HR Analytics Project- Understanding the Attrition in HR

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?

Today I am going to write about a complete ene to end project for HR analytics which should solve as guiding path for many data science aspirants .When I began my journey Into studing data science ,I was over whelmed by the contained that is available online and most of them seattered into chunks emphasizing ion a deeper knowledge that a newbie will be most likely not be able to comprehend or connect through .i agree that there are already many projects available into some deep web beneath multiple clicks and pahs hidden like a core of an onion projected by multiple layers .But here I just to my bit in making things easier for a new comer to understand the basic architecture that's require in the real world for creating a data science project.

Show without any further ado please allow me to explain the agenda for this blog post .In this article, I have jotted down all the technique in the form of soft topics that I will be explaining one by one.And those pointers as follows:

1. Problem definition

2.data analysis

3.EDA

4.preprocessing data

5.building machine learning projects

6.concusion

Lets start with the problem with the defination or short introduction on the project that I have choosen to elaborate and why it was made in the first place .

1. Problem definition

The project I am using for this article is for hr analytics which is basically a fictional dataset that created by the data scientist at ibm .You can find the data set here on the kaggle website . Basically ,every year a lot of companies hire a number of employees .The companies invest time and money in training those employee, not just this but there are raining programmes within the companies of their existing employees as well .The aim of these programme is to increase the effectiveness of their employees. But where does hr analytics fit in this ? And is it just about improving the performance of employees?

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

Attrition in human resources refers to the gradual loss of employees' overtime . In general, relatively high attrition is problematic for any company. HR professionals often assume a leadership rule in designing company compensation programs, work culture and motivation systems that help the organisation retain top employees. How does attrition affect companies and how does HR Analytics help in analysing 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, paper work and new hire trainings are some of the common expenses of losing employees and replacing them . Additionally, regular employee turnover prohibits an 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 too.

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!

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from scipy import stats
import feature_engine
import webbrowser
import wikipedia

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

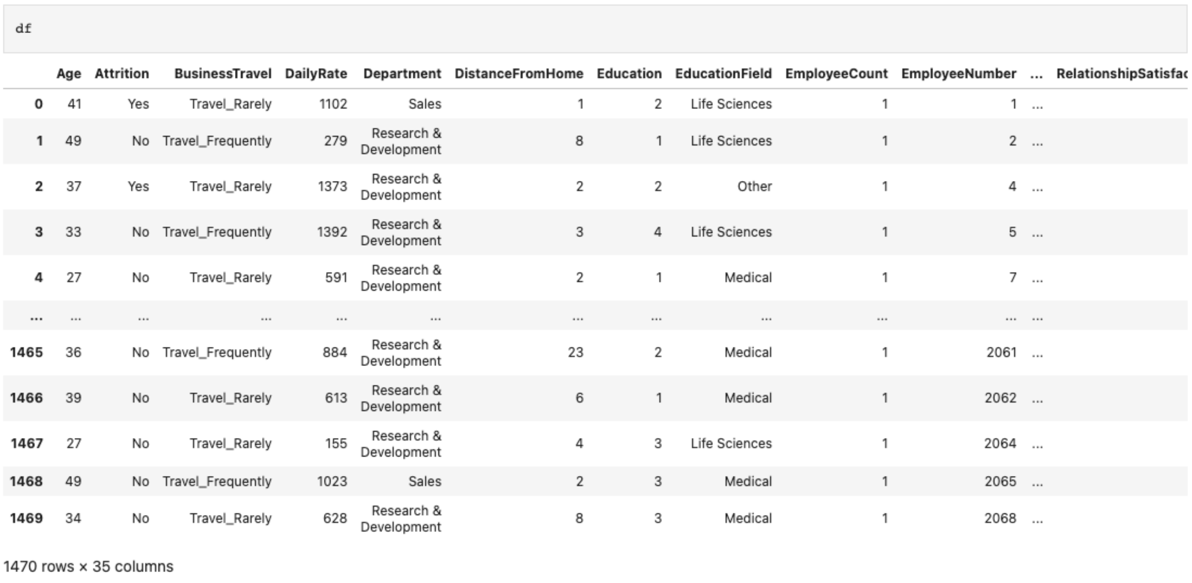
First we are going to import all the necessary dependencies here that will be used in our project and obtain the rest as and when required. Before we begin with any process, we need to get the data set in our Jupyter. Note book that can be achieved by a single step.

```python
df = pd.read_csv('https://raw.githubusercontent.com/Rubysmita/csvfile/main/WA_Fn-UseC_-HR-Employee-Attrition.csv')
```

This gives us our entire data sets stored in the variable name "df" for our data frame.

2. DATA ANALYSIS

For data analysis part we can simply eye ball the contents for our dataset trying to make sense of some columns, its related values and anything that comes to our mind.

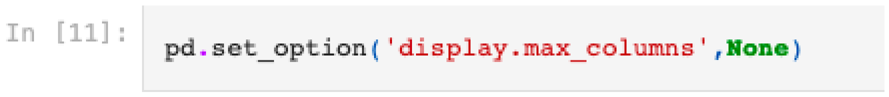


```
df
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	...	RelationshipSatisfac
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	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1465	36	No	Travel_Frequently	884	Research & Development	23	2	Medical	1	2061	...	
1466	39	No	Travel_Rarely	613	Research & Development	6	1	Medical	1	2062	...	
1467	27	No	Travel_Rarely	155	Research & Development	4	3	Life Sciences	1	2064	...	
1468	49	No	Travel_Frequently	1023	Sales	2	3	Medical	1	2065	...	
1469	34	No	Travel_Rarely	628	Research & Development	8	3	Medical	1	2068	...	

1470 rows × 35 columns

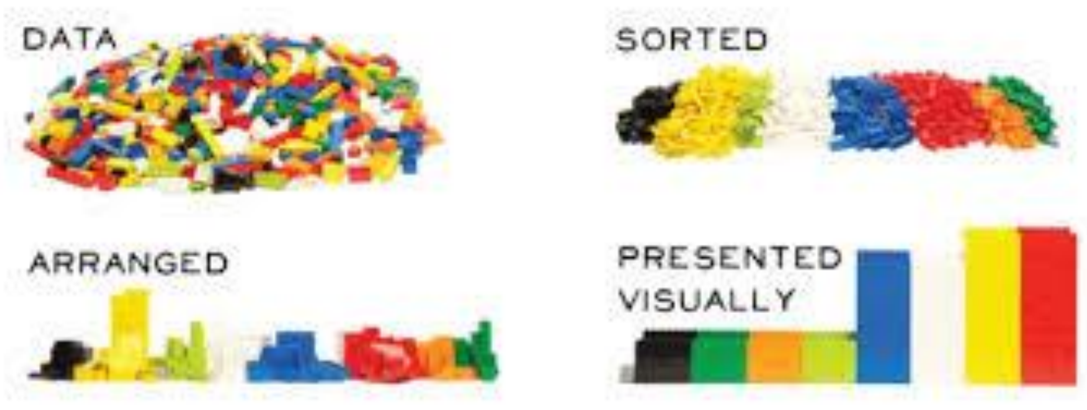
In the above line of code we can see that the total no of rows present in our data set is 1470 and the total number of columns are 35. Since it is a dataset with reasonably higher numbers of rows and columns the visualization gets truncated.



```
In [11]: pd.set_option('display.max_columns',None)
```

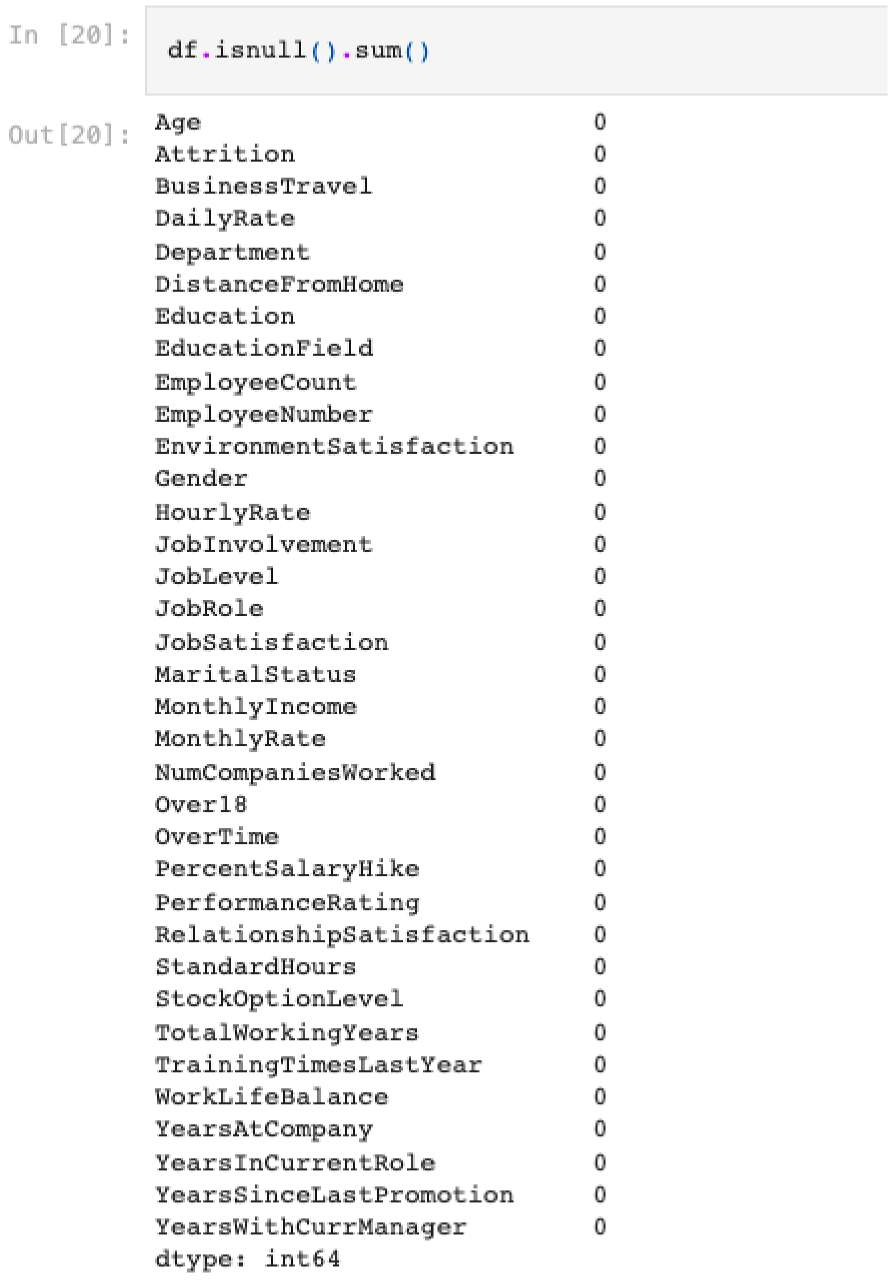
You can write this piece of code and then try to check the dataframe "df" again. It will show you entire row and column information on your jupyter notebook directly.

3. EDA



Eda is the first step in any Data Analytics project .It helps the Data analytics or data scientist to visualize ,summarize , and then interpret the hidden information in the data .Once EDA gets completed ,its features can be used for machine learning models.

Enough of the story telling let me show you the code since talk is cheap, but also necessary at times to explain what really is going on. The first thing I am going to take a look at is the missing data information in our dataset b using the codes below.

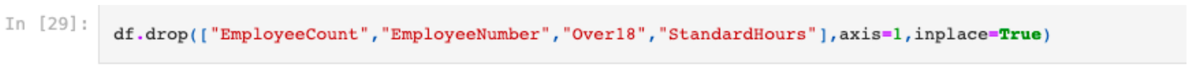


```
In [20]: df.isnull().sum()
```

```
Out[20]: 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
```

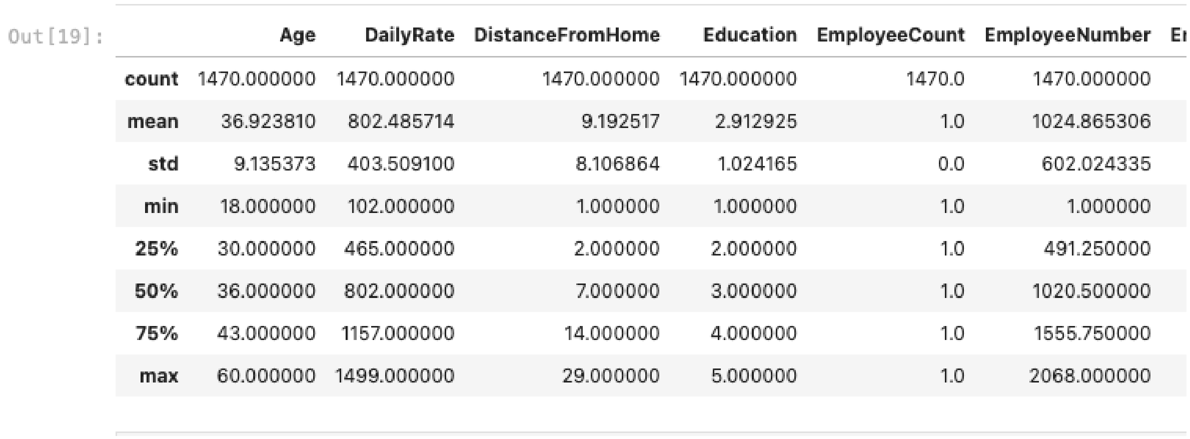
These two codes gives us the missing values information in a tabular and visual format that looks something like his.

Now that we were able to confirm our dataset being free of any missing data we will drop any duplicates that might be present using the code below.



```
In [29]: df.drop(["EmployeeCount","EmployeeNumber","Over18","StandardHours"],axis=1,inplace=True)
```

With the 'drop duplicates' option I was trying to get rid of all the duplicate data present in our dataset. However, we can see that there is no duplicate data existing in our dataset. Next we move on to describe to take a look at the count value, mean data , standard deviation information and the minimum, maximum ,25% quartile,50%quartile,75% quartile details. As the described method works best for numeric data all the object(text) type data gets ignored. Take a look at the below code and you will get an idea on how to use it.

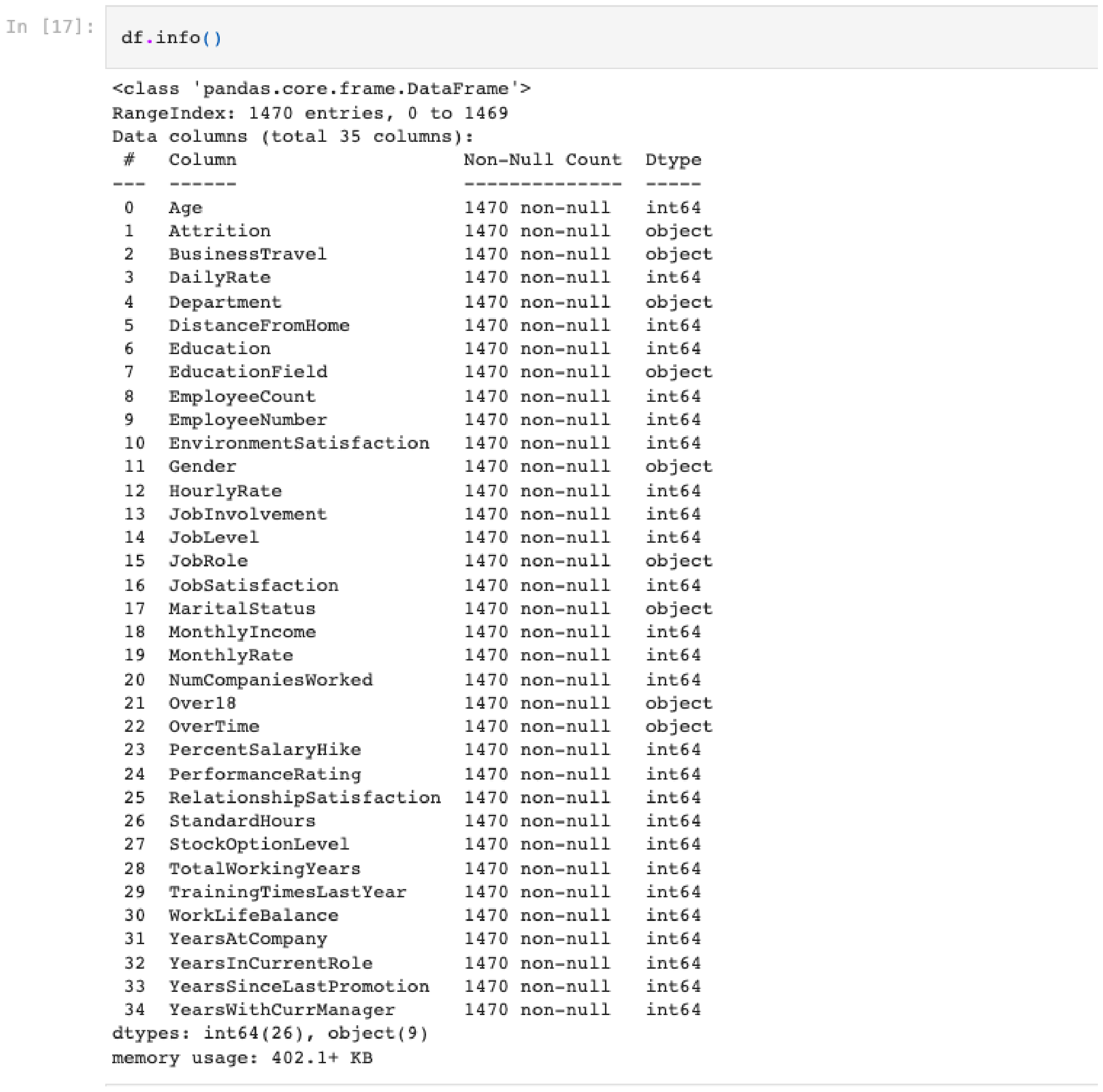


Out[19]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	Er
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.000000	
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.865306	
std	9.135373	403.509100	8.106864	1.024165	0.0	602.024335	
min	18.000000	102.000000	1.000000	1.000000	1.0	1.000000	
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.250000	
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.500000	
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.750000	
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.000000	

Once we have used the code the output provided are in transpose format accommodate all he columns from our dataset in tabular as well as visual format.

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



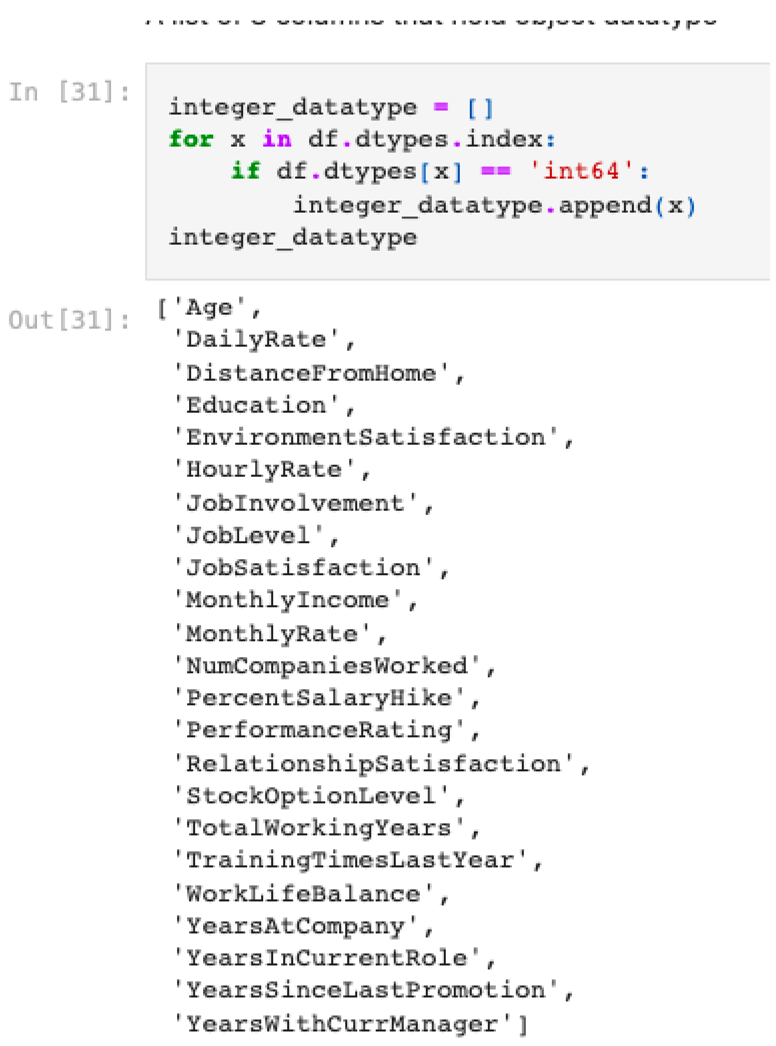
```
In [17]: df.info()
```

```
<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
```

This is the output hat I get explaining the data types of all the columns present in our dataframe.

We also get an opportunity to drop or remove any unwanted columns from the dataframe here.

One of the things I like to do is separate the object datatype and numeric data type values that allows for easier processing in future steps .The code to do that is a simple for loop usage.



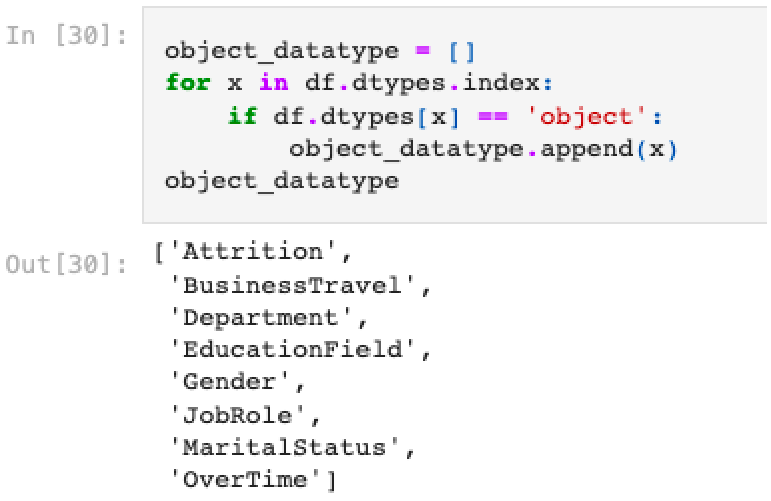
A list of 8 columns that hold object datatype

```
In [31]: integer_datatype = []
         for x in df.dtypes.index:
             if df.dtypes[x] == 'int64':
                 integer_datatype.append(x)
         integer_datatype
```

```
Out[31]: ['Age',
          'DailyRate',
          'DistanceFromHome',
          'Education',
          'EnvironmentSatisfaction',
          'HourlyRate',
          'JobInvolvement',
          'JobLevel',
          'JobSatisfaction',
          'MonthlyIncome',
          'MonthlyRate',
          'NumCompaniesWorked',
          'PercentSalaryHike',
          'PerformanceRating',
          'RelationshipSatisfaction',
          'StockOptionLevel',
          'TotalWorkingYears',
          'TrainingTimesLastYear',
          'WorkLifeBalance',
          'YearsAtCompany',
          'YearsInCurrentRole',
          'YearsSinceLastPromotion',
          'YearsWithCurrManager']
```

This allows us to store the column names in a list format within variables namely object-data type and integer-data type.

After I have bifurcated the data type column names in two separate lists, we will take a look at the overall unique values for all the columns and then the data numbers for only object data-type columns using the below codes.



```
In [30]:  object_datatype = []
          for x in df.dtypes.index:
              if df.dtypes[x] == 'object':
                  object_datatype.append(x)
          object_datatype

Out[30]: ['Attrition',
          'BusinessTravel',
          'Department',
          'EducationField',
          'Gender',
          'JobRole',
          'MaritalStatus',
          'OverTime']
```

These lines of codes provide us the output where we get an entire list of column names with unique data covered in the dataset rows providing a numerical data and then a description of those values for categorical object data-type columns.

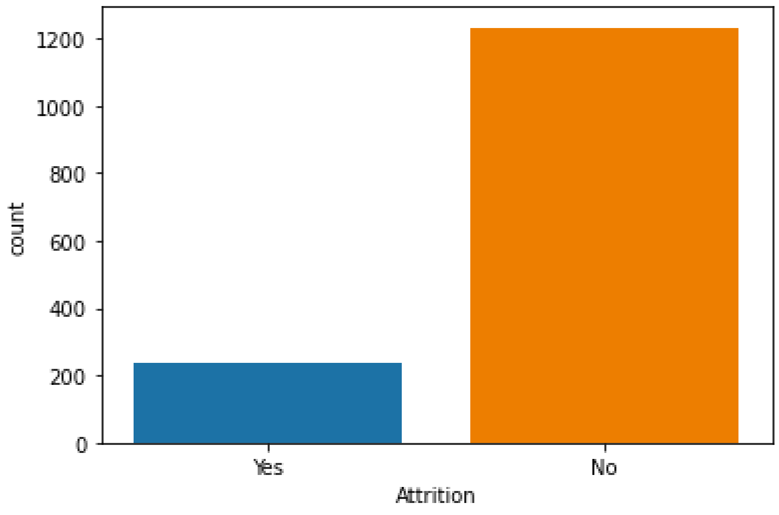
Considering the separation of object data I then take a visual on how many rows or count of rows these values cover in our data set. Usage of various visualization techniques allows me to optimize and analyse the columns further. It gives me an idea as to where data pre-processing will be needed and where removal of those data will benefit. Honestly all this can only be acquired from practising on different projects and as everyone says the more you work the more you acquire knowledge in that field working like a 6th sense in such project creation. I am giving this example here but here it does not mean that these are the only steps when it comes to creating a project. The architecture or the backbone of the project will remain the same however depending on what data you are working on the usage of techniques with all differ.

For example in this project I did not get any missing value therefore, I did not worry about handling them but there are data sets which have a lot of missing data which are then filled using various methods and are at times even discarded as a last resort if it is only going to make our machine learning model biased towards one data value or category .

Let me go ahead and list down all the visualization codes and their output for your reference.

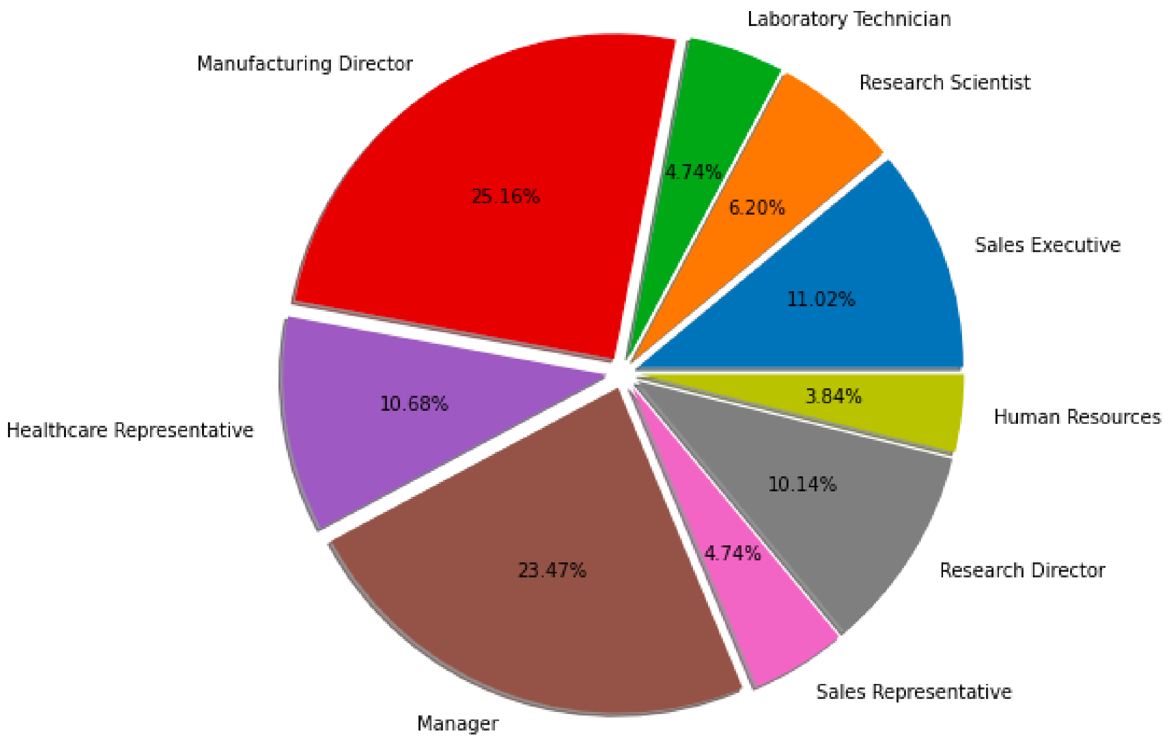
```
In [74]: sns.countplot(df.Attrition)
```

```
Out[74]: <AxesSubplot:xlabel='Attrition', ylabel='count'>
```

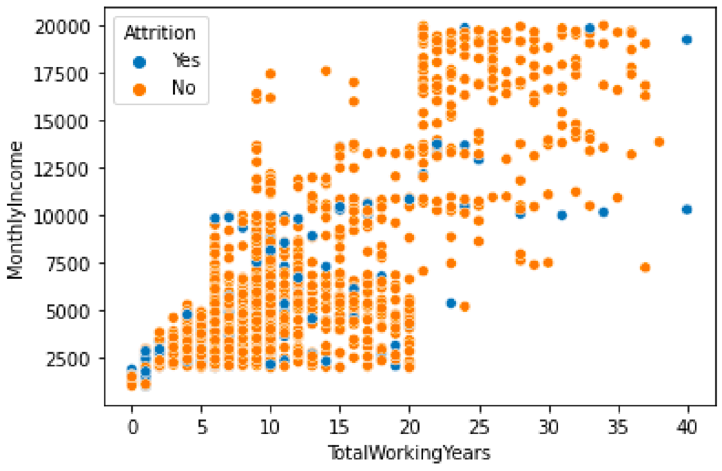


```
In [85]: plt.pie(x=df.groupby('JobRole').mean()['MonthlyIncome'],labels=df.JobRole.value_counts().index,radius=2,shadow=True,explode=np.repeat(0.1,
```

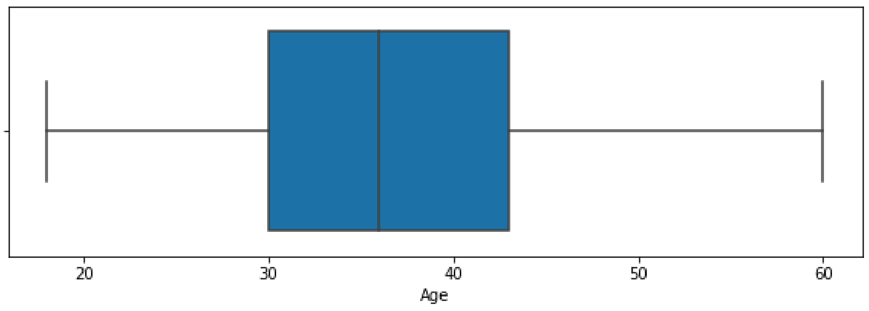
```
Out[85]: ([<matplotlib.patches.Wedge at 0xb7d9806880>,
           <matplotlib.patches.Wedge at 0xb7d98161f0>,
           <matplotlib.patches.Wedge at 0xb7d98169d0>,
```

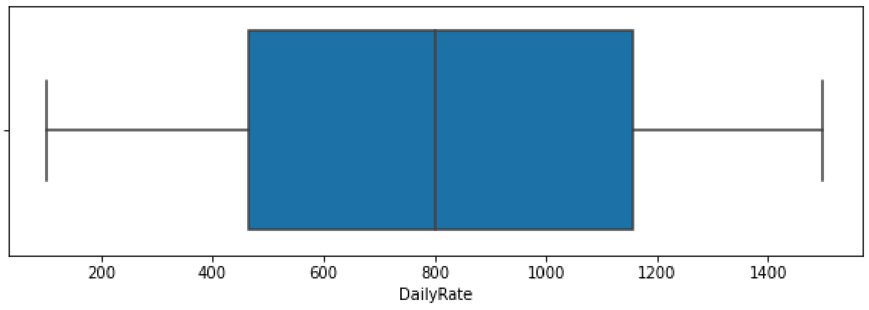


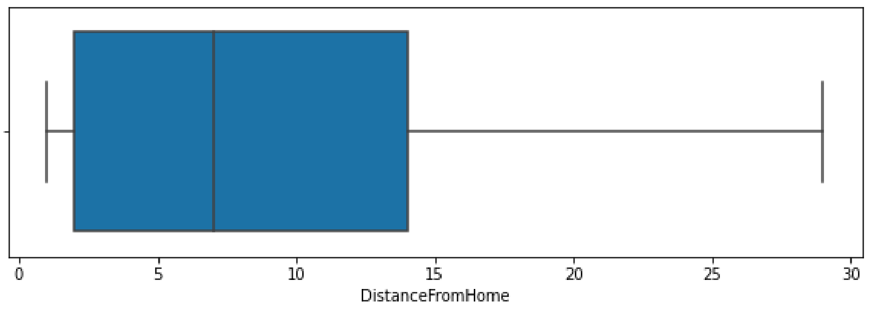
```
In [83]: sns.scatterplot(df.TotalWorkingYears,df.MonthlyIncome,hue=df.Attrition)

Out[83]: <AxesSubplot:xlabel='TotalWorkingYears', ylabel='MonthlyIncome'>
```



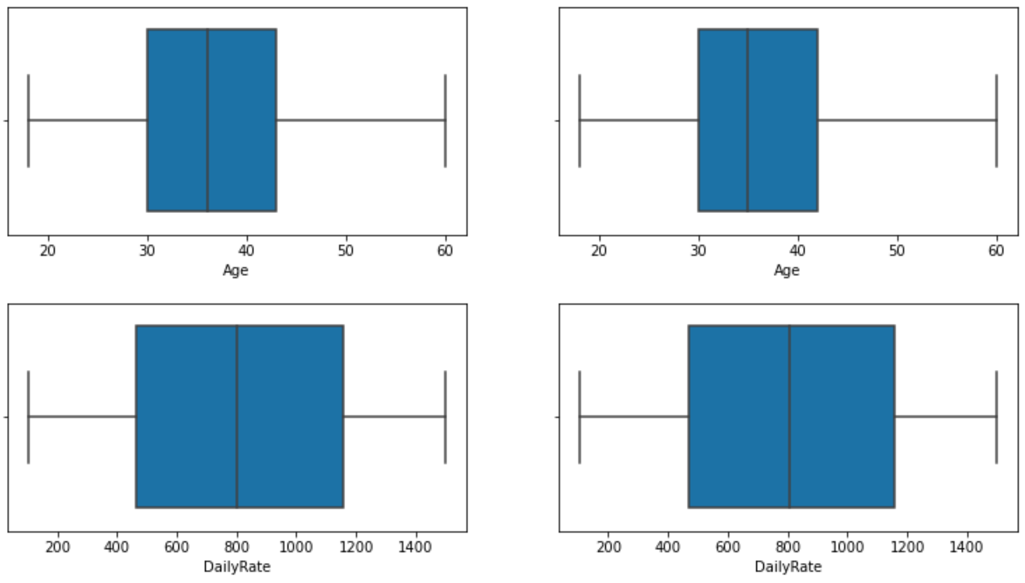
```
In [93]: for col in integer_datatype:
             fig_this_large(10,3)
             sns.boxplot(df_cleaned[col])
             plt.show()
```







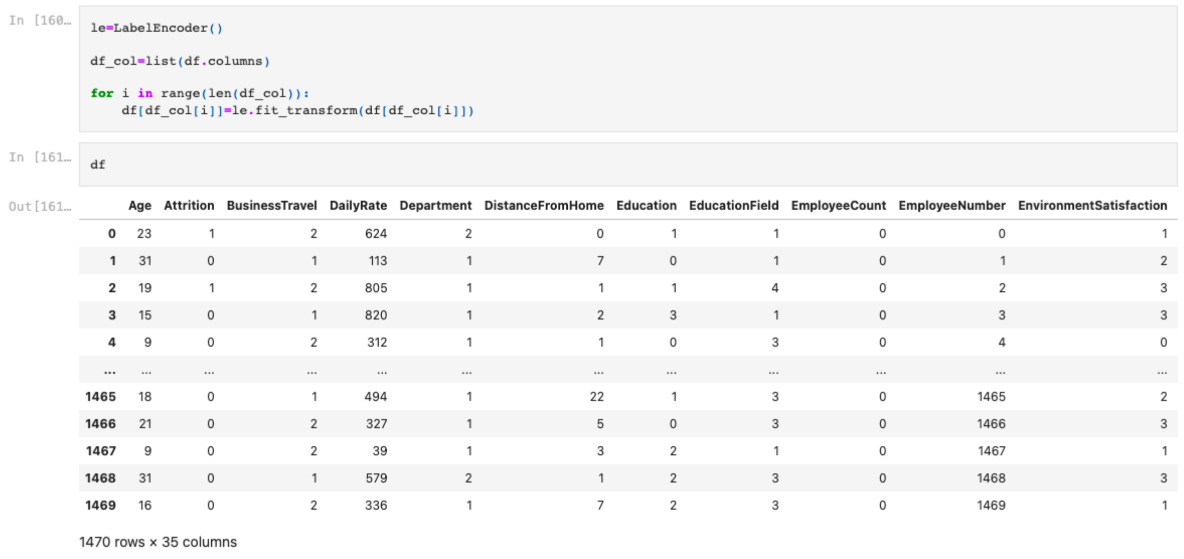
In [98]:
```
for col in integer_datatype:
    fig = fig_this_large(20,3)

    fig.add_subplot(131)
    sns.boxplot(df[col])

    fig.add_subplot(132)
    sns.boxplot(df_cleaned[col])

    plt.show()
```



You can see that with the help of above codes and getting the outputs I was able to take a look at all the column values/counts, the boxen plus 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

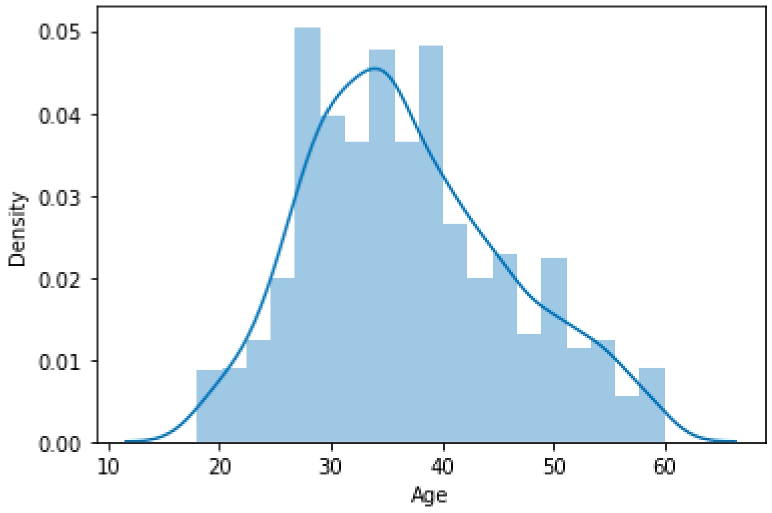
In the pre-processing step I am going to tackle all the misfits and fix them one by one starting with the problem that out data set as object data-type values where as our machine learning models can only understand numeric values. I am making use of the encoding methods to convert all the object data type values for our label I am using label encoder while for other categorical feature columns I am using the ordinal encoder. Instead of the ordinal encoder I could have used the one hot encoder but as I mentioned it is all about preference with trial and errors. The one hot encoder method increases the number columns while application of ordinal encoder on data values offering an order deems a better option to me. I have also seen many people apply label encoder on feature columns as well and it does not make any sense to me since the name itself says label encoder how much of a specification is required to understand that it is only for our label(s) columns. Hope my readers will not make the same mistake!

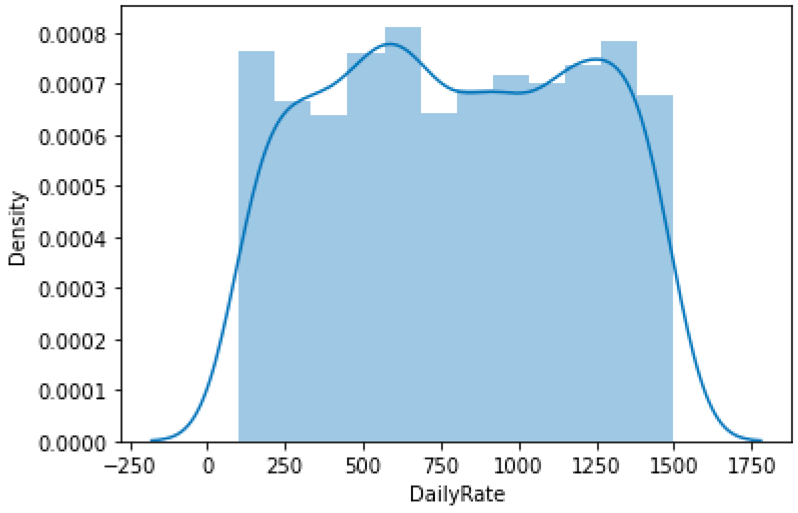


```
In [160…  le=LabelEncoder()

          df_col=list(df.columns)

          for i in range(len(df_col)):
              df[df_col[i]]=le.fit_transform(df[df_col[i]])

In [161…  df

Out[161…
        Age  Attrition  BusinessTravel  DailyRate  Department  DistanceFromHome  Education  EducationField  EmployeeCount  EmployeeNumber  EnvironmentSatisfaction
     0   23          1               2        624           2                 0          1               1              0               0                        1
     1   31          0               1        113           1                 7          0               1              0               1                        2
     2   19          1               2        805           1                 1          1               4              0               2                        3
     3   15          0               1        820           1                 2          3               1              0               3                        3
     4    9          0               2        312           1                 1          0               3              0               4                        0
   ...  ...        ...             ...        ...         ...               ...        ...             ...            ...             ...                      ...
  1465   18          0               1        494           1                22          1               3              0            1465                        2
  1466   21          0               2        327           1                 5          0               3              0            1466                        3
  1467    9          0               2         39           1                 3          2               1              0            1467                        1
  1468   31          0               1        579           2                 1          2               3              0            1468                        3
  1469   16          0               2        336           1                 7          2               3              0            1469                        1

1470 rows × 35 columns
```

After I have encoded all the columns in our data set, I am using a histogram to view the data distribution. Since histogram only consider numeric data it should be able to identify all the information from our encoded data frame.

In [99]:
```
for col in integer_datatype:
    sns.distplot(df[col])
    plt.show()
```

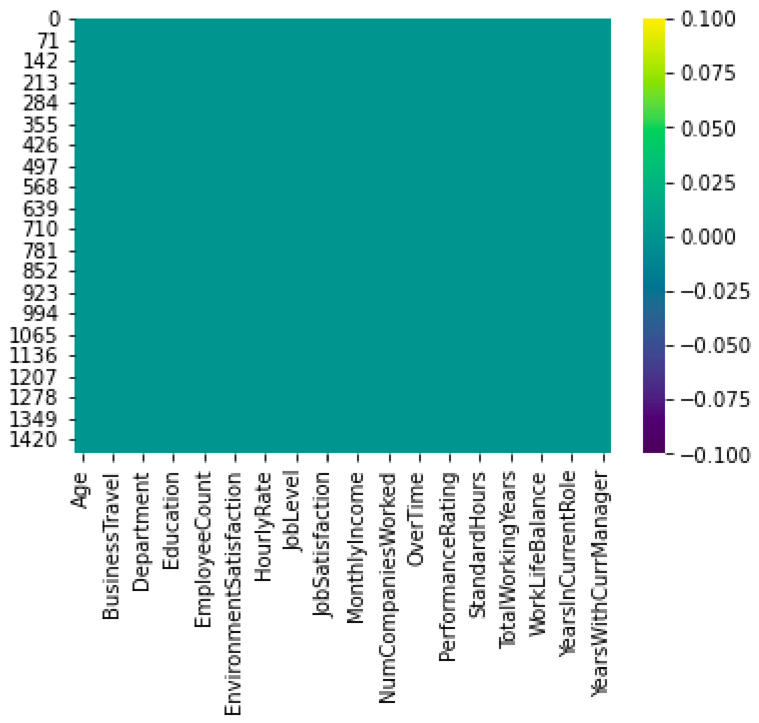




I now fill the need to check for correlation details in our data set through a heat map.For those who still feel confusion on correlation details let me break it down in two simple points that there are positive correlation – A correlation of + 1 indicates a perfect positive correlation, meaning that both variables move in the same direction together a negative correlation – A correlation of -1 in ates a perfect negative correlation, Meaning that as one variable goes up, the other goes down. The code to see this information is displayed below.

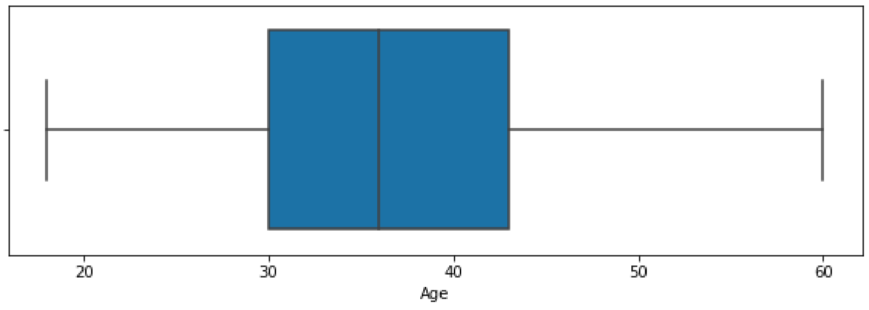
```
In [21]: sns.heatmap(df.isnull(),cmap='viridis')
```

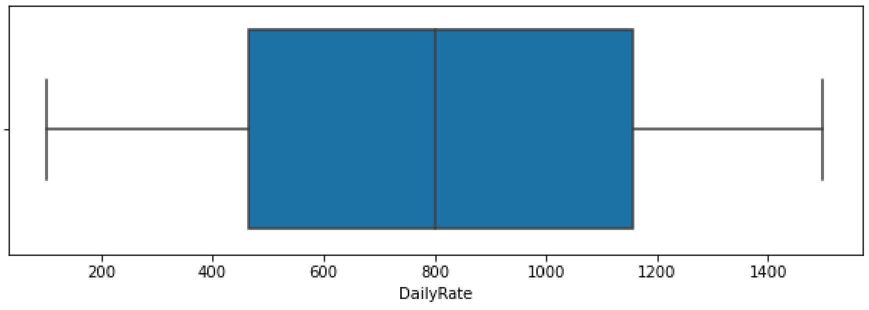
```
Out[21]: <AxesSubplot:>
```

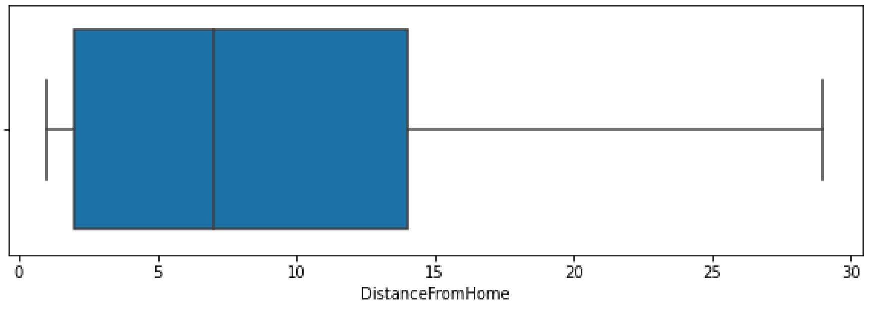


Trust me even on the jupyter Notebook it looks all tiny due to the high number of columns. So, in such scenarios I majorly look at the colours to understand if there is any multicollinearity issue among the feature columns and if there is still any column that I can drop. But to clearly view the correlation between our label and feature columns I use a bar plot comparison and you can find it's code here.

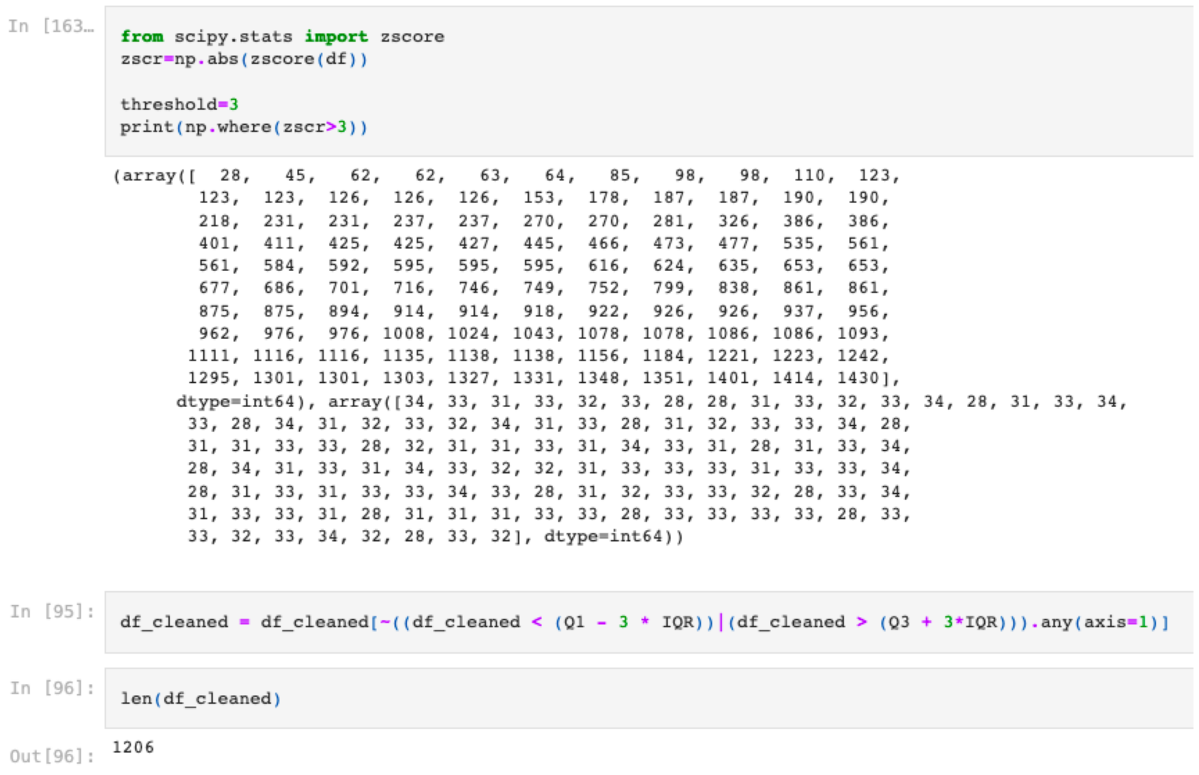
```
In [93]:  for col in integer_datatype:
              fig_this_large(10,3)
              sns.boxplot(df_cleaned[col])
              plt.show()
```







In the above Box 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 .

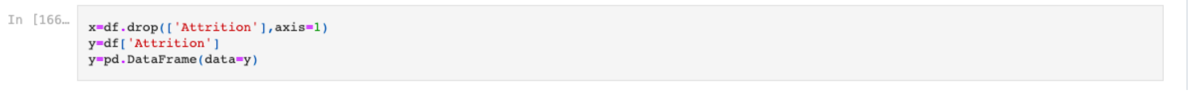


```
In [163…  from scipy.stats import zscore
          zscr=np.abs(zscore(df))

          threshold=3
          print(np.where(zscr>3))

          (array([  28,   45,   62,   62,   63,   64,   85,   98,   98,  110,  123,
                   123,  123,  126,  126,  126,  153,  178,  187,  187,  190,  190,
                   218,  231,  231,  237,  237,  270,  270,  281,  326,  386,  386,
                   401,  411,  425,  425,  427,  445,  466,  473,  477,  535,  561,
                   561,  584,  592,  595,  595,  595,  616,  624,  635,  653,  653,
                   677,  686,  701,  716,  746,  749,  752,  799,  838,  861,  861,
                   875,  875,  894,  914,  914,  918,  922,  926,  926,  937,  956,
                   962,  976,  976, 1008, 1024, 1043, 1078, 1078, 1086, 1086, 1093,
                  1111, 1116, 1116, 1135, 1138, 1138, 1156, 1184, 1221, 1223, 1242,
                  1295, 1301, 1301, 1303, 1327, 1331, 1348, 1351, 1401, 1414, 1430],
                 dtype=int64), array([34, 33, 31, 33, 32, 33, 28, 28, 31, 33, 32, 33, 34, 28, 31, 33, 34,
                  33, 28, 34, 31, 32, 33, 32, 34, 31, 33, 28, 31, 32, 33, 33, 34, 28,
                  31, 31, 33, 33, 28, 32, 31, 31, 33, 31, 34, 33, 31, 28, 31, 33, 34,
                  28, 34, 31, 33, 31, 34, 33, 32, 32, 31, 33, 33, 33, 31, 33, 33, 34,
                  28, 31, 33, 31, 33, 33, 34, 33, 28, 31, 32, 33, 33, 32, 28, 33, 34,
                  31, 33, 33, 31, 28, 31, 31, 31, 33, 33, 28, 33, 33, 33, 33, 28, 33,
                  33, 32, 33, 34, 32, 28, 33, 32], dtype=int64))

In [95]:  df_cleaned = df_cleaned[~((df_cleaned < (Q1 - 3 * IQR))|(df_cleaned > (Q3 + 3*IQR))).any(axis=1)]

In [96]:  len(df_cleaned)

Out[96]:  1206
```

As for the usage of Z score, I was able to lose only about 5% of data but when I used the IQR method it took away I believe 30% of the data. And as Data scientist retaining valuable data always takes priority and fixing it rather than simply deleting it unless it is the last resort. After this I am using the Log transformation to deal with the skewness since the acceptable range lies between +/-0.5 value for each column.

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 table column in the Y variable.

Code:



```
In [166…  x=df.drop(['Attrition'],axis=1)
          y=df['Attrition']
          y=pd.DataFrame(data=y)
```

But there was an imbalance between the label classes. If you would notice the value displayed in the count plot earlier, there was a huge difference between the “Yes” and “No” data. Therefore, I will have to resolve it as the imbalance can make our machine learning model biased towards the “No” value.

In [182…

```
oversample = SMOTE()
X, Y = oversample.fit_resample(X, Y)
```

Then I will also 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 as a higher importance when in real that won't be true due to difference in unit range.

In [184…

```
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X.head()
```

Out[184…

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	Genc
0	0.669884	0.686302	0.840190	1.328099	-1.687394	-0.966686	-1.050242	0.0	-2.298939	-0.610592	-1.3261
1	1.398973	-1.446242	-1.356560	-0.538313	0.182812	-1.908062	-1.050242	0.0	-2.284555	0.322349	0.8101
2	0.272323	0.686302	1.464534	-0.538313	-1.124781	-0.966686	1.256170	0.0	-2.272217	1.294972	0.8101
3	-0.155485	-1.446242	1.514391	-0.538313	-0.768821	1.168918	-1.050242	0.0	-2.261045	1.294972	-1.3261
4	-0.880334	0.686302	-0.379077	-0.538313	-1.124781	-1.908062	0.681445	0.0	-2.250666	-1.485670	0.8101

I would like to share a simple piece of code that allows us to choose a fitting random state for the machine learning models.

In [185…

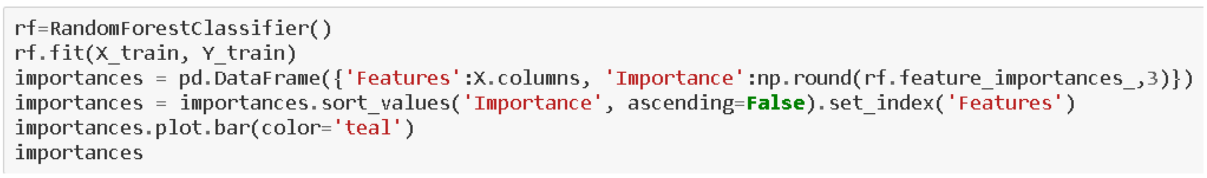
```
maxAccu=0
maxRS=0

for i in range(1,1000):
    X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25,random_state=i)
    lr=LogisticRegression()
    lr.fit(X_train,Y_train)
    pred=lr.predict(X_test)
    acc_score = (accuracy_score(Y_test,pred))*100

    if acc_score>maxAccu:
        maxAccu=acc_score
        maxRS=i

print("Best accuracy score is",maxAccu,"on Random State",maxRS)
```

Then I will use the train test split to bifurcate out entire data set into training data and testing data. Here I am using 75% data for training purpose and 25% data for testing purpose. Some people provide training and test data separately as well and hence it completely depends on you how you want to use this step.

In [186…

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25,random_state=659)
```

Now at this critical step before building my machine learning model I take a look at the importance of my feature columns. This gives me an insight on how the feature columns are involved and what kind of weightage they have in predicting my target label.



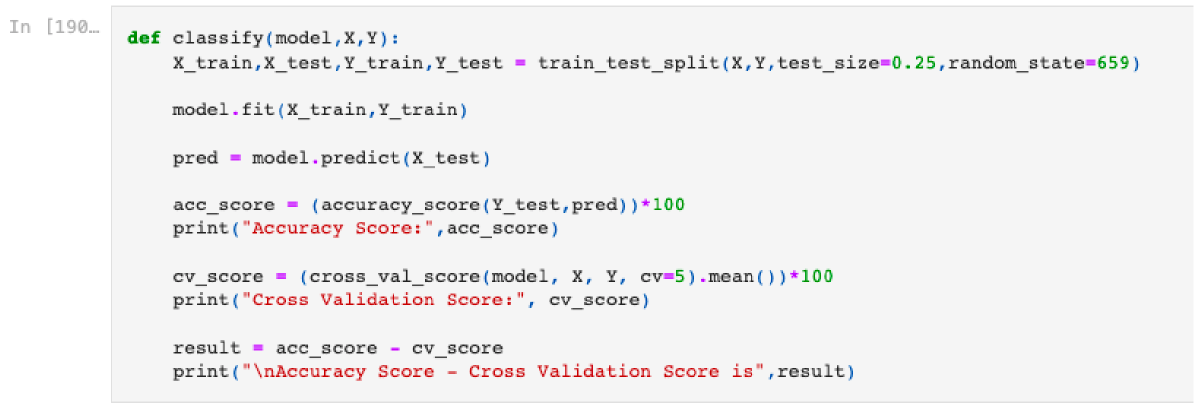
```
rf=RandomForestClassifier()
rf.fit(X_train, Y_train)
importances = pd.DataFrame({'Features':X.columns, 'Importance':np.round(rf.feature_importances_,3)})
importances = importances.sort_values('Importance', ascending=False).set_index('Features')
importances.plot.bar(color='teal')
importances
```

Once we have invested enough time in doing EDA and pre-processing our data comes the steps for which all the previous hard work was performed. That is to finally start building our machine learning model for classification purpose.

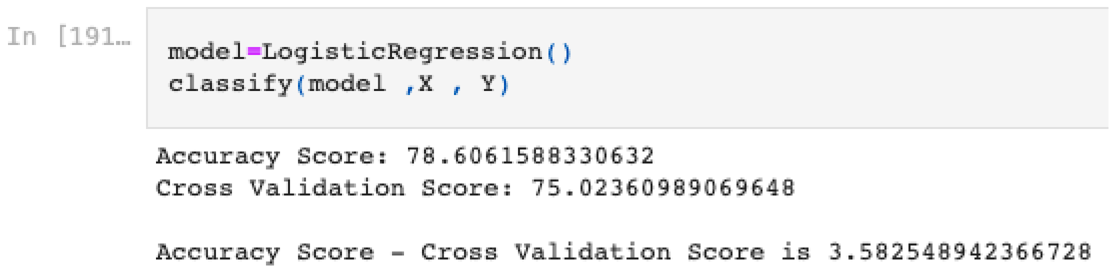
5. BUILDING MACHINE LEARNING MODELS

In order to build a classification method I have imported the necessary libraries and created function that contents 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.

Code:



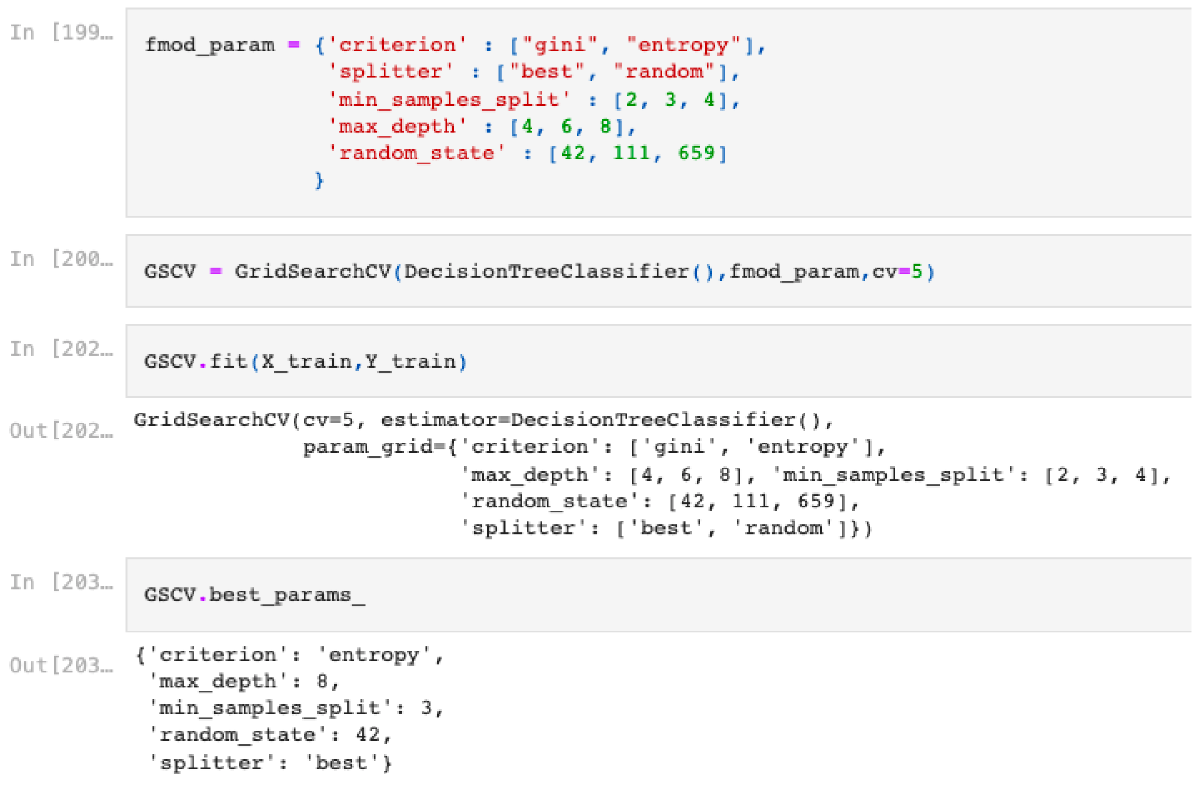
```
In [190…  def classify(model,X,Y):
              X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25,random_state=659)

              model.fit(X_train,Y_train)

              pred = model.predict(X_test)

              acc_score = (accuracy_score(Y_test,pred))*100
              print("Accuracy Score:",acc_score)

              cv_score = (cross_val_score(model, X, Y, cv=5).mean())*100
              print("Cross Validation Score:", cv_score)

              result = acc_score - cv_score
              print("\nAccuracy Score - Cross Validation Score is",result)
```



```
In [191…  model=LogisticRegression()
          classify(model ,X , Y)

Accuracy Score: 78.6061588330632
Cross Validation Score: 75.02360989069648

Accuracy Score - Cross Validation Score is 3.582548942366728
```

It is always advisable to build more than 5 machine learning models so that you can choose from the best performing model and then apply hyper parameter tuning to make it perform even better. I am going to use the extra trees classifier as my choice of classification model as I see it is doing better than the other models I used.

Code:

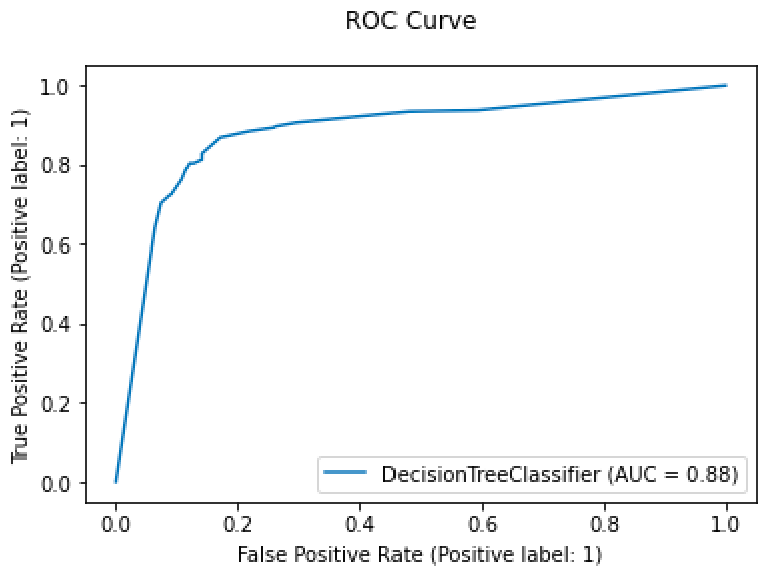


```
In [199…  fmod_param = {'criterion' : ["gini", "entropy"],
                        'splitter' : ["best", "random"],
                        'min_samples_split' : [2, 3, 4],
                        'max_depth' : [4, 6, 8],
                        'random_state' : [42, 111, 659]
                       }

In [200…  GSCV = GridSearchCV(DecisionTreeClassifier(),fmod_param,cv=5)

In [202…  GSCV.fit(X_train,Y_train)

Out[202… GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'max_depth': [4, 6, 8], 'min_samples_split': [2, 3, 4],
                                  'random_state': [42, 111, 659],
                                  'splitter': ['best', 'random']})

In [203…  GSCV.best_params_

Out[203… {'criterion': 'entropy',
          'max_depth': 8,
          'min_samples_split': 3,
          'random_state': 42,
          'splitter': 'best'}
```

After applying the above steps to get the best parameters list , I simply have to plug it into my final model and receive the output of it. I have created an ROC curve plot and confusion matrix for the final model.

Code:

```
In [208…
disp = metrics.plot_roc_curve(Final_Model,X_test,Y_test)
disp.figure_.suptitle("ROC Curve")
plt.show()
```



AUC score for m model is 92%

This also proves that your best performing model can keep changing at times even without changing your code and simply running it multiple times. But I am sure you can understand that the print statement can be changed as per your liking and the important stuff was on the utilization of the code showing the correct result.

Once you have gone through all the previous steps and you are satisfied with outcome you can then save the final model using either joblib or pickle. I have used the joblib method to save and load my model from the same saved filename.

Code:

```
In [207…
Final_Model = DecisionTreeClassifier(criterion="entropy",max_depth=8,min_samples_split=4,
                                     random_state=42,splitter="best")
Classifier = Final_Model.fit(X_train,Y_train)
fmod_pred = Final_Model.predict(X_test)
fmod_acc = (accuracy_score(Y_test,fmod_pred))*100
print("Accuracy score for the best Model is:",fmod_acc)
```

```
Accuracy score for the best Model is: 84.92706645056725
```

6. CONCLUDING REMARKS:

Kindly allow me to provide aquick recap on all the stepsthat 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.

What I do is code my entire project on my own and then take a peek at the internet to look through other’s coding style for inspiration and understand if I can incorporate anything to improvise further on accuracy or beautify the visuals. However, I have seen many people doing the complete opposite whereupon they don’t practise or create their own unique coding style first and rather copy paste lines from the web and perform some sort of messy patch work and when asked to explain might not be capable of conveying functioning or usage of those code blocks.

Before wrapping up my only advise to everyone is “No pain No gain” you will have to get your hands dirty with building your own code and trying out all the permutations and combinations. Create a self-made unique data story telling commandment list and follow it along with the standard project life cycle. Hope this at length article helps you in gaining the initial knowledge on building your first project from scratch.

DISCLAIMER: I am a newbie myself in this field with some accumulated knowledge over a year’s time studying Data science and felt like since sharing is caring someone who’s stepping in now can benefit from my experience. I am also open to get some feedback from anyone that will help me in improving too! The content that I have written is solely my view of the project but it’s definitely inspired by others over the internet who have worked on similar projects before me.