Problem Statement

Recruiting and retaining drivers is seen by industry watchers 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 Uber 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)

Column Profiling:

- MMMM-YY : Reporting Date (Monthly)
- Driver_ID : Unique id for drivers
- Age : Age of the driver
- Gender : Gender of the driver – Male : 0, Female: 1
- City : City Code of the driver
- Education_Level : Education level – 0 for 10+ ,1 for 12+ ,2 for graduate
- Income : Monthly average Income of the driver
- Date Of Joining : Joining date for the driver
- LastWorkingDate : Last date of working for the driver
- Joining Designation : Designation of the driver at the time of joining
- Grade : Grade of the driver at the time of reporting
- Total Business Value : The total business value acquired by the driver in a month (negative business indicates cancellation/refund or car EMI adjustments)
- Quarterly Rating : Quarterly rating of the driver: 1,2,3,4,5 (higher is better)

Concepts Tested:

- **Ensemble Learning- Bagging**
- **Ensemble Learning- Boosting**
- **KNN Imputation of Missing Values**
- **Working with an imbalanced dataset**

In [1]:
```
#For dealing with tables
import pandas as pd
#For dealing with linear algebra
import numpy as np
#For data visualization and plotting graphs
import matplotlib.pyplot as plt
import seaborn as sns
#For minmaxscaler
from sklearn.preprocessing import MinMaxScaler
#For shapiro test
from scipy.stats import shapiro
#For train-test split
from sklearn.model_selection import train_test_split,GridSearchCV
#For RandomForest
from sklearn.ensemble import RandomForestClassifier
#Accuracy score, confusion matrix, classification report, ROC curve, AUC
from sklearn.metrics import f1_score, roc_auc_score, RocCurveDisplay, roc_curve
#To ignore warnings
import warnings
warnings.filterwarnings("ignore")
from sklearn.impute import KNNImputer

# to balance data using SMOTE
from imblearn.over_sampling import SMOTE

from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report,confusion_matrix
```

In [2]:
```
# loading the data
df= pd.read_csv("ola_driver_scaler.csv")
```

In [3]:
```
df.head()
```

Out[3]:

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

In [4]:
```
print(f"Number of rows: {df.shape[0]}\nNumber of columns: {df.shape[1]}")
```

```
Number of rows: 19104
Number of columns: 14
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Unnamed: 0            19104 non-null  int64  
 1   MMM-YY                19104 non-null  object 
 2   Driver_ID             19104 non-null  int64  
 3   Age                   19043 non-null  float64
 4   Gender                19052 non-null  float64
 5   City                  19104 non-null  object 
 6   Education_Level       19104 non-null  int64  
 7   Income                19104 non-null  int64  
 8   Dateofjoining         19104 non-null  object 
 9   LastWorkingDate       1616 non-null   object 
 10  Joining Designation   19104 non-null  int64  
 11  Grade                 19104 non-null  int64  
 12  Total Business Value  19104 non-null  int64  
 13  Quarterly Rating      19104 non-null  int64  
dtypes: float64(2), int64(8), object(4)
memory usage: 2.0+ MB
```

In [6]:
```
# checking unique value count in each column
df.nunique()
```

Out[6]:
```
Unnamed: 0              19104
MMM-YY                     24
Driver_ID                2381
Age                        36
Gender                      2
City                       29
Education_Level             3
Income                   2383
Dateofjoining             869
LastWorkingDate           493
Joining Designation         5
Grade                       5
Total Business Value    10181
Quarterly Rating            4
dtype: int64
```

In [7]:
```
# checking duplicated values
df.duplicated().sum()
```

Out[7]:
```
0
```

- we can drop "Unnamed: 0" column

In [8]:
```
df.drop(columns=("Unnamed: 0"), axis=1, inplace=True)
```

In [9]:
```
df.columns
```

```
Out[9]: Index(['MMM-YY', 'Driver_ID', 'Age', 'Gender', 'City', 'Education_Level',
               'Income', 'Dateofjoining', 'LastWorkingDate', 'Joining Designation',
               'Grade', 'Total Business Value', 'Quarterly Rating'],
              dtype='object')
```

```
In [10]: df.describe(include="all").T
```

Out[10]:

	count	unique	top	freq	mean	std	
MMM-YY	19104	24	01/01/19	1022	NaN	NaN	N
Driver_ID	19104.0	NaN	NaN	NaN	1415.591133	810.705321	
Age	19043.0	NaN	NaN	NaN	34.668435	6.257912	2
Gender	19052.0	NaN	NaN	NaN	0.418749	0.493367	
City	19104	29	C20	1008	NaN	NaN	N
Education_Level	19104.0	NaN	NaN	NaN	1.021671	0.800167	
Income	19104.0	NaN	NaN	NaN	65652.025126	30914.515344	107
Dateofjoining	19104	869	23/07/15	192	NaN	NaN	N
LastWorkingDate	1616	493	29/07/20	70	NaN	NaN	N
Joining Designation	19104.0	NaN	NaN	NaN	1.690536	0.836984	
Grade	19104.0	NaN	NaN	NaN	2.25267	1.026512	
Total Business Value	19104.0	NaN	NaN	NaN	571662.074958	1128312.218461	-60000
Quarterly Rating	19104.0	NaN	NaN	NaN	2.008899	1.009832	

```
In [11]: # missing value percentage in each columns
         df.isna().sum()/df.shape[0]*100
```

```
Out[11]: 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
```

Observation till now:

- there are missing values and 91.5% most null values are present in column "LastWorkingDate" which is target variable here.
- **LastWorkingDate feature contains missing values which indicates the driver has not left the company yet.**
- some features showing wrong data types

In [12]:
```
### Converting features to respective data-types

df["MMM-YY"] = pd.to_datetime(df["MMM-YY"], format= "mixed")
df["Dateofjoining"] = pd.to_datetime(df["Dateofjoining"], format= "mixed")
df["LastWorkingDate"] = pd.to_datetime(df["LastWorkingDate"], format= "mixed")
```

In [13]:
```
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  datetime64[ns]
 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  datetime64[ns]
 8   LastWorkingDate       1616 non-null   datetime64[ns]
 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: datetime64[ns](3), float64(2), int64(7), object(1)
memory usage: 1.9+ MB
```

In [14]:
```
# getting numerical column name

num = df.select_dtypes(np.number)

num.columns
```

Out[14]:
```
Index(['Driver_ID', 'Age', 'Gender', 'Education_Level', 'Income',
       'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating'],
      dtype='object')
```

Missing Value treatment

KNN imputer

In [15]:
```
#KNN imputer
#Apply only to 'Age'
```

```python
imputer = KNNImputer(n_neighbors=5)
df['Age'] = imputer.fit_transform(df['Age'].values.reshape(-1,1))
```

- 2381 unique number of drivers so we can group the data

In [16]:
```python
#Fill gender null values backward fill

df['Gender'] = df['Gender'].bfill()
```

- "LastWorkingDate" feature contains missing values means driver has not left the company we can replace it with 0

In [17]:
```python
df['LastWorkingDate'].fillna(value=0, inplace=True)
df['LastWorkingDate'] = df['LastWorkingDate'].apply(lambda x: 0 if x == 0 else 1)
```

In [18]:
```python
df.isna().sum()
```

Out[18]:
```
MMM-YY                  0
Driver_ID               0
Age                     0
Gender                  0
City                    0
Education_Level         0
Income                  0
Dateofjoining           0
LastWorkingDate         0
Joining Designation     0
Grade                   0
Total Business Value    0
Quarterly Rating        0
dtype: int64
```

In [19]:
```python
df.describe().T
```

Out[19]:

	count	mean	min	25%	50%	75%	
MMM-YY	19104	2019-12-11 02:09:29.849246464	2019-01-01 00:00:00	2019-06-01 00:00:00	2019-12-01 00:00:00	2020-07-01 00:00:00	2 01
Driver_ID	19104.0	1415.591133	1.0	710.0	1417.0	2137.0	
Age	19104.0	34.668435	21.0	30.0	34.0	39.0	
Gender	19104.0	0.418656	0.0	0.0	0.0	1.0	
Education_Level	19104.0	1.021671	0.0	0.0	1.0	2.0	
Income	19104.0	65652.025126	10747.0	42383.0	60087.0	83969.0	1
Dateofjoining	19104	2018-04-28 20:52:54.874371840	2013-04-01 00:00:00	2016-11-29 12:00:00	2018-09-12 00:00:00	2019-11-05 00:00:00	2 28
LastWorkingDate	19104.0	0.08459	0.0	0.0	0.0	0.0	
Joining Designation	19104.0	1.690536	1.0	1.0	1.0	2.0	
Grade	19104.0	2.25267	1.0	1.0	2.0	3.0	
Total Business Value	19104.0	571662.074958	-6000000.0	0.0	250000.0	699700.0	337
Quarterly Rating	19104.0	2.008899	1.0	1.0	2.0	3.0	

- 75% of the drivers have age less than or equal to 39, minimum and maximum age of a driver 21 to 58
- As we can see in above there are not much diffrences between mean and median so we can say no or less outliers may present
- less than or equal to 50% of the drivers are earning around 60000rs per month on an average.

In [20]:
```python
df["Driver_ID"].nunique()   # 2381 Unique drivers
```

Out[20]: 2381

In [21]:
```python
df.loc[df["Driver_ID"]==56].shape
```

Out[21]: (24, 13)

Data is scattered in different rows we can aggregate the features accordingly

In [22]:
```python
agg_functions = {
    'MMM-YY':len,
    "Age": "max",
    "Gender": "first",
```

```
    "Education_Level": "max",
    "Income": "last",
    "Joining Designation": "last",
    "Grade": "last",
    "Total Business Value": "sum",
    "Quarterly Rating": "last",
    "LastWorkingDate": "last",
    "City": "first",
    "Dateofjoining": "last"
}

df1 = df.groupby("Driver_ID").aggregate(agg_functions).reset_index()
```

In [23]:
```
df1.shape
```

Out[23]:
```
(2381, 13)
```

In [24]:
```
df1.rename(columns={'MMM-YY': "No_of_records", }, inplace=True)
```

Creating a column which tells if the quarterly rating has increased for that employee

for those whose quarterly rating has increased assign the value 1

In [25]:
```
#Quarterly rating at the beginning
qrf = df.groupby('Driver_ID').agg({'Quarterly Rating':'first'})

#Quarterly rating at the end
qrl = df.groupby('Driver_ID').agg({'Quarterly Rating':'last'})

#The dataset which has the employee ids and a bollean value which tells if the rati
qr = (qrl['Quarterly Rating']>qrf['Quarterly Rating']).reset_index()

#the employee ids whose rating has increased
empid = qr[qr['Quarterly Rating']==True]['Driver_ID']

qri = []
for i in df1['Driver_ID']:
    if i in empid.values:
        qri.append(1)
    else:
        qri.append(0)

df1['Quarterly_Rating_Increased'] = qri
```

Creating a column which tells if the monthly income has increased for that employee

for those whose monthly income has increased we assign the value 1

In [26]:
```python
# Income at the beginning
sf = df.groupby('Driver_ID').agg({'Income':'first'})

# income at the end
sl = df.groupby('Driver_ID').agg({'Income':'last'})

#The dataset which has the employee ids and a bollean value which tells if the mont
s = (sl['Income']>sf['Income']).reset_index()

#the employee ids whose monthly income has increased
empid = s[s['Income']==True]['Driver_ID']

si = []
for i in df1['Driver_ID']:
    if i in empid.values:
        si.append(1)
    else:
        si.append(0)

df1['Income_Increased'] = si
```

In [27]:
```python
df1.head()
```

Out[27]:

	Driver_ID	No_of_records	Age	Gender	Education_Level	Income	Joining Designation	Grade
0	1	3	28.0	0.0	2	57387	1	1
1	2	2	31.0	0.0	2	67016	2	2
2	4	5	43.0	0.0	2	65603	2	2
3	5	3	29.0	0.0	0	46368	1	1
4	6	5	31.0	1.0	1	78728	3	3

In [28]:
```python
df1.describe().T
```

Out[28]:

	count	mean	min	25%	50%	
Driver_ID	2381.0	1397.559009	1.0	695.0	1400.0	2
No_of_records	2381.0	8.02352	1.0	3.0	5.0	
Age	2381.0	33.804322	21.0	30.0	33.0	
Gender	2381.0	0.410332	0.0	0.0	0.0	
Education_Level	2381.0	1.00756	0.0	0.0	1.0	
Income	2381.0	59334.157077	10747.0	39104.0	55315.0	75
Joining Designation	2381.0	1.820244	1.0	1.0	2.0	
Grade	2381.0	2.096598	1.0	1.0	2.0	
Total Business Value	2381.0	4586741.822764	-1385530.0	0.0	817680.0	4173
Quarterly Rating	2381.0	1.427971	1.0	1.0	1.0	
LastWorkingDate	2381.0	0.678706	0.0	0.0	1.0	
Dateofjoining	2381	2019-02-08 07:14:50.550189056	2013-04-01 00:00:00	2018-06-29 00:00:00	2019-07-21 00:00:00	202 00:
Quarterly_Rating_Increased	2381.0	0.150357	0.0	0.0	0.0	
Income_Increased	2381.0	0.01806	0.0	0.0	0.0	

- Max driver record is 23 while minimum is 1.
- 75% of the drivers have less than or equal to 10 number of records.
- 75% drivers have income less than or equal to 75986

In [29]:
```
## checking how many drivers income incresed
df1['Income_Increased'].value_counts()
```

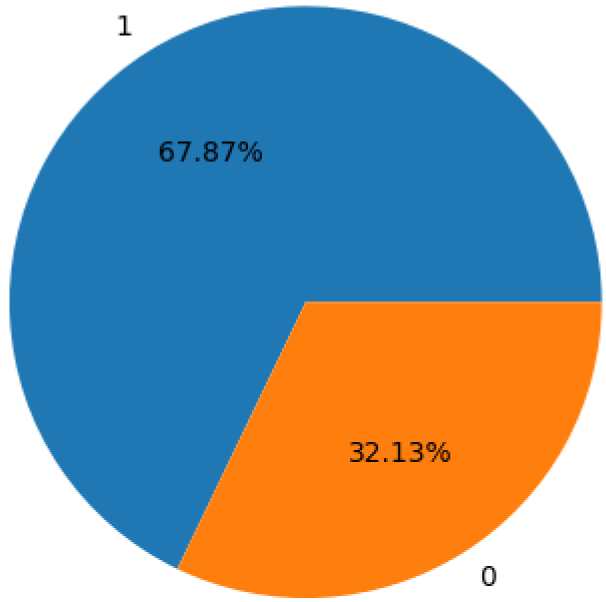
Out[29]:
```
Income_Increased
0    2338
1      43
Name: count, dtype: int64
```

- 43 drivers income increased

% of the drivers churn

In [30]:
```
chrn= df1["LastWorkingDate"].value_counts()
```

In [31]:
```
### percentage of churning of drivers

plt.pie(chrn, labels= chrn.index, autopct='%.2f%%' )
plt.show()
```



- 67.87 % drivers are churned

In [32]:
```
df1.describe(include="object")
```

Out[32]:

	City
count	2381
unique	29
top	C20
freq	152

- most of the drivers are from C20 city

In [33]:
```
cat_col= ['Gender','City','Education_Level','Joining Designation','Grade','Quarterl
# % of every cat in each feature
for i in cat_col:
    print(df1[i].value_counts(normalize=True)*100)
    print("-"*50)
```

```
Gender
0.0    58.966821
1.0    41.033179
Name: proportion, dtype: float64
--------------------------------------------------
City
C20    6.383872
C15    4.241915
C29    4.031919
C26    3.905922
C8     3.737925
C27    3.737925
C10    3.611928
C16    3.527929
C22    3.443931
C3     3.443931
C28    3.443931
C12    3.401932
C5     3.359933
C1     3.359933
C21    3.317934
C14    3.317934
C6     3.275934
C4     3.233935
C7     3.191936
C9     3.149937
C25    3.107938
C23    3.107938
C24    3.065939
C19    3.023940
C2     3.023940
C17    2.981940
C13    2.981940
C18    2.897942
C11    2.687946
Name: proportion, dtype: float64
--------------------------------------------------
Education_Level
2    33.683326
1    33.389332
0    32.927341
Name: proportion, dtype: float64
--------------------------------------------------
Joining Designation
1    43.091138
2    34.229315
3    20.705586
4     1.511970
5     0.461991
Name: proportion, dtype: float64
--------------------------------------------------
Grade
2    35.909282
1    31.121378
3    26.165477
4     5.795884
```

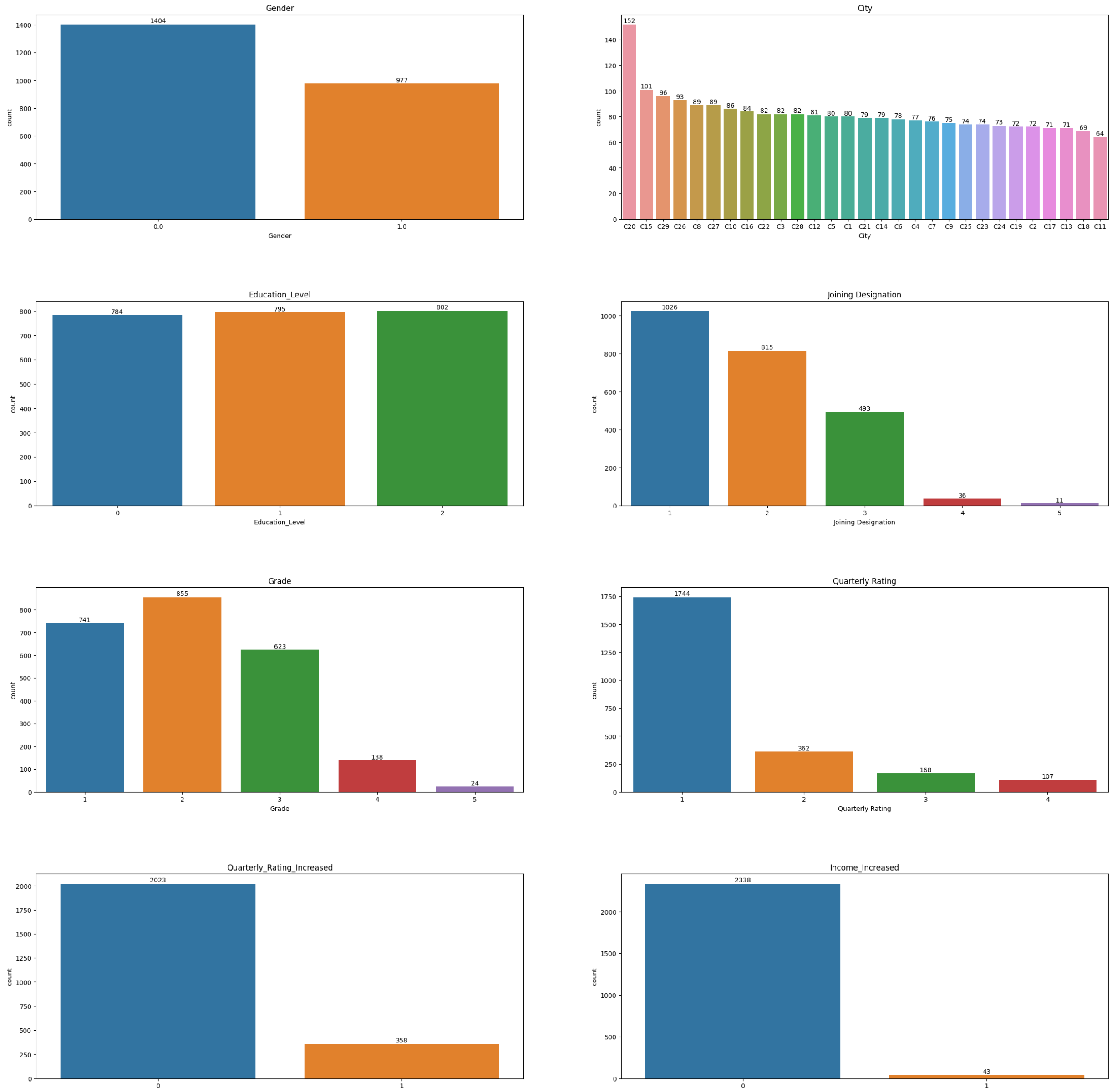
```
5     1.007980
Name: proportion, dtype: float64
--------------------------------------------------
Quarterly Rating
1    73.246535
2    15.203696
3     7.055859
4     4.493910
Name: proportion, dtype: float64
--------------------------------------------------
Quarterly_Rating_Increased
0    84.964301
1    15.035699
Name: proportion, dtype: float64
--------------------------------------------------
Income_Increased
0    98.194036
1     1.805964
Name: proportion, dtype: float64
--------------------------------------------------
```

- 58% drivers are male
- out of 29 cities 6% drivers are from C20 city
- 33.6% drivers are graduated and 33% are 12th pass
- 43% drivers joined company as 1 designation
- 84% drivers quarterly rating not increased
- 98% drivers income not increased

In [34]:
```
cat_col
```

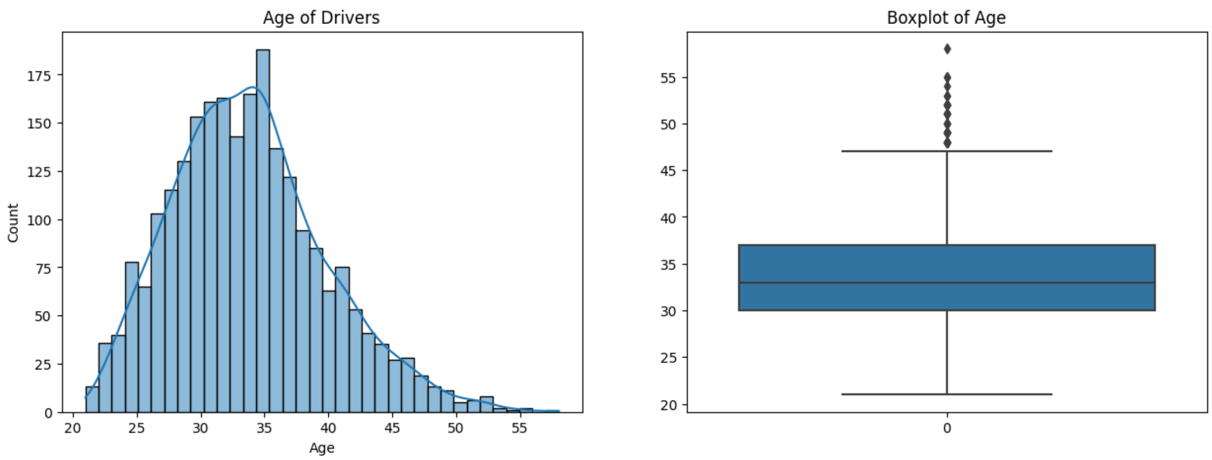
Out[34]:
```
['Gender',
 'City',
 'Education_Level',
 'Joining Designation',
 'Grade',
 'Quarterly Rating',
 'Quarterly_Rating_Increased',
 'Income_Increased']
```

In [35]:
```
plt.figure(figsize= (30,30))
# plt.subplots_adjust(wspace= 0.3)
plt.subplots_adjust(hspace= 0.4)
for i in range(len(cat_col)):
    plt.subplot(4,2,i+1)
    lb= sns.barplot(x= df1[cat_col[i]].value_counts().index, y=df1[cat_col[i]].valu
    plt.title(f"{cat_col[i]}")
    for i in lb.containers:
        lb.bar_label(i)
```



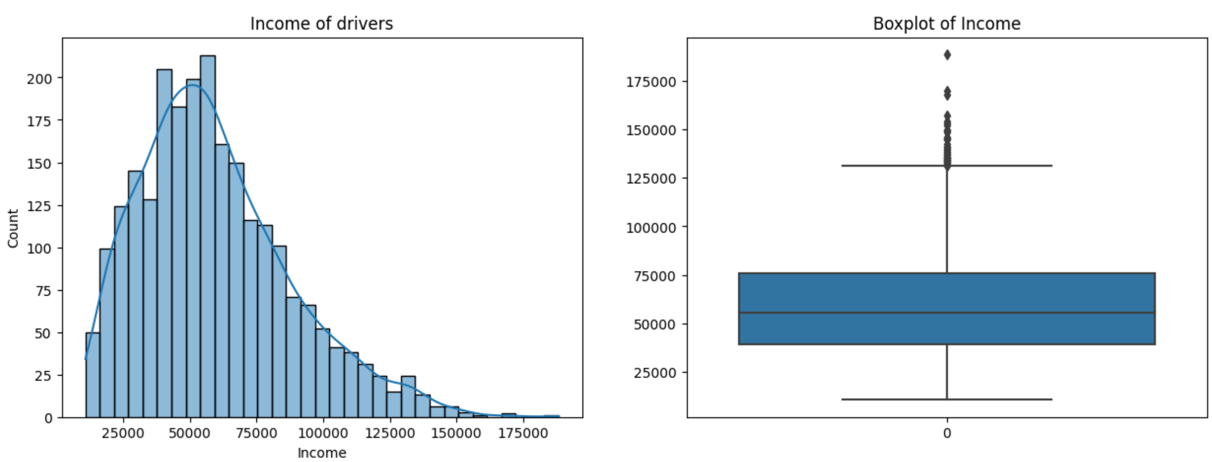
In [36]:

```
plt.subplots(figsize=(15,5))
plt.subplot(1,2,1)
sns.histplot(df1['Age'],kde= True)
plt.title("Age of Drivers")

plt.subplot(1,2,2)
sns.boxplot(df1['Age'])
plt.title('Boxplot of Age')

plt.show()
```



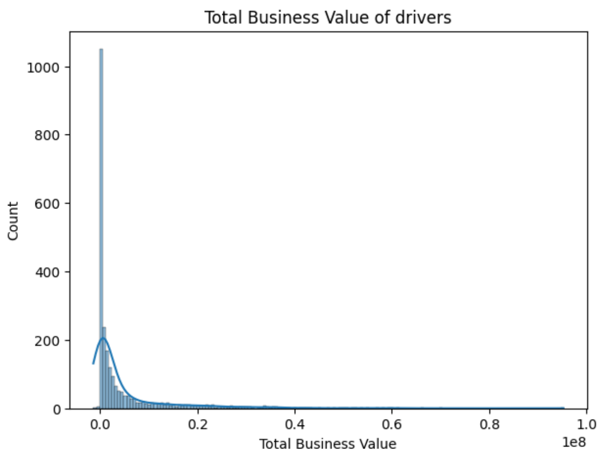
- **There are few outliers, The distribution is slightly towards right**

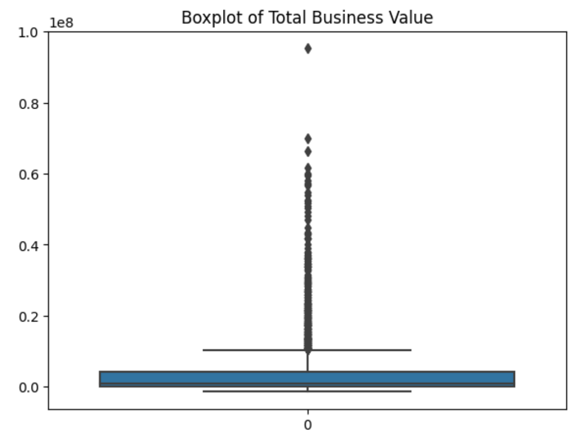
In [37]:
```
plt.subplots(figsize=(15,5))
plt.subplot(1,2,1)
sns.histplot(df1['Income'],kde= True)
plt.title("Income of drivers")

plt.subplot(1,2,2)
sns.boxplot(df1['Income'])
plt.title('Boxplot of Income')

plt.show()
```



- **There are outliers in Income feature and distribution is right skewed**

In [38]:
```
plt.subplots(figsize=(15,5))
plt.subplot(1,2,1)
sns.histplot(df1['Total Business Value'],kde= True)
plt.title("Total Business Value of drivers")

plt.subplot(1,2,2)
sns.boxplot(df1['Total Business Value'])
plt.title('Boxplot of Total Business Value')
plt.show()
```





- **The distribution of total business value is towards the right. There are a lot of outliers for the feature Total Business Value.**

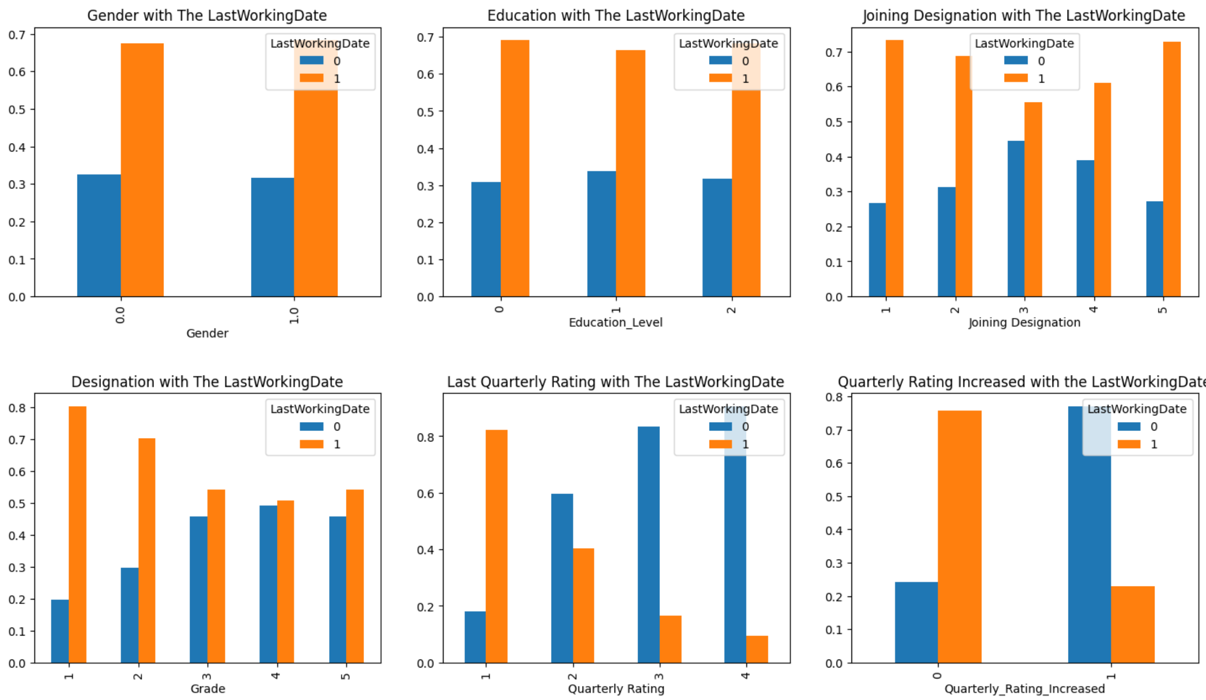
In [39]:
```
gender= pd.crosstab(df1['Gender'],df1['LastWorkingDate'])
gender
```

Out[39]:

LastWorkingDate	0	1
Gender		
0.0	456	948
1.0	309	668

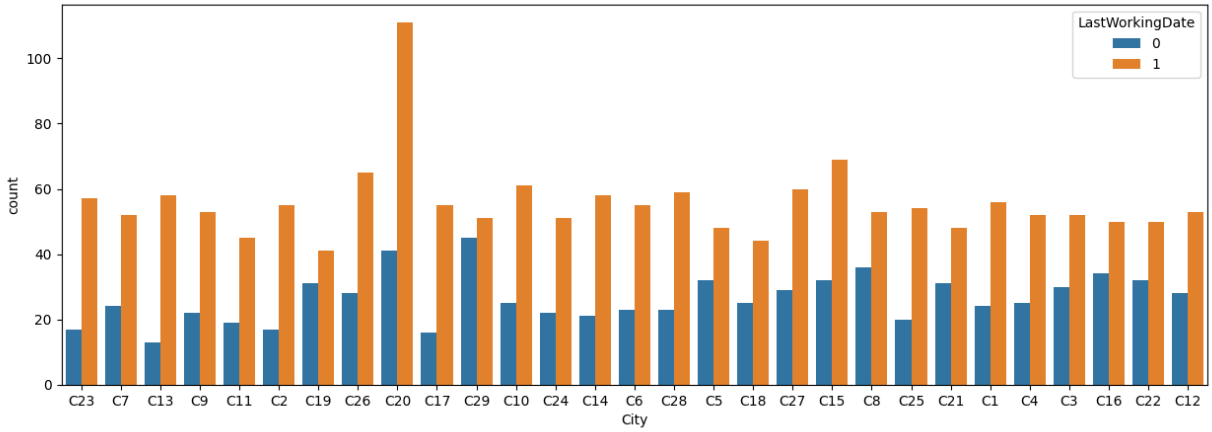
In [40]:
```
figure,axes=plt.subplots(2,3,figsize=(15,9))

#Gender feature with LastWorkingDate
gender = pd.crosstab(df1['Gender'],df1['LastWorkingDate'])
gender.div(gender.sum(1).astype(float),axis=0).plot(kind='bar',stacked=False,ax=axe

#Education feature with LastWorkingDate
education = pd.crosstab(df1['Education_Level'],df1['LastWorkingDate'])
education.div(education.sum(1).astype(float),axis=0).plot(kind='bar',stacked=False,
                                                          title="Education with The

#Joining Designation feature with LastWorkingDate
jde = pd.crosstab(df1['Joining Designation'],df1['LastWorkingDate'])
jde.div(jde.sum(1).astype(float),axis=0).plot(kind='bar',stacked=False,ax=axes[0,2]
                                                      title="Joining Designation wi

#Designation feature with Target
desig = pd.crosstab(df1['Grade'],df1['LastWorkingDate'])
desig.div(desig.sum(1).astype(float),axis=0).plot(kind='bar',stacked=False,ax=axes[
                                                     title="Designation with The Las

#Last Quarterly Rating feature with LastWorkingDate
lqrate = pd.crosstab(df1['Quarterly Rating'],df1['LastWorkingDate'])
lqrate.div(lqrate.sum(1).astype(float),axis=0).plot(kind='bar',stacked=False,ax=axe
                                                      title="Last Quarterly Rating
```

```
#Quarterly Rating Increased feature with LastWorkingDate
qratei = pd.crosstab(df1['Quarterly_Rating_Increased'],df1['LastWorkingDate'])
qratei.div(qratei.sum(1).astype(float),axis=0).plot(kind='bar',stacked=False,ax=axe
                                                      title="Quarterly Rating In

plt.tight_layout(pad=3)
```

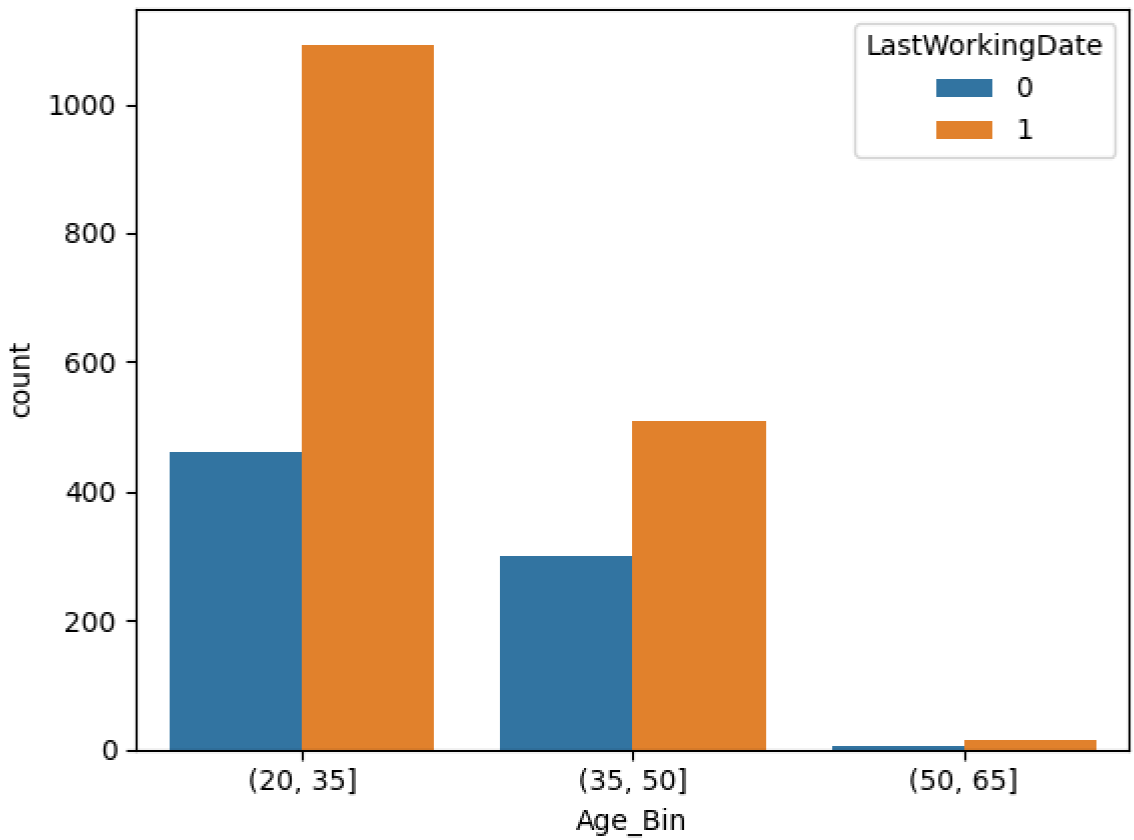


- Both Gender have same proportion of leaving and not leaving the company, same goes for Education level.
- Joining designation as 3 and 4 are less likely leave the company.
- The employees who have their grade as 3 or 4 at the time of reporting are less likely to leave the organization.
- The employees who have their last quarterly rating as 3 or 4 at the time of reporting are less likely to leave the organization.
- The employees whose quarterly rating has increased are less likely to leave the organization.

In [41]:
```
plt.figure(figsize=(15,5))
sns.countplot(data=df1, x="City", hue="LastWorkingDate")
plt.show()
```

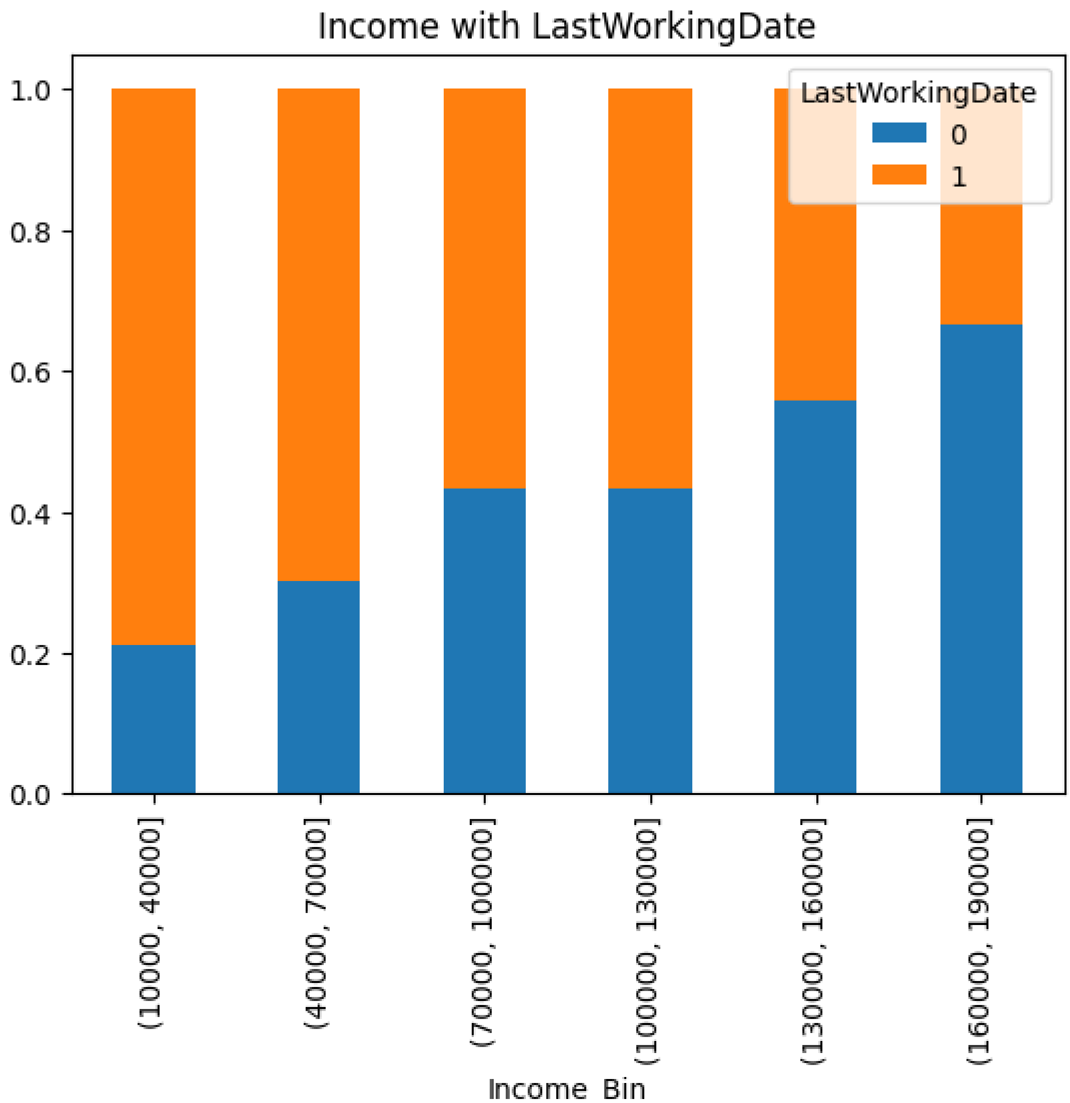


In [42]:
```python
#Binning the Age into categories
df1['Age_Bin'] = pd.cut(df1['Age'],bins=[20,35,50,65])

#Age feature with LastWorkingDate
sns.countplot(data=df1, x= "Age_Bin", hue="LastWorkingDate")
plt.show()
```



In [43]:
```python
#Binning the Income into categories
df1['Income_Bin'] = pd.cut(df1['Income'],bins=[10000, 40000, 70000, 100000, 130000,

#Salary feature with Target
salarybin = pd.crosstab(df1['Income_Bin'],df1['LastWorkingDate'])
salarybin.div(salarybin.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,t
```

Out[43]:
```
<Axes: title={'center': 'Income with LastWorkingDate'}, xlabel='Income_Bin'>
```



- **The employees whose monthly income is in 1,60,000-1,90,000 or 1,30,000-1,60,000 are less likely to leave the organization.**

In [44]:
```
#Dropping the bins columns and date_column
df1.drop(['Age_Bin','Income_Bin','Dateofjoining'],axis=1,inplace=True)
```

In [45]:
```
df1.head()
```

Out[45]:

	Driver_ID	No_of_records	Age	Gender	Education_Level	Income	Joining Designation	Grade
0	1	3	28.0	0.0	2	57387	1	1
1	2	2	31.0	0.0	2	67016	2	2
2	4	5	43.0	0.0	2	65603	2	2
3	5	3	29.0	0.0	0	46368	1	1
4	6	5	31.0	1.0	1	78728	3	3

In [46]:

```
plt.figure(figsize=(15, 15))
sns.heatmap(df1.corr(numeric_only=True) ,annot=True, cmap="RdYlGn")
```

Out[46]: <Axes: >

One Hot Encoding

In [47]:
```
# converting categorical features to numerical values
```

In [48]:
```
df1 = pd.concat([df1,pd.get_dummies(df1['City'],prefix='City')],axis=1)
```

In [49]:
```
df1.head()
```

Out[49]:

	Driver_ID	No_of_records	Age	Gender	Education_Level	Income	Joining Designation	Grade
0	1	3	28.0	0.0	2	57387	1	1
1	2	2	31.0	0.0	2	67016	2	2
2	4	5	43.0	0.0	2	65603	2	2
3	5	3	29.0	0.0	0	46368	1	1
4	6	5	31.0	1.0	1	78728	3	3

5 rows × 43 columns

In [50]:
```
## Independant variable
X = df1.drop(['Driver_ID','LastWorkingDate','City'],axis=1)
```

In [51]:
```
# Target Variable
y= df1["LastWorkingDate"]
```

In [52]:
```
# split into 80:20 ration
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_
```

In [53]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(1904, 40)
(477, 40)
```

Scaling the data

In [54]:
```
df1.columns
```

Out[54]:
```
Index(['Driver_ID', 'No_of_records', 'Age', 'Gender', 'Education_Level',
       'Income', 'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating', 'LastWorkingDate', 'City',
       'Quarterly_Rating_Increased', 'Income_Increased', 'City_C1', 'City_C10',
       'City_C11', 'City_C12', 'City_C13', 'City_C14', 'City_C15', 'City_C16',
       'City_C17', 'City_C18', 'City_C19', 'City_C2', 'City_C20', 'City_C21',
       'City_C22', 'City_C23', 'City_C24', 'City_C25', 'City_C26', 'City_C27',
       'City_C28', 'City_C29', 'City_C3', 'City_C4', 'City_C5', 'City_C6',
       'City_C7', 'City_C8', 'City_C9'],
      dtype='object')
```

In [55]:
```
#Feature Variables
X_cols=X_train.columns
# MinMaxScaler
scaler = MinMaxScaler()

#Mathematically learning the distribution
X_train=scaler.fit_transform(X_train)

X_test= scaler.fit_transform(X_test)
```

In [56]:
```
X_train=pd.DataFrame(X_train)
X_test= pd.DataFrame(X_test)
```

In [57]:
```
X_train.columns=X_cols
X_test.columns=X_cols
```

In [58]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(1904, 40)
(477, 40)
```

In [59]:
```
X_train.head()
```

Out[59]:

	No_of_records	Age	Gender	Education_Level	Income	Joining Designation	Grade	To Busin Val
0	0.043478	0.567568	0.0	0.5	0.268403	0.50	0.50	0.0052
1	0.391304	0.162162	1.0	1.0	0.098206	0.00	0.00	0.0605
2	1.000000	0.540541	1.0	0.5	0.266562	0.00	0.25	0.1752
3	0.173913	0.162162	0.0	0.0	0.348979	0.25	0.25	0.0052
4	0.086957	0.216216	0.0	0.0	0.266956	0.00	0.00	0.0072

5 rows × 40 columns

Balancing the data using oversampling SMOTE

In [60]:
```
sm=SMOTE(random_state=42)
```

```
X_train_baln, y_train_baln = sm.fit_resample(X_train,y_train.ravel())

print(f"Before OverSampling, count of label 1: {sum(y_train == 1)}")
print(f"Before OverSampling, count of label 0: {sum(y_train == 0)}")
print(f"After OverSampling, count of label 1: {sum(y_train_baln == 1)}")
print(f"After OverSampling, count of label 0: {sum(y_train_baln == 0)}")
```

```
Before OverSampling, count of label 1: 1287
Before OverSampling, count of label 0: 617
After OverSampling, count of label 1: 1287
After OverSampling, count of label 0: 1287
```

Random Forest Classifier

In [61]:
```
# Defining parameters -

params = {
          'n_estimators' : [50,100,150,200,250],
          'max_depth' : [2,3,5, 7, 9],
          'criterion' : ['gini', 'entropy']
         }
```

In [62]:
```
random_forest = RandomForestClassifier()

grid= GridSearchCV(estimator = random_forest,
                    param_grid = params,
                    scoring = 'f1',
                    cv = 3,
                    n_jobs=-1)
```

In [63]:
```
import datetime as dt
```

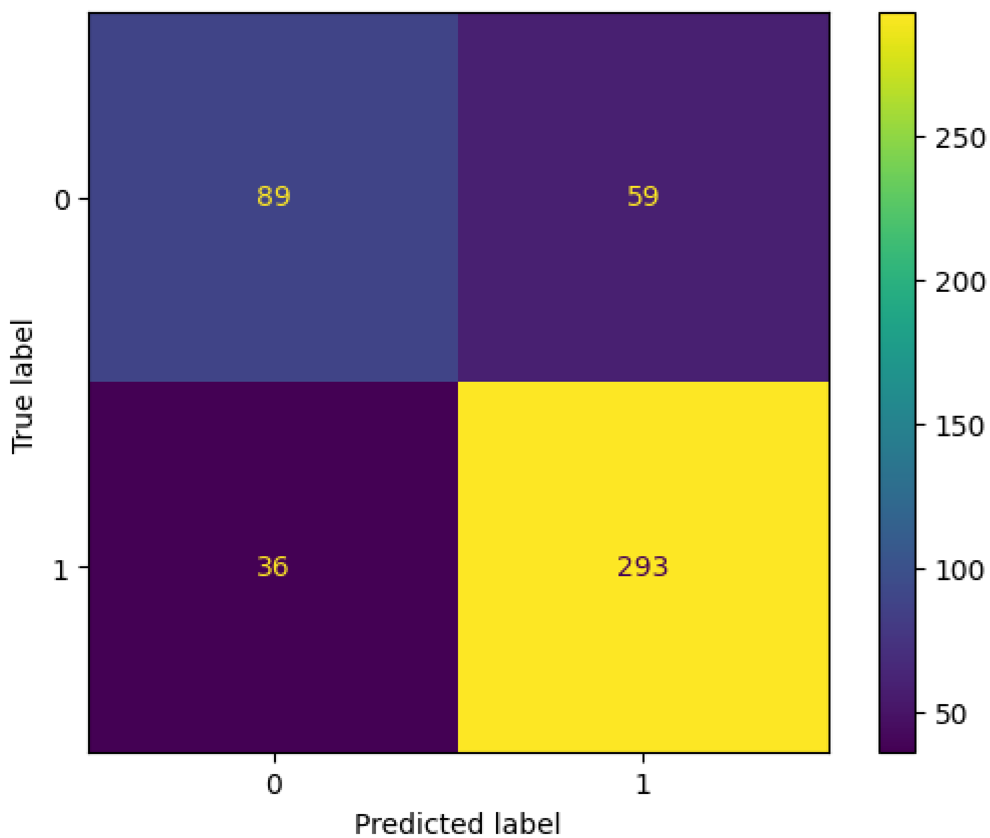
In [64]:
```
#training model
start = dt.datetime.now()

grid.fit(X_train_baln, y_train_baln)

print("Best params: ", grid.best_params_)
print("Best score: ", grid.best_score_)

end = dt.datetime.now()
print("time taken to train the model:", end- start)
```

```
Best params:  {'criterion': 'entropy', 'max_depth': 9, 'n_estimators': 250}
Best score:  0.8611974931157061
time taken to train the model: 0:00:18.953436
```

In [65]:
```
pred = grid.predict(X_test)
print(classification_report(y_test,pred))
print(confusion_matrix(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.71      0.60      0.65       148
           1       0.83      0.89      0.86       329

    accuracy                           0.80       477
   macro avg       0.77      0.75      0.76       477
weighted avg       0.80      0.80      0.80       477

[[ 89  59]
 [ 36 293]]
```

In [66]:
```
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay(confusion_matrix(y_test,pred)).plot()

plt.show()
```



- The Random Forest method out of all predicted 0 the measure of correctly predicted is 72%, and for 1 it is 84% (Precision).
- The Random Forest method out of all actual 0 the measure of correctly predicted is 66%, and for 1 it is 89% (Recall).
- Random forest model score is 0.86

Feature Importance for the best model using Random forest

In [67]:
```
# training random forest model with best parameter
```

```
best_model= RandomForestClassifier(criterion= 'gini', max_depth= 9, n_estimators= 1
```

In [68]:

```
start = dt.datetime.now()
best_model.fit(X_train_baln, y_train_baln)

end = dt.datetime.now()
print("time taken to train the model:", end- start)
```

```
time taken to train the model: 0:00:00.589972
```

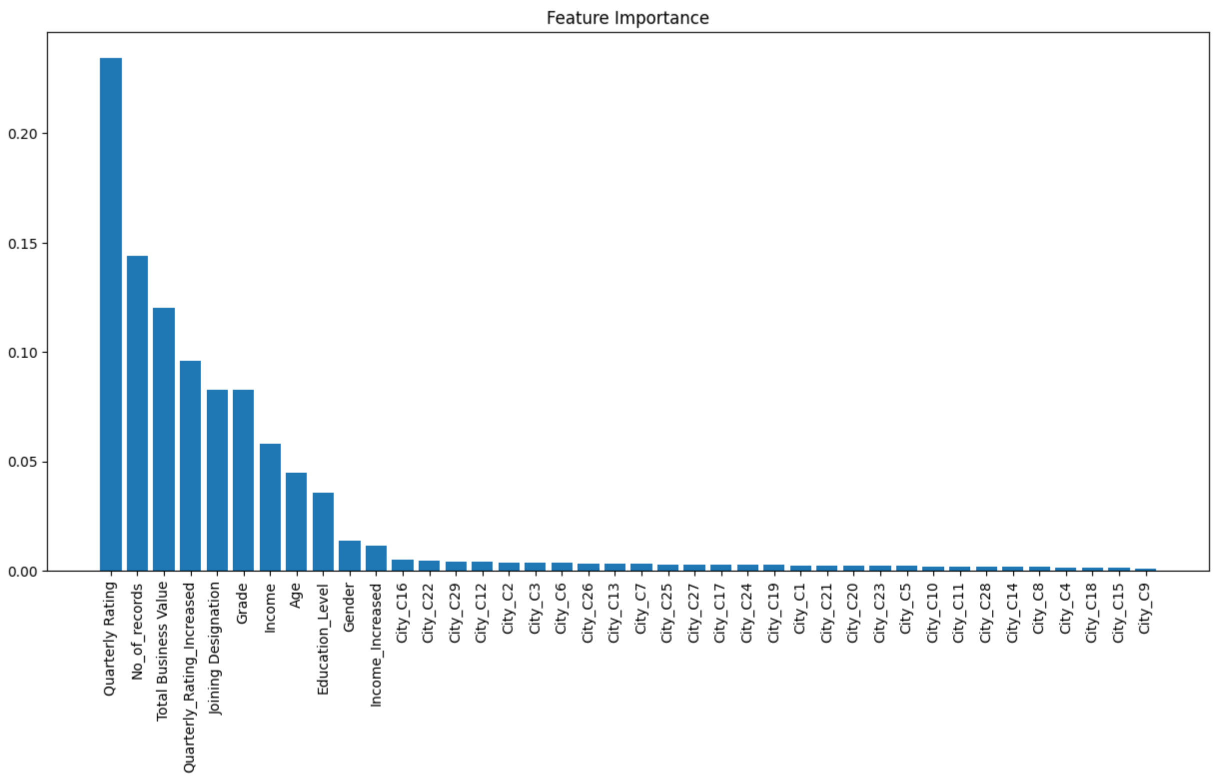
In [69]:

```
pred = best_model.predict(X_test)
print(classification_report(y_test,pred))
print(confusion_matrix(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.69      0.59      0.64       148
           1       0.83      0.88      0.85       329

    accuracy                           0.79       477
   macro avg       0.76      0.74      0.75       477
weighted avg       0.78      0.79      0.79       477

[[ 88  60]
 [ 40 289]]
```

In [70]:

```
# Feature Importance

import matplotlib.pyplot as plt

importances = best_model.feature_importances_

indices = np.argsort(importances)[::-1] # Sort feature importances in descending or
names = [X.columns[i] for i in indices] # Rearrange feature names so they match the

plt.figure(figsize=(15, 7)) # Create plot
plt.title("Feature Importance") # Create plot title
plt.bar(range(X.shape[1]), importances[indices]) # Add bars
plt.xticks(range(X.shape[1]), names, rotation=90) # Add feature names as x-axis lab
plt.show() # Show plot
```



- Most important feature is Quaterly Rating, No_of_records, Total Business Value

XGBoost Classifier

In [71]:
```
from xgboost import XGBClassifier
```

In [72]:
```
xgb= XGBClassifier(objective='multi:softmax', num_class= 2)

params = {
        "n_estimators": [50,100,150,200, 250],
        "max_depth" : [3, 4, 5, 7, 8,9],
        'criterion' : ['gini', 'entropy']
        }
```

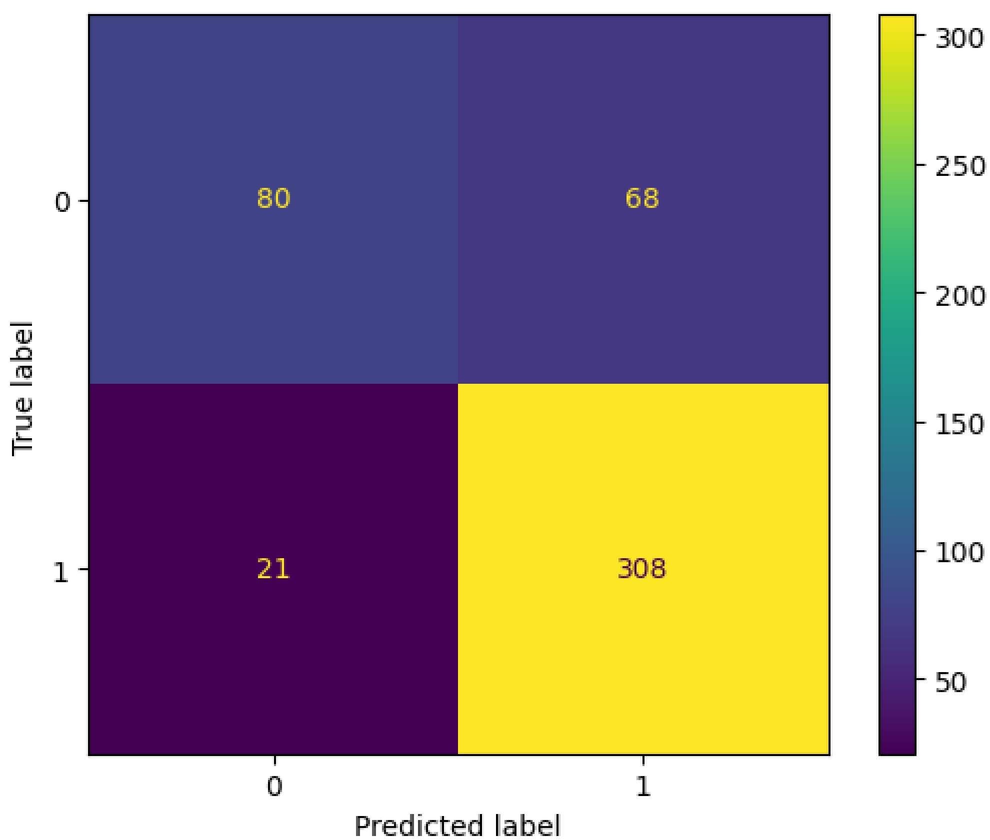
In [73]:
```
xgb_grid_model=  GridSearchCV(estimator = xgb,
                    param_grid = params,
                    scoring = 'f1',
                    cv = 3,
                    n_jobs=-1)

start = dt.datetime.now()
xgb_grid_model.fit(X_train, y_train)
end = dt.datetime.now()

print("Best params: ", xgb_grid_model.best_params_)
print("Best score: ", xgb_grid_model.best_score_)
print("time taken to train:", end-start)
```

```
Best params:  {'criterion': 'gini', 'max_depth': 3, 'n_estimators': 50}
Best score:  0.8822798886700521
time taken to train: 0:00:11.997788
```

In [74]:
```
xgb_best= xgb_grid_model.best_estimator_

pred = xgb_best.predict(X_test)
print(classification_report(y_test,pred))
print(confusion_matrix(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.54      0.64       148
           1       0.82      0.94      0.87       329

    accuracy                           0.81       477
   macro avg       0.81      0.74      0.76       477
weighted avg       0.81      0.81      0.80       477

[[ 80  68]
 [ 21 308]]
```

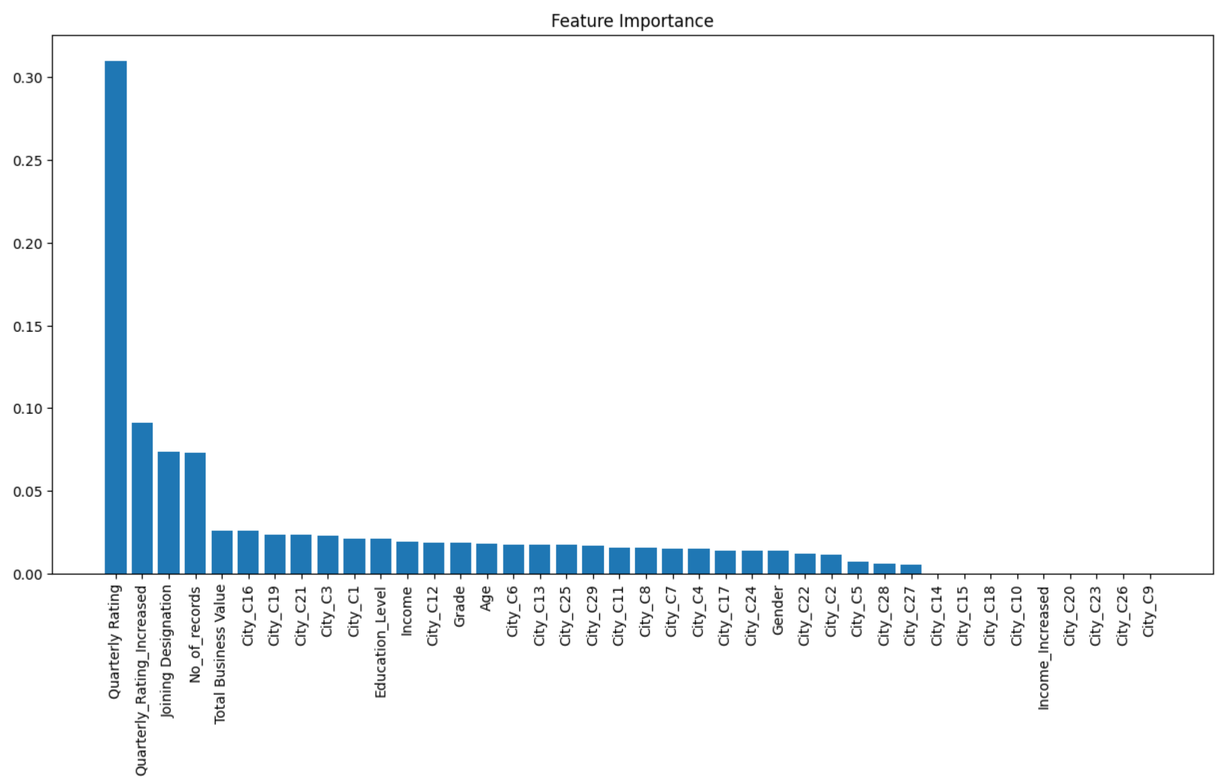
In [75]:
```
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay(confusion_matrix(y_test,pred)).plot()

plt.show()
```



- The XGBoost method out of all predicted 0 the measure of correctly predicted is 79%, and for 1 it is 82% (Precision).

- The XGBoost method out of all actual 0 the measure of correctly predicted is 54%, and for 1 it is 94% (Recall).
- XGBoost model best score is 0.88

In [76]:
```
# Feature Importance

import matplotlib.pyplot as plt

importances = xgb_best.feature_importances_

indices = np.argsort(importances)[::-1] # Sort feature importances in descending or
names = [X.columns[i] for i in indices] # Rearrange feature names so they match the

plt.figure(figsize=(15, 7)) # Create plot
plt.title("Feature Importance") # Create plot title
plt.bar(range(X.shape[1]), importances[indices]) # Add bars
plt.xticks(range(X.shape[1]), names, rotation=90) # Add feature names as x-axis lab
plt.show() # Show plot
```

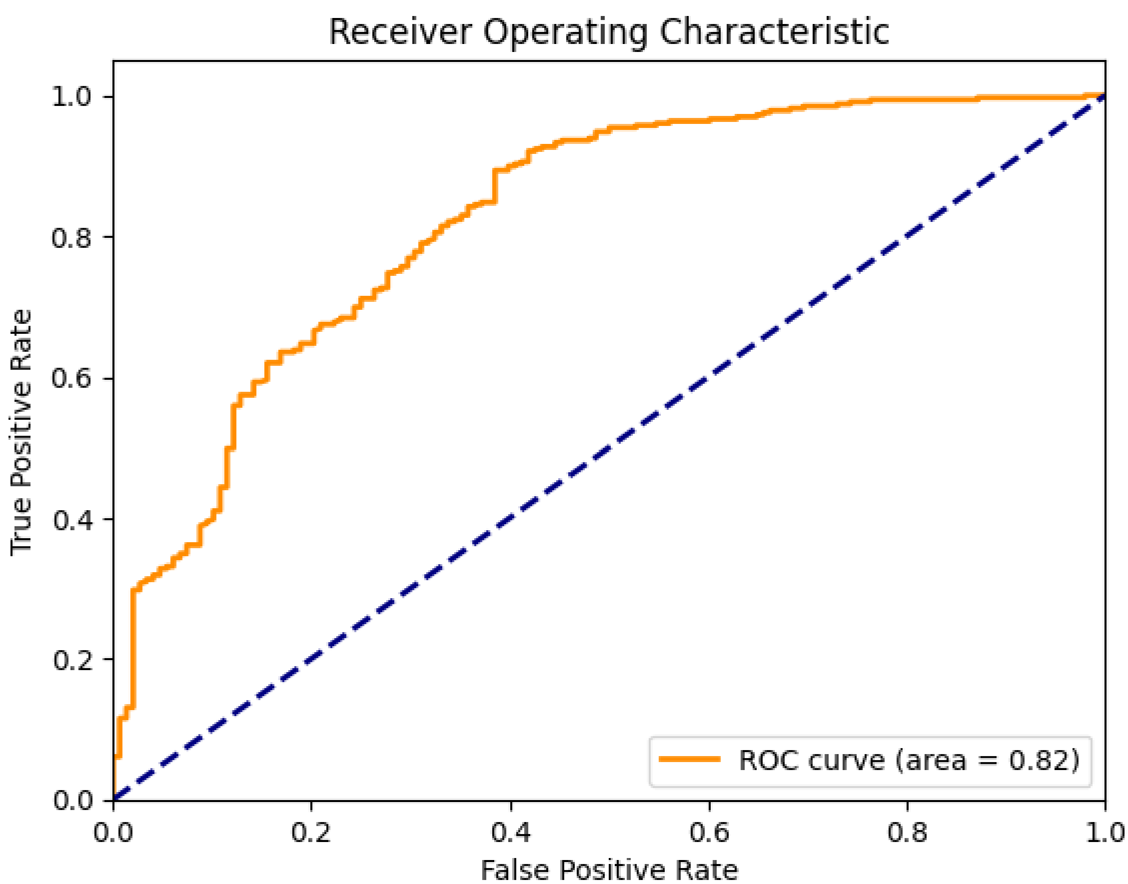


- Most important features are Quarterly rating, Quarterly rating increased, joining designation and number of records

ROC Curve & AUC

In [77]:
```
# Predict probabilities in test data
probs = xgb_best.predict_proba(X_test)[:,1]
```

```
# Computing the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, probs)

# Compute the area under the ROC curve
logit_roc_auc = roc_auc_score(y_test, probs)

# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % log
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



- we got AUC value 0.82 which is good that means model performance is good

Insights & Recommendation

- In data there are 58.96% of male drivers and 41.03% are female
- C20 have highest driver percentage (6.38%), followed by cities C15(4.24), C29(4.03%)
- Max driver record is 23 while minimum is 1.

- 75% of the drivers have less than or equal to 10 number of records.
- 75% drivers have income less than or equal to 75986
- 75% of the drivers have age less than or equal to 39, minimum and maximum age of a driver 21 to 58
- As we can see in above there are not much differences between mean and median so we can say no or less outliers may present
- less than or equal to 50% of the drivers are earning around 60000 rs per month on an average.
- 33.6% drivers are graduated and 33% are 12th pass
- 43% drivers joined company as 1 designation
- 84% drivers quarterly rating not increased
- 98% drivers income not
- Both Gender have same proportion of leaving and not leaving the company, same goes for Education level.
- Joining designation as 3 and 4 are less likely leave the company.
- The employees who have their grade as 3 or 4 at the time of reporting are less likely to leave the organization.
- The employees who have their last quarterly rating as 3 or 4 at the time of reporting are less likely to leave the organization.
- The employees whose quarterly rating has increased are less likely to leave the organization.
- In both model we can see quarterly rating most important feature
- Quarterly_Rating, Total_Business_Value & Quarterly_Rating_Increased, no. of records are the most important features. Company needs to tracks these features as predictors.
- Company should incourage the customers to rate the drivers genuinely good or bad on given scale
- XGBoost classifier doing good job in predicting having F1 score 0.88

In [ ]:

```

```