we noticed that difference between Training and Testing score is improved.

11.4 9.4) Random Forest after Feature Engineering

[61]:
```
params = {

    'n_estimators':range(10,100,10),
    'criterion':['gini','entropy'],
    #'splitter':['best','random'],
    'max_depth':range(2,10,1),
    'max_leaf_nodes':range(2,10,1),
    'max_features':['auto','log2']

}
```

```
Ran_ft = RandomForestClassifier()

Grid_Sr =␣
 ↪GridSearchCV(estimator=Ran_ft,param_grid=params,cv=3,scoring='recall',n_jobs=-1)
Grid_Sr.fit(X_new,Y)
```

[61]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 10),
                         'max_features': ['auto', 'log2'],
                         'max_leaf_nodes': range(2, 10),
                         'n_estimators': range(10, 100, 10)},
             scoring='recall')
```

[62]:
```
Grid_Sr.best_params_
```

[62]:
```
{'criterion': 'gini',
 'max_depth': 2,
 'max_features': 'auto',
 'max_leaf_nodes': 4,
 'n_estimators': 10}
```

[63]:
```
Ran_ft = RandomForestClassifier(**gs.best_params_)
Ran_ft.fit(x_train_f,y_train_f)
y_pred_rf = Ran_ft.predict(x_test_f)
y_pred_rf1 = Ran_ft.predict(x_train_f)
```

[64]:
```
print("Train score: {}%".format( 100 * accuracy_score(y_train_f, y_pred_rf1)))
print()
print("f1 score {}%".format( 100 * f1_score(y_test_f, y_pred_rf, average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(y_test_f, y_pred_rf)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(y_test_f, y_pred_rf)))
```

```
Train score: 85.03401360544217%

f1 score [90.58971142 11.76470588]%

accuracy score 82.99319727891157%

roc auc score 52.83466533466533%
```

[65]:
```
print(classification_report(y_test_f, y_pred_rf))
```

```
              precision    recall  f1-score   support
```

```
              0       0.83      0.99      0.91       364
              1       0.62      0.06      0.12        77

       accuracy                           0.83       441
      macro avg       0.73      0.53      0.51       441
   weighted avg       0.80      0.83      0.77       441
```

- So far after applying feature Selection we have noticed that our difference between Train and Test Accuracy

Note:

- Well now after applying feature engineering using SelectKbest and Chi2 we have noticed a bit improvement in our Pipeline so now we are applying one more Technique to see if can get any better results.So what we gonna do is that we are applying Backward Elimination so what does it do or how it works, it selects the optimal number of features from the given dataset by selecting all of them and searching for best P-value it's repetative task so it might slow down over pipeline but as for now time is not our concern but we want to eliminate the errors by selecting optimal features so thats why we are going with Backward elimination.

12 10) Backward Elimination on Imbalanced Data

[66]:
```
import statsmodels.api as sm

cols = list(x.columns)
pmax = 1
while (len(cols)>0):
    p= []
    X_1 = x[cols]
    X_1 = sm.add_constant(X_1)
    model = sm.OLS(y,X_1).fit()
    p = pd.Series(model.pvalues.values[1:],index = cols)
    pmax = max(p)
    feature_with_p_max = p.idxmax()
    if(pmax>0.05):
        cols.remove(feature_with_p_max)
    else:
        break
selected_features_BE = cols
print(selected_features_BE)
```

```
['Age', 'DistanceFromHome', 'EnvironmentSatisfaction', 'JobInvolvement',
'JobSatisfaction', 'NumCompaniesWorked', 'RelationshipSatisfaction',
'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
'YearsInCurrentRole', 'YearsSinceLastPromotion',
'BusinessTravel_Travel_Frequently', 'BusinessTravel_Travel_Rarely',
'Department_Sales', 'EducationField_Human Resources', 'EducationField_Technical
```

```
Degree', 'Gender_Female', 'Gender_Male', 'JobRole_Laboratory Technician',
'JobRole_Sales Representative', 'MaritalStatus_Divorced',
'MaritalStatus_Married', 'MaritalStatus_Single', 'OverTime_No', 'OverTime_Yes']
```

[67]:
```
x_new = HR_Emp_df[selected_features_BE]
x_new.head()
```

[67]:
```
   Age  DistanceFromHome  EnvironmentSatisfaction  JobInvolvement  \
0   41                 1                        2               3
1   49                 8                        3               2
2   37                 2                        4               2
3   33                 3                        4               3
4   27                 2                        1               3

   JobSatisfaction  NumCompaniesWorked  RelationshipSatisfaction  \
0                4                   8                         1
1                2                   1                         4
2                3                   6                         2
3                3                   1                         3
4                2                   9                         4

   TotalWorkingYears  TrainingTimesLastYear  WorkLifeBalance  \
0                  8                      0                1
1                 10                      3                3
2                  7                      3                3
3                  8                      3                3
4                  6                      3                3

   YearsInCurrentRole  YearsSinceLastPromotion  \
0                   4                        0
1                   7                        1
2                   0                        0
3                   7                        3
4                   2                        2

   BusinessTravel_Travel_Frequently  BusinessTravel_Travel_Rarely  \
0                                 0                             1
1                                 1                             0
2                                 0                             1
3                                 1                             0
4                                 0                             1

   Department_Sales  EducationField_Human Resources  \
0                 1                               0
1                 0                               0
2                 0                               0
3                 0                               0
```

```
4                 0                              0

   EducationField_Technical Degree  Gender_Female  Gender_Male  \
0                                0              1            0
1                                0              0            1
2                                0              0            1
3                                0              1            0
4                                0              0            1

   JobRole_Laboratory Technician  JobRole_Sales Representative  \
0                              0                             0
1                              0                             0
2                              1                             0
3                              0                             0
4                              1                             0

   MaritalStatus_Divorced  MaritalStatus_Married  MaritalStatus_Single  \
0                       0                      0                     1
1                       0                      1                     0
2                       0                      0                     1
3                       0                      1                     0
4                       0                      1                     0

   OverTime_No  OverTime_Yes
0            0             1
1            1             0
2            0             1
3            0             1
4            1             0
```

[68]:
```
y_new = HR_Emp_df['Attrition']
```

[69]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x_new, y_new, test_size=0.3,␣
 ↪random_state=1)
```

12.1 10.1) Logistic Regression after backward elimination

[70]:
```
lr = LogisticRegression()
lr.fit(xtrain, ytrain)
lr_pred = lr.predict(xtest)
lr_pred1 = lr.predict(xtrain)


print("Train score: {}%".format( 100 * accuracy_score(ytrain, lr_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(ytest, lr_pred, average=None)))
print()
```

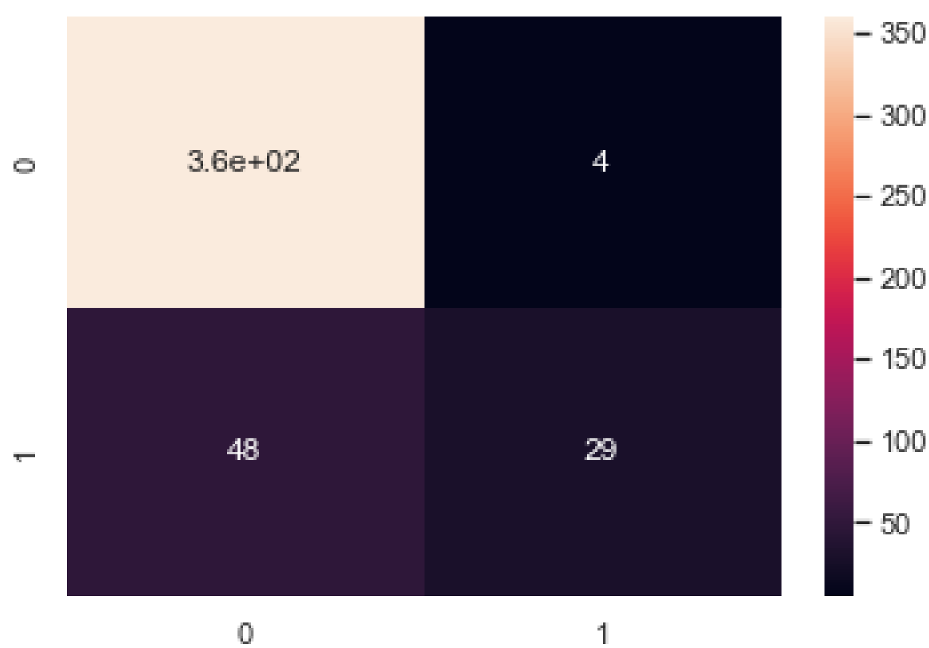
```
print("accuracy score {}%".format( 100 * accuracy_score(ytest, lr_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(ytest, lr_pred)))
```

```
Train score: 89.0184645286686%

f1 score [93.2642487  52.72727273]%

accuracy score 88.20861678004536%

roc auc score 68.2817182817183%
```

[71]:
```
print(classification_report(ytest,lr_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.93       364
           1       0.88      0.38      0.53        77

    accuracy                           0.88       441
   macro avg       0.88      0.68      0.73       441
weighted avg       0.88      0.88      0.86       441
```

[73]:
```
# Testing Set Performance

con_max = confusion_matrix(ytest,lr_pred)
sns.heatmap(con_max, annot=True);
```



- As we can see our model is much more improved our Testing accuracy is now very close to our training accuracy which is a very good Sign.

[72]:
```
Model.append('LR on Imbalanced Data after feature selection')
F1Score.append(f1_score(ytest, lr_pred, average=None))
Accuracy.append(accuracy_score(ytest, lr_pred))
```

12.2 10.2) Naive Bayes after Backward Elimination

[74]:
```
nb = GaussianNB()
nb.fit(xtrain,ytrain)
nb_pred = nb.predict(xtest)
nb_pred1 = nb.predict(xtrain)


print("Train score: {}%".format( 100 * accuracy_score(ytrain, nb_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(ytest, nb_pred, average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(ytest, nb_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(ytest, nb_pred)))
```

```
Train score: 86.20019436345967%

f1 score [89.31506849 48.68421053]%

accuracy score 82.31292517006803%

roc auc score 68.80619380619382%
```

[75]:
```
print(classification_report(ytest,nb_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.90      0.89       364
           1       0.49      0.48      0.49        77

    accuracy                           0.82       441
   macro avg       0.69      0.69      0.69       441
weighted avg       0.82      0.82      0.82       441
```

- Here we can see the change in the accuracy between the both training one and testing one is bit improved after backward Elimination, but still on Naive Bayes it didn't impacted that much like LR.

```
[76]: Model.append('NB on Imbalanced Data after feature selection')
F1Score.append(f1_score(ytest, nb_pred, average=None))
Accuracy.append(accuracy_score(ytest, nb_pred))
```

12.3 10.3) Decision Tree after Backward Elimination

```
[77]: gm = DecisionTreeClassifier(criterion='gini', random_state=100, max_depth=3,␣
  ↪min_samples_leaf=5)
gm.fit(xtrain, ytrain)
gm_pred = gm.predict(xtest)
gm_pred1= gm.predict(xtrain)

print("Train score: {}%".format( 100 * accuracy_score(ytrain, gm_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(ytest, gm_pred, average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(ytest, gm_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(ytest, gm_pred)))
```

```
Train score: 86.0058309037901%

f1 score [90.68010076 15.90909091]%

accuracy score 83.21995464852607%

roc auc score 53.99600399600399%
```

```
[78]: print(classification_report(ytest, gm_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.99      0.91       364
           1       0.64      0.09      0.16        77

    accuracy                           0.83       441
   macro avg       0.74      0.54      0.53       441
weighted avg       0.80      0.83      0.78       441
```

- After applying backwards Elimination on Decision tree we haven't noticed that much change it's approximately the same as before.

```
[79]: Model.append('Decision Tree on Imbalanced Data after feature selection')
F1Score.append(f1_score(ytest, gm_pred, average=None))
Accuracy.append(accuracy_score(ytest, gm_pred))
```

12.4 10.4) Random Forest after Backward Elimination

[80]:
```
params = {

    'n_estimators':range(10,100,10),
    'criterion':['gini','entropy'],
    #'splitter':['best','random'],
    'max_depth':range(2,10,1),
    'max_leaf_nodes':range(2,10,1),
    'max_features':['auto','log2']

}

rf = RandomForestClassifier()

gs =␣
 ↪GridSearchCV(estimator=rf,param_grid=params,cv=3,scoring='recall',n_jobs=-1)
gs.fit(x_new,y_new)
```

[80]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 10),
                         'max_features': ['auto', 'log2'],
                         'max_leaf_nodes': range(2, 10),
                         'n_estimators': range(10, 100, 10)},
             scoring='recall')
```

[81]:
```
gs.best_params_
```

[81]:
```
{'criterion': 'gini',
 'max_depth': 8,
 'max_features': 'auto',
 'max_leaf_nodes': 9,
 'n_estimators': 10}
```

[82]:
```
rf = RandomForestClassifier(**gs.best_params_)
rf.fit(xtrain,ytrain)
y_pred_rf = rf.predict(xtest)
y_pred_rf1 = rf.predict(xtrain)
```

[83]:
```
print("Train score: {}%".format( 100 * accuracy_score(ytrain, y_pred_rf1)))
print()
print("f1 score {}%".format( 100 * f1_score(ytest, y_pred_rf, average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(ytest, y_pred_rf)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(ytest, y_pred_rf)))
```

```
Train score: 85.9086491739553%

f1 score [90.63670412  7.40740741]%

accuracy score 82.99319727891157%

roc auc score 51.810689310689305%
```

[84]:
```python
print(classification_report(ytest, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.83      1.00      0.91       364
           1       0.75      0.04      0.07        77

    accuracy                           0.83       441
   macro avg       0.79      0.52      0.49       441
weighted avg       0.82      0.83      0.76       441
```

- So what we have so far analyzed after applying Backward Elimination we have noticed signinficient changes in improving the training and testing accuracy on Logistic Regression but on Decision Tree we haven't noticed much difference.

[85]:
```python
Model.append('Random Forest on Imbalanced Data after feature selection')
F1Score.append(f1_score(ytest, y_pred_rf, average=None))
Accuracy.append(accuracy_score(ytest, y_pred_rf))
```

13 11) Balancing the Dataset using Smote and StandardScaler

[102]:
```python
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
```

[103]:
```python
scaler=StandardScaler()
scaled_df=scaler.fit_transform(HR_Emp_df.drop(['Attrition'], axis=1))
X=scaled_df
Y=HR_Emp_df['Attrition']
SMOTE().fit_resample(X, Y)
X,Y = SMOTE().fit_resample(X, Y)
#split data
train, test, target_train, target_val = train_test_split(X,
                                                         Y,
                                                         train_size= 0.80,
                                                         random_state=0);
```

13.1 11.1) Logistic Regression after Balacing the Data

[104]:
```
LR_b = LogisticRegression(multi_class='auto')
LR_b.fit(train,target_train)
lr_pred_b = LR_b.predict(test)
lr_pred_b1 = LR_b.predict(train)
```

[105]:
```
print("Train score: {}%".format( 100 * accuracy_score(target_train,␣
    ↪lr_pred_b1)))
print()
print("f1 score {}%".format( 100 * f1_score(target_val, lr_pred_b,␣
    ↪average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(target_val, lr_pred_b)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(target_val, lr_pred_b)))
```

```
Train score: 81.7444219066937%

f1 score [77.09251101 80.52434457]%

accuracy score 78.94736842105263%

roc auc score 78.75331259361677%
```

[106]:
```
print(classification_report(target_val, lr_pred_b))
```

```
              precision    recall  f1-score   support

           0       0.78      0.76      0.77       231
           1       0.79      0.82      0.81       263

    accuracy                           0.79       494
   macro avg       0.79      0.79      0.79       494
weighted avg       0.79      0.79      0.79       494
```

- We have now noticed that accuracy scored is fallen to 79% approx

[107]:
```
Model.append('LR on Balanced Data')
F1Score.append(f1_score(target_val, lr_pred_b, average=None))
Accuracy.append(accuracy_score(target_val, lr_pred_b))
```

13.2 11.2) Naive Bayes after Balacing the Data

[108]:
```
nb_b = GaussianNB()
nb_b.fit(train,target_train)
nb_pred_b = nb_b.predict(test)
nb_pred_b1 = nb_b.predict(train)


print("Train score: {}%".format( 100 * accuracy_score(target_train,␣
    ↪nb_pred_b1)))
print()
print("f1 score {}%".format( 100 * f1_score(target_val, nb_pred_b,␣
    ↪average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(target_val, nb_pred_b)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(target_val, nb_pred_b)))
```

```
Train score: 70.58823529411765%

f1 score [65.85956416 75.47826087]%

accuracy score 71.45748987854252%

roc auc score 70.69198228894047%
```

[109]:
```
print(classification_report(target_val, nb_pred_b))
```

```
              precision    recall  f1-score   support

           0       0.75      0.59      0.66       231
           1       0.70      0.83      0.75       263

    accuracy                           0.71       494
   macro avg       0.72      0.71      0.71       494
weighted avg       0.72      0.71      0.71       494
```

- Here we can see that all over the accuracy has fallen but we can see that over testing accuracy is improved if compare to training accuracy.

[110]:
```
Model.append('NB on Balanced Data')
F1Score.append(f1_score(target_val, nb_pred_b, average=None))
Accuracy.append(accuracy_score(target_val, nb_pred_b))
```

13.3 11.3) Decision Tree after Balacing the Data

[111]:
```
gm_b = DecisionTreeClassifier(criterion='gini', random_state=100, max_depth=3,␣
  ↪min_samples_leaf=5)
gm_b.fit(train, target_train)
gm_pred_b = gm_b.predict(test)
gm_pred_b1 = gm_b.predict(train)


print("Train score: {}%".format( 100 * accuracy_score(target_train,␣
  ↪gm_pred_b1)))
print()
print("f1 score {}%".format( 100 * f1_score(target_val, gm_pred_b,␣
  ↪average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(target_val, gm_pred_b)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(target_val, gm_pred_b)))
```

```
Train score: 77.73833671399595%

f1 score [77.69516729 73.33333333]%

accuracy score 75.7085020242915%

roc auc score 76.60691653087092%
```

[112]:
```
print(classification_report(target_val, gm_pred_b))
```

```
              precision    recall  f1-score   support

           0       0.68      0.90      0.78       231
           1       0.88      0.63      0.73       263

    accuracy                           0.76       494
   macro avg       0.78      0.77      0.76       494
weighted avg       0.79      0.76      0.75       494
```

- Overall there is a significent drop is accuracy if we compare the results before balancing the dataset

[113]:
```
Model.append('Decision Tree on Balanced Data')
F1Score.append(f1_score(target_val, gm_pred_b, average=None))
Accuracy.append(accuracy_score(target_val, gm_pred_b))
```

13.4 11.4) Random Forest after Balacing the Data

[114]:
```
seed = 0
params = {
    'n_estimators':range(10,100,10),
    'criterion':['gini','entropy'],
    'max_depth':range(2,10,1),
    'max_leaf_nodes':range(2,10,1),
    'max_features':['auto','log2'],
    'verbose':[0]
}
rf = RandomForestClassifier()
rs = RandomizedSearchCV(rf, param_distributions=params, scoring='accuracy',␣
 ↪n_jobs=-1, cv=5, random_state=42)
rs.fit(X,Y)
```

[114]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': range(2, 10),
                                        'max_features': ['auto', 'log2'],
                                        'max_leaf_nodes': range(2, 10),
                                        'n_estimators': range(10, 100, 10),
                                        'verbose': [0]},
                   random_state=42, scoring='accuracy')
```

[115]:
```
rs.best_params_
```

[115]:
```
{'verbose': 0,
 'n_estimators': 60,
 'max_leaf_nodes': 9,
 'max_features': 'log2',
 'max_depth': 7,
 'criterion': 'gini'}
```

[116]:
```
rf = RandomForestClassifier(**rs.best_params_)
rf.fit(train, target_train)
rf_pred = rf.predict(test)
rf_pred1 = rf.predict(train)
```

[117]:
```
print("Train score: {}%".format( 100 * accuracy_score(target_train, rf_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(target_val, rf_pred, average=None)))
print()
print("accuracy score {}%".format( 100 * accuracy_score(target_val, rf_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(target_val, rf_pred)))
```

```
Train score: 87.62677484787018%
```

```
f1 score [85.12396694 85.71428571]%

accuracy score 85.4251012145749%

roc auc score 85.65338337201455%
```

[118]:
```python
print(classification_report(target_val, rf_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       231
           1       0.90      0.82      0.86       263

    accuracy                           0.85       494
   macro avg       0.86      0.86      0.85       494
weighted avg       0.86      0.85      0.85       494
```

- Here we can see that we got accuracy of 85% which is Significiently better than other algorithms we have run so far after balancing the Data.

[119]:
```python
Model.append('Random Forest on Balanced Data')
F1Score.append(f1_score(target_val, rf_pred, average=None))
Accuracy.append(accuracy_score(target_val, rf_pred))
```

13.5 11.5) Gradient Boosting after Balacing the Data

[120]:
```python
gb_params ={
    'n_estimators': 1500,
    'max_features': 0.9,
    'learning_rate' : 0.25,
    'max_depth': 4,
    'min_samples_leaf': 2,
    'subsample': 1,
    'max_features' : 'sqrt',
    'verbose': 0
}
```

[121]:
```python
gb = GradientBoostingClassifier(**gb_params)
gb.fit(train, target_train)
gb_pred = gb.predict(test)
gb_pred1 = gb.predict(train)
```

[122]:
```python
print("Train score: {}%".format( 100 * accuracy_score(target_train, gb_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(target_val, gb_pred, average=None)))
print()
```

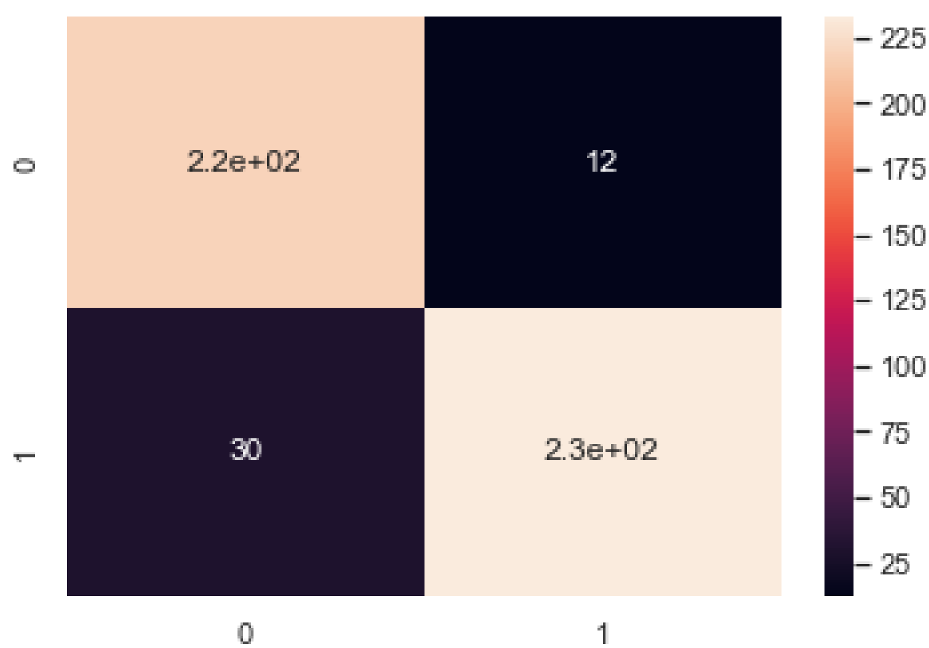
```
print("accuracy score {}%".format( 100 * accuracy_score(target_val, gb_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(target_val, gb_pred)))
```

```
Train score: 100.0%

f1 score [91.25       91.73228346]%

accuracy score 91.49797570850203%

roc auc score 91.69917534936548%
```

[123]:
```
print(classification_report(target_val, gb_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.91       231
           1       0.95      0.89      0.92       263

    accuracy                           0.91       494
   macro avg       0.92      0.92      0.91       494
weighted avg       0.92      0.91      0.92       494
```

[125]:
```
# Testing Set Performance

con_max = confusion_matrix(target_val, gb_pred)
sns.heatmap(con_max, annot=True);
```



- What we can notice here is that we got very good score on testing Accuracy as in start when we applied the Gradient Boosting before which was 84% and now its improved to 91% which is the best accuracy we got so far.

[124]:
```
Model.append('Gradient Boosting on Balanced Data')
F1Score.append(f1_score(target_val, gb_pred, average=None))
Accuracy.append(accuracy_score(target_val, gb_pred))
```

14 12) Feature Selection on Balanced Dataset

[127]:
```
X = HR_Emp_df.drop(['Attrition'], axis=1)
Y = HR_Emp_df[['Attrition']]
```

[128]:
```
# using select k best
SMOTE().fit_resample(X, Y)
X1,Y1 = SMOTE().fit_resample(X, Y)
```

[129]:
```
min_max_scaler = preprocessing.MinMaxScaler()
Scaled_X = min_max_scaler.fit_transform(X1)
Y_new=Y1
X_new = SelectKBest(chi2, k=2).fit_transform(Scaled_X, Y_new)
```

[130]:
```
x_train_f1,x_test_f1,y_train_f1,y_test_f1=train_test_split(X_new,Y_new,test_size=0.
↪30,random_state=1)
```

[131]:
```
gb_params ={
    'n_estimators': 1500,
    'max_features': 0.9,
    'learning_rate' : 0.25,
    'max_depth': 4,
    'min_samples_leaf': 2,
    'subsample': 1,
    'max_features' : 'sqrt',
    'verbose': 0
}
```

[132]:
```
gb = GradientBoostingClassifier(**gb_params)
gb.fit(x_train_f1, y_train_f1)
gb_pred = gb.predict(x_test_f1)
gb_pred1 = gb.predict(x_train_f1)
```

[133]:
```
print("Train score: {}%".format( 100 * accuracy_score(y_train_f1, gb_pred1)))
print()
print("f1 score {}%".format( 100 * f1_score(y_test_f1, gb_pred, average=None)))
```

```
print()
print("accuracy score {}%".format( 100 * accuracy_score(y_test_f1, gb_pred)))
print()
print("roc auc score {}%".format( 100 *roc_auc_score(y_test_f1, gb_pred)))
```

```
Train score: 78.62108922363848%

f1 score [74.96917386 69.65620329]%

accuracy score 72.56756756756756%

roc auc score 73.49916375693202%
```

[134]:
```
print(classification_report(y_test_f1, gb_pred))
```

```
              precision    recall  f1-score   support

           0       0.65      0.88      0.75       346
           1       0.85      0.59      0.70       394

    accuracy                           0.73       740
   macro avg       0.75      0.73      0.72       740
weighted avg       0.76      0.73      0.72       740
```

- So after getting the best Accuracy while using the Gradient Boosting Classification we applied feature Engineering again on the same algorithm to figure out will it make it better anyway, but we have seen a big drop in efficency of this model after getting the less Accuracy score of 73% approx.

15 13) Evaluation of the Models we have created So Far

[136]:
```
TN=cm[0,0]
TP=cm[1,1]
FN=cm[1,0]
FP=cm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
print('The acuuracy of the model = TP+TN / (TP+TN+FP+FN) = ',(TP+TN)/
 ↪float(TP+TN+FP+FN),'\n\n',

'The Miss-classification = 1-Accuracy = ',1-((TP+TN)/float(TP+TN+FP+FN)),'\n\n',

'Sensitivity or True Positive Rate = TP / (TP+FN) = ',TP/float(TP+FN),'\n\n',

'Specificity or True Negative Rate = TN / (TN+FP) = ',TN/float(TN+FP),'\n\n',
```

```
'Positive Predictive value = TP / (TP+FP) = ',TP/float(TP+FP),'\n\n',

'Negative predictive Value = TN / (TN+FN) = ',TN/float(TN+FN),'\n\n',

'Positive Likelihood Ratio = Sensitivity / (1-Specificity) = ',sensitivity/
 ↪(1-specificity),'\n\n',

'Negative likelihood Ratio = (1-Sensitivity) / Specificity = ',(1-sensitivity)/
 ↪specificity)
```

```
The acuuracy of the model = TP+TN / (TP+TN+FP+FN) =  0.9149797570850202

 The Miss-classification = 1-Accuracy =  0.08502024291497978

 Sensitivity or True Positive Rate = TP / (TP+FN) =  0.8859315589353612

 Specificity or True Negative Rate = TN / (TN+FP) =  0.948051948051948

 Positive Predictive value = TP / (TP+FP) =  0.9510204081632653

 Negative predictive Value = TN / (TN+FN) =  0.8795180722891566

 Positive Likelihood Ratio = Sensitivity / (1-Specificity) =  17.054182509505697

 Negative likelihood Ratio = (1-Sensitivity) / Specificity =
0.12031876660242719
```

Analyzing the results we got so far.

[135]:
```
# final_result = pd.DataFrame({'Model':Model,'Accuracy':Accuracy, 'F1Score':
 ↪F1Score})
# final_result

a = {'Model':Model, 'Accuracy':Accuracy, 'F1Score':F1Score}
final_result = pd.DataFrame.from_dict(a, orient='index')
final_result= final_result.transpose()
final_result
```

[135]:
```
                                                Model  Accuracy  \
0                                LR on Imbalanced Data  0.823129
1                                NB on Imbalanced Data  0.777778
2                     Decision Tree on Imbalanced Data    0.8322
3                     Random Forest on Imbalanced Data    0.8322
4                 Gradient Boosting on Imbalanced Data   0.84127
5        LR on Imbalanced Data after feature selection  0.882086
6        NB on Imbalanced Data after feature selection  0.823129
7   Decision Tree on Imbalanced Data after feature…    0.8322
8   Random Forest on Imbalanced Data after feature…  0.829932
```

```
9                                    LR on Balanced Data  0.789474
10                                   NB on Balanced Data  0.714575
11                        Decision Tree on Balanced Data  0.757085
12                        Random Forest on Balanced Data  0.854251
13                    Gradient Boosting on Balanced Data   0.91498

                                        F1Score
0   [0.9025000000000001, 0.04878048780487805]
1    [0.8558823529411765, 0.5148514851485149]
2    [0.9065656565656566, 0.1777777777777778]
3   [0.9072681704260652, 0.11904761904761903]
4   [0.9090909090909091, 0.37499999999999994]
5    [0.9326424870466321, 0.5272727272727273]
6    [0.893150684931507, 0.48684210526315785]
7     [0.906801007556675, 0.1590909090909091]
8   [0.9063670411985019, 0.07407407407407407]
9    [0.7709251101321586, 0.8052434456928839]
10   [0.6585956416464892, 0.7547826086956523]
11   [0.7769516728624535, 0.7333333333333333]
12   [0.8512396694214877, 0.8571428571428571]
13               [0.9125, 0.9173228346456693]
```

16 14) Conclusion & Actionable Insights

We have applied predictive analysis on the Attrition DataSet provided by IBM on Kaggle. We have applied multiple Classification Algorithms, Including Logistic Regression , Random Forest, Naive Bayes, Decision Tree, and Gradient Boosting. We have used multiple ensemble methods and tried multiple Hyper Tuning techniques as well we used Feature Engineering and Balancing Techniques like SMOTE and Scaling Techniques to find out which Model will be the best to use and now we have the answer That Gradient Boosting after oversampling and Scaling worked the best and we got 92% of Testing Accuracy score.This analyses additionally assessed the exploration question by showing the outcome that after balancing and feature engineering we got an effective result in predicting the Employee Attrition.

Well, We can say that this project has some limitations This research is restricted to a little dataset which may lack to train the model very well which could give low outcomes and getting private and confidential data make this research restricted to a limit to IBM dataset which is provided on internet. The second downside is with the model is restricted to just Supervised Learning that requires a tons of calculation time, moreover, there is a certain chance that decision boundary may get over trained when new feature is added and may requires user input.

This task can be stretched out in future as it has a great deal of possibilities to improve by applying Deep learning algorithms, pattern examination and time series analysis can be done in future to improve this pipeline.