HR Analytics Project

By

PUNEET JAWA

Today I am going to write about HR Analytics Project to understanding the Attrition in HR which should serve as a guiding path for many Data Science aspirants. When I began my journey into studying Data Science, I doubted myself , whether I have made correct decision by going through the content that is available online and most of them scattered into chunks emphasizing on a deeper knowledge that a newbie like me would be able to understand the concepts of this new technology named machine learning. I've done multiple projects now on this technology and with every project got new learning of data Science field and still going on.

So, now without any further delay, let me explain the agenda for this blog post. In this article, I have jotted down all the techniques in the form of sub-topics that I will be explaining one by one. And those pointers are as follows:

1. Problem Definition
2. Data Analysis
3. EDA
4. Pre-processing Data
5. Building Machine Learning Models
6. Concluding Remarks

Let's start with the problem definition or a short introduction on the project that I have chosen to elaborate.

1. Problem Definition

The project I am using for this article is for HR Analytics which is basically a fictional dataset that was created by the Data Scientists at IBM. the dataset is in the link provided. Basically, every year a lot of companies hire a number of employees. The companies invest time and money in training those employees, not just this but there are training programs within the companies for their existing employees as well. The aim of these programs is to increase the effectiveness of their employees. But where HR Analytics fit in this? and is it just about improving the performance of employees?

Human resource analytics (HR analytics) is an area in the field of analytics that refers to applying analytic processes to the human resource department of an organization in the hope of improving employee performance and therefore getting a better return on investment.

HR analytics does not just deal with gathering data on employee efficiency. Instead, it aims to provide insight into each process by gathering data and then using it to make relevant decisions about how to improve these processes.

Attrition in human resources refers to the gradual loss of employees overtime. In general, relatively high attrition is problematic for companies. HR professionals often assume a leadership role in designing company compensation programs, work culture, and motivation systems that help the organization retain top employees. How does Attrition affect companies? and how does HR Analytics help in analyzing attrition? We will discuss the first question here and for the second question, we will write the code and try to understand the process step by step.

Attrition affecting companies is a major problem since high employee attrition is its cost to an organization. Job postings, hiring processes, paperwork, and new hire training are some of the common expenses of losing employees and replacing them. Additionally, regular employee turnover prohibits your organization from increasing its collective knowledge base and experience over time. This is especially concerning if your business is customer-facing, as customers often prefer to interact with familiar people. Errors and issues are more likely if you constantly have new workers.

Therefore the major goal of this project is to identify the “Attrition” rate as a simple Yes or a No tag making this to be a classification problem!

First we are going to import all the necessary libraries that will be used in our project and obtain the rest as and when required.

```
#Data Analysis and Data Wrangling
import pandas as pd
import numpy as np

#Data Visulaziation
import matplotlib.pyplot as plt
import seaborn as sns

#for model building
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier


#for spliting dataset into train and test
from sklearn.model_selection import train_test_split

#for model evaluation
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

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

Before we begin with any process, we need to get the dataset in our Jupyter Notebook that can be achieved by a single step.

```
hr=pd.read_csv(r"D:\DT EVAL Projects\HR.csv")
```

This gives us our entire dataset stored in the variable name “hr” for our dataframe.

2. Data Analysis

For data analysis part, we will use different commands, that are as follows:

```
hr.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	...	RelationshipS
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1	...	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2	...	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4	...	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5	...	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7	...	

5 rows × 35 columns

head() function is used to display the columns heading in the dataset.

```
: #checking dimension of dataset

  hr.shape
```

```
: (1470, 35)
```

There are 1470 rows and 35 columns including target variable

```
: hr.columns
```

```
: 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')
```

shape function, we can interpret how much rows and columns are there in our dataset.

columns function, we can interpret what all columns are there in our dataset.

3. EDA

EDA also known as Exploratory Data Analysis and is considered the most important aspect in Data Science industry by most Data Scientist. After following a huge number of expert Data Scientist on various platforms I can confirm one thing that it boils down to a single important thing of conveying a story on how we are able to achieve each and every step via code showing the provided problem statement, the observation, the challenges faced and what was done to tackle or rectify those issues.

Building a good model comes only when we understand clearly what we are doing and why we are doing it. Making sure that we have a clean data in proper processed format to feed into our model and get appropriate result. Because no amount of Machine Learning model usage and hyper parameter tuning is going to help if we have not invested time to sort out and fix our data that's the only input we have in hand.

The first thing I am going to take a look at is the missing data information in our dataset by using the codes below, also including the output after executing the code.

```
#checking null values

hr.isnull().sum()
```

```
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
```

There are no null values in the dataset

Let's check once heatmap also to be double sure of null values, here is the code:

```
#Lets plot heatmap

sns.heatmap(hr.isnull())
plt.show()
```

Next, let's move on to using the describe method to take a look at the count value, mean data, standard deviation information and the minimum, maximum, 25% quartile, 50% quartile and 75% quartile details. As the describe method works best for numeric data all the object (text) type data gets ignored. Take a look at the below code and we will get an idea on how to use it.

```
hr.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Age	1470.0	36.923810	9.135373	18.0	30.00	36.0	43.00	60.0
DailyRate	1470.0	802.485714	403.509100	102.0	465.00	802.0	1157.00	1499.0
DistanceFromHome	1470.0	9.192517	8.106864	1.0	2.00	7.0	14.00	29.0
Education	1470.0	2.912925	1.024165	1.0	2.00	3.0	4.00	5.0
EmployeeCount	1470.0	1.000000	0.000000	1.0	1.00	1.0	1.00	1.0
EmployeeNumber	1470.0	1024.865306	602.024335	1.0	491.25	1020.5	1555.75	2068.0
EnvironmentSatisfaction	1470.0	2.721769	1.093082	1.0	2.00	3.0	4.00	4.0
HourlyRate	1470.0	65.891156	20.329428	30.0	48.00	66.0	83.75	100.0
JobInvolvement	1470.0	2.729932	0.711561	1.0	2.00	3.0	3.00	4.0
JobLevel	1470.0	2.063946	1.106940	1.0	1.00	2.0	3.00	5.0
JobSatisfaction	1470.0	2.728571	1.102846	1.0	2.00	3.0	4.00	4.0
MonthlyIncome	1470.0	6502.931293	4707.956783	1009.0	2911.00	4919.0	8379.00	19999.0
MonthlyRate	1470.0	14313.103401	7117.786044	2094.0	8047.00	14235.5	20461.50	26999.0
NumCompaniesWorked	1470.0	2.693197	2.498009	0.0	1.00	2.0	4.00	9.0
PercentSalaryHike	1470.0	15.209524	3.659938	11.0	12.00	14.0	18.00	25.0
PerformanceRating	1470.0	3.153741	0.360824	3.0	3.00	3.0	3.00	4.0
RelationshipSatisfaction	1470.0	2.712245	1.081209	1.0	2.00	3.0	4.00	4.0
StandardHours	1470.0	80.000000	0.000000	80.0	80.00	80.0	80.00	80.0
StockOptionLevel	1470.0	0.793878	0.852077	0.0	0.00	1.0	1.00	3.0
TotalWorkingYears	1470.0	11.279592	7.780782	0.0	6.00	10.0	15.00	40.0
TrainingTimesLastYear	1470.0	2.799320	1.289271	0.0	2.00	3.0	3.00	6.0
WorkLifeBalance	1470.0	2.761224	0.706476	1.0	2.00	3.0	3.00	4.0
YearsAtCompany	1470.0	7.008163	6.126525	0.0	3.00	5.0	9.00	40.0
YearsInCurrentRole	1470.0	4.229252	3.623137	0.0	2.00	3.0	7.00	18.0
YearsSinceLastPromotion	1470.0	2.187755	3.222430	0.0	0.00	1.0	3.00	15.0
YearsWithCurrManager	1470.0	4.123129	3.568136	0.0	2.00	3.0	7.00	17.0

When we are able to draw insights from the describe method we can take a look at the datatype information using the code below and that shall give us the list of all the columns marking them to be either integer, float or object datatype depending on the values present inside the columns.

Below is the code with output.

```
hr.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
```

This is explaining the datatypes of all the columns present in our data frame. We also get an opportunity to drop or remove any unwanted columns from the data frame here.

Next we will check the unique values and will drop the unnecessary columns also which is of no use for us.

```
for column in hr.columns:
    print(f"{column}: Number of unique values {hr[column].nunique()}")
```

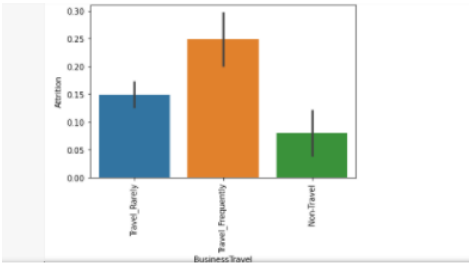
```
Age: Number of unique values 43
Attrition: Number of unique values 2
BusinessTravel: Number of unique values 3
DailyRate: Number of unique values 886
Department: Number of unique values 3
DistanceFromHome: Number of unique values 29
Education: Number of unique values 5
EducationField: Number of unique values 6
EmployeeCount: Number of unique values 1
EmployeeNumber: Number of unique values 1470
EnvironmentSatisfaction: Number of unique values 4
Gender: Number of unique values 2
HourlyRate: Number of unique values 71
JobInvolvement: Number of unique values 4
JobLevel: Number of unique values 5
JobRole: Number of unique values 9
JobSatisfaction: Number of unique values 4
MaritalStatus: Number of unique values 3
MonthlyIncome: Number of unique values 1349
MonthlyRate: Number of unique values 1427
NumCompaniesWorked: Number of unique values 10
Over18: Number of unique values 1
OverTime: Number of unique values 2
PercentSalaryHike: Number of unique values 15
PerformanceRating: Number of unique values 2
RelationshipSatisfaction: Number of unique values 4
StandardHours: Number of unique values 1
StockOptionLevel: Number of unique values 4
TotalWorkingYears: Number of unique values 40
TrainingTimesLastYear: Number of unique values 7
WorkLifeBalance: Number of unique values 4
YearsAtCompany: Number of unique values 37
YearsInCurrentRole: Number of unique values 19
YearsSinceLastPromotion: Number of unique values 16
YearsWithCurrManager: Number of unique values 18
```

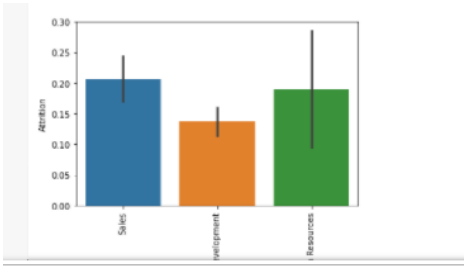
```
#removing unnessary columns which are not impacting target variable

hr.drop(['EmployeeCount','EmployeeNumber','Over18','StandardHours'],axis='columns',inplace=True)
```

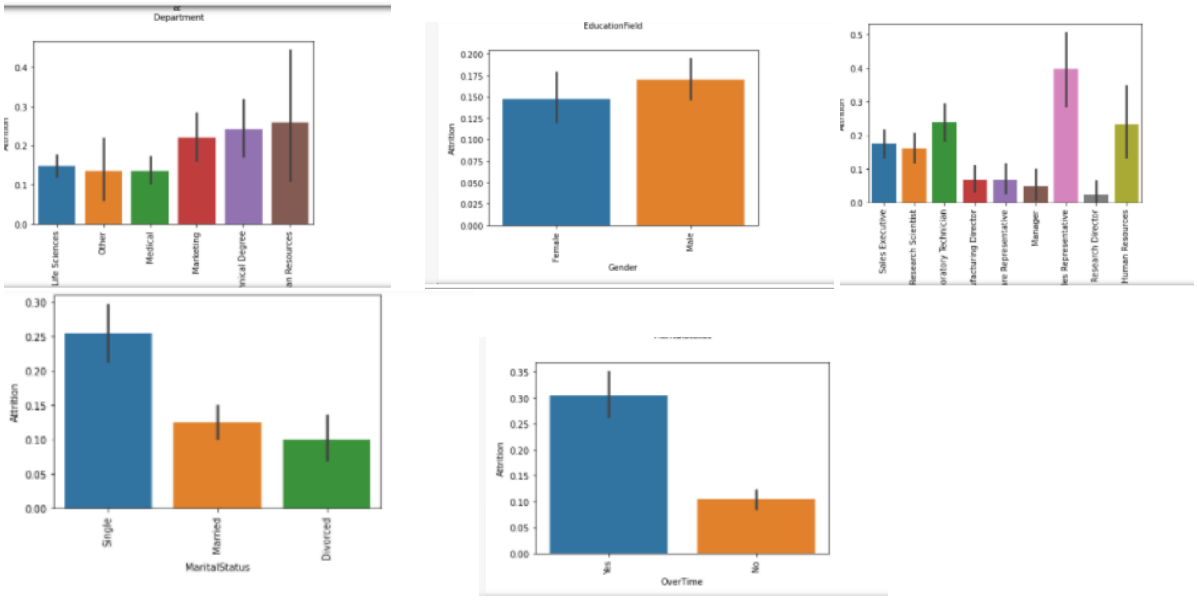
Next step is to list down all the visualization codes with their output.

```
In [15]: #generating charts that compare all the categorical variables with Attrition variable

columns=['BusinessTravel','Department','EducationField','Gender','JobRole','MaritalStatus','OverTime']
for i,col in enumerate(columns):
    plt.figure(i)
    sns.barplot(x=col,y='Attrition',data=hr)
    plt.xticks(rotation=90)
```

Output for above code, contains multiple snips.

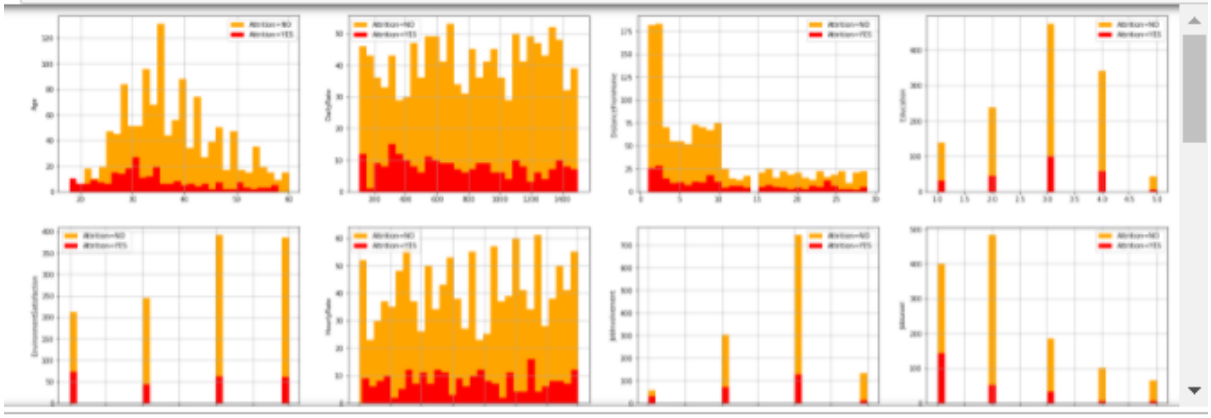






next code:

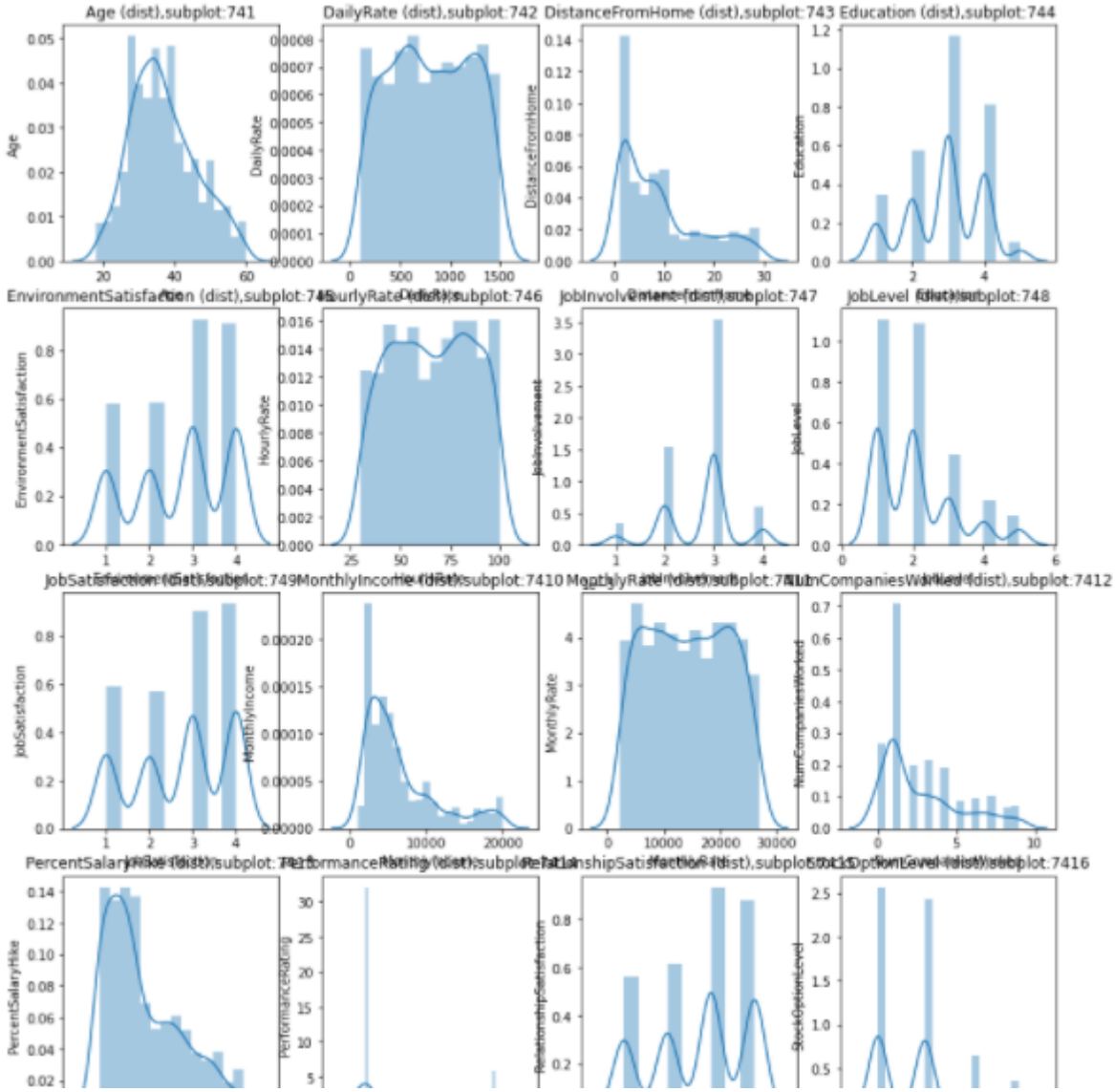
```
plt.figure(figsize=(30,40))

for i,column in enumerate(hr_count,1):
    plt.subplot(7,4,i)
    hr[hr["Attrition"]==0][column].hist(bins=30,color='orange',label='Attrition=NO',alpha=1)
    hr[hr["Attrition"]==1][column].hist(bins=30,color='red',label='Attrition=YES',alpha=1)
    plt.legend()
    plt.ylabel(column)
```



next code:

```
Columns=hr_count.columns

#plot Numearical Data

a= 7 #no of rows
b= 4 #no of columns
c= 1 #initialize plot counter

fig=plt.figure(figsize=(14,28))

for i in Columns:
    plt.subplot(a,b,c)
    plt.title('{} (dist),subplot:{}{}{}'.format(i,a,b,c))
    plt.ylabel(i)
    sns.distplot(hr[i])
    c=c+1

plt.show()
```

Output of above code:



We can see that with the help of above codes and the outputs I was able to take a look at all the column values/counts, bar/histplot gave me a view on the presence of outliers and the distribution plots showed me the skewness information that will needed to be treated. These are like the challenges that will need to be dealt with before I even think of building my Classification Machine Learning models.

4. Pre-processing Data

In the pre-processing step I am going to tackle all the miss fits and fix them one by one starting with the problem that out dataset has object datatype values where as our Machine Learning models can only understand numeric values. I am making use of the encoding methods to convert all the object datatype values. For our label I am using Label Encoder.

Code:

```
from sklearn.preprocessing import LabelEncoder

label=LabelEncoder()
hr['Attrition']=label.fit_transform(hr.Attrition)
```

```
#Label encoding object variables

from sklearn.preprocessing import LabelEncoder
cols=['BusinessTravel','Department','EducationField','Gender','JobRole','MaritalStatus','OverTime']
label=LabelEncoder()
hr[cols]=hr[cols].apply(LabelEncoder().fit_transform)

#print head

hr.head()
```

Output:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EnvironmentSatisfaction	Gender	...	PerformanceRatin
0	41	1	2	1102	2	1	2	1	2	0	...	
1	49	0	1	279	1	8	1	1	3	1	...	
2	37	1	2	1373	1	2	2	4	4	1	...	
3	33	0	1	1392	1	3	4	1	4	0	...	
4	27	0	2	591	1	2	1	3	1	1	...	

5 rows × 31 columns

After I have encoded all the columns in our dataset, next I'll check for correlation.

Here is the code:

```
hr.corr()
```

Output :

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EnvironmentSatisfaction
Age	1.000000	-0.159205	0.024751	0.010661	-0.031882	-0.001686	0.208034	-0.040873	0.010146
Attrition	-0.159205	1.000000	0.000074	-0.056652	0.063991	0.077924	-0.031373	0.026846	-0.103369
BusinessTravel	0.024751	0.000074	1.000000	-0.004086	-0.009044	-0.024469	0.000757	0.023724	0.004174
DailyRate	0.010661	-0.056652	-0.004086	1.000000	0.007109	-0.004985	-0.016806	0.037709	0.018355
Department	-0.031882	0.063991	-0.009044	0.007109	1.000000	0.017225	0.007996	0.013720	-0.019395
DistanceFromHome	-0.001686	0.077924	-0.024469	-0.004985	0.017225	1.000000	0.021042	0.002013	-0.016075
Education	0.208034	-0.031373	0.000757	-0.016806	0.007996	0.021042	1.000000	-0.039592	-0.027128
EducationField	-0.040873	0.026846	0.023724	0.037709	0.013720	0.002013	-0.039592	1.000000	0.043163
EnvironmentSatisfaction	0.010146	-0.103369	0.004174	0.018355	-0.019395	-0.016075	-0.027128	0.043163	1.000000
Gender	-0.036311	0.029453	-0.032981	-0.011716	-0.041583	-0.001851	-0.016547	-0.002504	0.000508
HourlyRate	0.024287	-0.006846	0.026528	0.023381	-0.004144	0.031131	0.016775	-0.021941	-0.049857
JobInvolvement	0.029820	-0.130016	0.039062	0.046135	-0.024586	0.008783	0.042438	-0.002655	-0.008278
JobLevel	0.509604	-0.169105	0.019311	0.002966	0.101963	0.005303	0.101589	-0.044933	0.001212
JobRole	-0.122427	0.067151	0.002724	-0.009472	0.662431	-0.001015	0.004236	0.015599	-0.017321
JobSatisfaction	-0.004892	-0.103481	-0.033962	0.030571	0.021001	-0.003669	-0.011296	-0.034401	-0.006784
MaritalStatus	-0.095029	0.162070	0.024001	-0.069586	0.056073	-0.014437	0.004053	0.014420	-0.003593
MonthlyIncome	0.497855	-0.159840	0.034319	0.007707	0.053130	-0.017014	0.094961	-0.041070	-0.006259
MonthlyRate	0.028051	0.015170	-0.014107	-0.032182	0.023642	0.027473	-0.026084	-0.027182	0.037600
NumCompaniesWorked	0.299635	0.043494	0.020875	0.038153	-0.035882	-0.029251	0.126317	-0.008663	0.012594
OverTime	0.028062	0.246118	0.016543	0.009135	0.007481	0.025514	-0.020322	0.002259	0.070132
PercentSalaryHike	0.003634	-0.013478	-0.029377	0.022704	-0.007840	0.040235	-0.011111	-0.011214	-0.031701
PerformanceRating	0.001904	0.002889	-0.026341	0.000473	-0.024604	0.027110	-0.024539	-0.005614	-0.029548
RelationshipSatisfaction	0.053535	-0.045872	-0.035986	0.007846	-0.022414	0.006557	-0.009118	-0.004378	0.007665
StockOptionLevel	0.037510	-0.137145	-0.016727	0.042143	-0.012193	0.044872	0.018422	-0.016185	0.003432
TotalWorkingYears	0.680381	-0.171063	0.034226	0.014515	-0.015762	0.004628	0.148280	-0.027848	-0.002693
TrainingTimesLastYear	-0.019621	-0.059478	0.015240	0.002453	0.036875	-0.036942	-0.025100	0.049195	-0.019359
WorkLifeBalance	-0.021490	-0.063939	-0.011256	-0.037848	0.026383	-0.026556	0.009819	0.041191	0.027627
YearsAtCompany	0.311309	-0.134392	-0.014575	-0.034055	0.022920	0.009508	0.069114	-0.018692	0.001458
YearsInCurrentRole	0.212901	-0.160545	-0.011497	0.009932	0.056315	0.018845	0.060236	-0.010506	0.018007
YearsSinceLastPromotion	0.216513	-0.033019	-0.032591	-0.033229	0.040061	0.010029	0.054254	0.002326	0.016194
YearsWithCurrManager	0.202089	-0.156199	-0.022636	-0.026363	0.034282	0.014406	0.069065	-0.004130	-0.004999

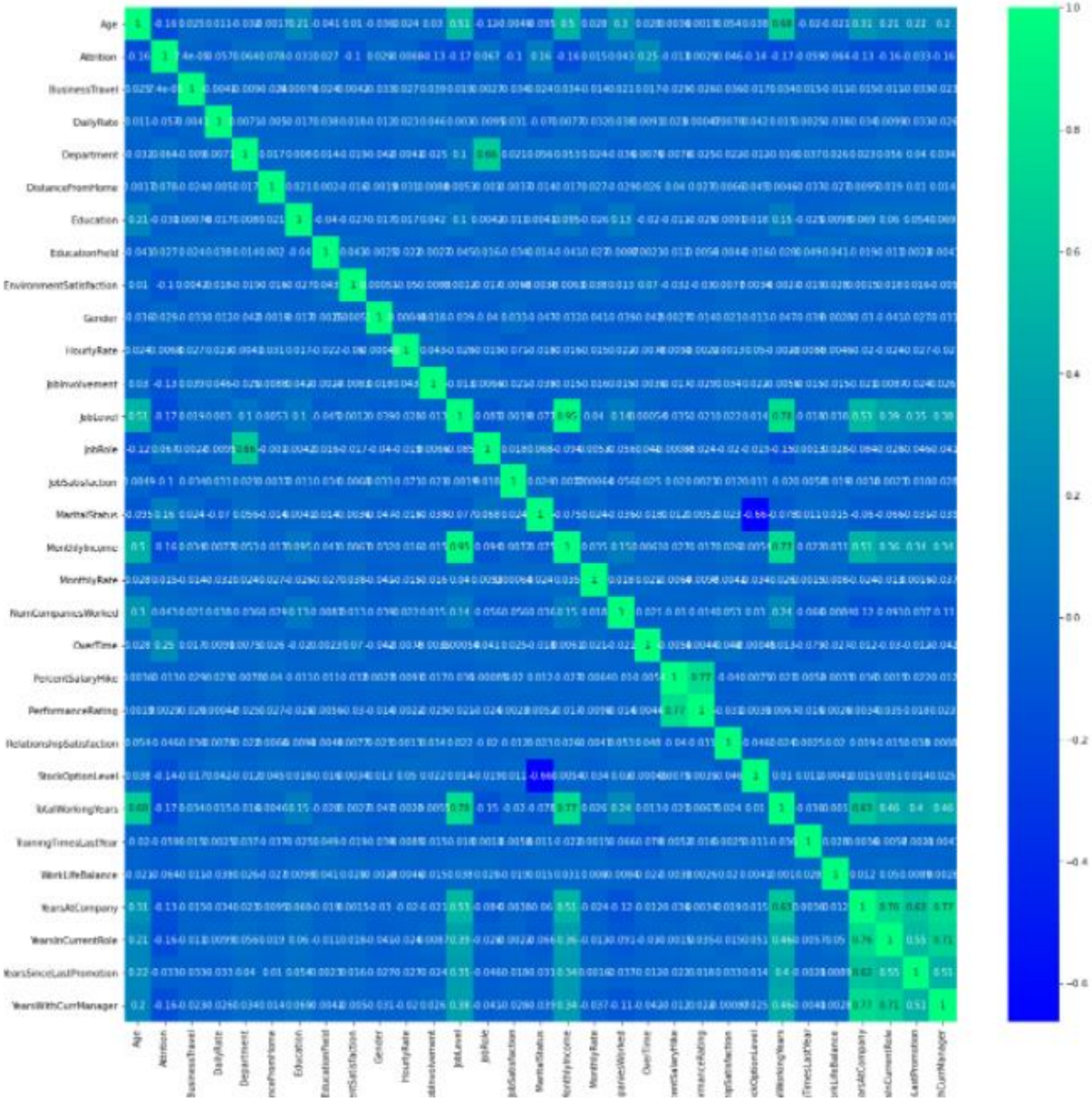
31 rows × 31 columns

Next is Heatmap for the correlation:

Code:

```
#heatmap for correlation

plt.figure(figsize=(20,20))
sns.heatmap(hr.corr(),annot=True,cmap='winter',annot_kws={"size":10})
```

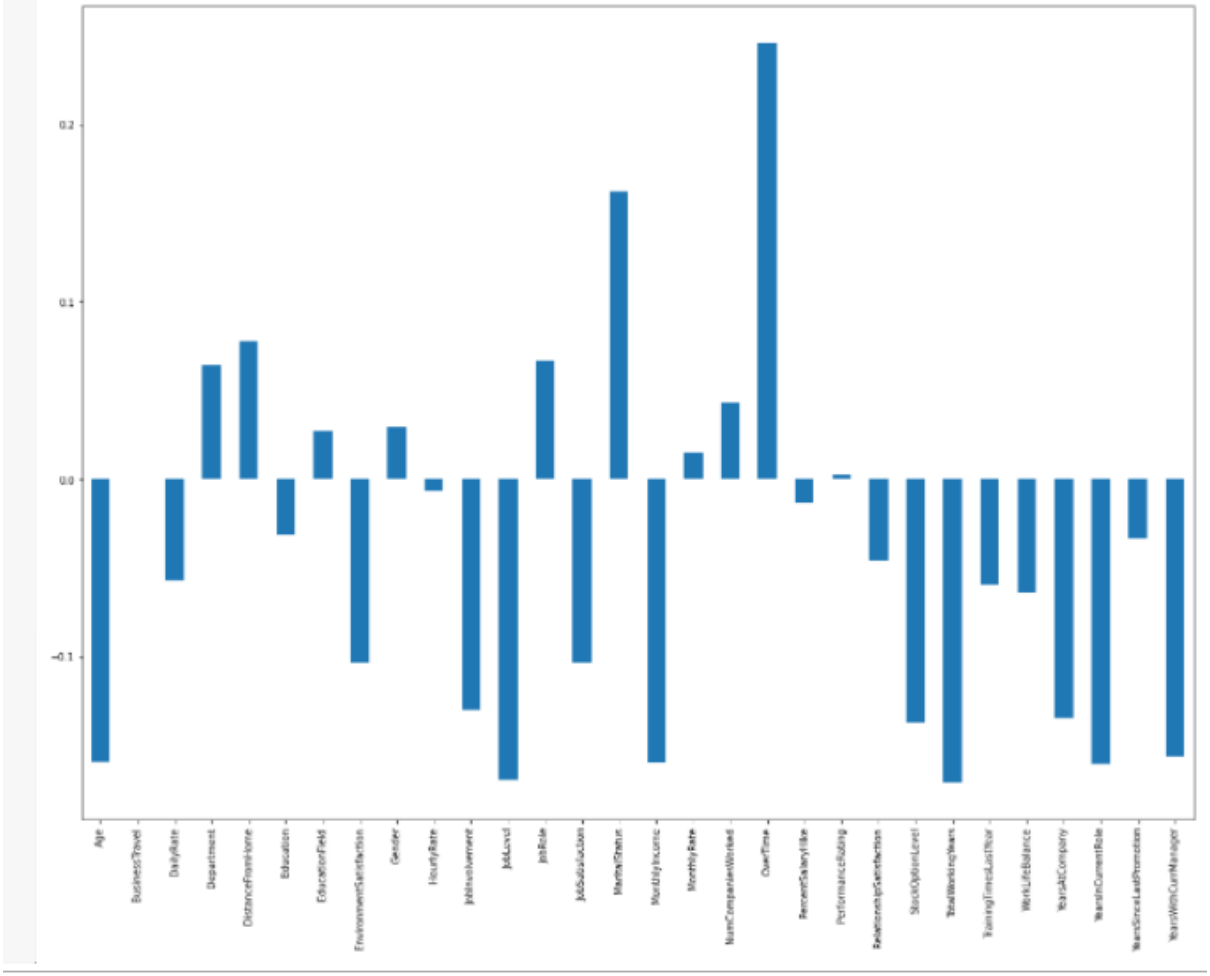
Output:



correlation between our label and feature columns I use a Bar Plot comparison and code is below:

```
hr.drop('Attrition',axis=1).corrwith(hr.Attrition).plot(kind='bar',figsize=(20,15))
```

Output:



In the above Bar Plot we are able to clearly define the feature columns that are positively correlated with our label and the feature columns that are negatively correlated with our label. Now coming back to the outlier and skewness concern in our dataset I will be using the Z score method.

Code:

```
# zscore technique

from scipy.stats import zscore
z=np.abs(zscore(hr))
```

```
hr_new=hr[(z<3).all(axis=1)]
print(hr.shape)
print(hr_new.shape)
```

```
(1470, 31)
(1387, 31)
```

```
loss=((1470-1387)/1470)*100
loss
```

```
5.646258503401361
```

only 5.6% of data are outliers. lets remove it

As for the usage of Z score, I was able to lose only about 5% of data.

To Remove Skewness, I will use yeo johnson method because of negative skewness.

Here is the code :

```
#removing skewness

from sklearn.preprocessing import power_transform

#using yeo-johnson due to negative skewness

X_new=power_transform(X,method='yeo-johnson')
X_new=pd.DataFrame(X_new,columns=X.columns)
```

After dealing with the data concerns I will then split our columns into feature and label. I am storing the feature columns in X and the target label column in the Y variable.

```
X=hr_new.drop('Attrition',axis=1)
y=hr_new['Attrition']
```

Now I will scale the feature columns that is stored in the X variable to avoid any kind of biasness over column values. Some integers cover thousands place and some cover hundreds or tens place then it can make the machine learning model assume the column with thousands place has a higher importance when in real that won't be true due to difference in unit range.
Code:

```
#to normalize the data

from sklearn.preprocessing import StandardScaler

scale=StandardScaler()
X=scale.fit_transform(X_new)
```

Next step is to check the best random state for the machine learning models.
Code:

```
Maxacc=0
MaxRS=0

for i in range(1,200):
    X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=i)
    Logic=LogisticRegression()
    Logic.fit(X_train,y_train)
    pred=Logic.predict(X_test)
    accur=accuracy_score(y_test,pred)
    if accur>Maxacc:
        Maxacc=accur
        MaxRS=i

print("Best Accuracy is ",Maxacc,"and random_state",MaxRS)
```

Now I will use the train test split to bifurcate our entire data set into training data and testing data. Here I am using 6% data for training purpose and 30% data for testing purpose. Some people provide training and test data separately as well and hence it completely depends on us how we want to use this step, code is below:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=6)
```

5. Building Machine Learning Models

In order to build a classification method I have imported the necessary libraries and created a function that contains all our machine learning model creation and its evaluation metrics steps. This makes our job easier since later on we just need to feed the model's name and get the result without repeating/rewriting the same code again and again.

There are 4 models that I've used here to evaluate the best from those 4.

a) KNeighbors Classifier
b) Random Forest Classifier
c) AdaBoost Classifier
d) Logistic Regression.

Code:

```
#for model building
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier


#for splitting dataset into train and test
from sklearn.model_selection import train_test_split

#for model evaluation
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

Code:

```
models=[LogisticRegression(),

        KNeighborsClassifier(),RandomForestClassifier(),
        AdaBoostClassifier()]
```

```
for model in range(len(models)):
    mode=models[model]
    mode.fit(X_train,y_train)
    pred=mode.predict(X_test)
    print(models[model],"accuracy score",accuracy_score(y_test,pred))
```

```
LogisticRegression() accuracy score 0.8992805755395683
KNeighborsClassifier() accuracy score 0.8465227817745803
RandomForestClassifier() accuracy score 0.8800959232613909
AdaBoostClassifier() accuracy score 0.8657074340527577
```

In above models we observe that LogisticRegression has good accuracy score. Lets do cross validation for best model

Code:

```
for model in range(len(models)):
    mode=models[model]
    mode.fit(X_train,y_train)
    pred=mode.predict(X_test)
    print("Confusion Matrix\n",confusion_matrix(y_test,pred))
    print("Classification Report\n",classification_report(y_test,pred))
```

Output:

```
Confusion Matrix
 [[343  10]
 [ 32  32]]
Classification Report
              precision    recall  f1-score   support

           0       0.91      0.97      0.94       353
           1       0.76      0.50      0.60        64

    accuracy                           0.90       417
   macro avg       0.84      0.74      0.77       417
weighted avg       0.89      0.90      0.89       417

Confusion Matrix
 [[343  10]
 [ 54  10]]
Classification Report
              precision    recall  f1-score   support

           0       0.86      0.97      0.91       353
           1       0.50      0.16      0.24        64

    accuracy                           0.85       417
   macro avg       0.68      0.56      0.58       417
weighted avg       0.81      0.85      0.81       417

Confusion Matrix
 [[351   2]
 [ 50  14]]
Classification Report
              precision    recall  f1-score   support

           0       0.88      0.99      0.93       353
           1       0.88      0.22      0.35        64

    accuracy                           0.88       417
   macro avg       0.88      0.61      0.64       417
weighted avg       0.88      0.88      0.84       417

Confusion Matrix
 [[339  14]
 [ 42  22]]
Classification Report
              precision    recall  f1-score   support

           0       0.89      0.96      0.92       353
           1       0.61      0.34      0.44        64

    accuracy                           0.87       417
   macro avg       0.75      0.65      0.68       417
weighted avg       0.85      0.87      0.85       417
```

Next step is checking cross val score:

```
from sklearn.model_selection import cross_val_score
```

```
for model in range(len(models)):
    cross=cross_val_score(models[model],X,y,cv=5)
    print("Cross val score for : ",models[model],cross.mean())
```

```
Cross val score for :  LogisticRegression() 0.8695036750383087
Cross val score for :  KNeighborsClassifier() 0.8478741916214322
Cross val score for :  RandomForestClassifier() 0.8594083577903019
Cross val score for :  AdaBoostClassifier() 0.8673402072565775
```

we can observe KNeighborsClassifier, AdaBoostClassifier have least difference between Accuracy_score and cross_val_score. so we take this 2 models and perform hypertuning and AUC_ROC curve for better accuracy

Next step is Hyper tunning the model:

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

```
from sklearn.model_selection import GridSearchCV
```

We have applied hyper parameter tunning for 2 models i.e KNN and Adaboost to find out our best model.

For KNN, here we got accuracy score of 85%

```
KNN_param={'n_neighbors':np.arange(1,16),
           'weights':('uniform','distance'),
           'algorithm':('auto','ball_tree','kd_tree','brute')}
```

```
KNN=GridSearchCV(KNeighborsClassifier(),KNN_param,cv=5)
```

```
KNN.fit(X_train,y_train)
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ('auto', 'ball_tree', 'kd_tree', 'brute'),
                         'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
                         'weights': ('uniform', 'distance')})
```

```
KNN.best_params_
```

```
{'algorithm': 'auto', 'n_neighbors': 10, 'weights': 'distance'}
```

```
KNN_mod=KNeighborsClassifier(algorithm='auto',n_neighbors=10,weights='distance')
KNN_mod.fit(X_train,y_train)
pred=KNN_mod.predict(X_test)
print(accuracy_score(y_test,pred)*100)
```

```
85.37170263788968
```

For ADABoost, here we got accuracy score of 87%

```
ABC_param={'n_estimators':range(5,20),'algorithm':('SAMME','SAMME.R'),'random_state':range(20,40)}
```

```
Ada=GridSearchCV(AdaBoostClassifier(),ABC_param,cv=5)
```

```
Ada.fit(X_train,y_train)
```

```
GridSearchCV(cv=5, estimator=AdaBoostClassifier(),
             param_grid={'algorithm': ('SAMME', 'SAMME.R'),
                         'n_estimators': range(5, 20),
                         'random_state': range(20, 40)})
```

```
Ada.best_params_
```

```
{'algorithm': 'SAMME', 'n_estimators': 16, 'random_state': 20}
```

```
Ada_mod=AdaBoostClassifier(algorithm='SAMME',n_estimators=16,random_state=20)
Ada_mod.fit(X_train,y_train)
pred=Ada_mod.predict(X_test)
print(accuracy_score(y_test,pred)*100)
```

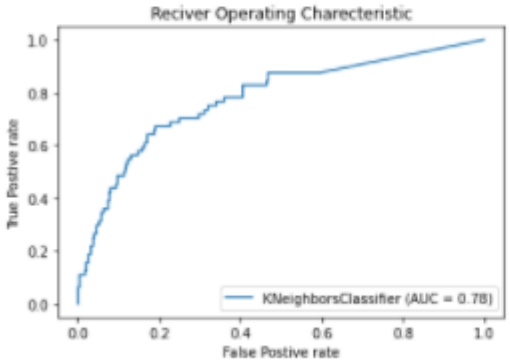
```
87.05035971223022
```

AUC/ROC Cure:

In Machine Learning, performance measurement is an essential task. When it comes to a classification problem, we can count on an AUC - ROC Curve. When we need to check or visualize the performance of the multi-class classification problem, we use the AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (Area Under the Receiver Operating Characteristics).

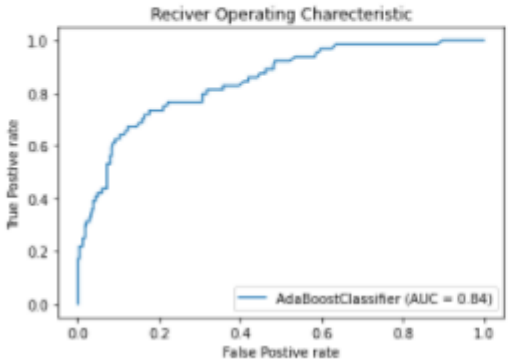
AUC score for KNN is 78%.

```
: from sklearn.metrics import plot_roc_curve

plot_roc_curve(KNN_mod,X_test,y_test)
plt.title("Reciver Operating Charecteristic")
plt.xlabel('False Postive rate')
plt.ylabel('True Postive rate')
plt.show()
```



AUC score for ADABOOST is 84%.

```
from sklearn.metrics import plot_roc_curve

plot_roc_curve(Ada_mod,X_test,y_test)
plt.title("Reciver Operating Charecteristic")
plt.xlabel('False Postive rate')
plt.ylabel('True Postive rate')
plt.show()
```



Best model is AdaBoostClassifier.

```
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score
finalabc=AdaBoostClassifier()
finalabc.fit(X_train,y_train)
print(finalabc)
prd=finalabc.predict(X_test)
ac=accuracy_score(y_test,prd)
rcal=recall_score(y_test,prd,average='binary')
prec=precision_score(y_test,prd,average='binary')
f1=f1_score(y_test,prd)

result=pd.DataFrame([['AdaBoostClassifier',ac,rcal,prec,f1]],
                    columns=['Model','Accuracy','Recall','Precision','F1 score'])

result
```

```
AdaBoostClassifier()
```

	Model	Accuracy	Recall	Precision	F1 score
0	AdaBoostClassifier	0.865707	0.34375	0.611111	0.44

Saving Model

```
]: import joblib
```

```
]: joblib.dump(finalabc,'HR.pkl')
```

```
]: ['HR.pkl']
```

conclusion

```
n [59]: abc=np.array(y_test)
        abc
```

```
ut[59]: array([0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
               0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
               1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
               0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
               1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
               0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
               1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
               0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
               1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
               0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0])
```

```
n [60]: predct=np.array(finalabc.predict(X_test))
```

```
n [61]: hr_pred=pd.DataFrame({"Original":abc,"predicted":predct},index=range(len(abc)))
```

```
In [60]: predct=np.array(finalabc.predict(X_test))

In [61]: hr_pred=pd.DataFrame({"Original":abc,"predicted":predct},index=range(len(abc)))

In [62]: hr_pred
```

Out[62]:

	Original	predicted
0	0	0
1	0	0
2	0	0
3	1	1
4	1	1
...	...	...
412	0	0
413	0	0
414	0	1
415	0	0
416	0	0

417 rows × 2 columns

6. **Concluding Remarks**

Let me provide a quick recap on all the steps that we went through starting from understanding the Problem Definition then going through the Data Analysis and EDA processes. We went through the necessary Pre-processing Data steps before the final Building Machine Learning Models step came into picture.

In this entire project I took help from internet to check further and improvise on accuracy or beautify the visuals. However I have observed that other people doing complete copy paste and don't even check it, what is mentioned in that code.

Before wrapping up my only advise to everyone is to try the code on your own and if u are not able to get any proper content or solution then refer to internet for the code.