HR Analytics Project- Understanding the Attrition in HR

Introduction

Human Resource in one of the six functional areas of managing business, Human resource (HR) often deals in managing employee, company hires new employee every year and invest time and money on them to train, it is said that newly hired employee takes average 3 months to get trained and become productive to the companies, apart from this many companies also organizes several training programs and webinar for professional development of their existing employee,

In-spite of various initiative, few companies are struggling with the problem of high attrition. Attrition means an employee leaving an existing company for a new company, company having high attrition rates often spends more time and money on training new employee as compared to the company where employee are staying for the longer period of time

HR Analytic plays an important role in the process improvement in Human Resource. HR analytic gathers data on employee efficiency, and also helps us to identify the reason behind the employee leaving the company, it also aid the company in making relevant business decision for improving the overall process and getting better return on investment

Problem statement

A company with high attrition rate often spends money on hiring and training new employees, it also requires significant amount of time and resources to look for a better replacement. Apart from this company with high attrition rate often struggles with collective knowledge base due to which overall development of the business become slow because employees have less knowledge of the business process. In addition to this new worker tends to make more mistake as compared to old employees.

Data Analysis

Data Analysis is the very curcial part in develop before developing the predicative model as it shows the trend and connection between features and labels. In-Depth analysis of the data-set provided has been done in Python using various libraries such as pandas(for data Manipulation), numpy (for numerical Calculations), Matplotlib and sea-born(for Visualization)

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline


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

Loading Data:

Importing the data and looking at the sample data

```
In [3]: 1 data = pd.read_csv(r'D:\Data Science\Evaluation projects\hr_analytics.csv')
        2 data.sample(10)
```

Out[3]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	...	Relati
243	40	No	Travel_Rarely	1300	Research & Development	24	2	Technical Degree	1	335	...	
662	20	Yes	Travel_Rarely	500	Sales	2	3	Medical	1	922	...	
296	18	Yes	Travel_Rarely	230	Research & Development	3	3	Life Sciences	1	405	...	
1111	53	Yes	Travel_Rarely	607	Research & Development	2	5	Technical Degree	1	1572	...	
1377	49	No	Travel_Frequently	1064	Research & Development	2	1	Life Sciences	1	1941	...	
276	35	No	Travel_Rarely	1315	Research & Development	22	3	Life Sciences	1	381	...	
505	26	No	Travel_Rarely	991	Research & Development	6	3	Life Sciences	1	686	...	
982	38	No	Travel_Frequently	693	Research & Development	7	3	Life Sciences	1	1382	...	
1341	31	No	Travel_Rarely	311	Research & Development	20	3	Life Sciences	1	1881	...	
83	38	No	Non-Travel	573	Research & Development	6	3	Medical	1	107	...	

10 rows × 35 columns

Data has been loaded in the Jupiter as a DataFrame Panda.read_csv function. Data-set contains 1470 rows and 35 columns.

Checking null/Missing Values :

```
5 data.isnull().sum()

(1470, 35)

[87]: Age                         0
      Attrition                   0
      BusinessTravel              0
      DailyRate                   0
      Department                  0
      DistanceFromHome            0
      Education                   0
      EducationField              0
      EmployeeCount               0
      EmployeeNumber              0
      EnvironmentSatisfaction     0
      Gender                      0
      HourlyRate                  0
      JobInvolvement              0
      JobLevel                    0
      JobRole                     0
      JobSatisfaction             0
      MaritalStatus               0
      MonthlyIncome               0
      MonthlyRate                 0
      NumCompaniesWorked          0
      Over18                      0
      OverTime                    0
      PercentSalaryHike           0
      PerformanceRating           0
      RelationshipSatisfaction    0
      StandardHours               0
      StockOptionLevel            0
      TotalWorkingYears           0
      TrainingTimesLastYear       0
      WorkLifeBalance             0
      YearsAtCompany              0
      YearsInCurrentRole          0
      YearsSinceLastPromotion     0
      YearsWithCurrManager        0
      dtype: int64
```

```
1 data.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
```

We checked for Null/ missing values pandas is-null function but we haven't found any missing values in the data-set. We also checked the data types of all the columns using pd.info() and we found that dataset and we found that data contains object and numerical columns. And it also confirms that there is no Missing or null value in the data-set

Data Exploration and Analysis:

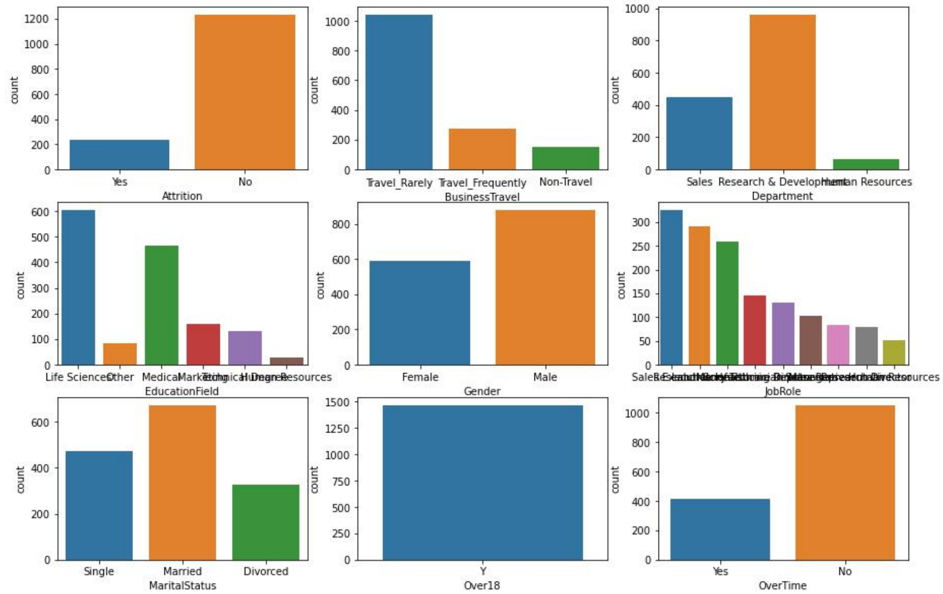
```
cat = []
for i in data.columns:
    if data[i].dtypes == object:
        cat.append(i)
cat
```

```
[88]: ['Attrition',
 'BusinessTravel',
 'Department',
 'EducationField',
 'Gender',
 'JobRole',
 'MaritalStatus',
 'Over18',
 'OverTime']
```

First we created a list of all the categorical(Object) columns in the data-set using th above formula, which will help us in data exploration

```
In [79]:
plt.figure(figsize=(15,10))
fignumber = 1

for column in data[cat]:
    if fignumber <=9:
        ax = plt.subplot(3,3,fignumber)
        sns.countplot(data[column])
        plt.xlabel(column, fontsize=10)
    fignumber+=1
plt.show()
```



After creating the list of all categorical columns, we have checked the distribution of all the categorical and we have found that data is imbalanced, as label data has 84% of the data as No and only 16 of the data as Yes. We have treated this going forward

```
# it seems that label(attrition) is unbalanced, lets check further

data['Attrition'].value_counts() / data['Attrition'].value_counts().sum() * 100
```

```
0]: No     83.877551
    Yes    16.122449
    Name: Attrition, dtype: float64
```

```
1 data.describe()
```

[5]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	HourlyRate	JobInvolvemen
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.000000	1470.000000	1470.000000	1470.00000
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.865306	2.721769	65.891156	2.72993
std	9.135373	403.509100	8.106864	1.024165	0.0	602.024335	1.093082	20.329428	0.71156
min	18.000000	102.000000	1.000000	1.000000	1.0	1.000000	1.000000	30.000000	1.00000
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.250000	2.000000	48.000000	2.00000
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.500000	3.000000	66.000000	3.00000
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.750000	4.000000	83.750000	3.00000
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.000000	4.000000	100.000000	4.00000

8 rows × 26 columns

We have extracted the structure of the data-set using pd.describe() function and following observations have been made

1) employeeCount - we have only one value for all the records in the dataset so we will drop this column

2) employeenumber - its more of an employee ID so it wont contribute in the predicting the attrtion so we will drop this column

3) Standardhoours - all the employee have standard hour of 80, so it wont contribution in identifying the attrition, so we will drop this column as well

4) over18 = from the structure of the data and countplot above we can see that all the employees are 18 plus so this column is not relevant so we will drop this column as well

we can also find some column with skewed data so we will deal with those data at a later stage

EDA :

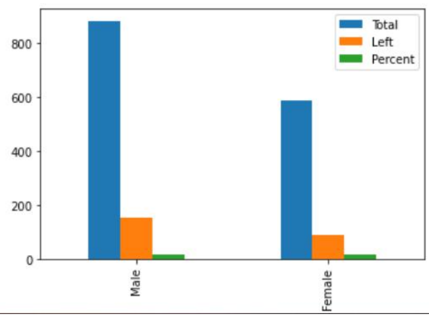
Comparing Various categorical columns with label data

In [113]:

```
count = data[data['Attrition']=='Yes']['Gender'].value_counts()
percent = data[data['Attrition']=='Yes']['Gender'].value_counts() / data['Gender'].value_counts()*100
total = data['Gender'].value_counts()
gender_attrition = pd.DataFrame({'Total':total,'Left':count,'Percent':percent})
gender_attrition.plot(kind='bar')
gender_attrition
```

Out[113]:

	Total	Left	Percent
Male	882	150	17.006803
Female	588	87	14.795918



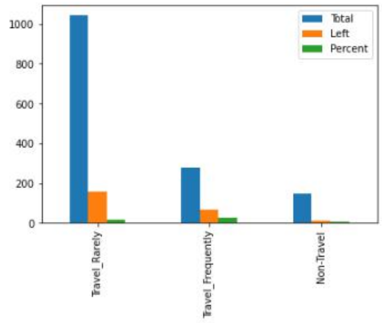
As we can see from the above plot around 17% of the male have left the job and around 15% of the female have left the job

In [114]:

```python
# lets see for businessTravel
count = data[data['Attrition']=='Yes']['BusinessTravel'].value_counts()
percent = data[data['Attrition']=='Yes']['BusinessTravel'].value_counts() / data['BusinessTravel'].value_counts()*100
total = data['BusinessTravel'].value_counts()
travel_attrition = pd.DataFrame({'Total':total,'Left':count,'Percent':percent})
travel_attrition.plot(kind='bar')
travel_attrition
```

Out[114]:

	Total	Left	Percent
Travel_Rarely	1043	156	14.956855
Travel_Frequently	277	69	24.909747
Non-Travel	150	12	8.000000



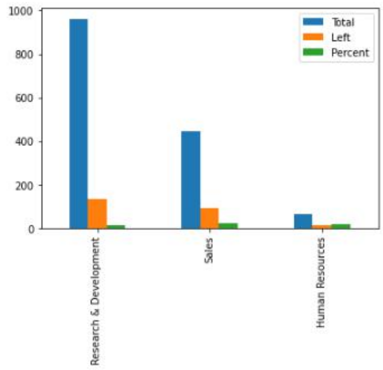
here we can see that the job which require frequent leads to higher attrition, however as the travel decreases attrition also decreases

In [115]:

```python
# lets check which department have the highest attrtion
count = data[data['Attrition']=='Yes']['Department'].value_counts()
percent = data[data['Attrition']=='Yes']['Department'].value_counts() / data['Department'].value_counts()*100
total = data['Department'].value_counts()
dept_attrition = pd.DataFrame({'Total':total,'Left':count,'Percent':percent})
dept_attrition.plot(kind='bar')
dept_attrition
```

Out[115]:

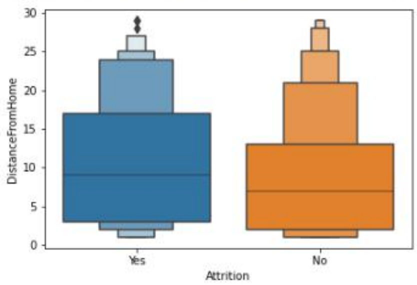
	Total	Left	Percent
Research & Development	961	133	13.839750
Sales	446	92	20.627803
Human Resources	63	12	19.047619



the highest attrition is in sales department followed by HR and then research & development

```python
#lets see how the distance from home impacts the attrition
print('avg distance of people who left --',data[data['Attrition']=='Yes']['DistanceFromHome'].mean())
print('avg distance of people who didnt left --',data[data['Attrition']=='No']['DistanceFromHome'].mean())
sns.boxenplot(x=data['Attrition'],y=data['DistanceFromHome'],data=data)
plt.show()
```

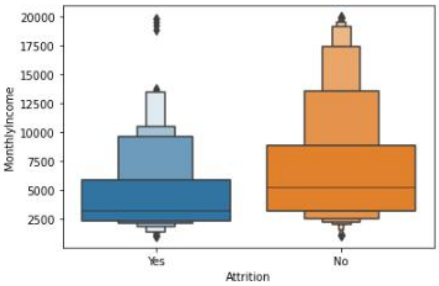
```
avg distance of people who left -- 10.632911392405063
avg distance of people who didnt left -- 8.915652879156529
```



as we can see people who live farther are more likely to leave the job, Commuting maybe the reason of people leaving the job

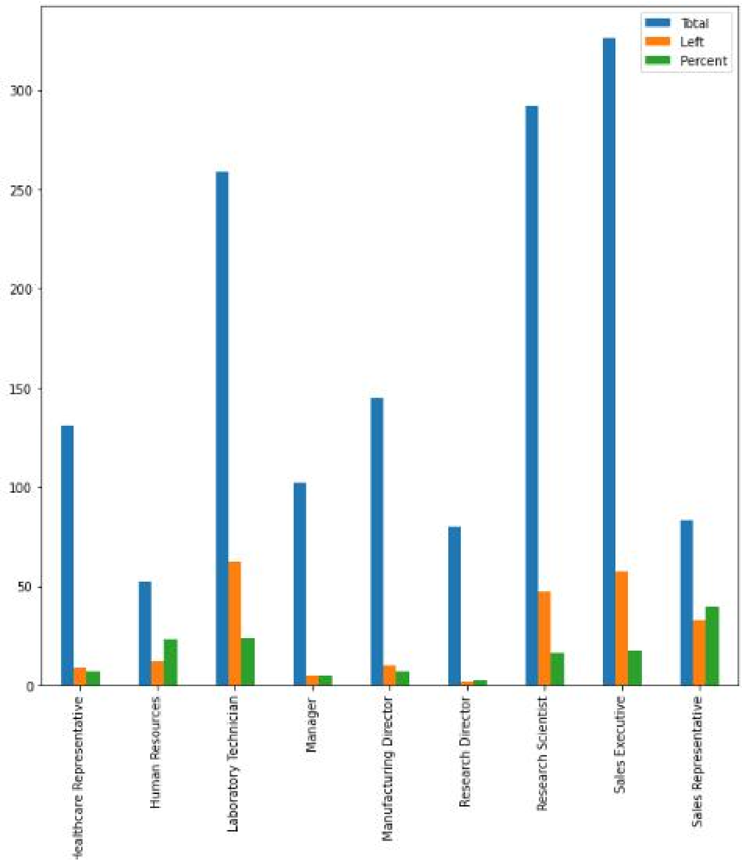
```
# lets check how monthly income is related to employee attrition
print('avg distance of people who left --',data[data['Attrition']=='Yes']['MonthlyIncome'].mean())
print('avg distance of people who didnt left --',data[data['Attrition']=='No']['MonthlyIncome'].mean())
sns.boxenplot(x=data['Attrition'],y=data['MonthlyIncome'],data=data)
plt.show()
```

```
avg distance of people who left -- 4787.0928270042195
avg distance of people who didnt left -- 6832.739659367397
```



As we can that average income of employee who left is 4787 however those who didnt left is 6832, which implies that lower income might be the reason employee leaving the job

	Total	Left	Percent	Avg Income
Sales Representative	83	33	39.759036	2626.000000
Laboratory Technician	259	62	23.938224	3237.169884
Human Resources	52	12	23.076923	4235.750000
Sales Executive	326	57	17.484663	6924.279141
Research Scientist	292	47	16.095890	3239.972603
Manufacturing Director	145	10	6.896552	7295.137931
Healthcare Representative	131	9	6.870229	7528.763359
Manager	102	5	4.901961	17181.676471
Research Director	80	2	2.500000	16033.550000



from the above table and graph it is clearly understandable the mean reason of employee leaving the company is because of salary,

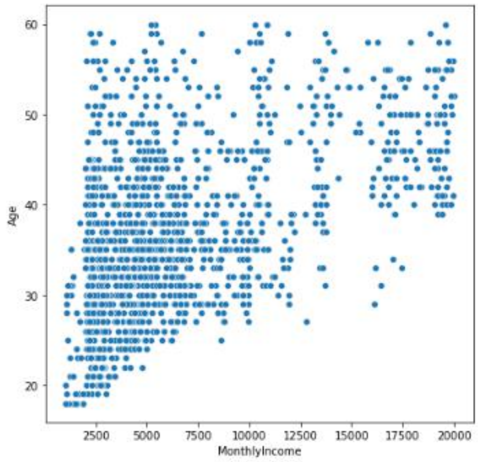
average salary of Manager and Research Director is the highest among the all profession (17181.67 & 16033.55), but in the contrary the attrition rate of both profession is merely 5% and 2.5% respectively

on the other hand, Sales Representative and Laboratory Technician age getting paid around 2626.00 and 3237.16, due to that reason the attrition rate is the highest

```
print('Average age of employee who left -- ',data[data.Attrition == 'Yes']['Age'].mean())
print('Average age of employee who stayed -- ',data[data.Attrition == 'No']['Age'].mean())
plt.figure(figsize=(7,7))
sns.scatterplot(x=data['MonthlyIncome'],y=data['Age'],data=data)
plt.show
```

```
Average age of employee who left --  33.607594936708864
Average age of employee who stayed --  37.561232765612324
```

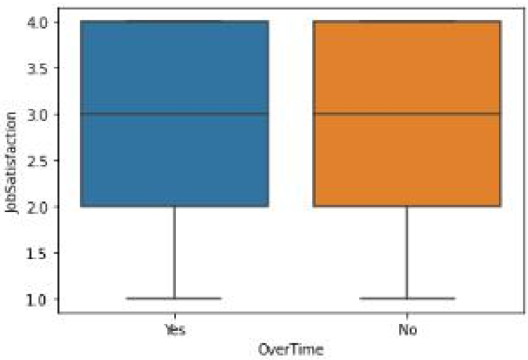
```
9]: <function matplotlib.pyplot.show(close=None, block=None)>
```



as we can see from the above data the younger people are more likely to leave the company,
from the scatter plot we can see that younger people are tends to gets lower wage, so younger people leave the job to search for high paying job

```
sns.boxplot(y=data['JobSatisfaction'],x=data['OverTime'],data=data)
```

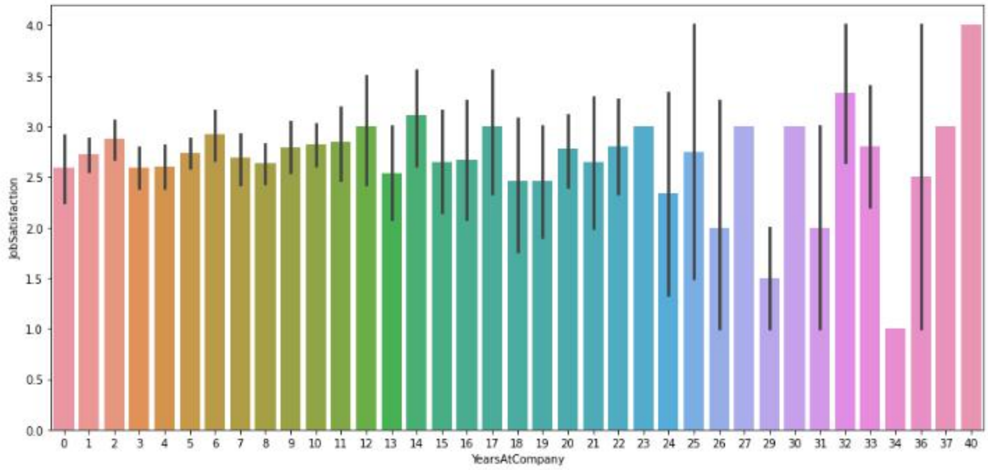
```
]: <AxesSubplot:xlabel='OverTime', ylabel='JobSatisfaction'>
```



from the above box plot we can conclude that there is no relation of job satisfaction with the workload

```python
plt.figure(figsize=(15,7))
sns.barplot(x=data['YearsAtCompany'],y=data['JobSatisfaction'],data=data)
```

```
7]: <AxesSubplot:xlabel='YearsAtCompany', ylabel='JobSatisfaction'>
```



From bar plot it is confirmed that there is no relation between year spent in the company and job satisfaction.

Pre-processing Pipeline

Encoding the Categorical Data

```python
# lets encode all the catergorical column using label encoder :
from sklearn.preprocessing import LabelEncoder

data[cat[1]] = LabelEncoder().fit_transform(data[cat[1]])
data[cat[2]] = LabelEncoder().fit_transform(data[cat[2]])
data[cat[3]] = LabelEncoder().fit_transform(data[cat[3]])
data[cat[4]] = LabelEncoder().fit_transform(data[cat[4]])
data[cat[5]] = LabelEncoder().fit_transform(data[cat[5]])
data[cat[6]] = LabelEncoder().fit_transform(data[cat[6]])
data[cat[7]] = LabelEncoder().fit_transform(data[cat[7]])
```

```python
data[cat]
```

1]:

	Attrition	BusinessTravel	Department	EducationField	Gender	JobRole	MaritalStatus	OverTime
0	1	2	2	1	0	7	2	1
1	0	1	1	1	1	6	1	0
2	1	2	1	4	1	2	2	1
3	0	1	1	1	0	6	1	1
4	0	2	1	3	1	2	1	0
...	...	...	...	...	...	...	...	...
1465	0	1	1	3	1	2	1	0
1466	0	2	1	3	1	0	1	0
1467	0	2	1	1	1	4	1	1
1468	0	1	2	3	1	7	1	0
1469	0	2	1	3	1	2	1	0

1470 rows × 8 columns

We have encoded all the categorical column using Label encoder

Balancing the Dataset

As mentioned earlier, the data-set in imbalanced so we have balanced the data-set using re-sampling technique, which means that we have created more data-points of Yes label so that we have more records of attrition = Yes

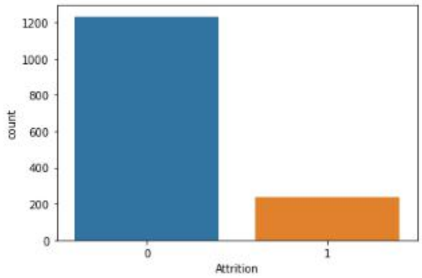
balancing the dataset

```
# as the data is unbalanced we will be balancing the dataset using resampling techniques
sns.countplot(x=data['Attrition'], data=data)

from sklearn.utils import resample
##
data_yes = data[data.Attrition==1]
data_no = data[data.Attrition==0]

data_yes = resample(data_yes,replace=True,n_samples=1045,random_state=25)

data_new = pd.concat([data_yes,data_no])
```




```
data_new
```

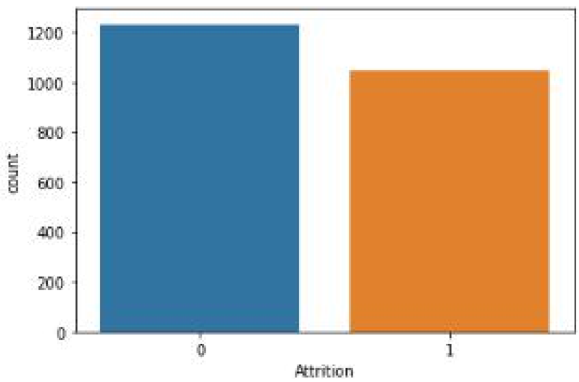
[17]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EnvironmentSatisfaction	Gender	...	Performa
796	25	1	2	1219	1	4	1	5	4	1	...	
415	34	1	1	296	2	6	2	2	4	0	...	
1326	32	1	2	414	2	2	4	2	3	1	...	
842	28	1	2	1485	1	12	1	1	3	0	...	
414	24	1	2	1448	2	1	1	5	1	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1465	36	0	1	884	1	23	2	3	3	1	...	
1466	39	0	2	613	1	6	1	3	4	1	...	
1467	27	0	2	155	1	4	3	1	2	1	...	
1468	49	0	1	1023	2	2	3	3	4	1	...	
1469	34	0	2	628	1	8	3	3	2	1	...	

2278 rows × 31 columns

```
sns.countplot(x=data_new['Attrition'], data=data)
```

[18]: <AxesSubplot:xlabel='Attrition', ylabel='count'>

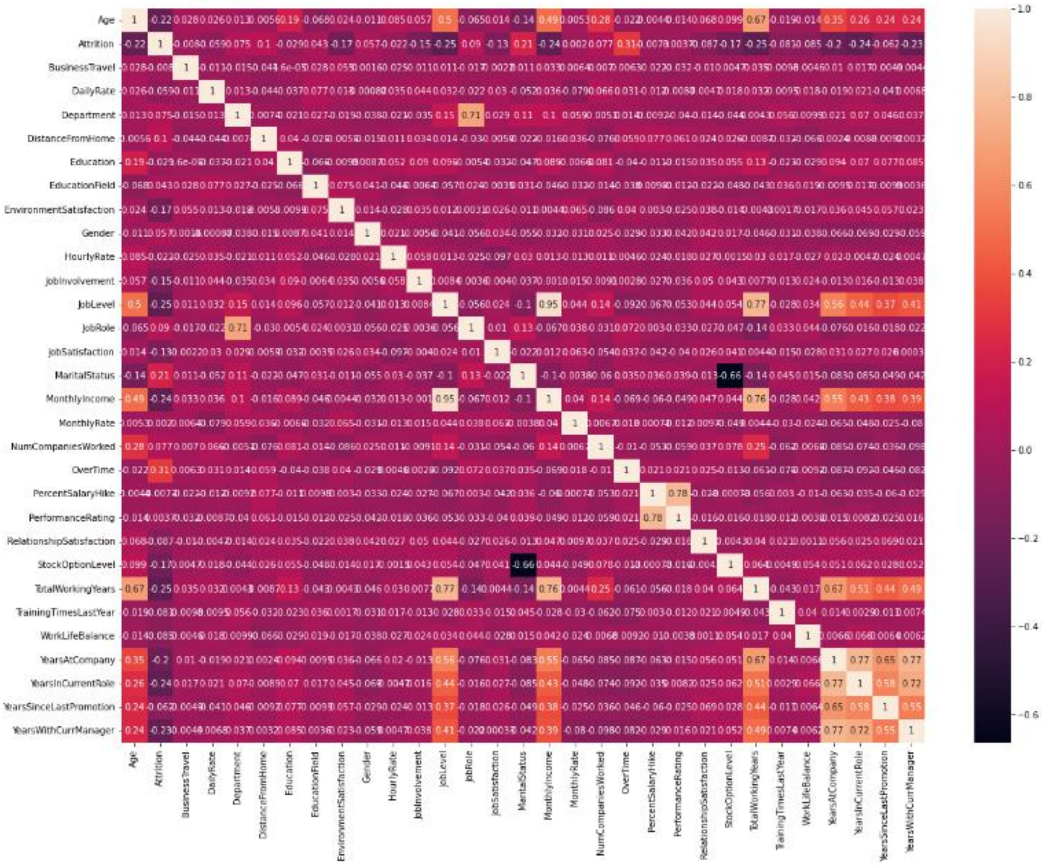


we have successfully balanced the dataset

Checking for multi co-linearity Problem:

After balancing the data set it is important to check the multi co-linearity because if independent features are highly co-related with each other, we might end up creating biased model, so it is very crucial to check for the multicoleariy before proceeding further

```python
plt.figure(figsize=(20,15))
sns.heatmap(data_new.corr(),annot=True)
plt.savefig('heatmap_hr.jpg')
```



We have used .corr() function to check for the correlation between the variables and used heat-map for visualizing the result.

We found Monthly Income and job level are highly correlated with each other, so we have to remove one of the columns, after analyzing we decided to drop Job level column because job level is less related to label data as compared to Monthly Income

```python
data_new = data_new.drop(columns='JobLevel')
data_new
```

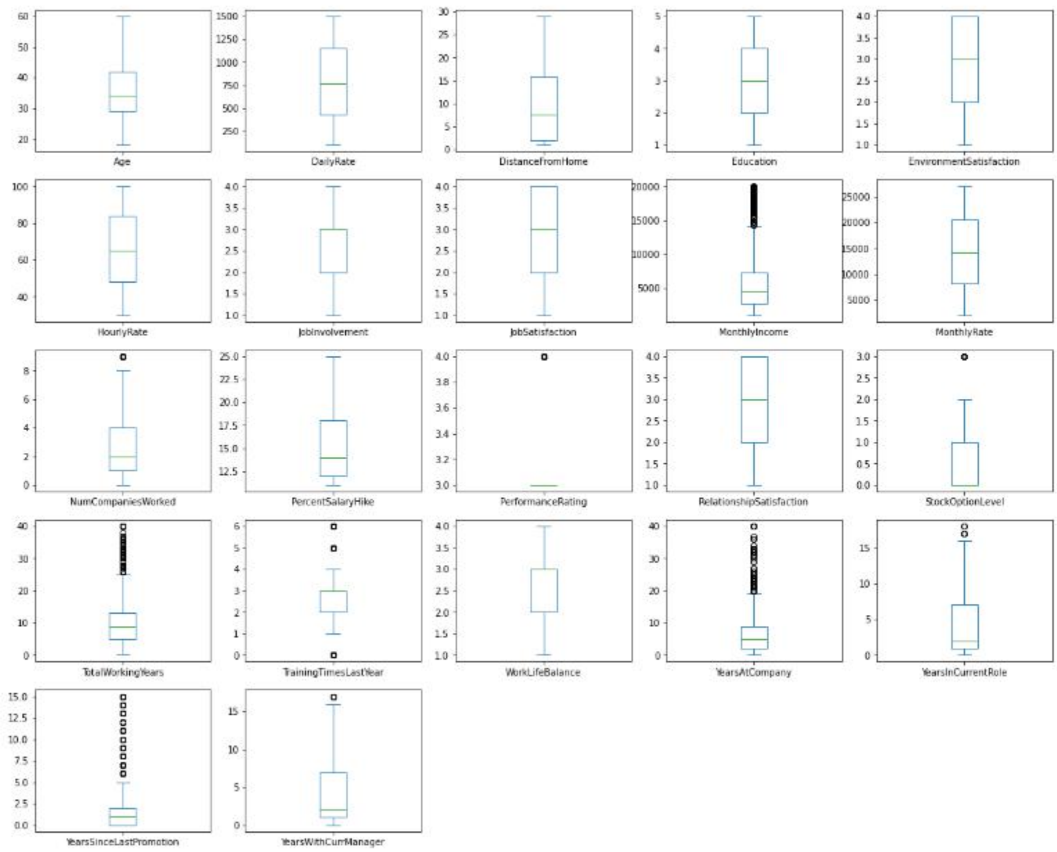
Outlier detection and Removal:

After Fixing the multicolinearity problem, the next import step is identifying the outlier and removing it, outlier have great impact on the central tendency of the data-set, for instance, if there are significant outlier on the higher end of the continuous data, it push the mean of the dataset to the higher side, these may have great impact on the accuracy of the model we build,

In this dataset we use the box plot to visualize the outlier in the continuous column

```python
# lets check for the outlier using box plot

data_new[cont_column].plot(kind='box',subplots=True,sharex=False,layout=(6,5),figsize=(20,20))
plt.show()
```



From the above box plots, we identified following columns have outliers

'MonthlyIncome','NumCompaniesWorked','TotalWorkingYears','TrainingTimesLastYear','YearsAtCompany','YearsInCurrentRole','YearsSinceLastPromotion','YearsWithCurrManager'

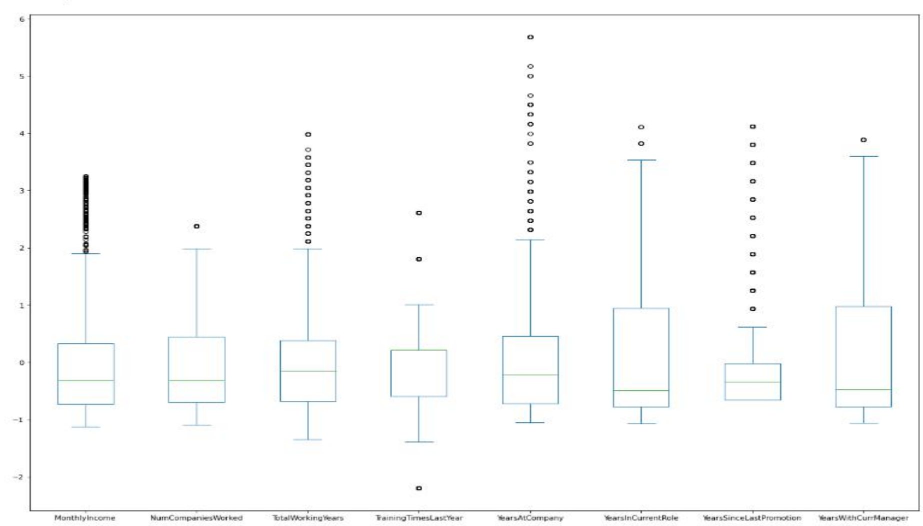
To further confirm, we will again use box plot to visualize the ouliers from the above columns, but this time we will use z score instead of normal value to visualize the data

```python
In [352]:
from scipy.stats import zscore


outlierlist = ['MonthlyIncome','NumCompaniesWorked','TotalWorkingYears','TrainingTimesLastYear','YearsAtCompany',
               'YearsInCurrentRole','YearsSinceLastPromotion','YearsWithCurrManager']

zscore(data_new[outlierlist]).plot(kind='box',figsize=(20,15))
```

```
Out[352]: <AxesSubplot:>
```



Based on the above chart and thumb rule of zscore (between +3 and -3) except Numcompaniesworked and TrainingTimesLastyear all other columns contains outier, so we will remove it using zscore

```python
#removing outliers using zscore

z_score= np.abs(zscore(data_new[outlierlist]))

data_new = data_new[(z_score<3).all(axis=1)]
```

```python
data_new
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EnvironmentSatisfaction	Gender	...	Performa
796	25	1	2	1219	1	4	1	5	4	1	...	
415	34	1	1	296	2	6	2	2	4	0	...	
1326	32	1	2	414	2	2	4	2	3	1	...	
842	28	1	2	1485	1	12	1	1	3	0	...	
414	24	1	2	1448	2	1	1	5	1	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1465	36	0	1	884	1	23	2	3	3	1	...	
1466	39	0	2	613	1	6	1	3	4	1	...	
1467	27	0	2	155	1	4	3	1	2	1	...	
1468	49	0	1	1023	2	2	3	3	4	1	...	
1469	34	0	2	628	1	8	3	3	2	1	...	

2140 rows × 30 columns

Feature reduction using chi-square :

After removing the outliers, we noticed that there are 29 features in the data set, some of them might not be important for predicting the label, and also too many labels can degrade the performance of the model,

Feature reduction using chi-square

```python
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import chi2

spercentile = SelectPercentile(score_func=chi2, percentile=75)
spercentile = spercentile.fit(X,y)

col = spercentile.get_support(indices=True)
features = X.columns[col]
```

```python
print(features)
print(col)
print(len(col))
```

```
Index(['Age', 'DailyRate', 'DistanceFromHome', 'EnvironmentSatisfaction',
       'HourlyRate', 'JobInvolvement', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'OverTime', 'RelationshipSatisfaction', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'YearsAtCompany',
       'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
[ 0  2  4  7  9 10 11 12 13 14 15 16 17 20 21 22 23 25 26 27 28]
21
```

We have used chi square technique to select the best features of the dataset, we have used the top 75% of the features and drop the least 25% of the features

Removing the skewness :

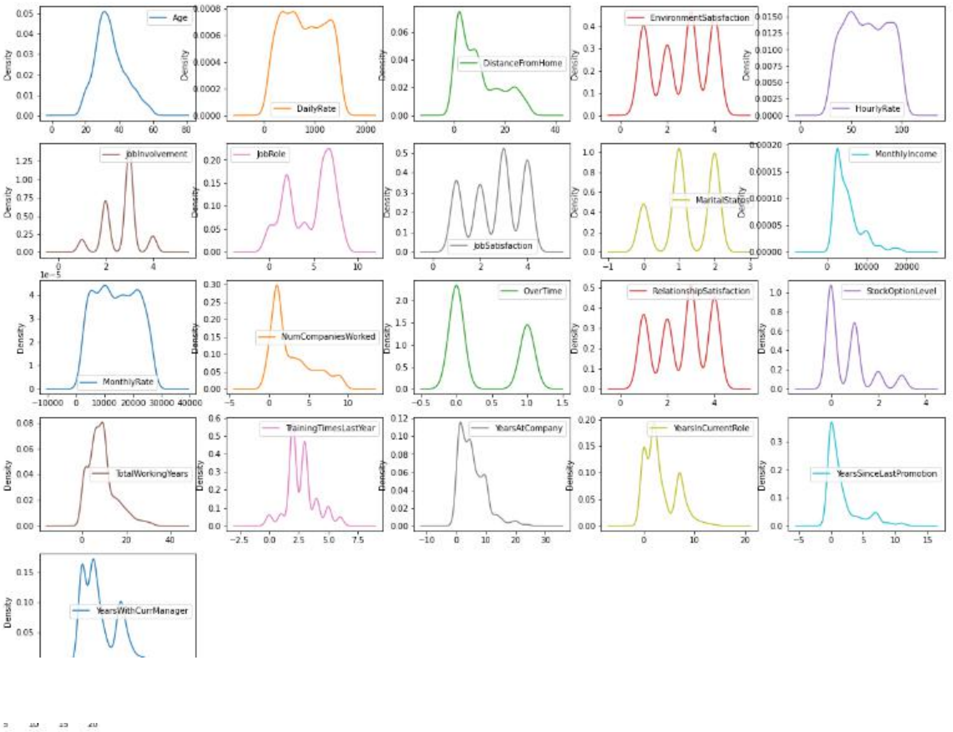
The next important step in the data preprocessing is to check and remove the skewness from all the continuous data, as Skewed data can prevent us from creating an efficient model

In [444]:

```
# checking the distribution of features

X.plot(kind='density',subplots=True,sharex=False,layout=(5,5),figsize=(20,15))
plt.show()

X.skew().sort_values(ascending=True)
```



```
JobInvolvement             -0.456961
JobRole                    -0.393597
MaritalStatus              -0.342004
JobSatisfaction            -0.233371
RelationshipSatisfaction   -0.206119
EnvironmentSatisfaction    -0.153992
HourlyRate                  0.025073
MonthlyRate                 0.040268
DailyRate                   0.056195
TrainingTimesLastYear       0.479097
OverTime                    0.482964
Age                         0.541150
YearsWithCurrManager        0.770705
DistanceFromHome            0.805995
YearsInCurrentRole          0.847199
NumCompaniesWorked          0.960069
TotalWorkingYears           1.084634
StockOptionLevel            1.189181
YearsAtCompany              1.263820
MonthlyIncome               1.574721
YearsSinceLastPromotion     1.802231
dtype: float64
```

After plot the distribution plot, we see that few of the columsn have skewenss in the data, so we have used Cube root technique to remove the skewness in the data

In [445]:

```
X.YearsInCurrentRole = np.cbrt(X.YearsInCurrentRole)
X.NumCompaniesWorked = np.cbrt(X.NumCompaniesWorked)
X.TotalWorkingYears = np.cbrt(X.TotalWorkingYears)
X.StockOptionLevel = np.cbrt(X.StockOptionLevel)
X.YearsAtCompany = np.cbrt(X.YearsAtCompany)
X.MonthlyIncome = np.cbrt(X.MonthlyIncome)
X.YearsSinceLastPromotion = np.cbrt(X.YearsSinceLastPromotion)
```

In [446]:

```
# vizualizing it usig ddensity plot
X.skew().sort_values(ascending=True)
```

Out[446]:

```
YearsAtCompany             -0.698917
YearsInCurrentRole         -0.678789
NumCompaniesWorked         -0.666163
TotalWorkingYears          -0.629244
JobInvolvement             -0.456961
JobRole                    -0.393597
MaritalStatus              -0.342004
JobSatisfaction            -0.233371
RelationshipSatisfaction   -0.206119
EnvironmentSatisfaction    -0.153992
HourlyRate                  0.025073
MonthlyRate                 0.040268
DailyRate                   0.056195
StockOptionLevel            0.198478
YearsSinceLastPromotion     0.273991
TrainingTimesLastYear       0.479097
OverTime                    0.482964
Age                         0.541150
MonthlyIncome               0.707172
YearsWithCurrManager        0.770705
DistanceFromHome            0.805995
dtype: float64
```

Scaling the data :

The final step in Data proprocessing is to scale the data, As we see that all the features in the data set have different unit of measure or scale, so to create an accurate model we need to scale all the feature so that all the measure should have same scale.

We used Standard Scalar to scale the dataset

Scaling the data

```
# scaled data using standard scalar
from sklearn.preprocessing import StandardScaler

x_scaled = StandardScaler().fit_transform(X)
x_scaled
```

```
51]: array([[-1.12176805,  1.04663199, -0.69198369, ...,  0.98326174,
          1.32884524,  0.84815131],
        [-0.1045237 , -1.21347202, -0.45154773, ...,  0.07549145,
          0.35454654, -1.06779922],
        [-0.330578  , -0.92453132, -0.93241965, ...,  0.07549145,
          0.71123573, -0.42914904],
        ...,
        [-0.89571375, -1.55873168, -0.69198369, ...,  0.07549145,
         -1.01775163, -0.10982396],
        [ 1.59088354,  0.56669658, -0.93241965, ...,  0.85007238,
         -1.01775163,  1.48680148],
        [-0.1045237 , -0.4005202 , -0.21111176, ...,  0.32895217,
          0.35454654, -0.42914904]])
```

Building Machine Learning Models.

After in-depth data analysis and all the data preprocessing its time to train the model, I have used multiple model for training and will choose the best model in terms of accuracy

I will be using following model for prediction:

Logistics Regression
Decsion Tree
Random Forest
Knn (K Nearest Neighbor)

Logistics Regression:

Confusion matrix and clssification report - Logistics Regression

```
0]:
#the random state from logisitics regression is 61, so we will use to generate confusion matrix and classification repor
# we got best result from random forest classifier we will use that result
x_train, x_test, y_train, y_test = train_test_split(x_scaled,y,train_size=.8,random_state=61)
lm = LogisticRegression()
lm.fit(x_train,y_train)
y_pred = lm.predict(x_test)
pacc = accuracy_score(y_test,y_pred)
pscore = lm.score(x_test,y_test)

print('Accuracy Score ---',accuracy_score(y_test,y_pred)*100,'%')
print('\n-------- Classification Report --------\n',classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
Accuracy Score --- 80.60747663551402 %

-------- Classification Report --------
              precision    recall  f1-score   support

           0       0.80      0.81      0.81       215
           1       0.81      0.80      0.80       213

    accuracy                           0.81       428
   macro avg       0.81      0.81      0.81       428
weighted avg       0.81      0.81      0.81       428

[[175  40]
 [ 43 170]]
```

First model I have used is logistic regression, First I have found the best random state for training, and after using the best random state(61), the accuracy score of the model is approx 81%

Decision Tree:

Confusion matrix and clssification report - Decision Tree

```
x_train, x_test, y_train, y_test = train_test_split(x_scaled,y,train_size=.8,random_state=97)
dt = DecisionTreeClassifier()
dt.fit(x_train,y_train)
y_pred = dt.predict(x_test)
pacc = accuracy_score(y_test,y_pred)
pscore = dt.score(x_test,y_test)

print('Accuracy Score ---',accuracy_score(y_test,y_pred)*100,'%')
print('\n-------- Classification Report --------\n',classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
Accuracy Score --- 94.39252336448598 %

-------- Classification Report --------
               precision    recall  f1-score   support

           0       0.99      0.90      0.94       226
           1       0.90      0.99      0.94       202

    accuracy                           0.94       428
   macro avg       0.95      0.95      0.94       428
weighted avg       0.95      0.94      0.94       428

[[204  22]
 [  2 200]]
```

Second model I have used is Decision Tree, First I have found the best random state for training, and after using the best random state(97), the accuracy score of the model is approx 94% which is better than logistics regression, which is a good sign

K Nearest Neighbour:

Confusion matrix and clssification report - KNN

```
[466]:
x_train, x_test, y_train, y_test = train_test_split(x_scaled,y,train_size=.8,random_state=49)
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
y_pred = knn.predict(x_test)
pacc = accuracy_score(y_test,y_pred)
pscore = knn.score(x_test,y_test)

print('Accuracy Score ---',accuracy_score(y_test,y_pred)*100,'%')
print('\n-------- Classification Report --------\n',classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
Accuracy Score --- 86.6822429906542 %

-------- Classification Report --------
               precision    recall  f1-score   support

           0       0.89      0.83      0.86       213
           1       0.84      0.90      0.87       215

    accuracy                           0.87       428
   macro avg       0.87      0.87      0.87       428
weighted avg       0.87      0.87      0.87       428

[[177  36]
 [ 21 194]]
```

Third Model is K nearest neighbour, I have used the same approach to find the best random state by running the for loops, then by plugging the best random state(49) the accuracy that I got Is 86.86% which is better than Logistics regression but still less than Decision Tree,

Lets check final Model

Random Forest:

Confusion matrix and clssification report - Random Forest

```
[469]:
x_train, x_test, y_train, y_test = train_test_split(x_scaled,y,train_size=.8,random_state=20)
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
y_pred = rf.predict(x_test)
pacc = accuracy_score(y_test,y_pred)
pscore = rf.score(x_test,y_test)

print('Accuracy Score ---',accuracy_score(y_test,y_pred)*100,'%')
print('\n-------- Classification Report --------\n',classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
Accuracy Score --- 99.06542056074767 %

-------- Classification Report --------
               precision    recall  f1-score   support

           0       1.00      0.99      0.99       205
           1       0.99      1.00      0.99       223

    accuracy                           0.99       428
   macro avg       0.99      0.99      0.99       428
weighted avg       0.99      0.99      0.99       428

[[202   3]
 [  1 222]]
```

After using the same, approach as previous three model, the accuracy we got for random forest is whooping 99% which is way better than previous three model, precision and recall rate is close to perfect for this model, so we will use Random forest as the final model

Hyper Parameter Tuning :

Although, we have got 99% accurate model, lets see if we can further improve the performance by tuning the parameter of random forest

Hyper Parameter tuning

```
from sklearn.model_selection import GridSearchCV
RandomForestClassifier()
param = {'criterion':['gini','entropy'],'min_samples_leaf': range(1,5),'min_samples_split': range(1,5),'max_depth':range

grd = GridSearchCV(rf, param_grid=param)

grd.fit(x_train, y_train)

print(grd.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 2, 'random_state': 3}
```

```
ri = 0
acc = 0
for i in range(1,100):
    rf = RandomForestClassifier(criterion='gini', max_depth=105, min_samples_leaf=1,min_samples_split=2,random_state=i)
    rf.fit(x_train,y_train)
    y_pred = rf.predict(x_test)
    pacc = accuracy_score(y_pred,y_test)

    if pacc > acc:
        acc = pacc
        ri = i
        score = pscore
        print('Accurancy Score - ',acc,'random state -',ri)
```

```
Accurancy Score -  0.985981308411215 random state - 1
Accurancy Score -  0.9883177570093458 random state - 3
Accurancy Score -  0.9906542056074766 random state - 6
Accurancy Score -  0.9929906542056075 random state - 22
```

After tweaking the parameter of the random forest, we are able to improve the model further by .24% so we will re train the model using the new parameter

```
x_train, x_test, y_train, y_test = train_test_split(x_scaled,y,train_size=.8,random_state=20)
rf = RandomForestClassifier(criterion='gini', max_depth=105, min_samples_leaf=1,min_samples_split=2,random_state=22)
rf.fit(x_train,y_train)
y_pred = rf.predict(x_test)
pacc = accuracy_score(y_test,y_pred)
pscore = rf.score(x_test,y_test)

print('Accuracy Score ---',accuracy_score(y_test,y_pred)*100,'%')
print('\n-------- Classification Report --------\n',classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
Accuracy Score --- 99.29906542056075 %

-------- Classification Report --------
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       205
           1       0.99      1.00      0.99       223

    accuracy                           0.99       428
   macro avg       0.99      0.99      0.99       428
weighted avg       0.99      0.99      0.99       428

[[203   2]
 [  1 222]]
```

Saving the model :

Now we have saved the model for future use

Saving the best Model - Random Forest

```
import pickle

filename = 'final_model.pkl'
pickle.dump(rf, open('rf.pkl', 'wb'))
```

Conclusion.

Dataset was quite clear, there was no missing value. It was mix of categorical and numerical features. We have performed multiple analyses to check that which factor plays important role in attrition. We checked outlier and found that few columns have some extreme value but it is very close to upper whisker and we didn't try treating them because the ensemble methods will deal with them. I have checked correlated of each features and found that couple of features were correlated so have deleted them.

As we saw at the initial phase of analysis that data was imbalance, we have corrected that by applying oversampling technique and then Model was trained. Random forest has given best F1 score and has taken it for final model.