```
# Problem Statement

# Recruiting and retaining drivers is seen by industry watchers as a tough battle for Ola. Ch

# As the companies get bigger, the high churn could become a bigger problem. To find new driv

# You are working as a data scientist with the Analytics Department of Ola, focused on driver

# Demographics (city, age, gender etc.)
# Tenure information (joining date, Last Date)
# Historical data regarding the performance of the driver (Quarterly rating, Monthly business
# Dataset:

# Dataset Link: ola_driver.csv




# 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
# 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
# What "good" looks like:

# Import the dataset and do usual exploratory analysis steps like checking the structure & ch

# Convert date-like features to their respective data type

# Check for missing values and Prepare data for KNN Imputation

# You may consider only numerical features for this purpose

# Aggregate data in order to remove multiple occurrences of same driver data (We did somethin
```

```
# You can start from storing unique Driver IDs in an empty dataframe and then bring all the f

# Feature Engineering Steps:

# Create a column which tells whether the quarterly rating has increased for that driver - fo

# Target variable creation: Create a column called target which tells whether the driver has

# Create a column which tells whether the monthly income has increased for that driver - for

# Statistical summary of the derived dataset

# Check correlation among independent variables and how they interact with each other

# One hot encoding of the categorical variable

# Class Imbalance Treatment

# Standardization of training data

# Using Ensemble learning - Bagging, Boosting methods with some hyper-parameter tuning
# Results Evaluation:

# Classification Report

# ROC AUC curve

# Provide actionable Insights & Recommendations


# Evaluation Criteria (100 Points):

# Define Problem Statement and perform Exploratory Data Analysis (10 points)
# Definition of problem (as per given problem statement with additional views)
# Observations on shape of data, data types of all the attributes, conversion of categorical
# Univariate Analysis (distribution plots of all the continuous variable(s) barplots/countplo
# Bivariate Analysis (Relationships between important variables)
# Illustrate the insights based on EDA
# Comments on range of attributes, outliers of various attributes
# Comments on the distribution of the variables and relationship between them
# Comments for each univariate and bivariate plots
# Data Preprocessing (50 Points)
# KNN Imputation
# Feature Engineering
# Class Imbalance treatment
# Standardization
# Encoding
# Model building (20 Points)
# 1 Ensemble - Bagging Algorithm
# 1 Ensemble - Boosting Algorithm
# Results Evaluation (10 Points)
```

```
# ROC AUC Curve & comments
# Classification Report (Confusion Matrix etc)
# Actionable Insights & Recommendations (10 Points)
```

```
# Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report,confusion_matrix,ConfusionMatrixDisplay,pre
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn import tree
from sklearn.model_selection import RandomizedSearchCV
```

```
df = pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/002/492/orig
```

```
df.head()
```

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

```
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
```

```
df.shape
```

```
(19104, 14)
```

Columns

```
df.columns
```

```
Index(['Unnamed: 0', 'MMM-YY', 'Driver_ID', 'Age', 'Gender', 'City',
       'Education_Level', 'Income', 'Dateofjoining', 'LastWorkingDate',
       'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating'],
      dtype='object')
```

After Exploring Dataset, We can observe that there are total 19104 rows with 14 columns.

1. MMMM-YY : Reporting Date (Monthly)
2. Driver_ID : Unique id for drivers
3. Age : Age of the driver
4. Gender : Gender of the driver – Male : 0, Female: 1
5. City : City Code of the driver
6. Education_Level : Education level – 0 for 10+ ,1 for 12+ ,2 for graduate
7. Income : Monthly average Income of the driver
8. Date Of Joining : Joining date for the driver
9. LastWorkingDate : Last date of working for the driver. This feature contains some null values or the driver has not resigned yet.
10. Joining Designation : Designation of the driver at the time of joining
11. Grade : Grade of the driver at the time of reporting
12. Total Business Value : The total business value acquired by the driver in a month (negative business indicates cancellation/refund or car EMI adjustments)
13. Quarterly Rating : Quarterly rating of the driver: 1,2,3,4,5 (higher is better)

Conclusions on the Basis of Stats

```
df.describe()
```

	Unnamed: 0	Driver_ID	Age	Gender	Education_Level	Inc
count	19104.000000	19104.000000	19043.000000	19052.000000	19104.000000	19104.000
mean	9551.500000	1415.591133	34.668435	0.418749	1.021671	65652.025
std	5514.994107	810.705321	6.257912	0.493367	0.800167	30914.515
min	0.000000	1.000000	21.000000	0.000000	0.000000	10747.000
25%	4775.750000	710.000000	30.000000	0.000000	0.000000	42383.000
50%	9551.500000	1417.000000	34.000000	0.000000	1.000000	60087.000
75%	14327.250000	2137.000000	39.000000	1.000000	2.000000	83969.000
max	19103.000000	2788.000000	58.000000	1.000000	2.000000	188418.000

1. Unnamed: 0 column is just a index column so need to drop it.
2. Driver_ID: As we can observe from the data, that the data has been observed over a period of time, so the Driver_ID is definetly going to be get duplicated.
3. Age: From Age we can observe that, the minimum age of driver is around 21 years and the maximum age is 58. Again the data is duplicated so the correct mean cannot be revealed.
4. Gender: Again Gender is a categorical feature, so is there any dominance by other gender, by mean we can say that Male drivers are more as compared to Female Drivers.
5. Education_Level: Again this feature is Categorical Feature consisting of 3 values, 0,1 and 2 with respect to Education Level. And we can see that mean is around 1, so most of the drivers education level is till 12th.
6. Income: It will be varied with different drivers and depending upon there Designation, so in further we will compute the dependency of this columns. For now we can see that the mean of the Income is near to 65K. And also we can observe there is much variation in the data.
7. Joining Designation : As there is different levels of Designations available, and mostly the company gives 1 Designation to its driver, dependency again on the age and Experience. Need to see the Dependency of Age, Income and Gender on this feature as there are very few IDs having high Designation.
8. Grade: Again similar feature, depending on the Back Experience of any Driver.
9. Total Business Value: May be one of important feature as it will depend on the behaviour of Drivers.

10. Quaterly Rating: Again Important features which will help to understand whether the Driver can be promoted or there behaviour towards customers.

Checking Null Values

```
df.isnull().sum()
```

```
Unnamed: 0                0
MMM-YY                    0
Driver_ID                 0
Age                      61
Gender                   52
City                      0
Education_Level           0
Income                    0
Dateofjoining             0
LastWorkingDate       17488
Joining Designation       0
Grade                     0
Total Business Value      0
Quarterly Rating          0
dtype: int64
```

Seems like there are null Values present in Age, Gender and Last_WorkingDate Features, needed Feature Engineering on these Features.

Checking DataTypes:

```
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
```

There fixing of columns is needed in Date Columns and rest features seems to be good in there respective Datatypes, only needed to check for City column.

Feature Engineering

Converting to Date-Time Datatype

```
df['Dateofjoining'] = pd.to_datetime(df['Dateofjoining'])
```

```
df['LastWorkingDate'] = pd.to_datetime(df['LastWorkingDate'])
```

```
df['MMM-YY'] = pd.to_datetime(df['MMM-YY'])
```

```
df['Dateofjoining'].nunique()
```

```
869
```

For Checking Date Difference in Joining and Leaving:

```
df['Date_Difference'] = df['LastWorkingDate'] - df['Dateofjoining']
```

```
df.head()
```

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining
0	0	2019-01-01	1	28.0	0.0	C23	2	57387	2018-12-24

```
df['City'].value_counts()
```

```
C20    1008
C29     900
C26     869
C22     809
C27     786
C15     761
C10     744
C12     727
C8      712
C16     709
C28     683
C1      677
C6      660
C5      656
C14     648
C3      637
C24     614
C7      609
C21     603
C25     584
C19     579
C4      578
C13     569
C18     544
C23     538
C9      520
C2      472
C11     468
C17     440
Name: City, dtype: int64
```

```
df['Gender'].value_counts()
```

```
0.0    11074
1.0     7978
Name: Gender, dtype: int64
```

```
new_df = df.groupby(['Driver_ID'])['Age','Gender','Education_Level','Income','Joining Designa
       'Quarterly Rating', 'Date_Difference'].agg({'Age':'median','Gender':'median','Educatio
                                            'Grade':'median','Total Business Value':'s
```

```
<ipython-input-22-35aafc255fda>:1: FutureWarning: Indexing with multiple keys (implicit
  new_df = df.groupby(['Driver_ID'])['Age','Gender','Education_Level','Income','Joining
```

```
new_df['City'] = df.groupby(['Driver_ID'])['City'].agg(pd.Series.mode)
```

As we can observe in our Main Data that the data is observed over a period of Time so for that purpose many duplicate rows can be observed. So, now new Dataframe is created in which the whole data will Grouped by w.r.t Driver_ID. Where we can observe different behaviors from different features in main data, same behaviour is tried to maintain in the new dataframe.

```
new_df['Days_Difference']=new_df['Date_Difference'].dt.days
churn = []
for i in new_df['Days_Difference']:
  if i>=1:
    churn.append(1)
  else:
    churn.append(0)

new_df['Churn_Rate'] = churn
new_df['Churn_Rate'].value_counts()
```

```
1    1612
0     769
Name: Churn_Rate, dtype: int64
```

From the New Dataframe, we can say that the all data is pivoted w.r.t Driver_ID, So the process of Finding out if the Driver Leaved or not is simple. For this process, firstly calculated the number of days for Each Drivers and if the value is coming null for any driver_id then the driver is still working for ola, hence Churn rate for that driver is 0 hence for other case the value will be 1.

```
q_rating_min = df.groupby(['Driver_ID'])['Quarterly Rating'].min()
q_rating_max = df.groupby(['Driver_ID'])['Quarterly Rating'].max()
q=q_rating_max-q_rating_min
q.value_counts()
```

```
0    1277
1     521
2     397
3     186
Name: Quarterly Rating, dtype: int64
```

```
qtr_change = []
for i in q:
  if i<1:
    qtr_change.append(0)
  else:
    qtr_change.append(1)
```

```
new_df['Quater_Rating_Change'] = qtr_change
```

```
new_df['Quater_Rating_Change'].value_counts()
```

```
0    1277
1    1104
Name: Quater_Rating_Change, dtype: int64
```

For Checking if the Drivers Quaterly rating is increased or not, we have summoned the minimum quater rating of the driver w.r.t to driver_id and similarly summoned the maximum rating of the same and hence after finding the difference between these two variables we can conclude that whether the driver's rating is increased or not. Same process can be observed in above code block. And hence naming the feature Quater_Rating_Change

```
new_df
```

	Age	Gender	Education_Level	Income	Grade	Total Business Value	Quarterly Rating	Date_Diffe
Driver_ID								
1	28.0	0.0	2.0	57387.0	1.0	1715580	2.0	7
2	31.0	0.0	2.0	67016.0	2.0	0	1.0	
4	43.0	0.0	2.0	65603.0	2.0	350000	1.0	14
5	29.0	0.0	0.0	46368.0	1.0	120360	1.0	5
6	31.0	1.0	1.0	78728.0	3.0	1265000	2.0	
...	...	...	...	...	...	...	...	
2784	33.5	0.0	0.0	82815.0	3.0	21748820	3.0	
2785	34.0	1.0	0.0	12105.0	1.0	0	1.0	6
2786	45.0	0.0	0.0	35370.0	2.0	2815090	2.0	41
2787	28.0	1.0	2.0	69498.0	1.0	977830	1.5	33
2788	30.0	0.0	2.0	70254.0	2.0	2298240	2.0	

2381 rows × 12 columns

```
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2381 entries, 1 to 2788
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
```

```
 0   Age                   2381 non-null   float64
 1   Gender                2381 non-null   float64
 2   Education_Level       2381 non-null   float64
 3   Income                2381 non-null   float64
 4   Grade                 2381 non-null   float64
 5   Total Business Value  2381 non-null   int64
 6   Quarterly Rating      2381 non-null   float64
 7   Date_Difference       1616 non-null   timedelta64[ns]
 8   City                  2381 non-null   object
 9   Days_Difference       1616 non-null   float64
 10  Churn_Rate            2381 non-null   int64
 11  Quater_Rating_Change  2381 non-null   int64
dtypes: float64(7), int64(3), object(1), timedelta64[ns](1)
memory usage: 241.8+ KB
```

```
new_df.describe()
```

	Age	Gender	Education_Level	Income	Grade	Tota Busines Valu
count	2381.000000	2381.000000	2381.00000	2381.000000	2381.000000	2.381000e+0
mean	33.370223	0.410332	1.00756	59209.060899	2.078538	4.586742e+0
std	5.893555	0.491997	0.81629	28275.899087	0.931321	9.127115e+0
min	21.000000	0.000000	0.00000	10747.000000	1.000000	-1.385530e+0
25%	29.000000	0.000000	0.00000	39104.000000	1.000000	0.000000e+0
50%	33.000000	0.000000	1.00000	55276.000000	2.000000	8.176800e+0
75%	37.000000	1.000000	2.00000	75765.000000	3.000000	4.173650e+0
max	58.000000	1.000000	2.00000	188418.000000	5.000000	9.533106e+0

```
new_df.dtypes
```

```
Age                               float64
Gender                            float64
Education_Level                   float64
Income                            float64
Grade                             float64
Total Business Value                int64
Quarterly Rating                  float64
Date_Difference           timedelta64[ns]
City                               object
Days_Difference                   float64
Churn_Rate                          int64
```

```
     Quater_Rating_Change              int64
     dtype: object
```

```
new_df.columns
```

```
     Index(['Age', 'Gender', 'Education_Level', 'Income', 'Grade',
            'Total Business Value', 'Quarterly Rating', 'Date_Difference', 'City',
            'Days_Difference', 'Churn_Rate', 'Quater_Rating_Change'],
           dtype='object')
```

```
cols = ['Age', 'Gender', 'Education_Level', 'Income', 'Grade',
       'Total Business Value', 'City', 'Days_Difference', 'Churn_Rate',
       'Quater_Rating_Change']
for i in cols:
  print(i," ",new_df[i].nunique())
```

```
     Age   59
     Gender   2
     Education_Level   3
     Income   2339
     Grade   5
     Total Business Value   1629
     City   29
     Days_Difference   680
     Churn_Rate   2
     Quater_Rating_Change   2
```

```
new_df.isnull().sum()
```

```
     Age                       0
     Gender                    0
     Education_Level           0
     Income                    0
     Grade                     0
     Total Business Value      0
     Quarterly Rating          0
     Date_Difference         765
     City                      0
     Days_Difference         765
     Churn_Rate                0
     Quater_Rating_Change      0
     dtype: int64
```

```
y1=[]
for i in new_df['City']:
  y1.append(int(i[1:]))

new_df['City'] = y1
```

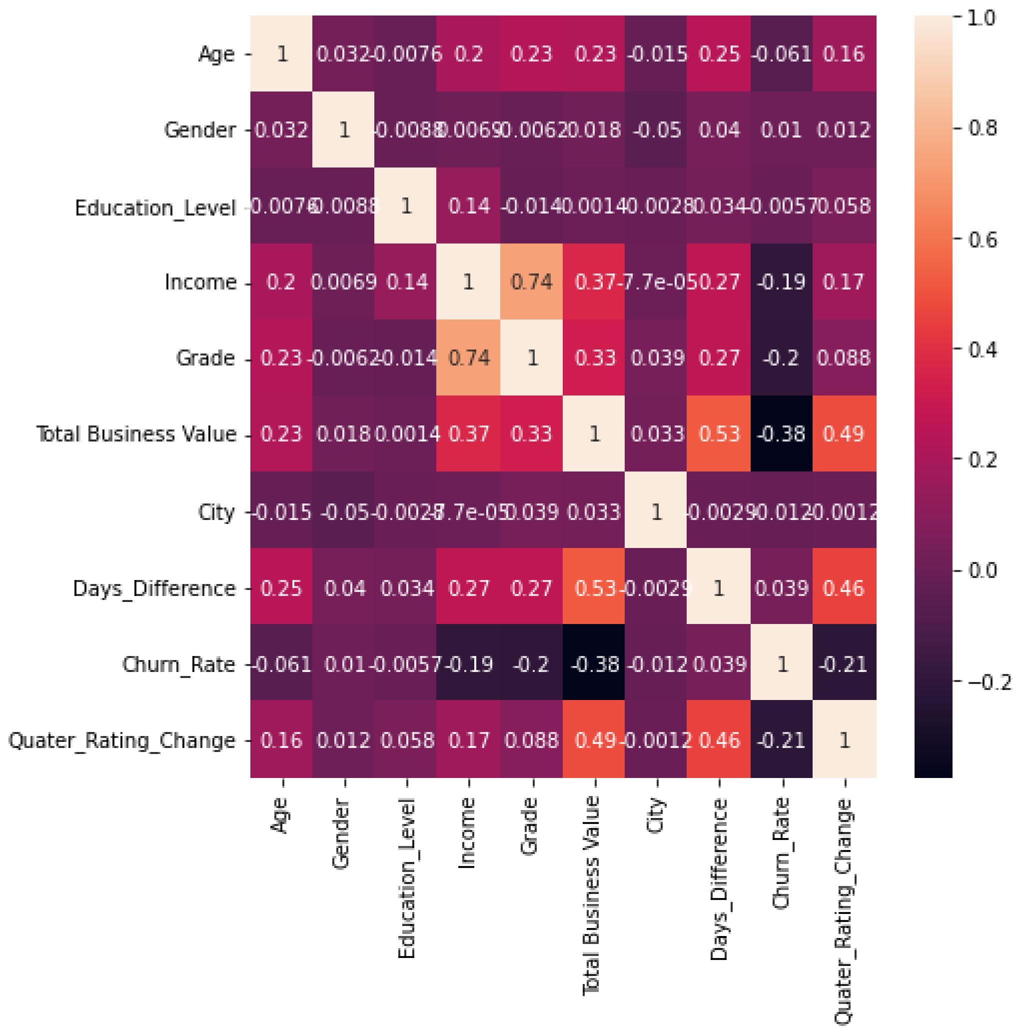
```
new_df.drop(['Date_Difference','Quarterly Rating'],axis=1,inplace=True)
```

```
cols = ['Age', 'Gender', 'Education_Level', 'Income', 'Grade',
       'Total Business Value', 'City',
       'Quater_Rating_Change']
target = ['Churn_Rate']
```

```
new_df['Gender'].value_counts()
```

```
0.0    1404
1.0     977
Name: Gender, dtype: int64
```

```
plt.figure(figsize=(7,7))
sns.heatmap(new_df.corr(),annot=True)
plt.show()
```



Splitting the Overall dataset into Train, Validation and Test

```
from sklearn.model_selection import train_test_split
```

```
X_tr_cv, X_test, y_tr_cv, y_test = train_test_split(new_df[cols], new_df[target], test_size=0
X_train,X_val,y_train,y_val = train_test_split(X_tr_cv,y_tr_cv,test_size = 0.25,random_state
print("X_train: ",X_train.shape,"X_validation: ", X_val.shape,"X_test: ",X_test.shape,"Y_trai
```

```
X_train:  (1428, 8) X_validation:  (476, 8) X_test:  (477, 8) Y_train (1428, 1) Y_val:
```

```
X_train.isnull().sum()
```

```
Age                    0
Gender                 0
Education_Level        0
Income                 0
Grade                  0
Total Business Value   0
City                   0
Quater_Rating_Change   0
dtype: int64
```

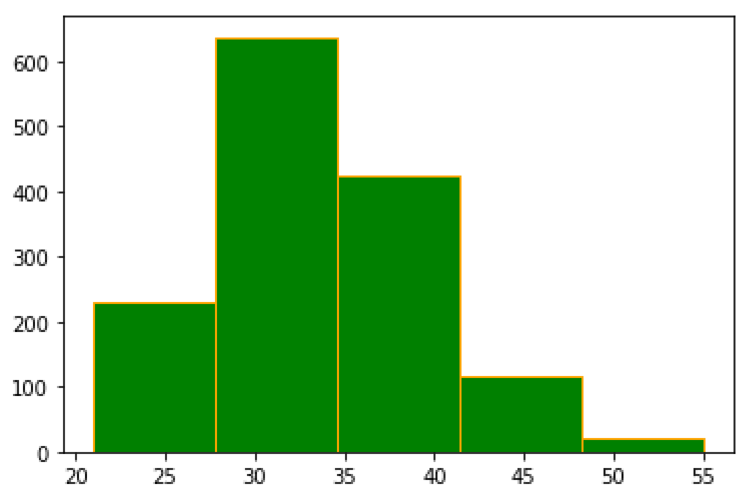
As we can see there are no null values in any columns, so no need of Doing KNN Imputation.

Univariate Analysis

```
new_df.columns
```

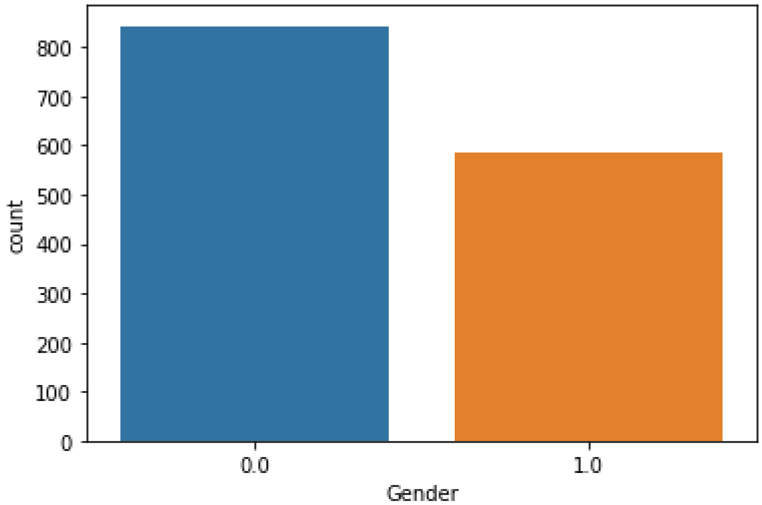
```
Index(['Age', 'Gender', 'Education_Level', 'Income', 'Grade',
       'Total Business Value', 'City', 'Days_Difference', 'Churn_Rate',
       'Quater_Rating_Change'],
      dtype='object')
```

```
plt.hist(X_train['Age'],color = 'green',edgecolor="orange",bins=5,)
plt.show()
```



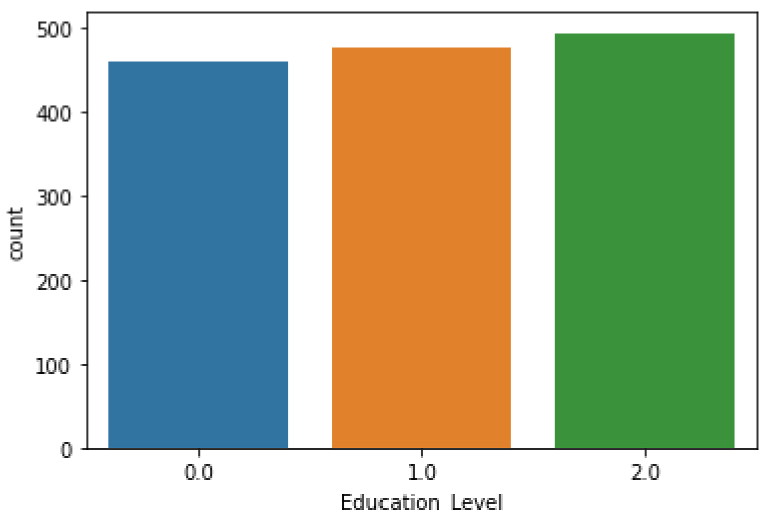
```
sns.countplot(X_train['Gender'])
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```

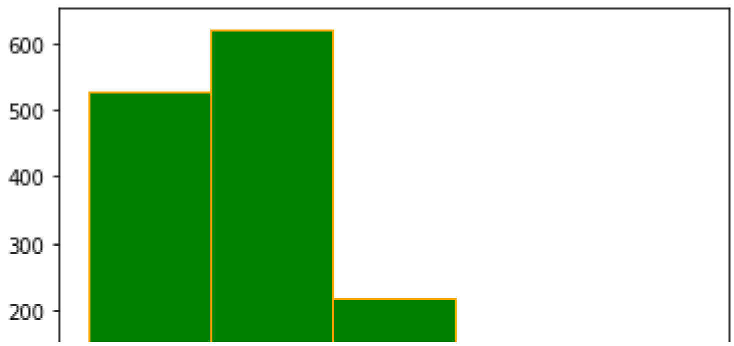



```
sns.countplot(X_train['Education_Level'])
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```

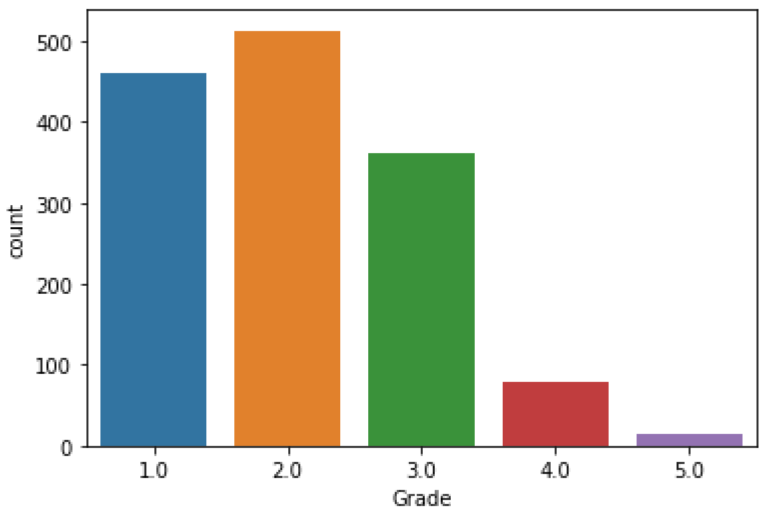



```
plt.hist(X_train['Income'],color = 'green',edgecolor="orange",bins=5,)
plt.show()
```




```
sns.countplot(X_train['Grade'])
plt.show()
```

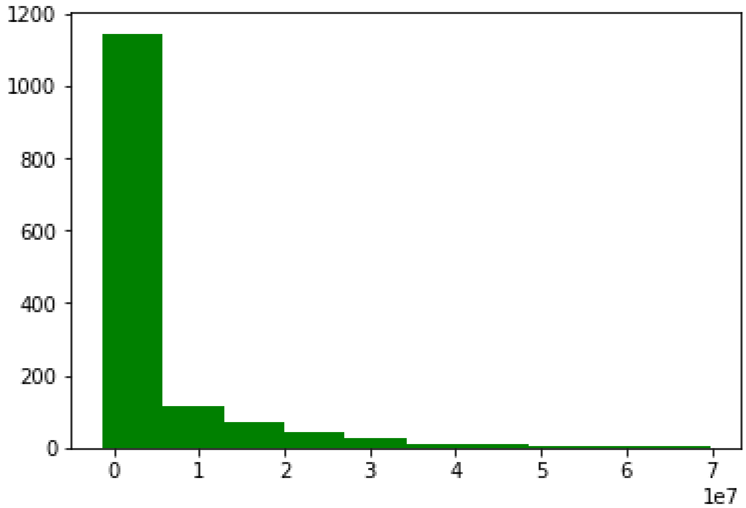
```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass tl
  warnings.warn(
```




```
plt.figure(figsize=(15,5))
sns.countplot(X_train['City'])
plt.show()
```

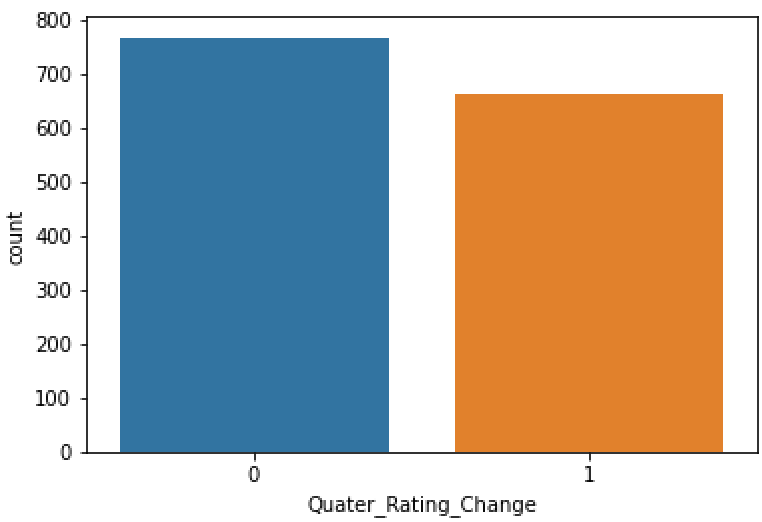
```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```


```
plt.hist(X_train['Total Business Value'],color = 'green',bins=10)
plt.show()
```

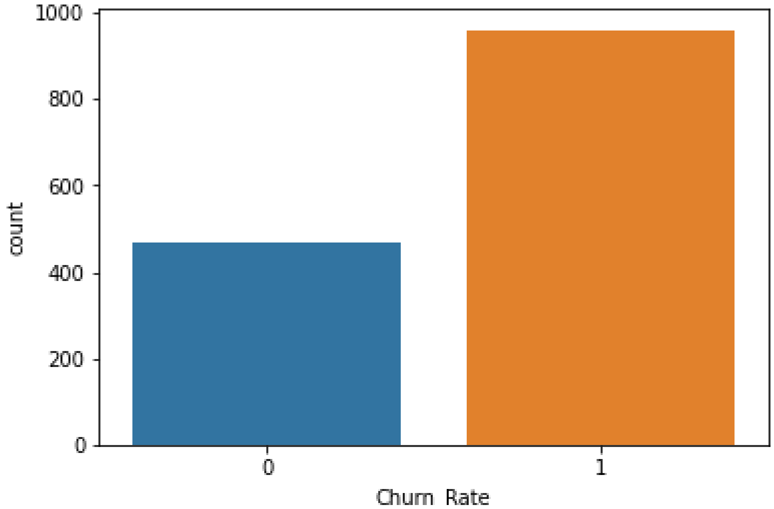



```
sns.countplot(X_train['Quater_Rating_Change'])
plt.show()
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass th
  warnings.warn(
```



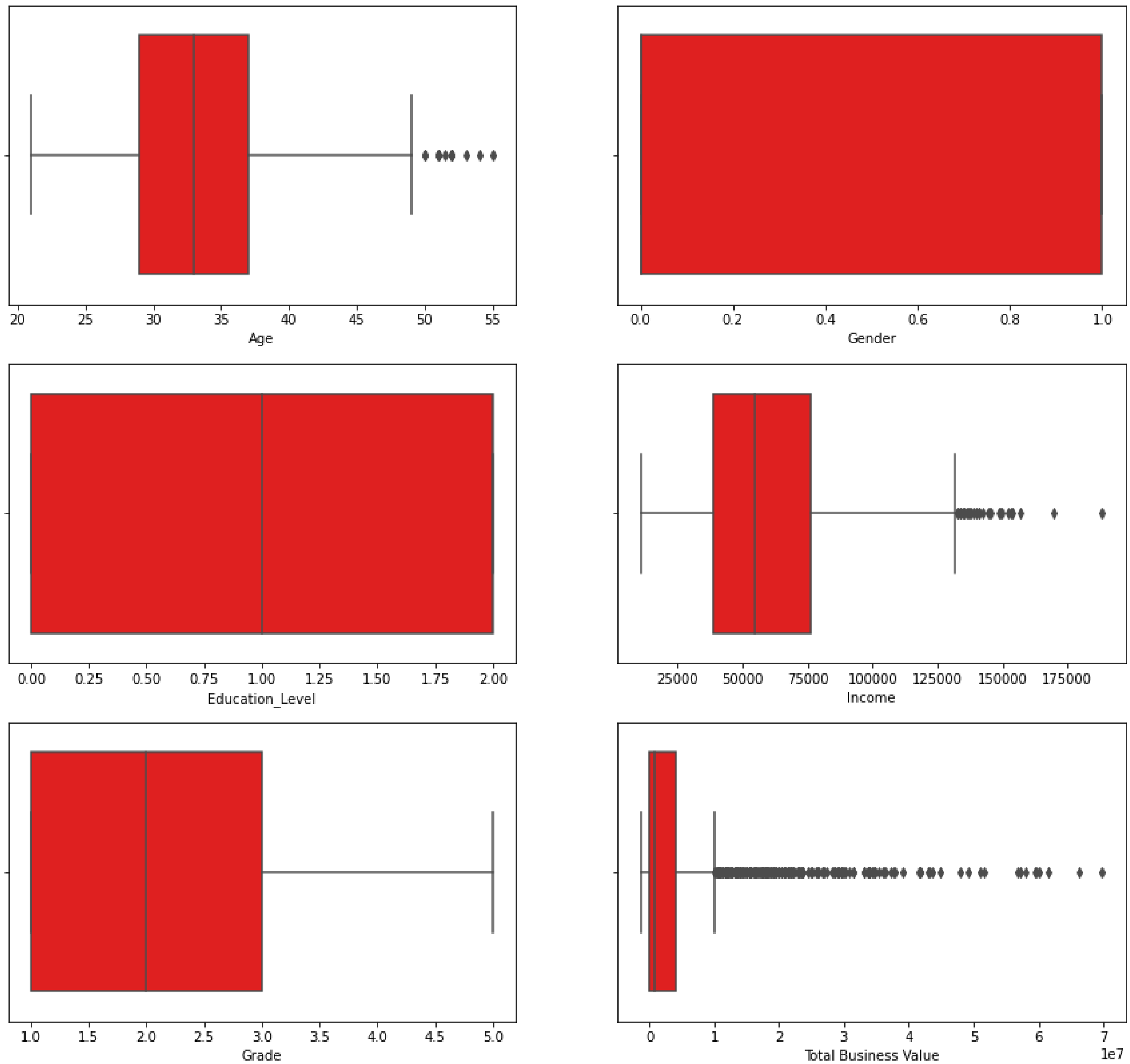

```
sns.countplot(x = y_train['Churn_Rate'])
plt.show()
```



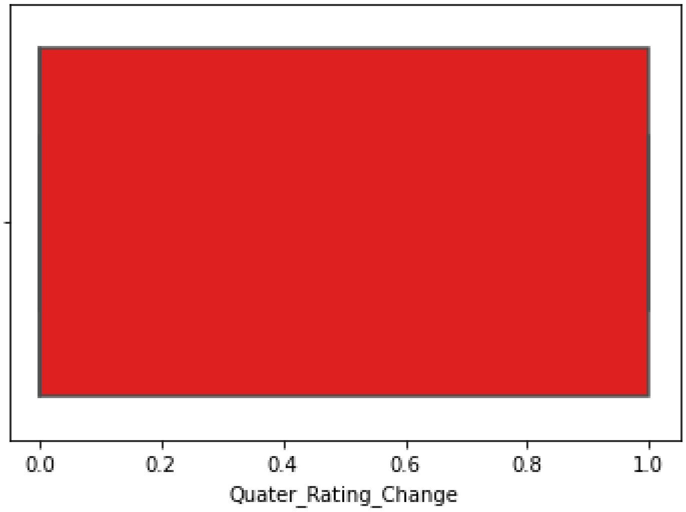
1. From the above Histogram, we can observe that the age of the drivers are mostly in the range of 30-35 years and after that 35-40 years. So we can conclude that most of the drivers main occupation is Driving Cab.
2. We can clearly see gender domination of Men over Women. So most of the Cab drivers are Men.
3. Education Level: Most of the drivers Education Level lies in between 1.0 -2.0 means the drivers have completed there Higher Secondary Schooling.
4. In Income Histogram we can clearly see a certain Spike in Income in range till 75000, later there is sudden drop in the income. Maybe Higher Grade drivers are getting paid as compared to lower grade drivers. Hence the Churn Rate is High for lower grade drivers.
5. We can clearly see that the Grades are basically distributed in 5 categories, but the most of the population belongs to Grade - 1,2,3 and there is around 10% of drivers present in the Higher Grade, Similar distribution we have noticed in the Salary grade.
6. In City distribution, we can clearly see that it seems to be equally distributed and city 21 and 29 has highest number of drivers present.
7. Total Business Value also says similar behaviour as of Salary and Grade, so we can check collinearlity between these columns.
8. On the basis of all drivers, those who have churned or not, most of the drivers ratings have been not changed and maybe they have churned before increased in rating.
9. Churn Rate where 0 is represented as not Churned and 1 is represented on Churned. We can definitely see the difference between these as more people churns and few continues with Ola.

Checking For Outliers:

```
fig,axes = plt.subplots(nrows = 3,ncols = 2,figsize=(15, 14))
sns.boxplot(x = X_train['Age'],  color ='red',ax = axes[0,0])
sns.boxplot(x = X_train['Gender'],  color ='red',ax = axes[0,1])
sns.boxplot(x = X_train['Education_Level'], color ='red',ax = axes[1,0])
sns.boxplot(x = X_train['Income'],  color ='red',ax = axes[1,1])
sns.boxplot(x = X_train['Grade'],  color ='red',ax = axes[2,0])
sns.boxplot(x = X_train['Total Business Value'],  color ='red',ax = axes[2,1])
plt.show()
```



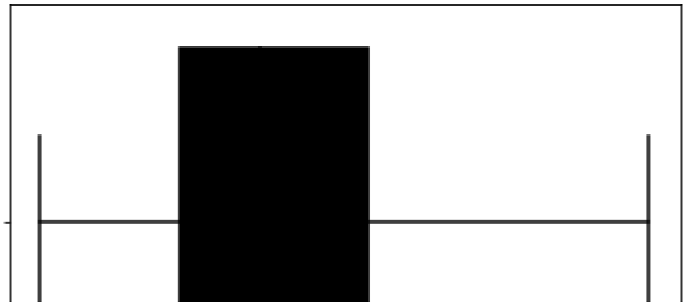
```
sns.boxplot(x = X_train['Quater_Rating_Change'],  color ='red')
plt.show()
```



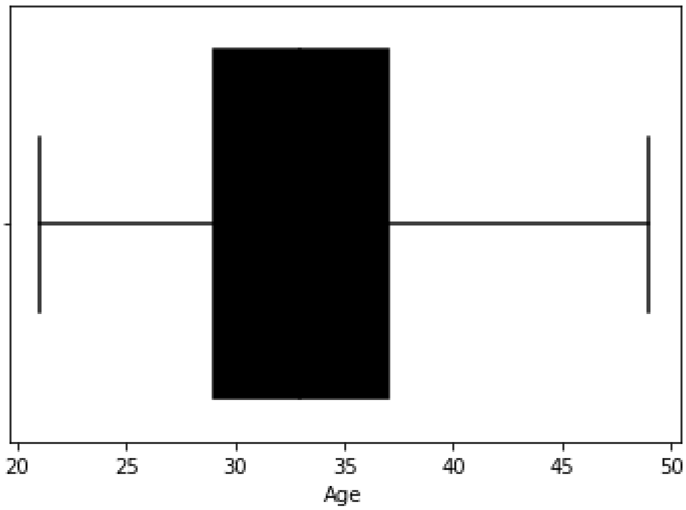
Fixing Outliers:

```
cols2 = ['Age','Income','Total Business Value']
for i in cols2:
  open_acc_25 = X_train[i].quantile(0.25)
  open_acc_75 = X_train[i].quantile(0.75)
  iqr2 = open_acc_75 - open_acc_25
  upper_limit2= open_acc_75 + 1.5 * iqr2
  lower_limit2 = open_acc_25 - 1.5 * iqr2

  X_train[i] = np.where(
      X_train[i] > upper_limit2,
      upper_limit2,
      np.where(
          X_train[i] < lower_limit2,
          lower_limit2,
          X_train[i]
      )
  )
```

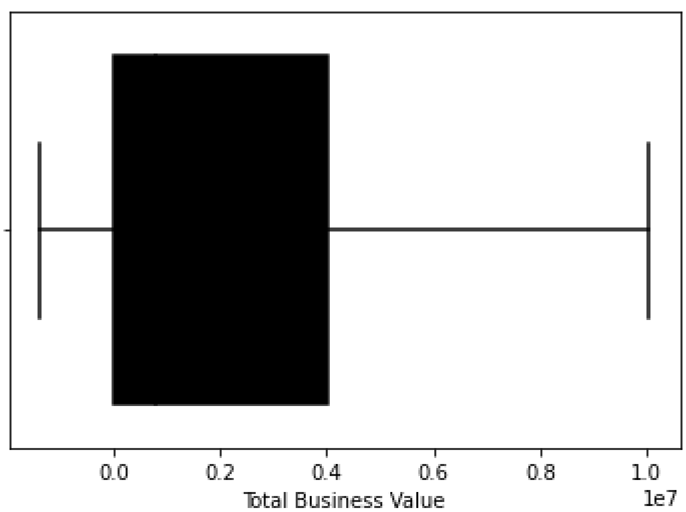
```
sns.boxplot(x = X_train['Income'],color ='black')
plt.show()
```



```
sns.boxplot(x = X_train['Age'],  color ='black')
plt.show()
```



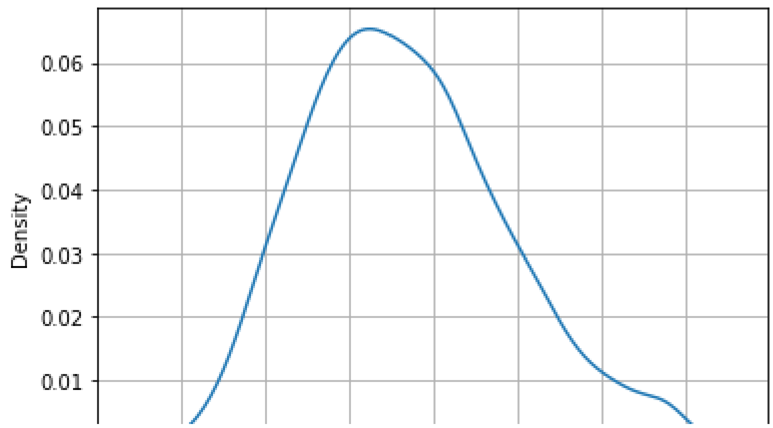

```
sns.boxplot(x = X_train['Total Business Value'],  color ='black')
plt.show()
```




```
X_train.columns
```

```
Index(['Age', 'Gender', 'Education_Level', 'Income', 'Grade',
       'Total Business Value', 'City', 'Quater_Rating_Change'],
      dtype='object')
```

```
def kde_plots(data,col):
  sns.kdeplot(data=data, x=col)
  plt.grid()
  plt.show()


for col in ['Age', 'Gender', 'Education_Level', 'Income', 'Grade',
       'Total Business Value', 'City', 'Quater_Rating_Change']:
       kde_plots(X_train, col)
```

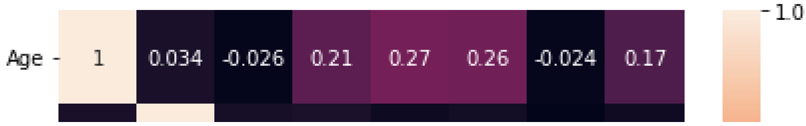


Bi-Variate Analysis

```
X_train.columns
```

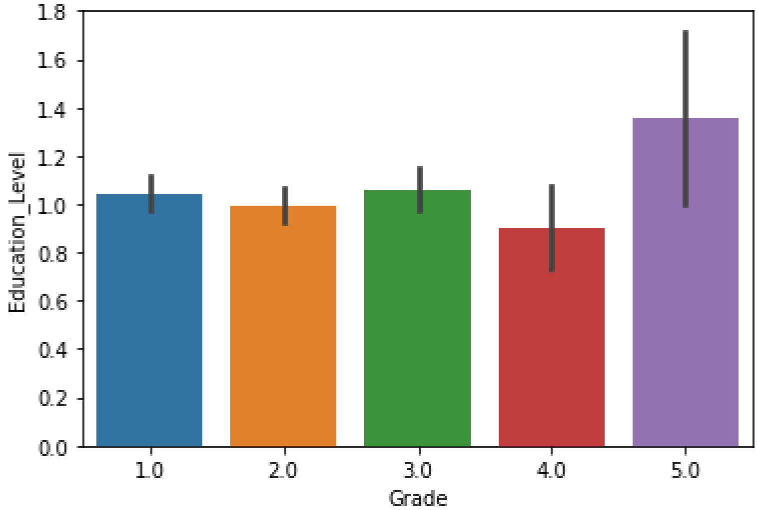
```
Index(['Age', 'Gender', 'Education_Level', 'Income', 'Grade',
       'Total Business Value', 'City', 'Quater_Rating_Change'],
      dtype='object')
```

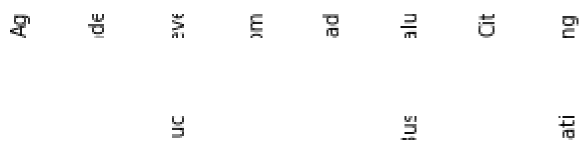
```
plt.figure(figsize=(7,7))
sns.heatmap(X_train.corr(),annot=True)
plt.show()
```




```
sns.barplot(data = X_train,x='Grade',y='Education_Level')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f86442f0c70>



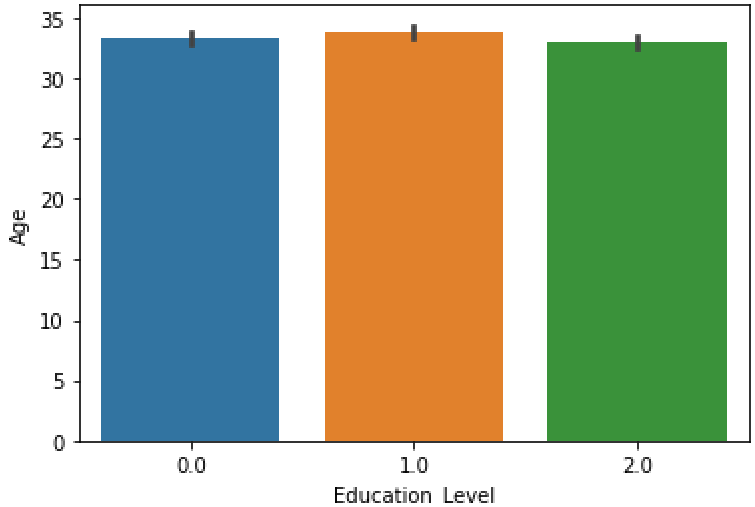


```
plt.figure(figsize = (25,5))
sns.jointplot(x ='Education_Level', y ='Age', data = X_train)
```

```
<seaborn.axisgrid.JointGrid at 0x7f864426ed60>
<Figure size 1800x360 with 0 Axes>
```

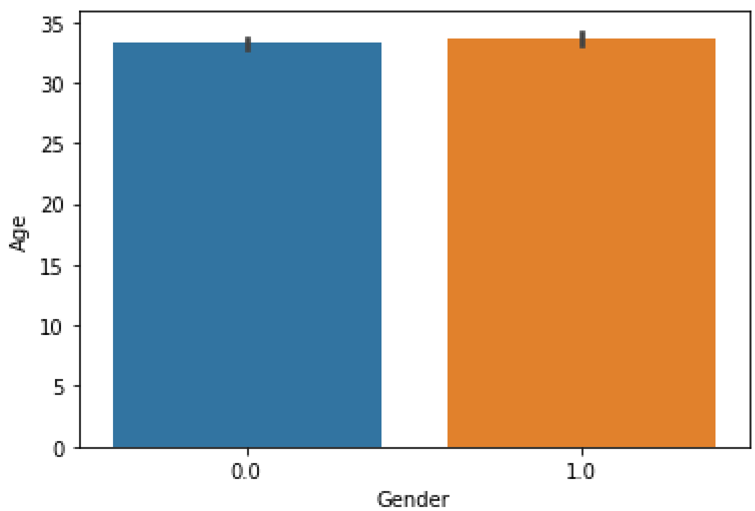
```
sns.barplot(data = X_train,x='Education_Level',y='Age')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8646ece190>
```



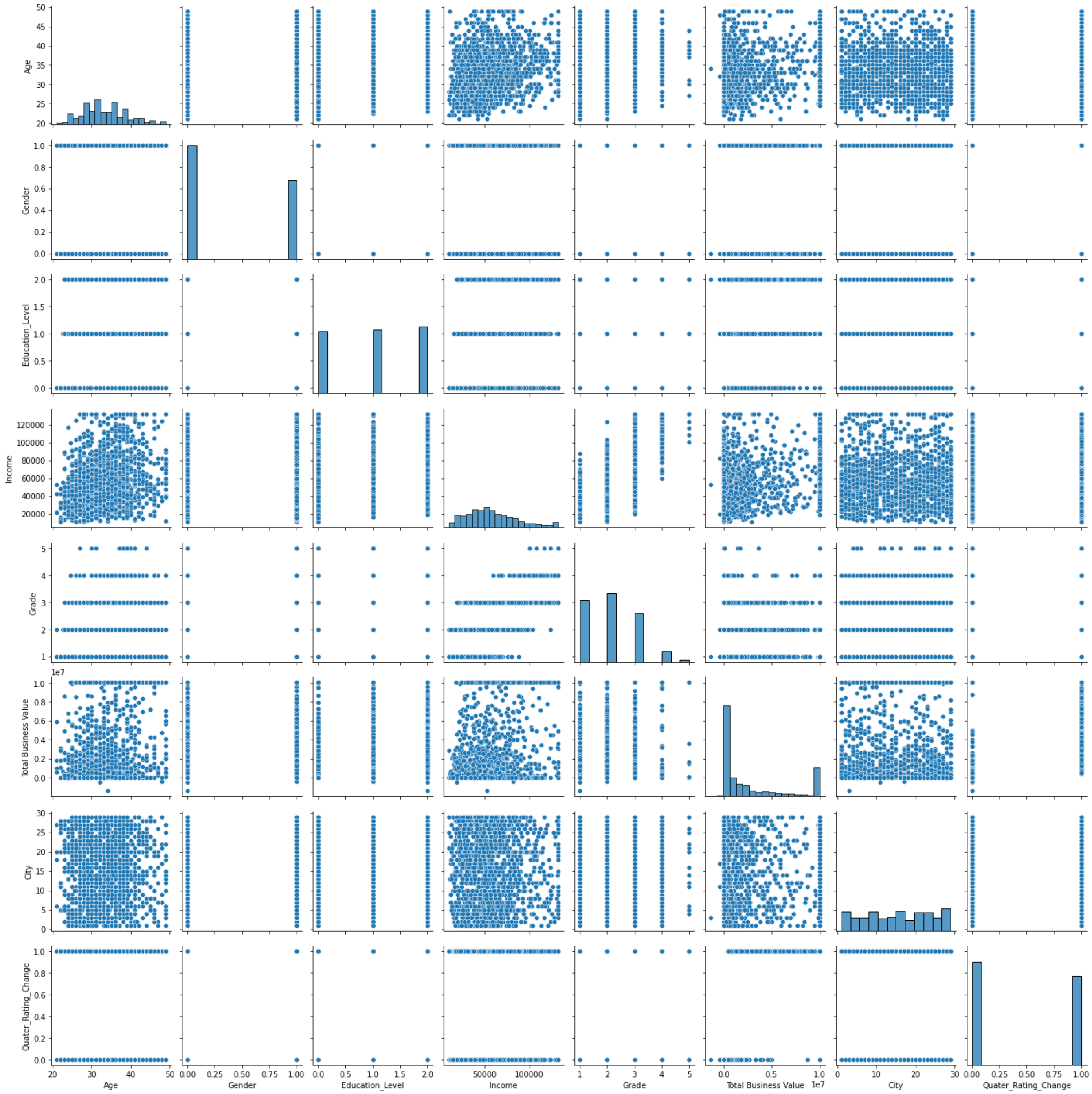

```
sns.barplot(data = X_train,x='Gender',y='Age')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8648d68c40>
```




```
sns.pairplot(X_train)
```

```
<seaborn.axisgrid.PairGrid at 0x7f867da069d0>
```



1. As mentioned in above insights we can see some strong correlation between Grade and Income, as Grade of a Driver Increases it is likely to say that Income of the same can be also increased.
2. Other than above, we can also say that Quater Rating Change and Total Business value has some correlation between them.

3. In Education_Level and Grade, we can see that the higher the Education Level the higher the Grade is assigned to the Driver.
4. We can see that there is less amount of drivers has been spreaded over Education Level 2, and we can see that those who have Education Level 0 is spreaded over all age groups.

Double-click (or enter) to edit

Balancing Data

```
from imblearn.over_sampling import SMOTE
smt = SMOTE()
```

```
from imblearn.over_sampling import SMOTE
smt = SMOTE(random_state = 30)
X_sm,y_sm = smt.fit_resample(X_train,y_train)
```

```
y_sm.value_counts()
```

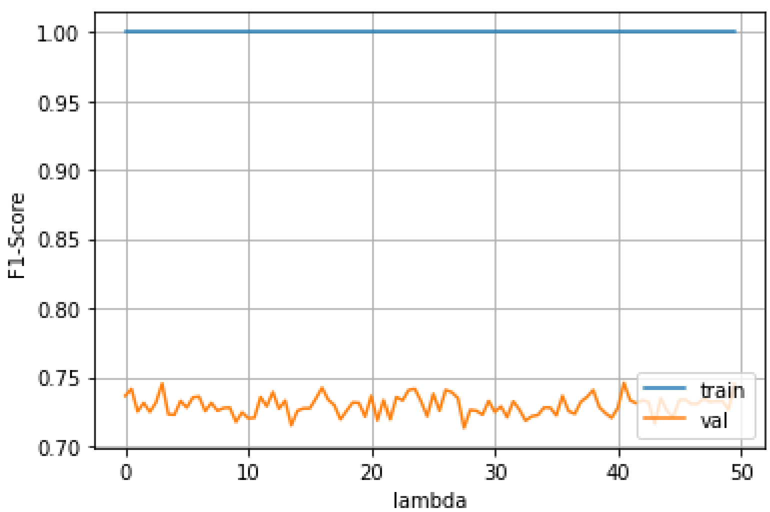
```
Churn_Rate
0             959
1             959
dtype: int64
```

Standardization:

```
from sklearn.preprocessing import StandardScaler

scale =StandardScaler()
X_train = scale.fit_transform(X_sm)
X_val = scale.transform(X_val)
X_test = scale.transform(X_test)
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score

train_scores = []
val_scores = []
scaler = StandardScaler()
l = 0.01
h = 50.0
```

```
d = 0.5

for la in np.arange(l,h,d):
  scaled_lr = make_pipeline( scaler, DecisionTreeClassifier())
  scaled_lr.fit(X_train, y_sm)
  train_y_pred = scaled_lr.predict(X_train)
  val_y_pred = scaled_lr.predict(X_val)
  train_score = f1_score(y_sm, train_y_pred)
  val_score = f1_score(y_val, val_y_pred)
  train_scores.append(train_score)
  val_scores.append(val_score)

plt.figure()
plt.plot(list(np.arange(l,h,d)), train_scores, label="train")
plt.plot(list(np.arange(l,h,d)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("F1-Score")
plt.grid()
plt.show()
```

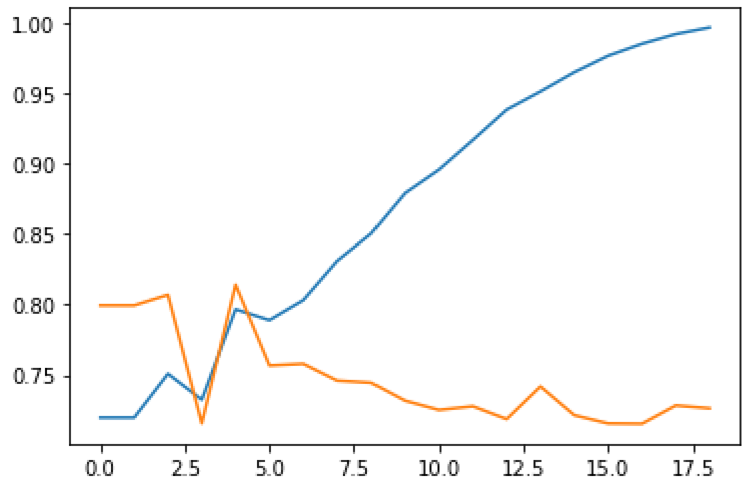


1. Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier


train_score = 0
val_score = 0
ts = []
vs = []
for i in range(1,20):
  tree_clf = DecisionTreeClassifier(max_depth=i,random_state = 40)
  tree_clf.fit(X_train,y_sm)
  train_y_pred = tree_clf.predict(X_train)
```

```
  val_y_pred = tree_clf.predict(X_val)
  train_score = f1_score(train_y_pred,y_sm)
  val_score = f1_score(y_val, val_y_pred)
  ts.append(train_score)
  vs.append(val_score)

plt.plot(ts)
plt.plot(vs)
```

```
[<matplotlib.lines.Line2D at 0x7f863e0e0a60>]
```




```
from sklearn.tree import DecisionTreeClassifier as DTC
from sklearn import tree
from sklearn.model_selection import GridSearchCV

params = {
    "max_depth" : [3, 5, 7,10],
    "max_leaf_nodes" : [15, 20, 25]
}
# Using GridSearchCV for Getting Hyper-tuned Parameters
model1 = DTC()
clf = GridSearchCV(model1, params, scoring = "accuracy", cv=5)

clf.fit(X_train, y_sm)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [3, 5, 7, 10],
                         'max_leaf_nodes': [15, 20, 25]},
             scoring='accuracy')
```

```
res = clf.cv_results_

for i in range(len(res["params"])):
  print(f"Parameters:{res['params'][i]} Mean_score: {res['mean_test_score'][i]} Rank: {res['r
```

```
Parameters:{'max_depth': 3, 'max_leaf_nodes': 15} Mean_score: 0.6934576261966927 Rank: 1
Parameters:{'max_depth': 3, 'max_leaf_nodes': 20} Mean_score: 0.6934576261966927 Rank: 1
Parameters:{'max_depth': 3, 'max_leaf_nodes': 25} Mean_score: 0.6934576261966927 Rank: 1
Parameters:{'max_depth': 5, 'max_leaf_nodes': 15} Mean_score: 0.7466941362053958 Rank: 3
Parameters:{'max_depth': 5, 'max_leaf_nodes': 20} Mean_score: 0.7456551892950392 Rank: 5
Parameters:{'max_depth': 5, 'max_leaf_nodes': 25} Mean_score: 0.7446148825065274 Rank: 6
Parameters:{'max_depth': 7, 'max_leaf_nodes': 15} Mean_score: 0.7377814947780679 Rank: 8
Parameters:{'max_depth': 7, 'max_leaf_nodes': 20} Mean_score: 0.7440450935596171 Rank: 7
Parameters:{'max_depth': 7, 'max_leaf_nodes': 25} Mean_score: 0.7471768929503917 Rank: 2
Parameters:{'max_depth': 10, 'max_leaf_nodes': 15} Mean_score: 0.7377814947780679 Rank:
Parameters:{'max_depth': 10, 'max_leaf_nodes': 20} Mean_score: 0.7461311466492603 Rank:
Parameters:{'max_depth': 10, 'max_leaf_nodes': 25} Mean_score: 0.7576139577893821 Rank:
```

```
print(clf.best_estimator_)
```

```
DecisionTreeClassifier(max_depth=10, max_leaf_nodes=25)
```

```
tree_clf = DecisionTreeClassifier(max_depth=10,max_leaf_nodes=25)
tree_clf.fit(X_train,y_sm)
train_y_pred = tree_clf.predict(X_train)
val_y_pred = tree_clf.predict(X_val)
train_score = f1_score(train_y_pred,y_sm)
val_score = f1_score(y_val, val_y_pred)
```

```
train_score
```

```
0.812127236580517
```

```
val_score
```

```
0.7830045523520485
```

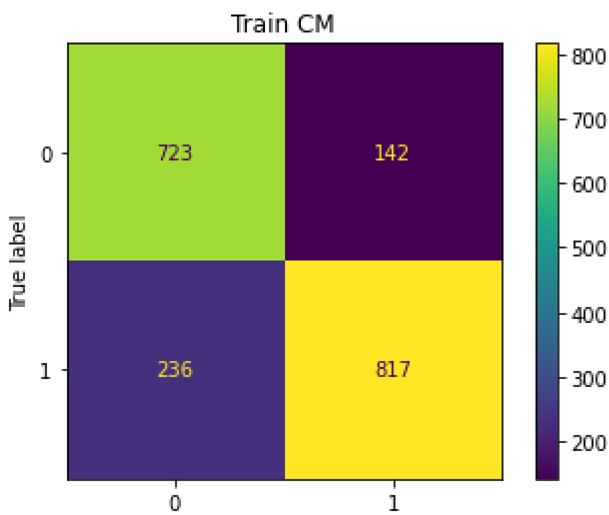
```
print(classification_report(train_y_pred,y_sm,target_names=['Continued','Churn']))
```

```
              precision    recall  f1-score   support

   Continued       0.75      0.84      0.79       865
       Churn       0.85      0.78      0.81      1053

    accuracy                           0.80      1918
   macro avg       0.80      0.81      0.80      1918
weighted avg       0.81      0.80      0.80      1918
```

```
print(classification_report(y_val, val_y_pred,target_names=['Continued','Churn']))
```

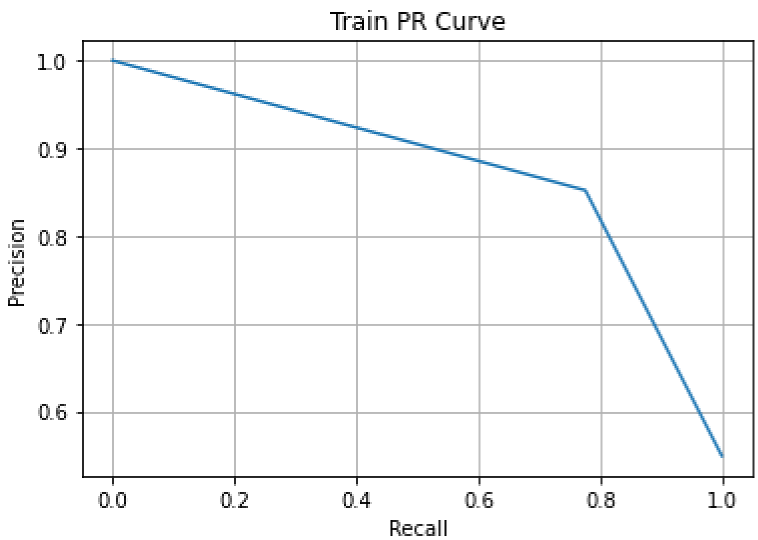
```
              precision    recall  f1-score   support

   Continued       0.55      0.48      0.51       156
       Churn       0.76      0.81      0.78       320

    accuracy                           0.70       476
   macro avg       0.65      0.64      0.65       476
weighted avg       0.69      0.70      0.69       476
```

```
conf_matrix = confusion_matrix(train_y_pred,y_sm)
ConfusionMatrixDisplay(conf_matrix).plot()
plt.title("Train CM")
plt.show()
```



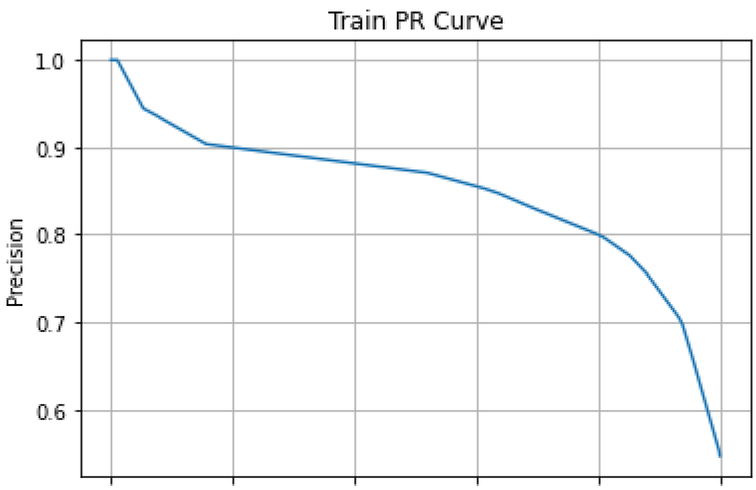

```
conf_matrix = confusion_matrix(y_val, val_y_pred)
ConfusionMatrixDisplay(conf_matrix).plot()
plt.title("CV CM")
plt.show()
```

```
precision, recall, thresholds = precision_recall_curve(train_y_pred,y_sm)
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Train PR Curve")
plt.grid()
plt.show()
```

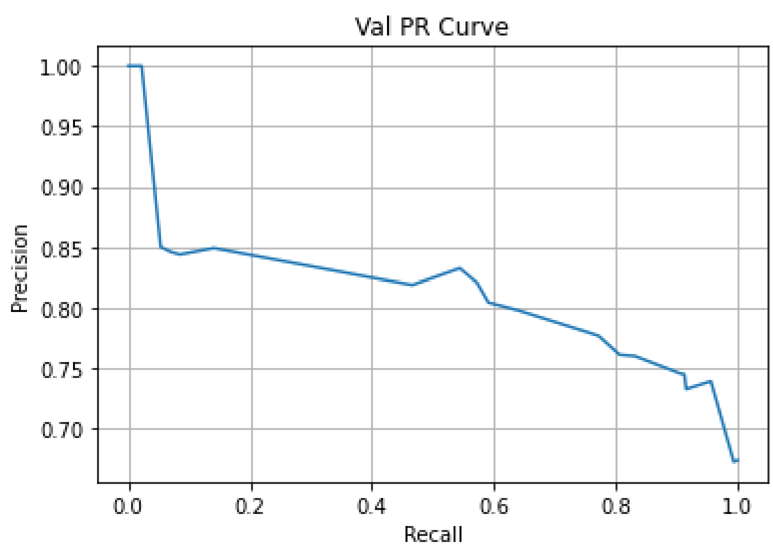



```
predicted_proba_train = tree_clf.predict_proba(X_train)
predicted_proba_cv = tree_clf.predict_proba(X_val)
train_f1_scores = []
cv_f1_scores = []
thresholds = np.arange(0.05, 1, 0.025)
for threshold in thresholds:
  train_preds = (predicted_proba_train[:,1] >= threshold).astype('int')
  cv_preds = (predicted_proba_cv[:,1] >= threshold).astype('int')
  trainF1Score = f1_score(y_sm, train_preds, average='weighted')
  cvF1Score = f1_score(y_val, cv_preds, average='weighted')
  train_f1_scores.append(trainF1Score)
  cv_f1_scores.append(cvF1Score)
```

```
precision, recall, thresholds = precision_recall_curve(np.asarray(y_sm), predicted_proba_trai
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Train PR Curve")
plt.grid()
plt.show()
```




```
precision, recall, thresholds = precision_recall_curve(np.asarray(y_val), predicted_proba_cv[
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Val PR Curve")
plt.grid()
plt.show()
```



Using Decision Tree Classifier on the Training data, using Hyper-Parameter tuning, we see that:

1. Training Score:- 81.21% where Testing Score:- 78.31
2. Precision - 0.76
3. Recall - 0.81

Confusion Matrix is also giving good results on Validation Data as True Positives are greater in Number as compared to Others.

We can conclude that, Precision - Recall in Decision Tree is observed to be good, can we get High Precision and Recall in other models, after observing all models we will test on Test Data.

2. Random Forest Classifier

```
params = {
    "n_estimators": [10,25,50,100,150,200],
    "max_depth" : [3, 5, 7,10,15,20],
    "max_leaf_nodes" : [15, 20, 25,30,35,40]
}

model2 = RFC()
clf = GridSearchCV(model2, params, scoring = "accuracy", cv=5)

clf.fit(X_train, np.ravel(y_sm))
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [3, 5, 7, 10, 15, 20],
                         'max_leaf_nodes': [15, 20, 25, 30, 35, 40],
                         'n_estimators': [10, 25, 50, 100, 150, 200]},
             scoring='accuracy')
```

```
res = clf.cv_results_

for i in range(len(res["params"])):
  print(f"Parameters:{res['params'][i]} Mean_score: {res['mean_test_score'][i]} Rank: {res['r
```

```
Parameters:{'max_depth': 3, 'max_leaf_nodes': 15, 'n_estimators': 10} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 15, 'n_estimators': 25} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 15, 'n_estimators': 50} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 15, 'n_estimators': 100} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 15, 'n_estimators': 150} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 15, 'n_estimators': 200} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 20, 'n_estimators': 10} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 20, 'n_estimators': 25} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 20, 'n_estimators': 50} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 20, 'n_estimators': 100} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 20, 'n_estimators': 150} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 20, 'n_estimators': 200} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 25, 'n_estimators': 10} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 25, 'n_estimators': 25} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 25, 'n_estimators': 50} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 25, 'n_estimators': 100} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 25, 'n_estimators': 150} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 25, 'n_estimators': 200} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 30, 'n_estimators': 10} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 30, 'n_estimators': 25} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 30, 'n_estimators': 50} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 30, 'n_estimators': 100} Mean_score: 0.
```

```
Parameters:{'max_depth': 3, 'max_leaf_nodes': 30, 'n_estimators': 150} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 30, 'n_estimators': 200} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 35, 'n_estimators': 10} Mean_score: 0.6
Parameters:{'max_depth': 3, 'max_leaf_nodes': 35, 'n_estimators': 25} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 35, 'n_estimators': 50} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 35, 'n_estimators': 100} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 35, 'n_estimators': 150} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 35, 'n_estimators': 200} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 40, 'n_estimators': 10} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 40, 'n_estimators': 25} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 40, 'n_estimators': 50} Mean_score: 0.7
Parameters:{'max_depth': 3, 'max_leaf_nodes': 40, 'n_estimators': 100} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 40, 'n_estimators': 150} Mean_score: 0.
Parameters:{'max_depth': 3, 'max_leaf_nodes': 40, 'n_estimators': 200} Mean_score: 0.
Parameters:{'max_depth': 5, 'max_leaf_nodes': 15, 'n_estimators': 10} Mean_score: 0.7
Parameters:{'max_depth': 5, 'max_leaf_nodes': 15, 'n_estimators': 25} Mean_score: 0.7
Parameters:{'max_depth': 5, 'max_leaf_nodes': 15, 'n_estimators': 50} Mean_score: 0.7
Parameters:{'max_depth': 5, 'max_leaf_nodes': 15, 'n_estimators': 100} Mean_score: 0.
Parameters:{'max_depth': 5, 'max_leaf_nodes': 15, 'n_estimators': 150} Mean_score: 0.
Parameters:{'max_depth': 5, 'max_leaf_nodes': 15, 'n_estimators': 200} Mean_score: 0.
Parameters:{'max_depth': 5, 'max_leaf_nodes': 20, 'n_estimators': 10} Mean_score: 0.7
Parameters:{'max_depth': 5, 'max_leaf_nodes': 20, 'n_estimators': 25} Mean_score: 0.7
Parameters:{'max_depth': 5, 'max_leaf_nodes': 20, 'n_estimators': 50} Mean_score: 0.7
Parameters:{'max_depth': 5, 'max_leaf_nodes': 20, 'n_estimators': 100} Mean_score: 0.
Parameters:{'max_depth': 5, 'max_leaf_nodes': 20, 'n_estimators': 150} Mean_score: 0.
Parameters:{'max_depth': 5, 'max_leaf_nodes': 20, 'n_estimators': 200} Mean_score: 0.
Parameters:{'max_depth': 5, 'max_leaf_nodes': 25, 'n_estimators': 10} Mean_score: 0.7
Parameters:{'max_depth': 5, 'max_leaf_nodes': 25, 'n_estimators': 25} Mean_score: 0.7
Parameters:{'max_depth': 5, 'max_leaf_nodes': 25, 'n_estimators': 50} Mean_score: 0.7
Parameters:{'max_depth': 5, 'max_leaf_nodes': 25, 'n_estimators': 100} Mean_score: 0.
Parameters:{'max_depth': 5, 'max_leaf_nodes': 25, 'n_estimators': 150} Mean_score: 0.
Parameters:{'max_depth': 5, 'max_leaf_nodes': 25, 'n_estimators': 200} Mean_score: 0.
Parameters:{'max_depth': 5, 'max_leaf_nodes': 30, 'n_estimators': 10} Mean_score: 0.7
```

```
print(clf.best_estimator_)
```

```
RandomForestClassifier(max_depth=20, max_leaf_nodes=35)
```

```
rf = clf.best_estimator_

rf.fit(X_train, np.ravel(y_sm))
```

```
RandomForestClassifier(max_depth=20, max_leaf_nodes=35)
```

```
rfc_clf = RFC(max_depth=10, max_leaf_nodes=40, n_estimators=100,n_jobs=-1)
rfc_clf.fit(X_train,np.ravel(y_sm))
train_y_pred = rfc_clf.predict(X_train)
val_y_pred = rfc_clf.predict(X_val)
train_score = f1_score(train_y_pred,np.ravel(y_sm))
val_score = f1_score(y_val, val_y_pred)
```

```
train_score

    0.856575682382134

val_score

    0.7781155015197568
```

```
print(classification_report(train_y_pred,y_sm,target_names=['Continued','Churn']))
```

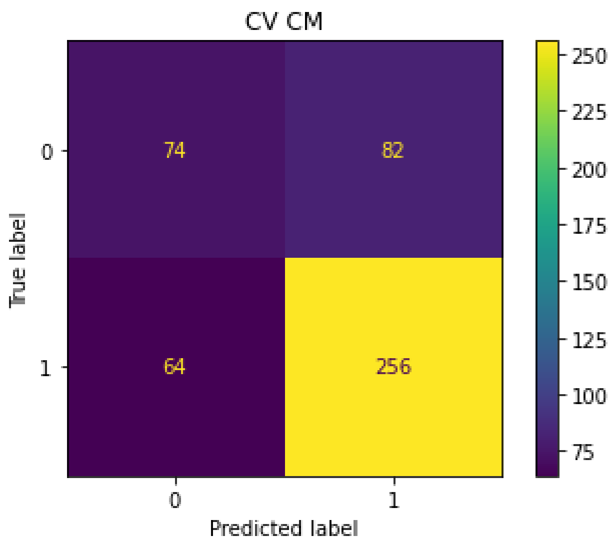
```
              precision    recall  f1-score   support

   Continued       0.80      0.89      0.84       862
       Churn       0.90      0.82      0.86      1056

    accuracy                           0.85      1918
   macro avg       0.85      0.85      0.85      1918
weighted avg       0.85      0.85      0.85      1918
```

```
print(classification_report(val_y_pred,y_val,target_names=['Continued','Churn']))
```

```
              precision    recall  f1-score   support

   Continued       0.47      0.54      0.50       138
       Churn       0.80      0.76      0.78       338

    accuracy                           0.69       476
   macro avg       0.64      0.65      0.64       476
weighted avg       0.71      0.69      0.70       476
```

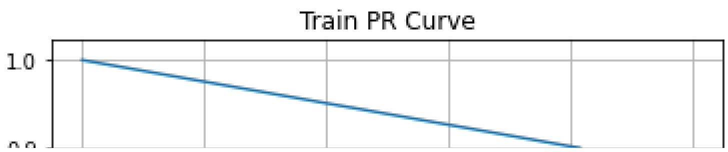
```
conf_matrix = confusion_matrix(train_y_pred,y_sm)
ConfusionMatrixDisplay(conf_matrix).plot()
plt.title("Train CM")
plt.show()
```


```
conf_matrix = confusion_matrix(y_val, val_y_pred)
ConfusionMatrixDisplay(conf_matrix).plot()
plt.title("CV CM")
plt.show()
```




```
precision, recall, thresholds = precision_recall_curve(train_y_pred,y_sm)
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Train PR Curve")
plt.grid()
plt.show()

predicted_proba_train = rfc_clf.predict_proba(X_train)
predicted_proba_cv = rfc_clf.predict_proba(X_val)
train_f1_scores = []
cv_f1_scores = []
thresholds = np.arange(0.05, 1, 0.025)
for threshold in thresholds:
  train_preds = (predicted_proba_train[:,1] >= threshold).astype('int')
  cv_preds = (predicted_proba_cv[:,1] >= threshold).astype('int')
  trainF1Score = f1_score(y_sm, train_preds, average='weighted')
  cvF1Score = f1_score(y_val, cv_preds, average='weighted')
  train_f1_scores.append(trainF1Score)
  cv_f1_scores.append(cvF1Score)

precision, recall, thresholds = precision_recall_curve(np.asarray(y_sm), predicted_proba_trai
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Train PR Curve")
```

```
plt.grid()
plt.show()

precision, recall, thresholds = precision_recall_curve(np.asarray(y_val), predicted_proba_cv[
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Val PR Curve")
plt.grid()
plt.show()
```



Using Random-Forest Classifier on the Training data, using Hyper-Parameter tuning, we see that:

1. Training Score:- 85.65% where Testing Score:- 78.81%
2. Precision - 0.80
3. Recall - 0.76

Confusion Matrix is also giving good results on Validation Data as True Positives are greater in Number as compared to Others.

We can conclude that, Precision - Recall in Random Forest Classifier is observed to be good, can we get High Precision and Recall in other models, after observing all models we will test on Test Data, and compared with the Descision Tree classifier we can also say that Random Forest classifier seems to be Overfitting. If we compare these two models we can say that Random Forest can give good performance on Testing Data if again Hyper Parameter are tuned correctly.

Precision - Recall Curve is on Validation seems to be getting more stable after some iteration but as we compared with decision tree, the curve was showing High Variance. Hence Random Forest Classifier model is good as compared with Descision tree.

3. Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=10)
gbc.fit(X_train, np.ravel(y_sm))
```

```
GradientBoostingClassifier(learning_rate=1.0, max_depth=10)
```

```
params = {
    "n_estimators": [10,25,50,100,150,200],
    "max_depth" : [3, 5, 7,10,15,20],
    "max_leaf_nodes" : [15, 20, 25,30,35,40]
}

model3 = GradientBoostingClassifier()
clf = GridSearchCV(model3, params, scoring = "accuracy", cv=5)

clf.fit(X_train, np.ravel(y_sm))
```

```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': [3, 5, 7, 10, 15, 20],
                         'max_leaf_nodes': [15, 20, 25, 30, 35, 40],
                         'n_estimators': [10, 25, 50, 100, 150, 200]},
             scoring='accuracy')
```

```
clf.best_params_
```

```
{'max_depth': 15, 'max_leaf_nodes': 40, 'n_estimators': 150}
```

```
gbc_clf = GradientBoostingClassifier(max_depth=5, max_leaf_nodes=40, n_estimators=150)
gbc_clf.fit(X_train,np.ravel(y_sm))
train_y_pred = gbc_clf.predict(X_train)
val_y_pred = gbc_clf.predict(X_val)
train_score = f1_score(train_y_pred,np.ravel(y_sm))
val_score = f1_score(y_val, val_y_pred)
```

```
test_y_pred = gbc_clf.predict(X_test)
test_score = f1_score(test_y_pred,y_test)
```

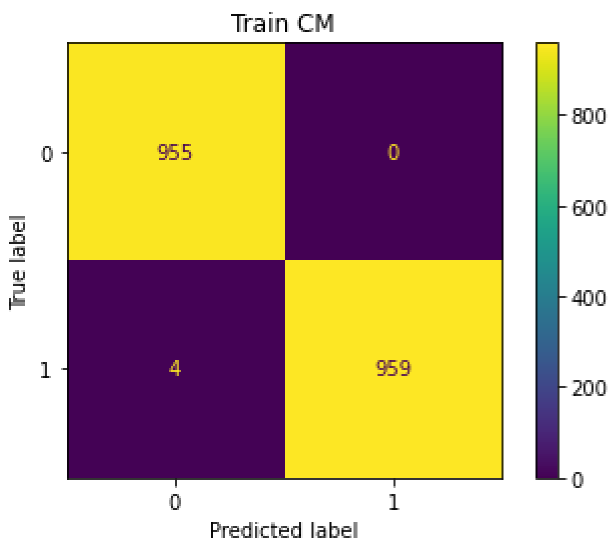
```
test_score
```

```
0.8976710334788937
```

```
print(classification_report(train_y_pred,y_sm,target_names=['Continued','Churn']))
print(classification_report(val_y_pred,y_val,target_names=['Continued','Churn']))

conf_matrix = confusion_matrix(train_y_pred,y_sm)
ConfusionMatrixDisplay(conf_matrix).plot()
plt.title("Train CM")
plt.show()

conf_matrix = confusion_matrix(y_val, val_y_pred)
ConfusionMatrixDisplay(conf_matrix).plot()
plt.title("CV CM")
plt.show()

conf_matrix = confusion_matrix(y_val, val_y_pred)
ConfusionMatrixDisplay(conf_matrix).plot()
plt.title("CV CM")
plt.show()

precision, recall, thresholds = precision_recall_curve(train_y_pred,y_sm)
plt.plot(recall, precision)
plt.xlabel("Recall")
```

```
plt.ylabel("Precision")
plt.title("Train PR Curve")
plt.grid()
plt.show()

predicted_proba_train = rfc_clf.predict_proba(X_train)
predicted_proba_cv = rfc_clf.predict_proba(X_val)
train_f1_scores = []
cv_f1_scores = []
thresholds = np.arange(0.05, 1, 0.025)
for threshold in thresholds:
  train_preds = (predicted_proba_train[:,1] >= threshold).astype('int')
  cv_preds = (predicted_proba_cv[:,1] >= threshold).astype('int')
  trainF1Score = f1_score(y_sm, train_preds, average='weighted')
  cvF1Score = f1_score(y_val, cv_preds, average='weighted')
  train_f1_scores.append(trainF1Score)
  cv_f1_scores.append(cvF1Score)

precision, recall, thresholds = precision_recall_curve(np.asarray(y_sm), predicted_proba_trai
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Train PR Curve")
plt.grid()
plt.show()

precision, recall, thresholds = precision_recall_curve(np.asarray(y_val), predicted_proba_cv[
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Val PR Curve")
plt.grid()
plt.show()
```

```
              precision    recall  f1-score   support

   Continued       1.00      1.00      1.00       955
       Churn       1.00      1.00      1.00       963

    accuracy                           1.00      1918
   macro avg       1.00      1.00      1.00      1918
weighted avg       1.00      1.00      1.00      1918

              precision    recall  f1-score   support

   Continued       0.46      0.55      0.50       129
       Churn       0.82      0.76      0.79       347

    accuracy                           0.70       476
   macro avg       0.64      0.65      0.64       476
weighted avg       0.72      0.70      0.71       476
```



Using Random-Forest Classifier on the Training data, using Hyper-Parameter tuning, we see that:

1. Training Score:- 99.8% where Validation Score:- 95.8% and Testing Score = 89.8%
2. Precision - 0.80
3. Recall - 0.76

Confusion Matrix is also giving good results on Validation Data as True Positives are greater in Number as compared to Others.

Insights:

1. From the above Histogram, we can observe that the age of the drivers are mostly in the range of 30-35 years and after that 35-40 years. So we can conclude that most of the drivers main

occupation is Driving Cab.

2. We can clearly see gender domination of Men over Women. So most of the Cab drivers are Men.
3. Education Level: Most of the drivers Education Level lies in between 1.0 -2.0 means the drivers have completed there Higher Secondary Schooling.
4. In Income Histogram we can clearly see a certain Spike in Income in range till 75000, later there is sudden drop in the income. Maybe Higher Grade drivers are getting paid as compared to lower grade drivers. Hence the Churn Rate is High for lower grade drivers.
5. We can clearly see that the Grades are basically distributed in 5 categories, but the most of the population belongs to Grade - 1,2,3 and there is around 10% of drivers present in the Higher Grade, Similar distribution we have noticed in the Salary grade.
6. In City distribution, we can clearly see that it seems to be equally distributed and city 21 and 29 has highest number of drivers present.
7. Total Business Value also says similar behaviour as of Salary and Grade, so we can check collinearlity between these columns.
8. On the basis of all drivers, those who have churned or not, most of the drivers ratings have been not changed and maybe they have churned before increased in rating.
9. Churn Rate where 0 is represented as not Churned and 1 is represented on Churned. We can definitely see the difference between these as more people churns and few continues with Ola.
10. As mentioned in above insights we can see some strong correlation between Grade and Income, as Grade of a Driver Increases it is likely to say that Income of the same can be also increased.
11. Other than above, we can also say that Quater Rating Change and Total Business value has some correlation between them.
12. In Education_Level and Grade, we can see that the higher the Education Level the higher the Grade is assigned to the Driver.
13. We can see that there is less amount of drivers has been spreaded over Education Level 2, and we can see that those who have Education Level 0 is spreaded over all age groups.

Actions:

1. As from above data we can see that there are high number of Churn Rate in Lower Grades Drivers, as they are not getting a good pay-day depending on there grade. Hence by increasing there grade after a subsequent period can help Ola to reduce Churn Rate.
2. We can also observe that those who have higher education are getting a good grade and hence it is helping them to get a high income. Hence if somehow Ola motivates there driver to pursue Higher Education will led to Low Churn Rate.

3. There is also seen some gender differentiation in driver and from confusion matrix we can also say that female gender is highly unlikely to churn, hence by increasing the Female Drivers can also possibly reduce the Churn Rate. For that introducing new schemes for female drivers can also be applied.
4. In Income grade, there is high spikes in the region till 75000, hence the driver seems to not getting enough income, hence that's why they can also opt for other services. So by increasing there per ride income and can try to match the services of other competitors can also help to reduce the churn rate.