MA336_2310628

Project Name : OLA - Driver Sustain Ensemble

Introduction (Problem Statement) :

OLA is leading transportation industry. Reducing drivers is seen by industry observers as a tough battle for Ola. Churn among drivers is high and it's very easy for drivers to stop working for the service on the fly or jump to other transportation services depending on the rates.

As the companies get bigger, the high churn could become a bigger problem. To find new drivers, Ola is casting a wide net, including people who don't have cars for jobs. But this acquisition is really costly. Losing drivers frequently impacts the morale of the organization and acquiring new drivers is more expensive than retaining existing ones.

You are working as a Data Scientist with the Analytics Department of Ola, focused on driver team attrition. You are provided with the monthly information for a segment of drivers for 2019 and 2020 and tasked to predict whether a driver will be leaving the company or not based on their attributes like

- Demographics (city, age, gender etc.)
- Tenure information (joining date, Last Date)
- Historical data regarding the performance of the driver (Quarterly rating, Monthly business acquired, grade, Income)

In [1]:

```
# Importing all necessary Libraries to run data efficiently

import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
from sklearn.impute import KNNImputer
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier


from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

```
from sklearn.metrics import confusion_matrix, classification_report, f1_score, roc_c
```

Dataset:

In [2]:

```
# Importing Dataset

df = pd.read_csv("ola_driver.csv")
```

In [3]:

```
df.head()
```

Out[3]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastW
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	

In [4]:

```
# Removing Unamed column because there is no use of it

df.drop(columns = ['Unnamed: 0'], axis = 1, inplace = True)
```

In [5]:

```
# Data Information

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   MMM-YY                19104 non-null  object 
 1   Driver_ID             19104 non-null  int64  
 2   Age                   19043 non-null  float64
 3   Gender                19052 non-null  float64
 4   City                  19104 non-null  object 
 5   Education_Level       19104 non-null  int64  
 6   Income                19104 non-null  int64  
 7   Dateofjoining         19104 non-null  object 
 8   LastWorkingDate       1616 non-null   object 
 9   Joining Designation   19104 non-null  int64  
 10  Grade                 19104 non-null  int64  
 11  Total Business Value  19104 non-null  int64  
 12  Quarterly Rating      19104 non-null  int64  
dtypes: float64(2), int64(7), object(4)
memory usage: 1.9+ MB
```

In [6]:

```
# Generating descriptive statistics

df.describe(include = 'all')
```

Out[6]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income
count	19104	19104.000000	19043.000000	19052.000000	19104	19104.000000	19104.000000
unique	24	NaN	NaN	NaN	29	NaN	NaN
top	01/01/19	NaN	NaN	NaN	C20	NaN	NaN
freq	1022	NaN	NaN	NaN	1008	NaN	NaN
mean	NaN	1415.591133	34.668435	0.418749	NaN	1.021671	65652.025126
std	NaN	810.705321	6.257912	0.493367	NaN	0.800167	30914.515344
min	NaN	1.000000	21.000000	0.000000	NaN	0.000000	10747.000000
25%	NaN	710.000000	30.000000	0.000000	NaN	0.000000	42383.000000
50%	NaN	1417.000000	34.000000	0.000000	NaN	1.000000	60087.000000
75%	NaN	2137.000000	39.000000	1.000000	NaN	2.000000	83969.000000
max	NaN	2788.000000	58.000000	1.000000	NaN	2.000000	188418.000000

PRELIMINARY ANALYSIS:

Filling Null values :

In [7]:

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

Out[7]:

```
MMM-YY                   0.000000
Driver_ID                0.000000
Age                      0.319305
Gender                   0.272194
City                     0.000000
Education_Level          0.000000
Income                   0.000000
Dateofjoining            0.000000
LastWorkingDate         91.541039
Joining Designation      0.000000
Grade                    0.000000
Total Business Value     0.000000
Quarterly Rating         0.000000
dtype: float64
```

- In this we have 91.54% null values in last working days
- So, the values are not present in the columns which means they are not leaving the company so, we can fill it with the 0.
- In age column also we have missing values that are filled with the preceding values, same for gender also using ffill.

In [8]:

```
df.head()
```

Out[8]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate
0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN
1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN
2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	03/11/19
3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN
4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN

Column Pre- Processing:

- In LastWorkingDate column we have a date of leaving that particular date is not needed as we need only the value that is 1 or 0.
- So, if we have that date in a row we fill that with 1 so that the driver is leaving that quarter.
- Here we can split the reporting MMM-YY to re_Day, re_Month, re_Year.
- Split the column date of joining to jo_Day, jo_Month, jo_Year.

In [9]:

```
df['LastWorkingDate'].fillna(value=0, inplace=True)
df['LastWorkingDate'] = df['LastWorkingDate'].apply(lambda x: 0 if x == 0 else 1)

df['re_Month'] = df['MMM-YY'].apply(lambda x: int(str(x).split('/')[0]) )
df['re_Day'] = df['MMM-YY'].apply(lambda x: int(str(x).split('/')[1]) )
df['re_Year'] = df['MMM-YY'].apply(lambda x: int(str(x).split('/')[2]) )

df['city'] = df['City'].apply(lambda x: str(x)[1:] )

df['jo_Month'] = df['Dateofjoining'].apply(lambda x: int(str(x).split('/')[0]) )
df['jo_Day'] = df['Dateofjoining'].apply(lambda x: int(str(x).split('/')[1]) )
df['jo_Year'] = df['Dateofjoining'].apply(lambda x: int(str(x).split('/')[2]) )

df1 = df[['re_Day', 're_Month',
       're_Year','Driver_ID', 'Age', 'Gender', 'city', 'Education_Level',
       'Income', 'jo_Day', 'jo_Month', 'jo_Year', 'LastWorkingDate', 'Joining Design
       'Grade', 'Total Business Value', 'Quarterly Rating']]
```

Knn Imputer:

In [10]:

```
imputer = KNNImputer(n_neighbors=2)
transformed = imputer.fit_transform(df1)
df2 = pd.DataFrame(transformed)

df2.rename(columns= {0:'re_Day', 1:'re_Month',2:'re_Year',3:'Driver_ID',4:'Age',5:'G
                     9:'jo_Day',10:'jo_Month',11:'jo_Year',12:'LastWorkingDate',13:'
                     15:'TotalBusinessValue',16:'QuarterlyRating'}, inplace=True)
df2
```

Out[10]:

	re_Day	re_Month	re_Year	Driver_ID	Age	Gender	City	Education_Level	Income	jo_Day	j
0	1.0	1.0	19.0	1.0	28.0	0.0	23.0	2.0	57387.0	12.0	
1	1.0	2.0	19.0	1.0	28.0	0.0	23.0	2.0	57387.0	12.0	
2	1.0	3.0	19.0	1.0	28.0	0.0	23.0	2.0	57387.0	12.0	
3	1.0	11.0	20.0	2.0	31.0	0.0	7.0	2.0	67016.0	6.0	
4	1.0	12.0	20.0	2.0	31.0	0.0	7.0	2.0	67016.0	6.0	
...	...	...	...	...	...	...	...	...	...	...	
19099	1.0	8.0	20.0	2788.0	30.0	0.0	27.0	2.0	70254.0	8.0	
19100	1.0	9.0	20.0	2788.0	30.0	0.0	27.0	2.0	70254.0	8.0	
19101	1.0	10.0	20.0	2788.0	30.0	0.0	27.0	2.0	70254.0	8.0	
19102	1.0	11.0	20.0	2788.0	30.0	0.0	27.0	2.0	70254.0	8.0	
19103	1.0	12.0	20.0	2788.0	30.0	0.0	27.0	2.0	70254.0	8.0	

19104 rows × 17 columns

In [11]:

```
100*df2.isnull().sum() / len(df2)
```

Out[11]:

```
re_Day               0.0
re_Month             0.0
re_Year              0.0
Driver_ID            0.0
Age                  0.0
Gender               0.0
City                 0.0
Education_Level      0.0
Income               0.0
jo_Day               0.0
jo_Month             0.0
jo_Year              0.0
LastWorkingDate      0.0
JoiningDesignation   0.0
Grade                0.0
TotalBusinessValue   0.0
QuarterlyRating      0.0
dtype: float64
```

- Using KNN Imputer no null value is present.
- By checking some columns using Imputer there is problem in Age and DriverID.

Checking KNN Imputer:

In [12]:

```
df[df['Age'].isnull()]
```

Out[12]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorking
72	02/01/20	20	NaN	1.0	C19	0	40342	25/10/19	
97	10/01/19	22	NaN	0.0	C10	2	31224	25/05/18	

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorking
110	07/01/19	24	NaN	0.0	C24	2	76308	25/05/18	
212	11/01/19	40	NaN	0.0	C15	0	59182	11/08/19	
261	05/01/19	49	NaN	0.0	C20	0	53039	25/05/18	
...	...	...	...	...	...	...	...	...	
18395	05/01/20	2690	NaN	0.0	C11	2	77662	17/07/18	
18722	08/01/20	2730	NaN	1.0	C16	2	69924	07/08/19	
18780	03/01/19	2738	NaN	0.0	C17	0	23068	09/08/18	
18843	01/01/19	2751	NaN	0.0	C17	2	53115	11/05/15	
19024	02/01/19	2774	NaN	0.0	C15	1	42313	21/07/18	

61 rows × 20 columns

In [13]:

```
df2[df2['Driver_ID'] == 22]
```

Out[13]:

	re_Day	re_Month	re_Year	Driver_ID	Age	Gender	City	Education_Level	Income	jo_Day	jo_
88	1.0	1.0	19.0	22.0	40.0	0.0	10.0	2.0	31224.0	5.0	
89	1.0	2.0	19.0	22.0	40.0	0.0	10.0	2.0	31224.0	5.0	
90	1.0	3.0	19.0	22.0	40.0	0.0	10.0	2.0	31224.0	5.0	
91	1.0	4.0	19.0	22.0	40.0	0.0	10.0	2.0	31224.0	5.0	
92	1.0	5.0	19.0	22.0	40.0	0.0	10.0	2.0	31224.0	5.0	
93	1.0	6.0	19.0	22.0	40.0	0.0	10.0	2.0	31224.0	5.0	
94	1.0	7.0	19.0	22.0	40.0	0.0	10.0	2.0	31224.0	5.0	
95	1.0	8.0	19.0	22.0	40.0	0.0	10.0	2.0	31224.0	5.0	
96	1.0	9.0	19.0	22.0	40.0	0.0	10.0	2.0	31224.0	5.0	
97	1.0	10.0	19.0	22.0	36.0	0.0	10.0	2.0	31224.0	5.0	
98	1.0	11.0	19.0	22.0	41.0	0.0	10.0	2.0	31224.0	5.0	
99	1.0	12.0	19.0	22.0	41.0	0.0	10.0	2.0	31224.0	5.0	
100	1.0	1.0	20.0	22.0	41.0	0.0	10.0	2.0	31224.0	5.0	
101	1.0	2.0	20.0	22.0	41.0	0.0	10.0	2.0	31224.0	5.0	
102	1.0	3.0	20.0	22.0	41.0	0.0	10.0	2.0	31224.0	5.0	
103	1.0	4.0	20.0	22.0	41.0	0.0	10.0	2.0	31224.0	5.0	

- From this we come to know that KNN inputer is not working good (index 97 columns Age).
- In this case as we have a group of so we can determine the null vales but is not more accurate, so will fill all the null vales using the ffill in fillna.

fillNa:

In [14]:
```
df1['Age'].fillna(method= 'ffill', inplace=True)
df1['Gender'].fillna(method= 'ffill', inplace=True)

df1[df1['Driver_ID'] == 22]
```

Out[14]:

	re_Day	re_Month	re_Year	Driver_ID	Age	Gender	city	Education_Level	Income	jo_Day	jo_
88	1	1	19	22	40.0	0.0	10	2	31224	5	
89	1	2	19	22	40.0	0.0	10	2	31224	5	
90	1	3	19	22	40.0	0.0	10	2	31224	5	
91	1	4	19	22	40.0	0.0	10	2	31224	5	
92	1	5	19	22	40.0	0.0	10	2	31224	5	
93	1	6	19	22	40.0	0.0	10	2	31224	5	
94	1	7	19	22	40.0	0.0	10	2	31224	5	
95	1	8	19	22	40.0	0.0	10	2	31224	5	
96	1	9	19	22	40.0	0.0	10	2	31224	5	
97	1	10	19	22	40.0	0.0	10	2	31224	5	
98	1	11	19	22	41.0	0.0	10	2	31224	5	
99	1	12	19	22	41.0	0.0	10	2	31224	5	
100	1	1	20	22	41.0	0.0	10	2	31224	5	
101	1	2	20	22	41.0	0.0	10	2	31224	5	
102	1	3	20	22	41.0	0.0	10	2	31224	5	
103	1	4	20	22	41.0	0.0	10	2	31224	5	

In [15]:
```
for i in df1.columns:
  print(i, '--->', df1[i].unique())
```

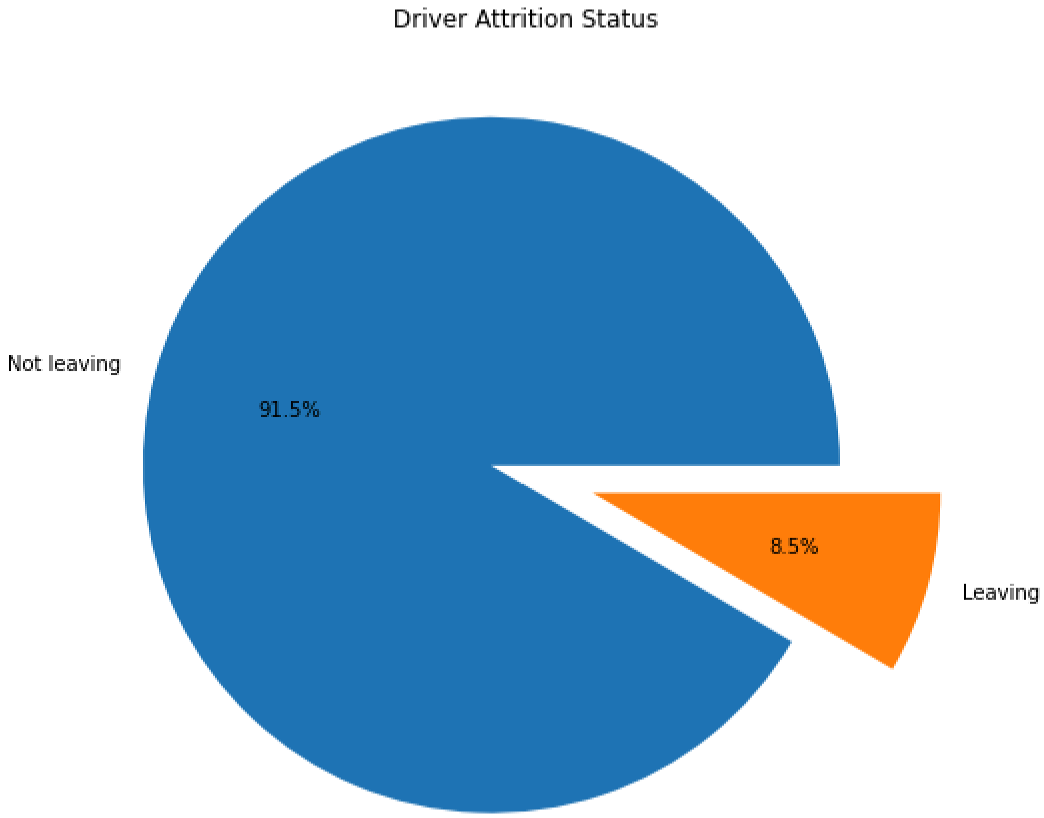
```
re_Day ---> [1]
re_Month ---> [ 1  2  3 11 12  4  8  9 10  7  5  6]
re_Year ---> [19 20]
Driver_ID ---> [   1    2    4 ... 2786 2787 2788]
Age ---> [28. 31. 43. 29. 34. 35. 30. 39. 42. 27. 26. 33. 40. 41. 32. 22. 44. 36.
 21. 49. 37. 38. 46. 47. 48. 25. 24. 45. 51. 52. 23. 50. 53. 54. 55. 58.]
Gender ---> [0. 1.]
city ---> ['23' '7' '13' '9' '11' '2' '19' '26' '20' '17' '29' '10' '24' '14' '6'
 '28' '5' '18' '27' '15' '8' '25' '21' '1' '4' '3' '16' '22' '12']
Education_Level ---> [2 0 1]
Income ---> [57387 67016 65603 ... 35370 69498 70254]
jo_Day ---> [12  6  7  9  5 10 11  3  4  1  8  2]
jo_Month ---> [24 11 12  1 31 19 29 28 16 30  3 25  5 20 10  7  8 13 22 21 26 15 17 1
8
 14  6  4 27 23  9  2]
jo_Year ---> [18 20 19 15 17 16 13 14]
LastWorkingDate ---> [0 1]
```

```
Joining Designation ---> [1 2 3 4 5]
Grade ---> [1 2 3 4 5]
Total Business Value ---> [2381060 -665480       0 ...  497690  740280  448370]
Quarterly Rating ---> [2 1 4 3]
```

Analysis :

In [16]:

```
# Set the size of the figure
plt.figure(figsize=(8, 8))

# Your pie chart code
plt.pie(df1['LastWorkingDate'].value_counts(), labels=['Not leaving', 'Leaving'], ex

plt.title('Driver Attrition Status')


# Display the pie chart
plt.show()
```



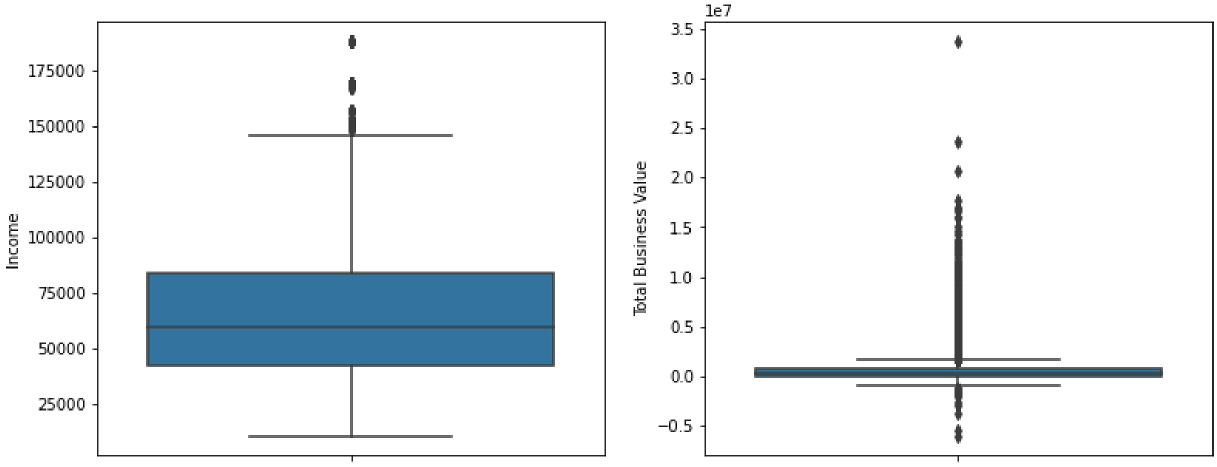
- According to pie chart, we can see 8.5% are leaving the company whereas 91.5% are working.

Univarient analysis : Numerical

In [17]:

```
uni_aly = ['Income', 'Total Business Value']
count = 0

plt.figure(figsize=(20,30))
for i in uni_aly:
    count += 1
```

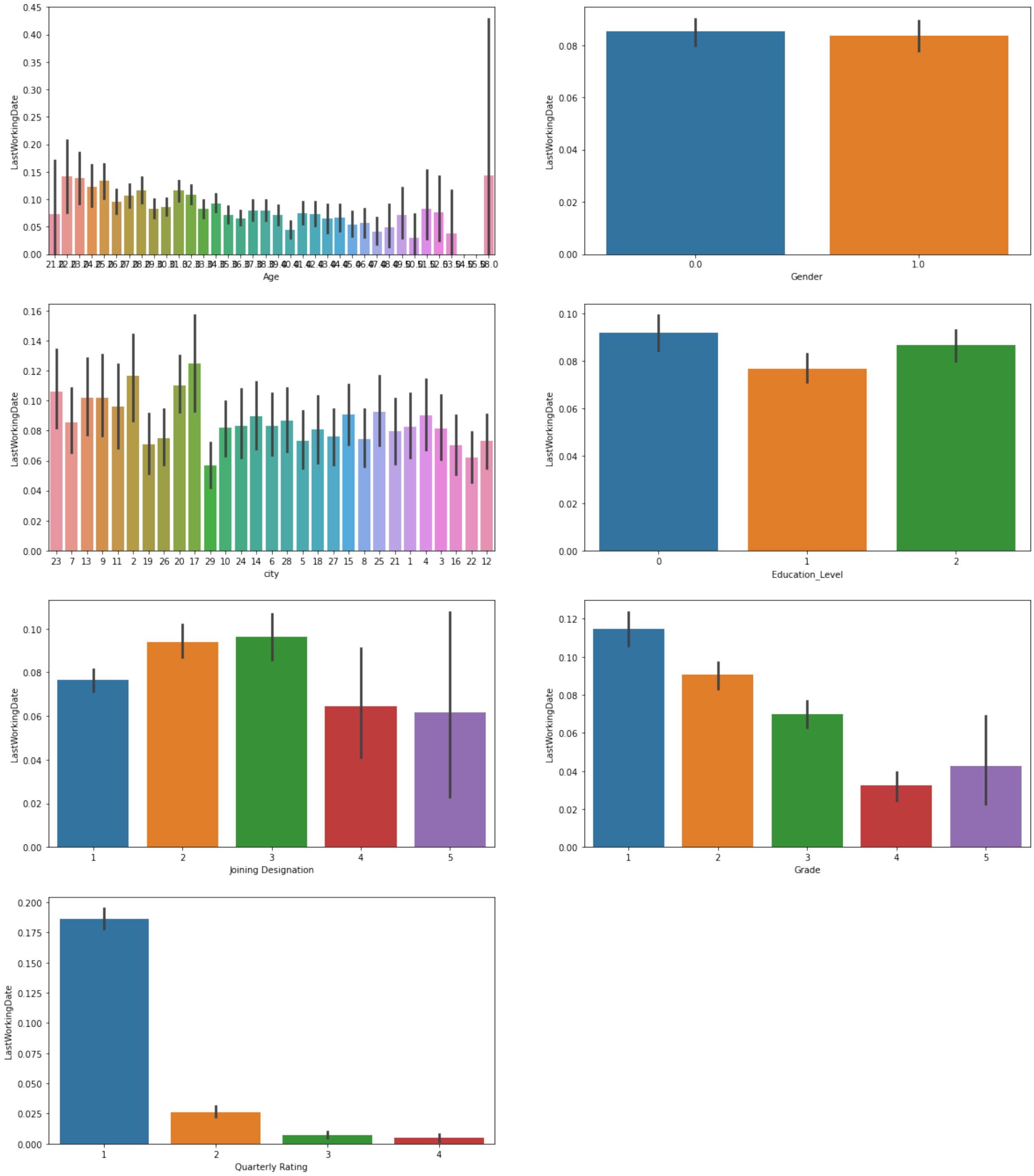
```
    plt.subplot(5,3,count)
    sns.boxplot(y= df1[i])


plt.show()
```



Univariate Analysis : Categorical

In [18]:

```
uni_aly = ['Age','Gender', 'city','Education_Level','Joining Designation', 'Grade',
count = 0
plt.figure(figsize=(20,30))
for i in uni_aly:
    count += 1
    plt.subplot(5,2,count)
    sns.barplot(x= df1[i], y= df1['LastWorkingDate'])
```

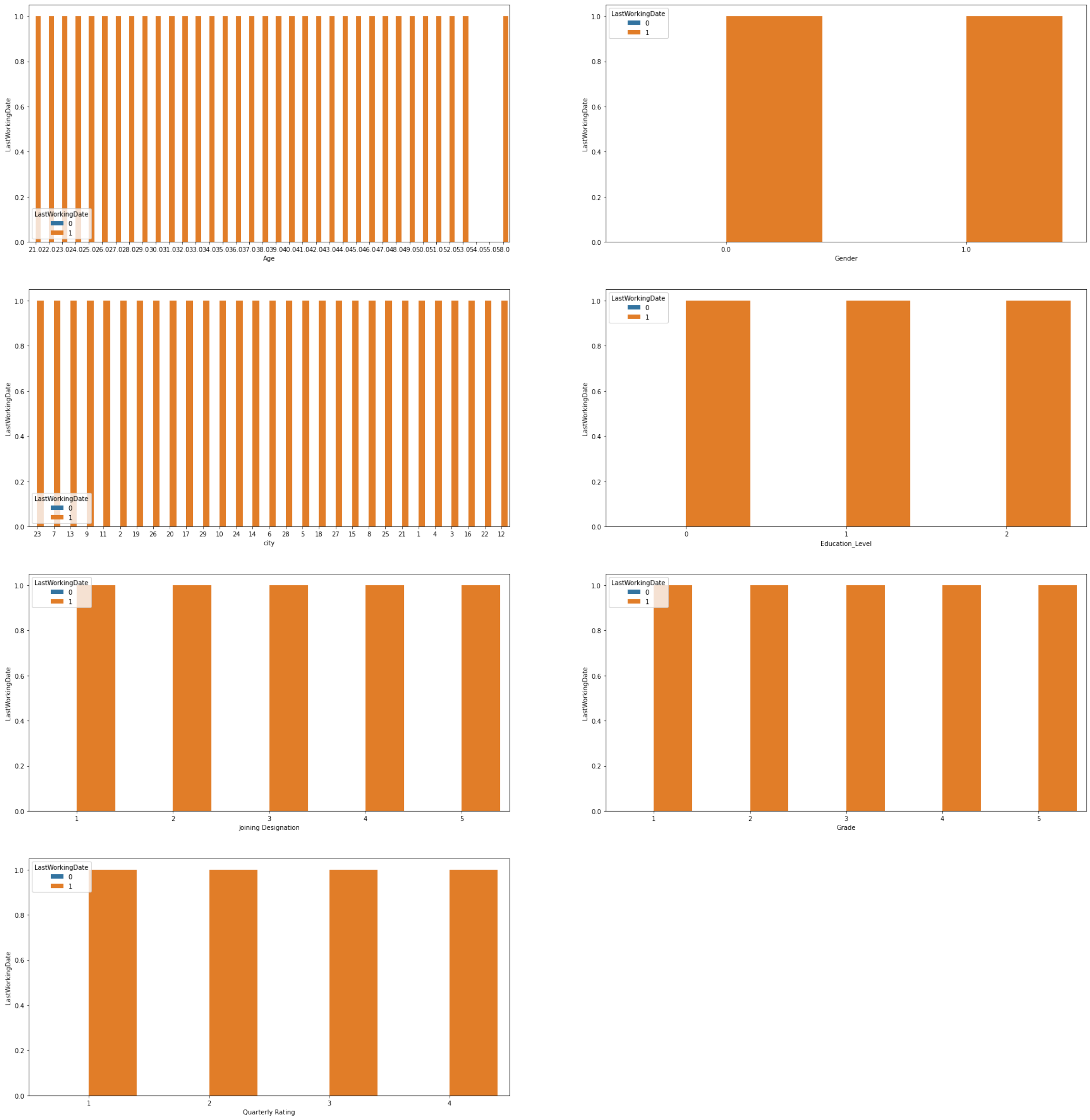


- By univariant analysis. Depending on categorical law. Variables gender box that are equal.
- City are almost equal.
- Education level is also equal.
- Joining designation it is major for two and three.
- When comparing with the grades, it is most dependent on grade one and two. 345 or very less.
- In quarterly re-rating, so one is more dependent on last working date. And others are very less.

Bivariate Analysis:

In [19]:

```
bi_ana = ['Age','Gender', 'city','Education_Level','Joining Designation', 'Grade', '
count = 0

plt.figure(figsize = (30,40))
for i in bi_ana:
    count+=1
    plt.subplot(5,2,count)
    sns.barplot(x= df1[i], y = df1['LastWorkingDate'], hue = df1['LastWorkingDate'])
```

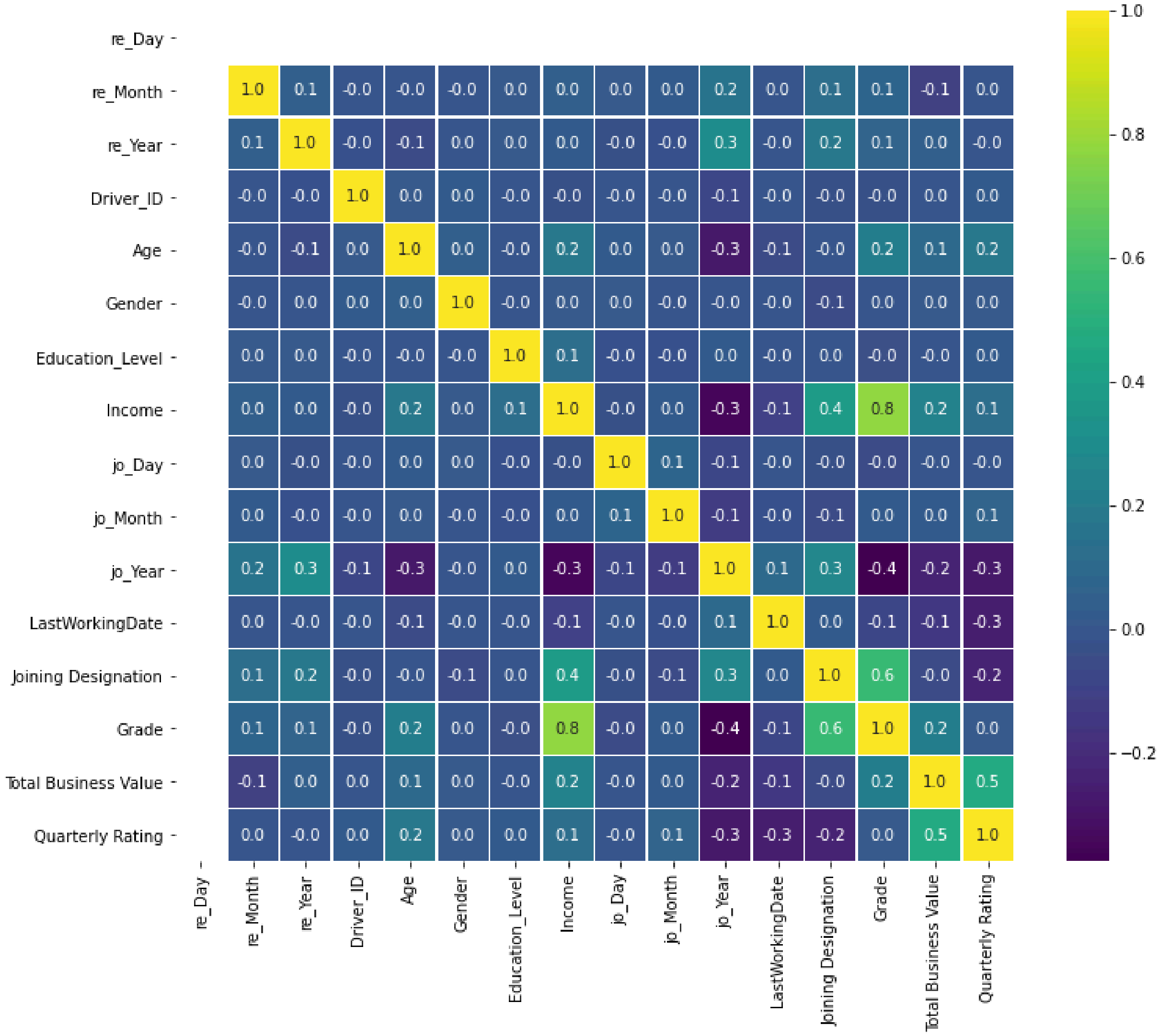


- From this bivarient analysis, we can see that the count of lastWorkingDate 0 is less so, we can't come to a conclusion with this we have to treat with the imbalance data.

Correlation analysis:

In [20]:

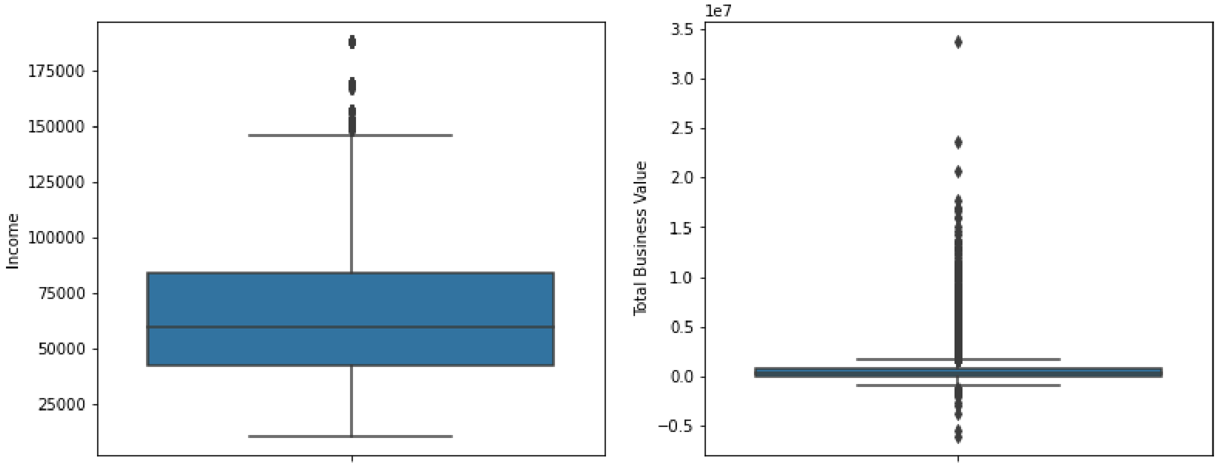
```
plt.figure(figsize=(12, 10))
sns.heatmap(df1.corr(), annot=True, fmt='.1f', cmap="viridis", linewidth=.5)
plt.show()
```



- From this correlation map. We can see that our target variable LastWorkingDate.
- It does not even much more dependent on any other 15 columns.
- So it is very difficult to find a correlation between them.

Outlier Treatment :

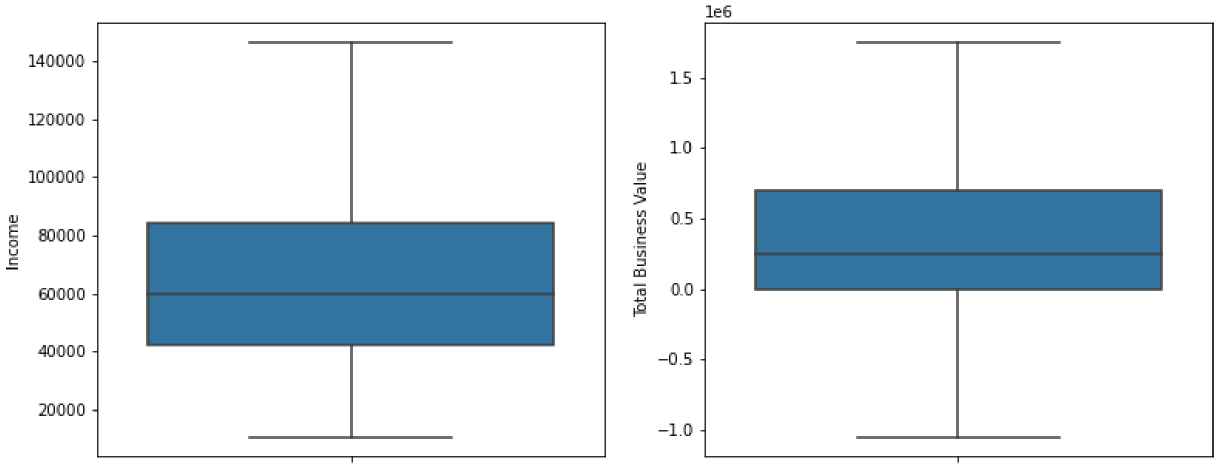
In [21]:
```
outliers = ['Income', 'Total Business Value']
count = 0
plt.figure(figsize=(20,30))
for i in outliers:
    count += 1
    plt.subplot(5,3,count)
    sns.boxplot(y= df[i])
```



In [22]:
```
for col in ['Income', 'Total Business Value']:

    mean = df[col].mean()
    std = df[col].std()
    q1 = np.percentile(df[col], 25)
    q2 = np.percentile(df[col], 50)
    q3 = np.percentile(df[col], 75)
    IQR = q3-q1
    lower_limt, upper_limit = q1-1.5*IQR , q3+1.5*IQR
    df[col] = df[col].apply(lambda x: lower_limt if x < lower_limt else x)
    df[col] = df[col].apply(lambda x: upper_limit if x > upper_limit else x)
df.shape
```

Out[22]: (19104, 20)

In [23]:
```
outliers = ['Income', 'Total Business Value']
count = 0
plt.figure(figsize=(20,30))
for i in outliers:
    count += 1
    plt.subplot(5,3,count)
    sns.boxplot(y= df[i])
```



- In this outliers, we will make the values that are more than Upper whisker and Lower whisker to inside the range.
- If we do that what happens is that all the values from total business value will compress and the mean value 0 will be shifted from 0 to higher value so that will also affect the output.

- Even if we drop the null values then also we won't be left with more number of values.
- So in this case its better to not treat.

EDA(Exploratory Data Analysis)/ FE(Feature Engineering):

In [24]:
```
df.columns
```

Out[24]:
```
Index(['MMM-YY', 'Driver_ID', 'Age', 'Gender', 'City', 'Education_Level',
       'Income', 'Dateofjoining', 'LastWorkingDate', 'Joining Designation',
       'Grade', 'Total Business Value', 'Quarterly Rating', 're_Month',
       're_Day', 're_Year', 'city', 'jo_Month', 'jo_Day', 'jo_Year'],
      dtype='object')
```

In [25]:
```
df2 = df[['MMM-YY', 'Driver_ID', 'Age', 'Gender', 'City', 'Education_Level',
       'Income', 'Dateofjoining', 'LastWorkingDate', 'Joining Designation',
       'Grade', 'Total Business Value', 'Quarterly Rating']]

df2['City'] = df2['City'].apply(lambda x: int(str(x)[1:]) )
df2['MMM-YY'] = pd.to_datetime(df2['MMM-YY'])
df2['Dateofjoining'] = pd.to_datetime(df2['Dateofjoining'])
df2['TotalexpinDays'] = (df2['MMM-YY'] - df2['Dateofjoining']).dt.days
```

GroupBy

- In MMM-YY we can see that the date are differing by exactly one month so that we can count in its month - month served.
- Dateofjoining will not change.
- Quarterly rating to mean reating(total rating/ total months).

In [26]:
```
df3 = df2.groupby(by=['Driver_ID', "Gender", 'Dateofjoining', 'Joining Designation']









df3.rename(columns={'MMM-YY' : 'TotalexpMonths', 'Income': 'tot_income'}, inplace=Tr
df3['hasNegBusiValue'] = df3['Total Business Value'].apply(lambda x : 1 if min(x) <
df3['totBusiValue'] = df3['Total Business Value'].apply(lambda x: sum(x))
df3['avg_income'] = df3['tot_income']/ df3['TotalexpMonths']
```

In [27]:
```
df4 = df3[['Gender', 'Joining Designation',
       'TotalexpMonths', 'Age', 'City', 'Education_Level', 'tot_income','avg_income'
        'Grade', 'Quarterly Rating',
       'TotalexpinDays', 'hasNegBusiValue', 'totBusiValue', 'LastWorkingDate']]
df4
```

Out[27]:

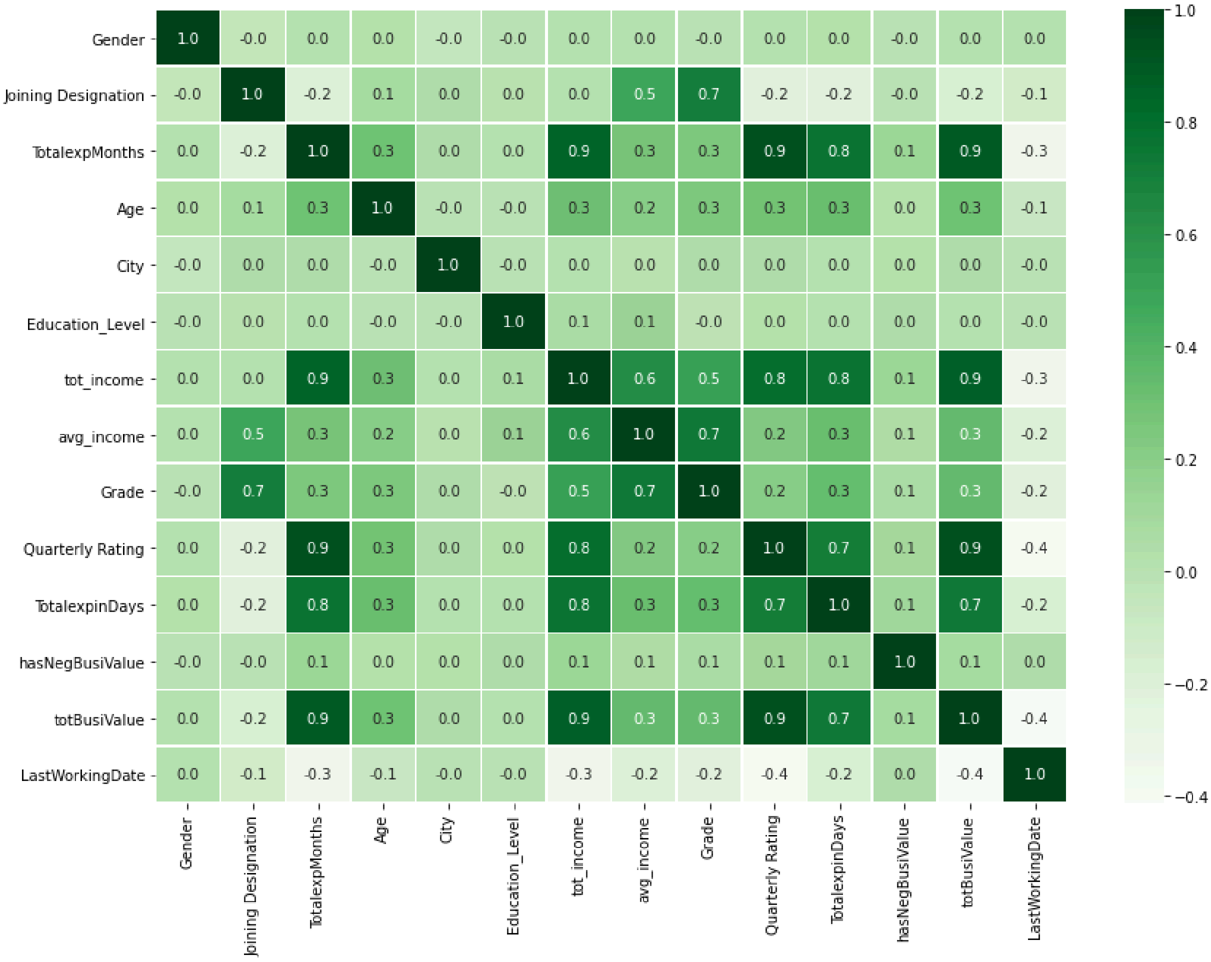
	Gender	Joining Designation	TotalexpMonths	Age	City	Education_Level	tot_income	avg_income	G
0	0.0	1	3	28.0	23	2	172161.0	57387.0	
1	0.0	2	2	31.0	7	2	134032.0	67016.0	
2	0.0	2	5	43.0	13	2	328015.0	65603.0	
3	0.0	1	3	29.0	9	0	139104.0	46368.0	
4	1.0	3	5	31.0	11	1	393640.0	78728.0	
...	...	...	...	...	...	...	...	...	
2376	0.0	2	24	34.0	24	0	1987560.0	82815.0	
2377	1.0	1	3	34.0	9	0	36315.0	12105.0	
2378	0.0	2	9	45.0	19	0	318330.0	35370.0	
2379	1.0	1	6	28.0	20	2	416988.0	69498.0	
2380	0.0	2	7	30.0	27	2	491778.0	70254.0	

2381 rows × 14 columns

In [28]:

```
plt.figure(figsize=(14,10))
sns.heatmap(df4.corr(), annot=True, fmt='.1f', cmap="Greens" , linewidth=.5)
```

Out[28]: <AxesSubplot:>



Encoding:

- Encoding is better to use here as it will work numeric for all categorical columns from range 0 to n.

In [29]:
```
df4.head()
```

Out[29]:

	Gender	Joining Designation	TotalexpMonths	Age	City	Education_Level	tot_income	avg_income	Grad
0	0.0	1	3	28.0	23	2	172161.0	57387.0	
1	0.0	2	2	31.0	7	2	134032.0	67016.0	
2	0.0	2	5	43.0	13	2	328015.0	65603.0	
3	0.0	1	3	29.0	9	0	139104.0	46368.0	
4	1.0	3	5	31.0	11	1	393640.0	78728.0	

In [30]:
```
df4.columns
```

Out[30]:
```
Index(['Gender', 'Joining Designation', 'TotalexpMonths', 'Age', 'City',
       'Education_Level', 'tot_income', 'avg_income', 'Grade',
       'Quarterly Rating', 'TotalexpinDays', 'hasNegBusiValue', 'totBusiValue',
       'LastWorkingDate'],
      dtype='object')
```

In [31]:
```
labelenc = LabelEncoder()

for i in ['Gender', 'Joining Designation', 'TotalexpMonths', 'Age', 'City','Educatio
  df4[i] = labelenc.fit_transform(df4[i])
```

In [32]:
```
df4.head()
```

Out[32]:

	Gender	Joining Designation	TotalexpMonths	Age	City	Education_Level	tot_income	avg_income	Grad
0	0	0	2	7	22	2	172161.0	57387.0	
1	0	1	1	10	6	2	134032.0	67016.0	
2	0	1	4	22	12	2	328015.0	65603.0	
3	0	0	2	8	8	0	139104.0	46368.0	
4	1	2	4	10	10	1	393640.0	78728.0	

- By label encoding, data looks better for further steps i.e Training and Testing.

Methods:

Train Test Split :

- Our data is finalised and we split the data for scalling and train our model : data without balancing.

Scalling

In [33]:
```
X = df4.drop(columns=['LastWorkingDate'], axis=True)
y = df4['LastWorkingDate']
```

In [34]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[34]:
```
((1904, 13), (477, 13), (1904,), (477,))
```

Machine Learning Model with imbalance data:

Logistic Regression

In [35]:
```
model = LogisticRegression()
model.fit(X_train, y_train)

print("training score",model.score(X_train, y_train))
print("test score",model.score(X_test, y_test))

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
training score 0.7510504201680672
test score 0.7693920335429769
              precision    recall  f1-score   support

           0       0.74      0.44      0.55       154
           1       0.78      0.93      0.84       323

    accuracy                           0.77       477
   macro avg       0.76      0.68      0.70       477
weighted avg       0.76      0.77      0.75       477
```

KNN classifier

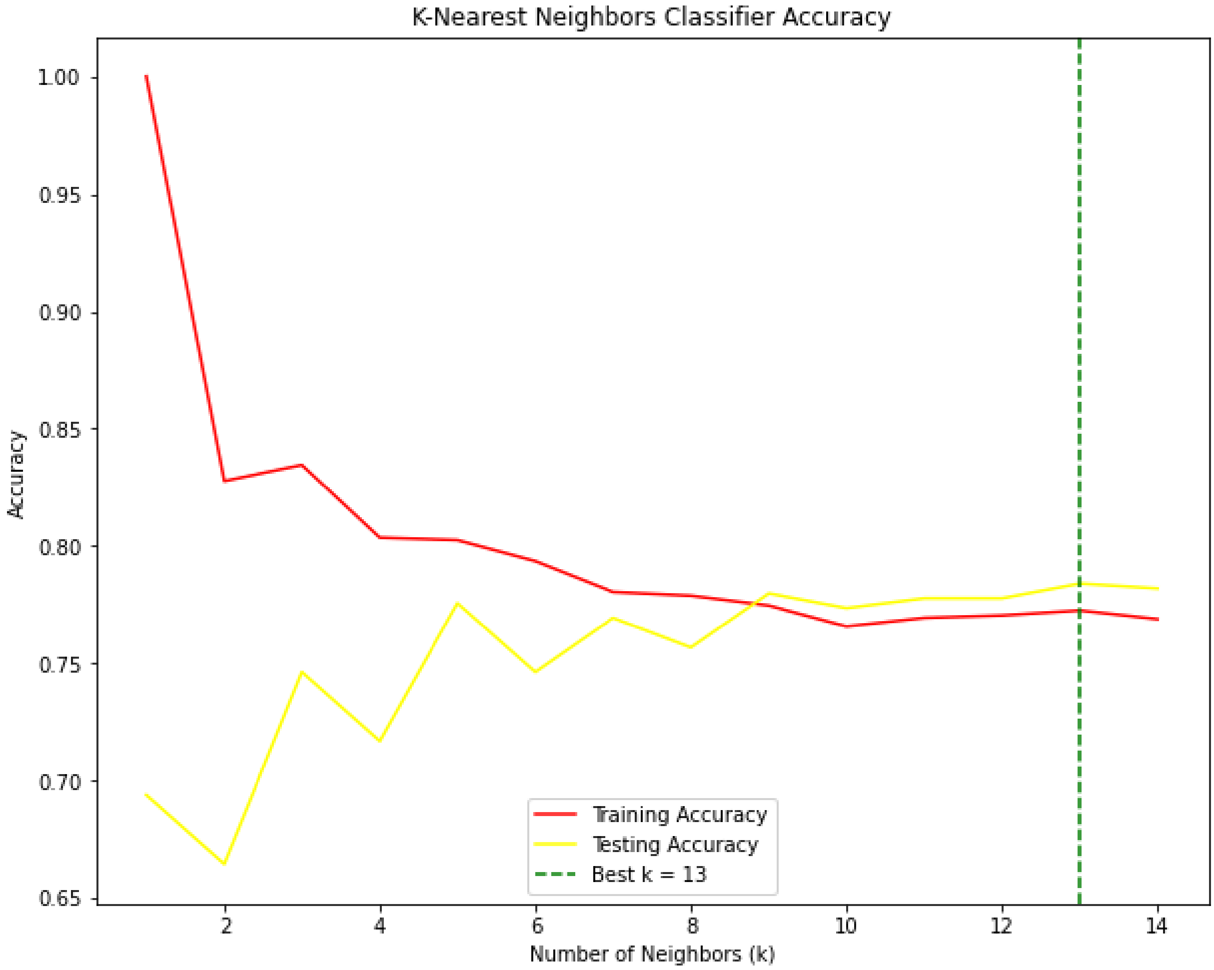
In [36]:
```
model = KNeighborsClassifier()
model.fit(X_train, y_train)

print("training score",model.score(X_train, y_train))
print("test score",model.score(X_test, y_test))

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
training score 0.8025210084033614
test score 0.7756813417190775
              precision    recall  f1-score   support

           0       0.72      0.50      0.59       154
```

```
           1       0.79      0.91      0.85       323

    accuracy                           0.78       477
   macro avg       0.76      0.70      0.72       477
weighted avg       0.77      0.78      0.76       477
```

In [37]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier

# Initialize empty lists to store training and testing scores
y_train_score = []
y_test_score = []

# Loop through different values of k
for i in range(1, 15):
    # Initialize the KNN classifier with i neighbors
    model = KNeighborsClassifier(n_neighbors=i)

    # Train the model on the training data
    model.fit(X_train, y_train)

    # Compute and store the training accuracy
    y_train_score.append(model.score(X_train, y_train))

    # Compute and store the testing accuracy
    y_test_score.append(model.score(X_test, y_test))

plt.figure(figsize=(10, 8))

# Plot the training and testing accuracy scores
sns.lineplot(range(1, 15), y_train_score, color='red', label='Training Accuracy')
sns.lineplot(range(1, 15), y_test_score, color='yellow', label='Testing Accuracy')

# Add a vertical dashed line at the index where the maximum testing accuracy occurs
best_k = y_test_score.index(max(y_test_score)) + 1  # Add 1 to convert index to k va
plt.axvline(x=best_k, linestyle='--', color='green', label=f'Best k = {best_k}')

# Set plot labels and title
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.title('K-Nearest Neighbors Classifier Accuracy')
plt.legend()
plt.show()
```



In [38]:

```
max(y_test_score)
```

Out[38]: 0.7840670859538784

In [39]:

```
pip install pydot
```

```
Requirement already satisfied: pydot in c:\users\asus\anaconda3\lib\site-packages (2.
0.0)Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: pyparsing>=3 in c:\users\asus\anaconda3\lib\site-packa
ges (from pydot) (3.1.2)
```

In [40]:

```
pip install pyparsing pydot
```

```
Requirement already satisfied: pyparsing in c:\users\asus\anaconda3\lib\site-packages
(3.1.2)
Requirement already satisfied: pydot in c:\users\asus\anaconda3\lib\site-packages (2.
0.0)
Note: you may need to restart the kernel to use updated packages.
```

Decision Tree Classifier:

In [41]:

```
features = list((df4.drop(columns=['LastWorkingDate'])).columns)

model = DecisionTreeClassifier(criterion = 'gini')
model.fit(X_train, y_train)

print("training score",model.score(X_train, y_train))
print("test score",model.score(X_test, y_test))
```

```
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

dot_data = StringIO()
export_graphviz(model, out_file=dot_data, feature_names=features, filled=True)
```

```
training score 1.0
test score 0.7819706498951782
              precision    recall  f1-score   support

           0       0.65      0.71      0.68       154
           1       0.85      0.82      0.84       323

    accuracy                           0.78       477
   macro avg       0.75      0.76      0.76       477
weighted avg       0.79      0.78      0.78       477
```

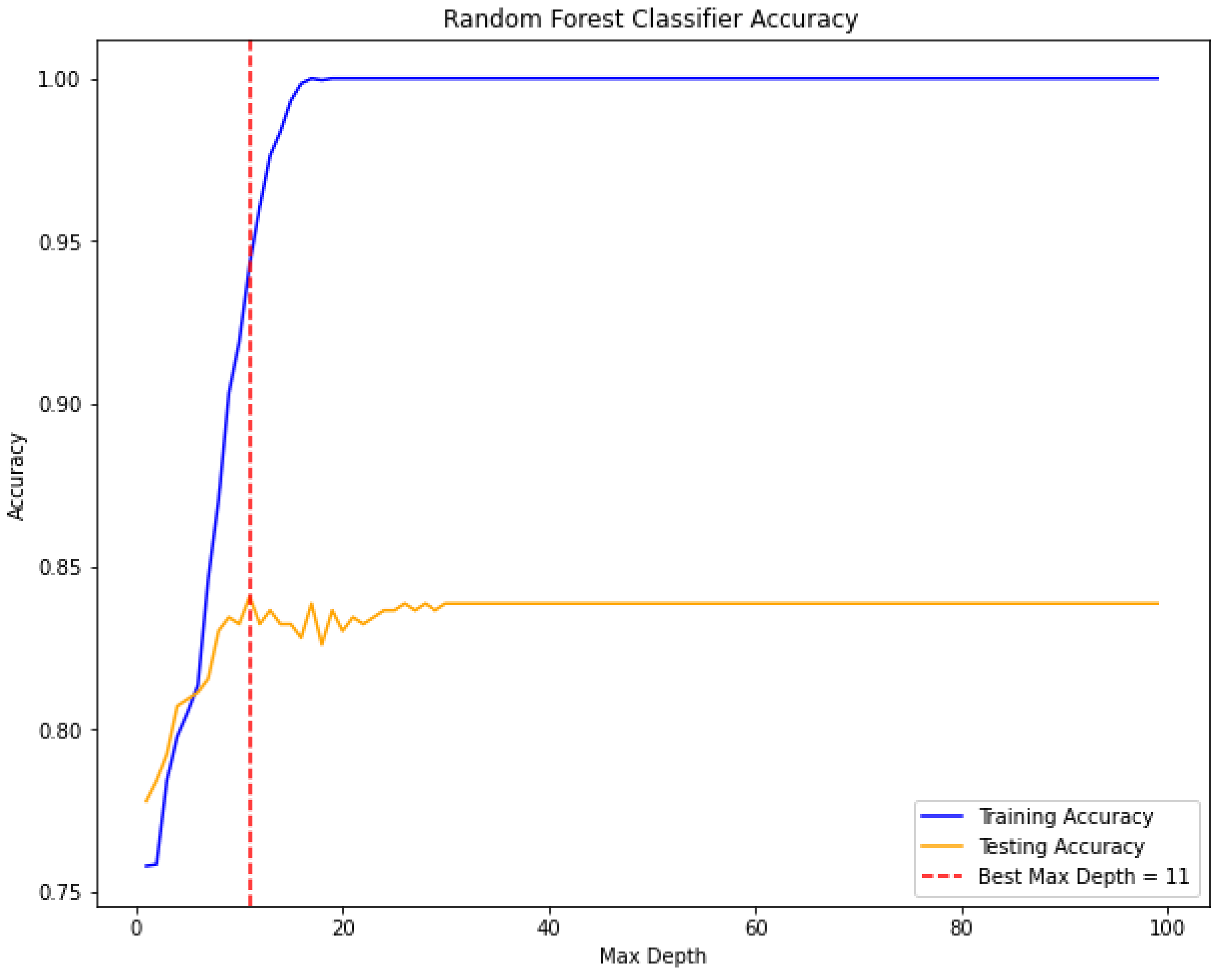
RandomForestClassifier <-- Selected Model:

In [42]:
```
model = RandomForestClassifier()
model.fit(X_train, y_train)

print("training score",model.score(X_train, y_train))
print("test score",model.score(X_test, y_test))

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
training score 1.0
test score 0.8301886792452831
              precision    recall  f1-score   support

           0       0.79      0.64      0.71       154
           1       0.84      0.92      0.88       323

    accuracy                           0.83       477
   macro avg       0.82      0.78      0.79       477
weighted avg       0.83      0.83      0.83       477
```

In [43]:
```
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize empty lists to store training and testing scores
y_train_score = []
y_test_score = []

# Loop through different values of max_depth
for i in range(1, 100):
    # Initialize the Random Forest classifier with max_depth=i
    model = RandomForestClassifier(max_depth=i, random_state=42)  # random_state for

    # Train the model on the training data
    model.fit(X_train, y_train)

    # Compute and store the training accuracy
    y_train_score.append(model.score(X_train, y_train))

    # Compute and store the testing accuracy
    y_test_score.append(model.score(X_test, y_test))
```

```
plt.figure(figsize=(10,8))
# Plot the training and testing accuracy scores
sns.lineplot(range(1, 100), y_train_score, color='blue', label='Training Accuracy')
sns.lineplot(range(1, 100), y_test_score, color='orange', label='Testing Accuracy')

# Add a vertical dashed line at the index where the maximum testing accuracy occurs
best_max_depth = y_test_score.index(max(y_test_score)) + 1  # Add 1 to convert index
plt.axvline(x=best_max_depth, linestyle='--', color='red', label=f'Best Max Depth =

# Set plot labels and title
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.title('Random Forest Classifier Accuracy')
plt.legend()
plt.show()
```



Conclusion:

- From this we come to know that all the 3 algorithms are good but not enough for modelling, will do the following precess:
- 1. Balance data
- 1. Bagging
- 1. Boosting algorithms
- We have all the precission value to be around 80% only, but we need above 90 atleast.

Machine Learning Model with balanced data and comparision:

In [44]:
```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\asus\anaconda3\lib\site-p
ackages (0.12.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\asus\anaconda3\lib\si
te-packages (from imbalanced-learn) (2.1.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\asus\anaconda3\lib\site-pack
ages (from imbalanced-learn) (1.20.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\asus\anaconda3\lib\sit
e-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: scipy>=1.5.0 in c:\users\asus\anaconda3\lib\site-packa
ges (from imbalanced-learn) (1.6.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\asus\anaconda3\lib\site-pack
ages (from imbalanced-learn) (1.3.2)
Note: you may need to restart the kernel to use updated packages.
```

In [45]:
```
from imblearn.over_sampling import SMOTE
```

In [46]:
```
smt = SMOTE()

print('Before SMOTE')
print(y_train.value_counts())

x_sm, y_sm = smt.fit_resample(X_train, y_train)
print('\nAfter SMOTE')
print(y_sm.value_counts())
```

```
Before SMOTE
1    1288
0     616
Name: LastWorkingDate, dtype: int64

After SMOTE
0    1288
1    1288
Name: LastWorkingDate, dtype: int64
```

- The purpose of this code is to demonstrate how the class distribution changes before and after applying SMOTE, which is a technique commonly used to address class imbalance in classification problems.
- By balancing the classes, it helps improve the performance of machine learning models, especially when dealing with imbalanced datasets.

Logistic Regression

In [47]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
print('for Normal Data')
print("training score",model.score(X_train, y_train))
print("test score",model.score(X_test, y_test))

y_pred = model.predict(X_test)
print('\n',classification_report(y_test, y_pred))
```

```python
model = LogisticRegression()
model.fit(x_sm, y_sm)
print('for Balanced Data')
print("training score",model.score(x_sm, y_sm))
print("test score",model.score(X_test, y_test))

y_pred = model.predict(X_test)
print('\n',classification_report(y_test, y_pred))
```

```
for Normal Data
training score 0.7510504201680672
test score 0.7693920335429769

              precision    recall  f1-score   support

           0       0.74      0.44      0.55       154
           1       0.78      0.93      0.84       323

    accuracy                           0.77       477
   macro avg       0.76      0.68      0.70       477
weighted avg       0.76      0.77      0.75       477

for Balanced Data
training score 0.6909937888198758
test score 0.7756813417190775

              precision    recall  f1-score   support

           0       0.67      0.60      0.63       154
           1       0.82      0.86      0.84       323

    accuracy                           0.78       477
   macro avg       0.74      0.73      0.74       477
weighted avg       0.77      0.78      0.77       477
```

KNN classification

In [48]:

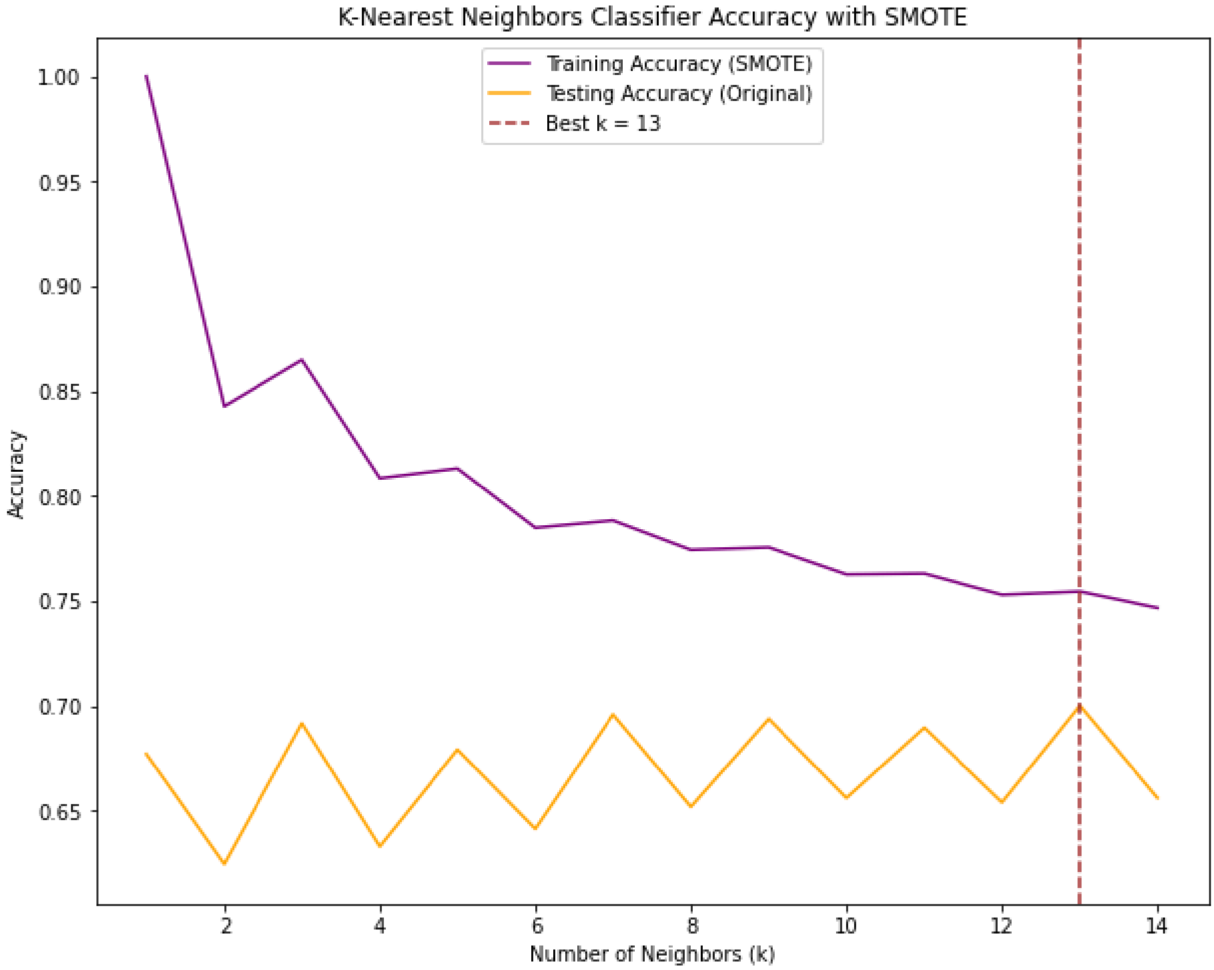
```python
model = KNeighborsClassifier()
model.fit(X_train, y_train)
print('for Normal Data')
print("training score",model.score(X_train, y_train))
print("test score",model.score(X_test, y_test))

y_pred = model.predict(X_test)
print('\n',classification_report(y_test, y_pred))

model = KNeighborsClassifier()
model.fit(x_sm, y_sm)
print('for Balanced Data')
print("training score",model.score(x_sm, y_sm))
print("test score",model.score(X_test, y_test))

y_pred = model.predict(X_test)
print('\n',classification_report(y_test, y_pred))
```

```
for Normal Data
training score 0.8025210084033614
test score 0.7756813417190775

              precision    recall  f1-score   support

           0       0.72      0.50      0.59       154
           1       0.79      0.91      0.85       323
```

```
    accuracy                           0.78       477
   macro avg       0.76      0.70      0.72       477
weighted avg       0.77      0.78      0.76       477

for Balanced Data
training score 0.8132763975155279
test score 0.6792452830188679

              precision    recall  f1-score   support

           0       0.50      0.66      0.57       154
           1       0.81      0.69      0.74       323

    accuracy                           0.68       477
   macro avg       0.66      0.67      0.66       477
weighted avg       0.71      0.68      0.69       477
```

In [49]:

```python
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize empty lists to store training and testing scores
y_train_score = []
y_test_score = []

# Loop through different values of k
for i in range(1, 15):
    # Initialize the KNN classifier with i neighbors
    model = KNeighborsClassifier(n_neighbors=i)

    # Train the model on the balanced training data (after SMOTE)
    model.fit(x_sm, y_sm)

    # Compute and store the training accuracy on the balanced data
    y_train_score.append(model.score(x_sm, y_sm))

    # Compute and store the testing accuracy on the original testing data
    y_test_score.append(model.score(X_test, y_test))

plt.figure(figsize = (10,8))

# Plot the training and testing accuracy scores
sns.lineplot(range(1, 15), y_train_score, color='purple', label='Training Accuracy (
sns.lineplot(range(1, 15), y_test_score, color='orange', label='Testing Accuracy (Or

# Add a vertical dashed line at the index where the maximum testing accuracy occurs
best_k = y_test_score.index(max(y_test_score)) + 1  # Add 1 to convert index to k va
plt.axvline(x=best_k, linestyle='--', color='brown', label=f'Best k = {best_k}')

# Set plot labels and title
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.title('K-Nearest Neighbors Classifier Accuracy with SMOTE')
plt.legend()
plt.show()
```



DecisionTree Classifier

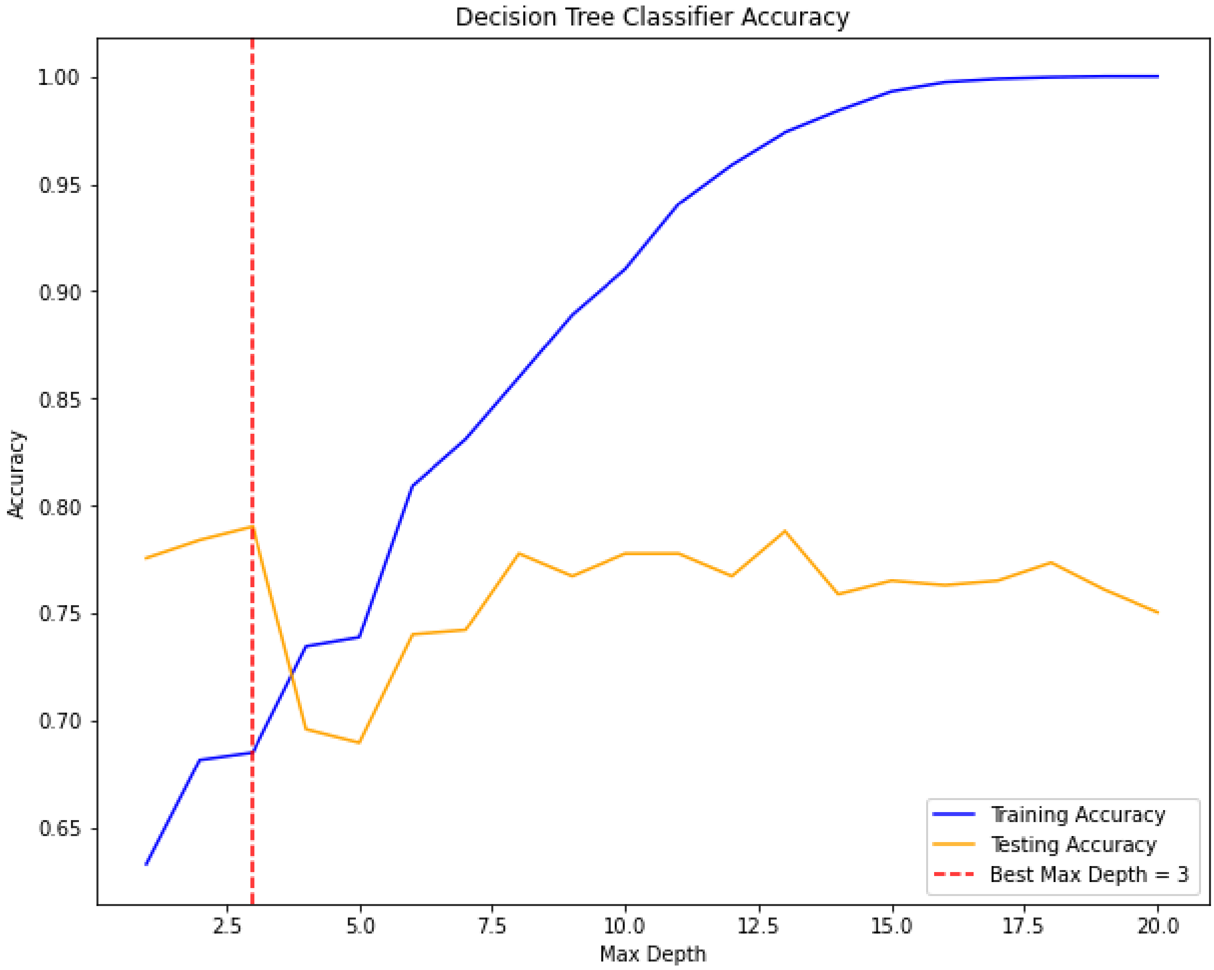
In [50]:
```
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
print('for Normal Data')
print("training score",model.score(X_train, y_train))
print("test score",model.score(X_test, y_test))

y_pred = model.predict(X_test)
print('\n',classification_report(y_test, y_pred))

model = DecisionTreeClassifier()
model.fit(x_sm, y_sm)
print('for Balanced Data')
print("training score",model.score(x_sm, y_sm))
print("test score",model.score(X_test, y_test))

y_pred = model.predict(X_test)
print('\n',classification_report(y_test, y_pred))
```

```
for Normal Data
training score 1.0
test score 0.7966457023060797

              precision    recall  f1-score   support

           0       0.67      0.72      0.70       154
           1       0.86      0.83      0.85       323

    accuracy                           0.80       477
   macro avg       0.77      0.78      0.77       477
weighted avg       0.80      0.80      0.80       477
```

```
for Balanced Data
training score 1.0
test score 0.7714884696016772

              precision    recall  f1-score   support

           0       0.63      0.73      0.67       154
           1       0.86      0.79      0.82       323

    accuracy                           0.77       477
   macro avg       0.74      0.76      0.75       477
weighted avg       0.78      0.77      0.78       477
```

In [51]:

```
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import seaborn as sns

y_train_score = []
y_test_score = []

max_depth_range = range(1, 21)  # Choose a range of max_depth values

for depth in max_depth_range:
    model = DecisionTreeClassifier(max_depth=depth)
    model.fit(x_sm, y_sm)
    y_train_score.append(model.score(x_sm, y_sm))
    y_test_score.append(model.score(X_test, y_test))

plt.figure(figsize=(10, 8))
sns.lineplot(max_depth_range, y_train_score, color='blue', label='Training Accuracy'
sns.lineplot(max_depth_range, y_test_score, color='orange', label='Testing Accuracy'
best_max_depth = y_test_score.index(max(y_test_score)) + 1
plt.axvline(x=best_max_depth, linestyle='--', color='red', label=f'Best Max Depth =
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.title('Decision Tree Classifier Accuracy')
plt.legend()
plt.show()
```



Hyperparameter Tunning :

- For hyperparameter tunning we use random forest with its hyperparameters.

In [52]:
```
model_rfc = RandomForestClassifier(criterion='gini', n_jobs=-1)
model_rfc.fit(X_train, y_train)
```

Out[52]:
```
▾      RandomForestClassifier
RandomForestClassifier(n_jobs=-1)
```

In [53]:
```
print('train score',model_rfc.score(X_train, y_train))
print('test score',model_rfc.score(X_test, y_test))
```

```
train score 1.0
test score 0.8343815513626834
```

In [54]:
```
model_rfc.feature_importances_
```

Out[54]:
```
array([0.01582872, 0.03588476, 0.11444229, 0.06483209, 0.07354901,
       0.02391556, 0.10873604, 0.08784599, 0.02288927, 0.11157216,
       0.19412834, 0.00379895, 0.14257683])
```

- From this we can see that the train and test score are in a big difference means we have a overfit model.

- We have to get rid of this overfit model.

In [55]:
```
hyp_prams = {
    "n_estimators": [100,200,300,400,500],
    "max_depth" : [10, 20, 30,40,50,60,70,80,90,100]
}

rfc = RandomForestClassifier(criterion='gini', n_jobs=-1)

# model_hyp = GridSearchCV(rfc, hyp_prams)
model_hyp = RandomizedSearchCV(rfc, hyp_prams)
model_hyp.fit(X_train, y_train)

print(model_hyp.best_params_)
```

```
{'n_estimators': 100, 'max_depth': 100}
```

In [56]:
```
model_rfc = RandomForestClassifier(criterion='gini', n_jobs=-1, **model_hyp.best_par
model_rfc.fit(X_train, y_train)

print('train score',model_rfc.score(X_train, y_train))
print('test score',model_rfc.score(X_test, y_test))
```

```
train score 0.9994747899159664
test score 0.8155136268343816
```

In [57]:
```
y_pred = model_rfc.predict(X_test)
cr = classification_report(y_test, y_pred)
print('cm', cr)
cm = confusion_matrix(y_test, y_pred)
print('cm', cm)
```

```
cm               precision    recall  f1-score   support

           0       0.75      0.64      0.69       154
           1       0.84      0.90      0.87       323

    accuracy                           0.82       477
   macro avg       0.80      0.77      0.78       477
weighted avg       0.81      0.82      0.81       477

cm [[ 98  56]
 [ 32 291]]
```

Bagging :

In [58]:
```
model_baga = BaggingClassifier()
model_baga.fit(X_train, y_train)
```

Out[58]:
```
▾ BaggingClassifier

BaggingClassifier()
```

In [59]:
```
print('train score',model_baga.score(X_train, y_train))
print('test score',model_baga.score(X_test, y_test))
y_pred = model_baga.predict(X_test)
cr = classification_report(y_test, y_pred)
print('cm', cr)
```

```
cm = confusion_matrix(y_test, y_pred)
print('cm', cm)
```

```
train score 0.9894957983193278
test score 0.8155136268343816
cm               precision    recall  f1-score   support

           0       0.69      0.78      0.73       154
           1       0.89      0.83      0.86       323

    accuracy                           0.82       477
   macro avg       0.79      0.81      0.80       477
weighted avg       0.82      0.82      0.82       477

cm [[120  34]
 [ 54 269]]
```

Boosting : <-- Best Model:

XGBoost

In [60]:

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\asus\anaconda3\lib\site-packages
(2.0.3)
Requirement already satisfied: numpy in c:\users\asus\anaconda3\lib\site-packages (fr
om xgboost) (1.20.1)
Requirement already satisfied: scipy in c:\users\asus\anaconda3\lib\site-packages (fr
om xgboost) (1.6.2)
Note: you may need to restart the kernel to use updated packages.
```

In [61]:

```
from xgboost import XGBClassifier
import xgboost
model_bost = XGBClassifier()
model_bost.fit(X_train, y_train)
```

Out[61]:

```
▾                          XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=N
one,
              enable_categorical=False, eval_metric=None, feature_types=N
one,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=N
one,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
```

In [62]:

```
print('train score',model_bost.score(X_train, y_train))
print('test score',model_bost.score(X_test, y_test))
y_pred = model_bost.predict(X_test)
cr = classification_report(y_test, y_pred)
print('cm', cr)
cm = confusion_matrix(y_test, y_pred)
print('cm', cm)
```

```
train score 0.9989495798319328
test score 0.8176100628930818
```

```
cm               precision    recall  f1-score   support

           0       0.73      0.69      0.71       154
           1       0.86      0.88      0.87       323

    accuracy                           0.82       477
   macro avg       0.79      0.79      0.79       477
weighted avg       0.82      0.82      0.82       477

cm [[107  47]
 [ 40 283]]
```

Hyperparmeter tunning

In [63]:

```
params = {
        "n_estimators": [150,200, 250, 300],
        "max_depth" : [2, 3, 4, 5, 7],
        "learning_rate": [0.01, 0.02, 0.05, 0.07],
        'subsample': [0.4, 0.5,0.6, 0.8],
        'colsample_bytree': [0.6, 0.8, 1.0],
        }

xgb = XGBClassifier(objective='multi:softmax', num_class=20, silent=True)
random_search = RandomizedSearchCV( xgb, param_distributions = params,scoring='accur

random_search.fit(X_train, y_train)
```

Out[63]:

```
▸ RandomizedSearchCV
▸ estimator: XGBClassifier
  ▸ XGBClassifier
```

In [64]:

```
random_search.best_params_
```

Out[64]:

```
{'subsample': 0.4,
 'n_estimators': 200,
 'max_depth': 2,
 'learning_rate': 0.05,
 'colsample_bytree': 0.8}
```

In [65]:

```
xgb = XGBClassifier(**random_search.best_params_ , num_classes=20)
xgb.fit(X_train, y_train)
print('train score',xgb.score(X_train, y_train))
print('test score',xgb.score(X_test, y_test))
y_pred = xgb.predict(X_test)
cr = classification_report(y_test, y_pred)
print('cm', cr)
cm = confusion_matrix(y_test, y_pred)
print('cm', cm)
```

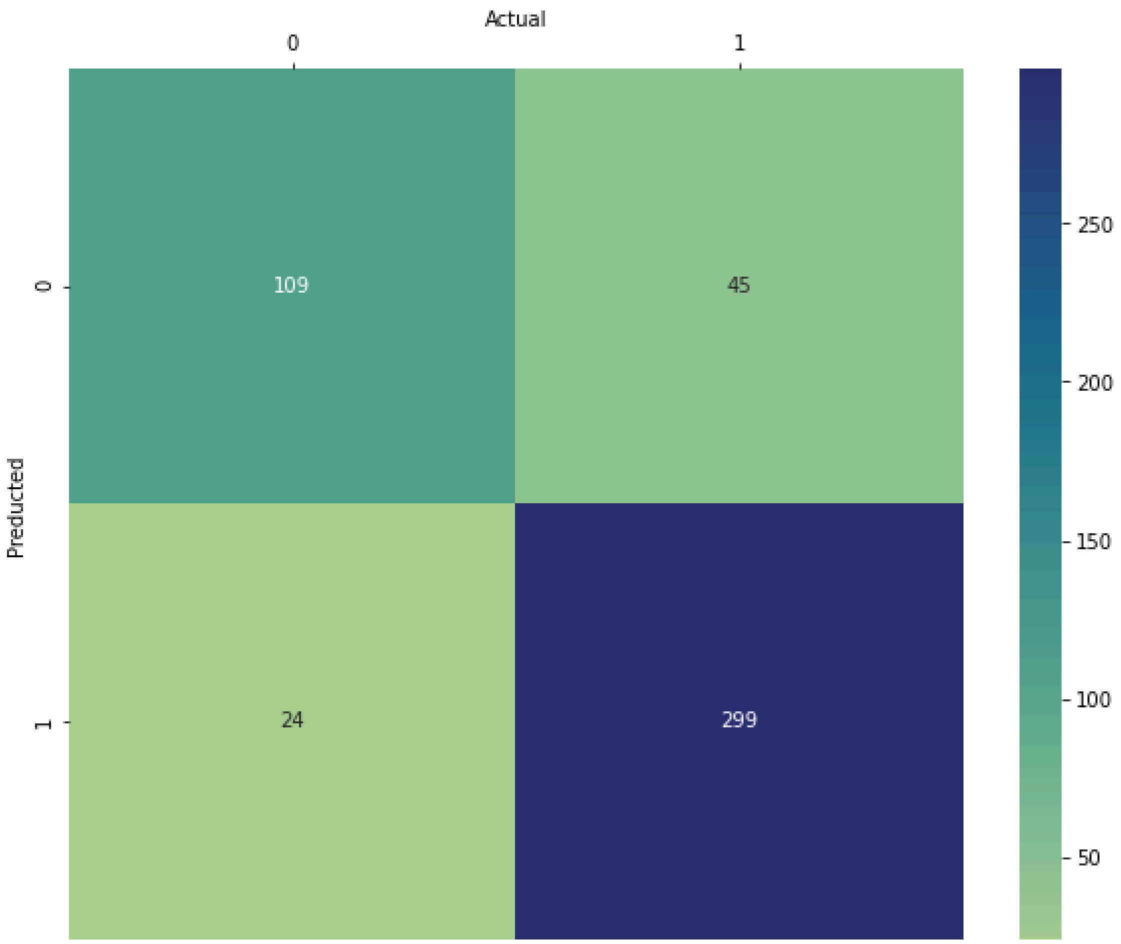
```
train score 0.854516806722689
test score 0.8553459119496856
cm               precision    recall  f1-score   support

           0       0.82      0.71      0.76       154
           1       0.87      0.93      0.90       323

    accuracy                           0.86       477
   macro avg       0.84      0.82      0.83       477
weighted avg       0.85      0.86      0.85       477
```

```
cm [[109  45]
 [ 24 299]]
```

- The best Model for training and it precession is good and all points are discovered.
- Model has fit into the Prefect-model , No-Overfir or No-Underfit.

In [66]:

```
plt.figure(figsize = (10,8))
ax = sns.heatmap(cm, annot=True, cmap="crest", fmt='g')
ax.set(xlabel="Actual", ylabel="Preducted")
ax.xaxis.set_label_position('top')
ax.xaxis.tick_top()
```

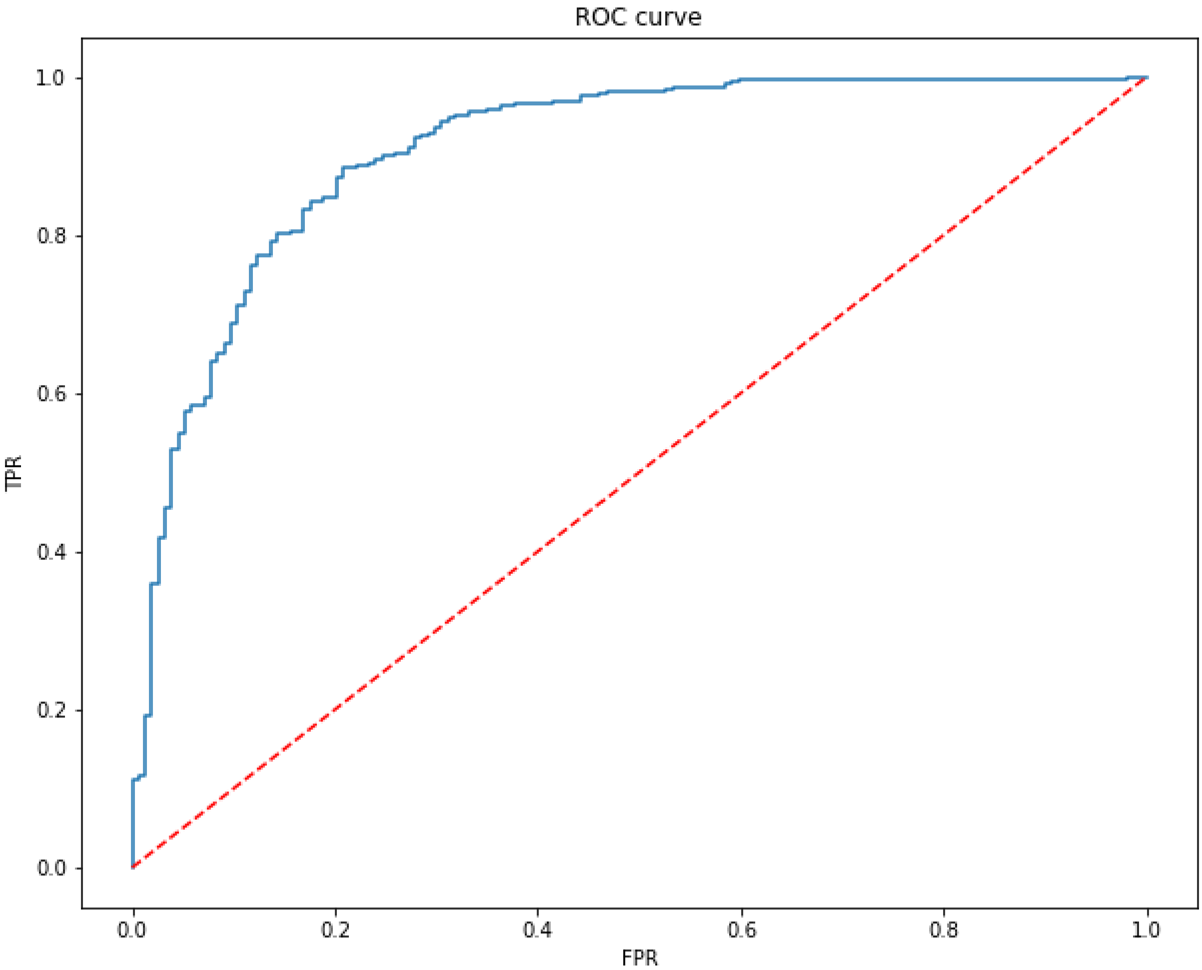


- Here we have predicted 299 events that are actually 1. We also predicted as 1. This is changeable.
- But actual 1 and we predicted 0 which has account of 45.

In [67]:

```
prob = (xgb.predict_proba(X_test))[:,1]
fpr, tpr, thr = roc_curve(y_test, prob)
plt.figure(figsize = (10,8))

plt.plot(fpr,tpr)
plt.plot(fpr,fpr,'--',color='red' )
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```

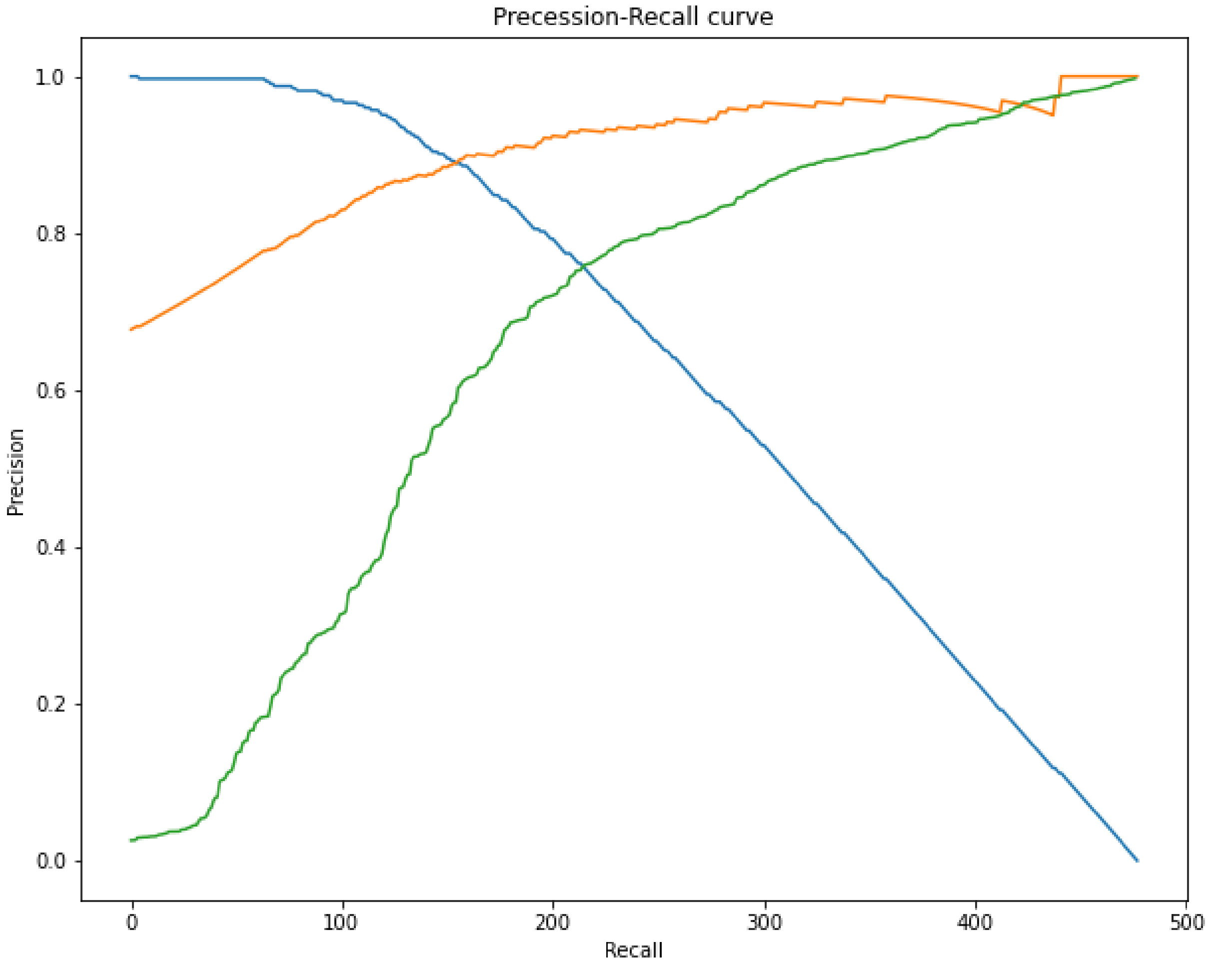
```
print('\nAUC-ROC score : ',roc_auc_score(y_test,prob))
```




```
AUC-ROC score :  0.9073016766515218
```

In [68]:

```
precision, recall, thr = precision_recall_curve(y_test, prob)

plt.figure(figsize = (10,8))

plt.plot(recall) #blue
plt.plot(precision) # orange
plt.plot(thr) # green

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precession-Recall curve')
plt.show()
```



- We have a good P-R curve but its ok to have a curve like this.

Results :

- Before diving into specific recommendations, it's essential to understand what ensemble learning is and how it works. Ensemble learning combines multiple machine learning models to improve prediction accuracy and robustness over individual models. There are various ensemble methods such as bagging, boosting, and stacking. Preprocessing:
- From the analysis and feature selections, we get the idea how much driver are working and leaving.
- Encoding used for data efficiency.
- Played with Imbalanced and Balanced data using Logistic Regression, KNN classifier, DecisionTree Classifier, RandomForest, Hyperparameter Tuning, Bagging Boosting to see which algorithm is good for this data.
- By doing Comparison between balanced and imbalanced data we get know that balanced is good compared to imbalanced. By ROC curve we get know balanced data is about 90% which is better.
- Experiment with different types of algorithms to capture diverse patterns in the data.

- Perform hyperparameter tuning for both individual models and ensemble methods to optimize their performance.
- Use techniques like grid search or random search to efficiently search the hyperparameter space.

Conclusion :

- In conclusion, the analysis of driver team attrition at Ola presents several key findings and recommendations.
- Firstly, the high churn rate among drivers poses a significant challenge for the company, impacting morale and incurring substantial costs associated with driver acquisition.
- Through the examination of monthly data for 2019 and 2020, it is evident that demographic factors such as age, gender, and city, alongside tenure information and historical performance metrics, play crucial roles in predicting driver attrition.
- Leveraging ensemble learning techniques such as bagging and boosting, as well as KNN imputation for handling missing values, proves to be effective in developing predictive models for identifying drivers at risk of leaving the company.
- Additionally, given the imbalanced nature of the dataset, strategies for working with imbalanced data, such as oversampling or incorporating class weights, are essential for achieving accurate predictions.
- Moving forward, Ola can use these insights to implement targeted retention strategies, focusing on factors identified as significant predictors of attrition.
- By addressing the root causes of driver churn and prioritizing the retention of existing drivers, Ola can mitigate the adverse effects of high turnover rates and ensure the stability and sustainability of its driver workforce.