Mobility_Platform_Churn_Prediction_by_Diptyajit_Das

January 26, 2025

0.0.1 Problem Statement

The primary challenge is to predict driver attrition in a ride-hailing service based on their demographics, tenure, performance, and historical data. High driver churn impacts operational efficiency, organizational morale, and increases the cost of acquiring new drivers. Retaining drivers is more cost-effective than replacing them, making it crucial to identify patterns and factors contributing to attrition.

The dataset includes driver demographics (age, gender, education level, city), tenure details (joining and last working dates), and historical performance metrics such as quarterly ratings, monthly income, and total business value. The goal is to create a robust predictive model to classify whether a driver is likely to leave the service or not.

This requires addressing deriving new features such as income or rating trends, handling class imbalance, and standardizing the data. Advanced ensemble methods like Bagging and Boosting will be utilized for prediction, along with appropriate hyperparameter tuning. The success of the model will be evaluated through metrics like classification reports, ROC-AUC curves, and actionable insights derived from the results to improve driver retention strategies.

[78]:
```
#!pip install imblearn --break-system-packages
#!pip install lightgbm --break-system-packages
#!pip install category_encoders --break-system-packages
#!pip install optuna --break-system-packages

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler,label_binarize,MinMaxScaler
from category_encoders import TargetEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.metrics import f1_score,classification_report, accuracy_score,␣
↪confusion_matrix,roc_curve, auc, precision_recall_curve,␣
↪average_precision_score
from sklearn.metrics import RocCurveDisplay,PrecisionRecallDisplay,␣
↪ConfusionMatrixDisplay
```

```python
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB,MultinomialNB
import optuna
from xgboost import XGBClassifier
import xgboost as xgb
from lightgbm import LGBMClassifier
from itertools import product
import warnings
from statsmodels.tsa.seasonal import seasonal_decompose
warnings.simplefilter('ignore')


#SEED=42
SEED=95
```

[79]:

```python
df = pd.read_csv('driver.csv').drop(columns=['Unnamed: 0'])
# look at the datatypes of the columns
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   MMM-YY                19104 non-null  object
 1   Driver_ID             19104 non-null  int64
 2   Age                   19043 non-null  float64
 3   Gender                19052 non-null  float64
 4   City                  19104 non-null  object
 5   Education_Level       19104 non-null  int64
 6   Income                19104 non-null  int64
 7   Dateofjoining         19104 non-null  object
 8   LastWorkingDate       1616 non-null   object
 9   Joining Designation   19104 non-null  int64
 10  Grade                 19104 non-null  int64
 11  Total Business Value  19104 non-null  int64
 12  Quarterly Rating      19104 non-null  int64
dtypes: float64(2), int64(7), object(4)
memory usage: 1.9+ MB
None
```

[80]:

```python
print(f'Shape of the dataset is {df.shape}')
```

```
Shape of the dataset is (19104, 13)
```

[81]:

```python
print(f'Number of nan/null values in each column: \n{df.isna().sum()}')
```

```
Number of nan/null values in each column:
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
```

[82]:
```
print(f'Number of unique values in each column: \n{df.nunique()}')
```

```
Number of unique values in each column:
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
```

[83]:
```
print(f'Duplicate entries: \n{df.duplicated().value_counts()}')
```

```
Duplicate entries:
False    19104
Name: count, dtype: int64
```

[84]:
```
df.head()
```

[84]:
```
     MMM-YY  Driver_ID   Age  Gender City  Education_Level  Income  \
0  01/01/19          1  28.0     0.0  C23                2   57387
1  02/01/19          1  28.0     0.0  C23                2   57387
2  03/01/19          1  28.0     0.0  C23                2   57387
3  11/01/20          2  31.0     0.0   C7                2   67016
4  12/01/20          2  31.0     0.0   C7                2   67016
```

```
  Dateofjoining LastWorkingDate  Joining Designation  Grade  \
0      24/12/18             NaN                    1      1
1      24/12/18             NaN                    1      1
2      24/12/18        03/11/19                    1      1
3      11/06/20             NaN                    2      2
4      11/06/20             NaN                    2      2

   Total Business Value  Quarterly Rating
0               2381060                 2
1               -665480                 2
2                     0                 2
3                     0                 1
4                     0                 1
```

[85]:
```python
df.describe()
```

[85]:
```
          Driver_ID           Age        Gender  Education_Level  \
count  19104.000000  19043.000000  19052.000000     19104.000000
mean    1415.591133     34.668435      0.418749         1.021671
std      810.705321      6.257912      0.493367         0.800167
min        1.000000     21.000000      0.000000         0.000000
25%      710.000000     30.000000      0.000000         0.000000
50%     1417.000000     34.000000      0.000000         1.000000
75%     2137.000000     39.000000      1.000000         2.000000
max     2788.000000     58.000000      1.000000         2.000000

              Income  Joining Designation         Grade  Total Business Value  \
count   19104.000000         19104.000000  19104.000000          1.910400e+04
mean    65652.025126             1.690536      2.252670          5.716621e+05
std     30914.515344             0.836984      1.026512          1.128312e+06
min     10747.000000             1.000000      1.000000         -6.000000e+06
25%     42383.000000             1.000000      1.000000          0.000000e+00
50%     60087.000000             1.000000      2.000000          2.500000e+05
75%     83969.000000             2.000000      3.000000          6.997000e+05
max    188418.000000             5.000000      5.000000          3.374772e+07

       Quarterly Rating
count      19104.000000
mean           2.008899
std            1.009832
min            1.000000
25%            1.000000
50%            2.000000
75%            3.000000
max            4.000000
```

[86]:
```python
df.describe(include='object')
```

[86]:
```
          MMM-YY   City Dateofjoining LastWorkingDate
count      19104  19104         19104            1616
unique        24     29           869             493
top     01/01/19    C20      23/07/15        29/07/20
freq        1022   1008           192              70
```

0.0.2 Insight

- There are **19104** entries with 14 columns
- There are **61 null/missing values in *Age*, 52 in *Gender* and 17488 in *LastWorkingDate***
- There are **2381 unique drivers**
- There are no **duplicates**
- The column ***Unnamed: 0*** can be dropped as it doesnt provide any new information
- The columns ***Gender, City, Education_Level, Joining Designation, Grade*** and ***Quarterly Rating*** can be converted to **categorical** datatype
- The columns ***MMM-YY, Dateofjoining*** and ***LastWorkingDate*** can be converted to **datetime** datatype
- Drivers who donot have valid ***LastWorkingDate*** can be considered as **churned**

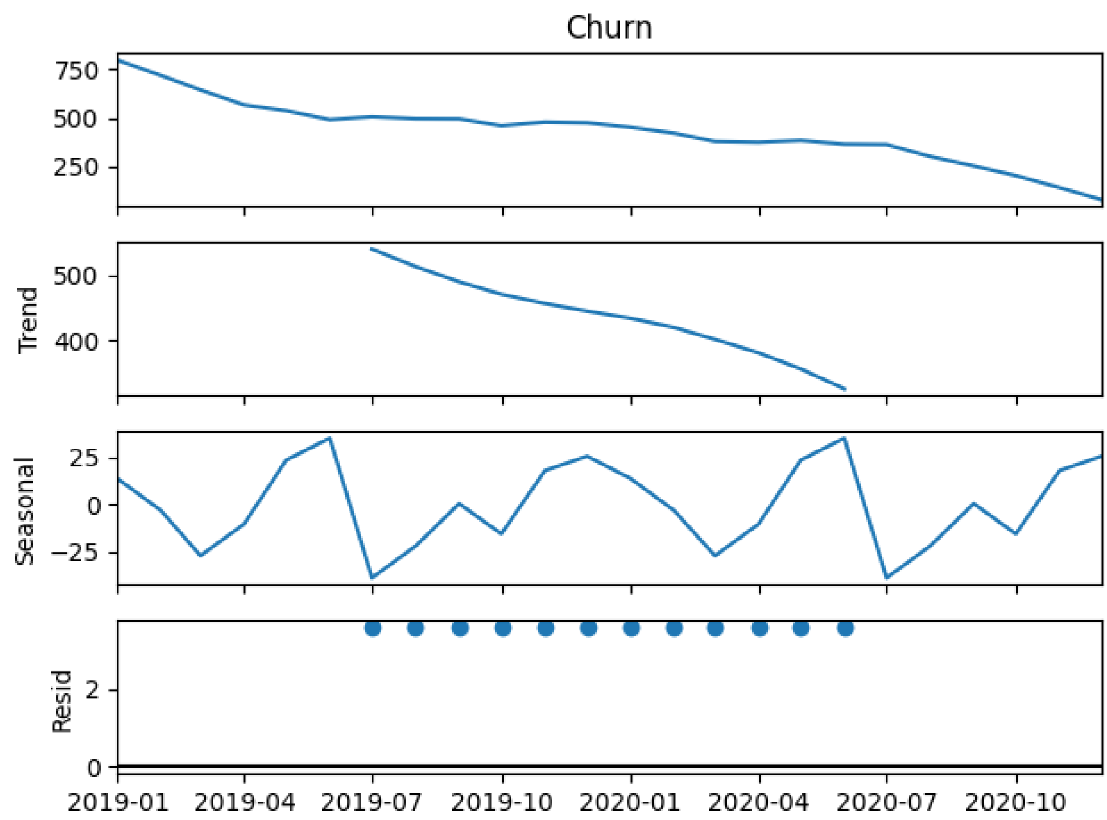
[87]:
```python
# Convert to category
categorical_columns = ['Gender', 'City', 'Education_Level', 'Joining␣
 ↪Designation', 'Grade']
df[categorical_columns] = df[categorical_columns].astype('category')
df['Gender'].replace({0.0:'Male', 1.0: 'Female'}, inplace=True)
df['Education_Level'].replace({0:'10+', 1:'12+', 2:'Graduate'}, inplace=True)

# Convert to datetime
df['MMM-YY'] = pd.to_datetime(df['MMM-YY'], format='%m/%d/%y')
df['Dateofjoining'] = pd.to_datetime(df['Dateofjoining'], format='%d/%m/%y')
df['LastWorkingDate'] = pd.to_datetime(df['LastWorkingDate'], format='%d/%m/%y')

# Rename 'MMM-YY' to 'ReportingMonthYear'
df.rename(columns={'MMM-YY':'ReportingMonthYear'}, inplace=True)
df['ReportingMonthYear'] = df['ReportingMonthYear'].dt.to_period('M')
df['ReportingYear'] = df['ReportingMonthYear'].dt.year

# Extract month and year from 'Dateofjoining'
df['Monthofjoining'] = df['Dateofjoining'].dt.month
df['Yearofjoining'] = df['Dateofjoining'].dt.year

# Find drivers who haved churned
df['Churn'] = df.groupby('Driver_ID')['LastWorkingDate'].transform('last')
df['Churn'] = df['Churn'].apply(lambda x: 0 if pd.isnull(x) else 1)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   ReportingMonthYear    19104 non-null  period[M]
 1   Driver_ID             19104 non-null  int64
 2   Age                   19043 non-null  float64
 3   Gender                19052 non-null  category
 4   City                  19104 non-null  category
 5   Education_Level       19104 non-null  category
 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  category
 10  Grade                 19104 non-null  category
 11  Total Business Value  19104 non-null  int64
 12  Quarterly Rating      19104 non-null  int64
 13  ReportingYear         19104 non-null  int64
 14  Monthofjoining        19104 non-null  int32
 15  Yearofjoining         19104 non-null  int32
 16  Churn                 19104 non-null  int64
dtypes: category(5), datetime64[ns](2), float64(1), int32(2), int64(6),
period[M](1)
memory usage: 1.7 MB
```

0.1 Time Series Features

[88]:
```
df_=df.copy()
df['ReportingMonthYear'] = pd.to_datetime(df['ReportingMonthYear'].dt.
 ↪to_timestamp())
df.set_index('ReportingMonthYear', inplace=True)
monthly_churn = df.groupby('ReportingMonthYear')['Churn'].sum()
decomposition = seasonal_decompose(monthly_churn, model='additive', period=12)
decomposition.plot()
plt.show()

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

df['trend'] = trend
df['seasonal'] = seasonal
df['residual'] = residual

df.fillna(method='bfill', inplace=True)
df=df.reset_index()
```



[89]:
```
df.head()
```

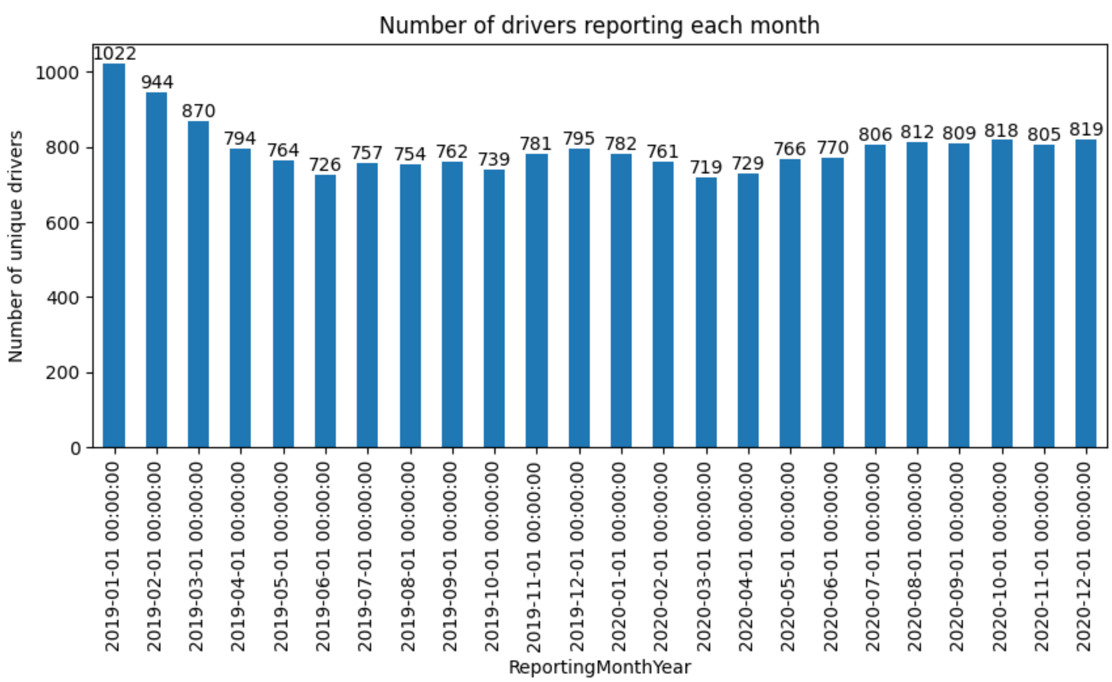
[89]:
```
  ReportingMonthYear  Driver_ID   Age Gender City Education_Level  Income  \
0         2019-01-01          1  28.0   Male  C23        Graduate   57387
1         2019-02-01          1  28.0   Male  C23        Graduate   57387
2         2019-03-01          1  28.0   Male  C23        Graduate   57387
3         2020-11-01          2  31.0   Male   C7        Graduate   67016
4         2020-12-01          2  31.0   Male   C7        Graduate   67016

  Dateofjoining LastWorkingDate  Joining Designation  Grade  \
0    2018-12-24      2019-11-03                    1      1
1    2018-12-24      2019-11-03                    1      1
2    2018-12-24      2019-11-03                    1      1
3    2020-06-11      2020-04-27                    2      2
4    2020-06-11      2020-04-27                    2      2

   Total Business Value  Quarterly Rating  ReportingYear  Monthofjoining  \
0               2381060                 2           2019              12
1               -665480                 2           2019              12
2                     0                 2           2019              12
3                     0                 1           2020               6
```

```
4                     0                1          2020                 6

   Yearofjoining  Churn        trend   seasonal  residual
0           2018      1   444.833333  13.774306  3.600694
1           2018      1   444.833333  -3.184028  3.600694
2           2018      1   444.833333 -27.017361  3.600694
3           2020      0   444.833333  17.940972  3.600694
4           2020      0   444.833333  25.565972  3.600694
```

1 Exploratory Data Analysis

1.1 Univariate analysis

[90]:
```python
plt.figure(figsize=(10,4))
temp_df = df.groupby('ReportingMonthYear')['Driver_ID'].nunique()
ax = temp_df.plot(kind='bar')
ax.bar_label(ax.containers[0])
plt.ylabel('Number of unique drivers')
plt.title('Number of drivers reporting each month')
plt.show()
```

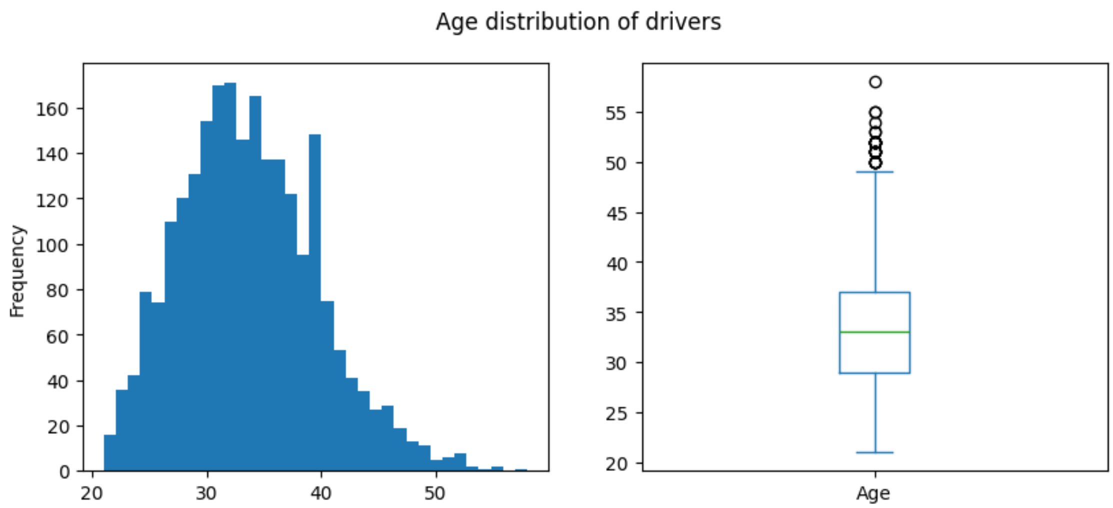


1.1.1 Insight

- The **month** during which **maximum** number of **drivers reported is January 2019**. A total of **1022 drivers** reported on January 2019

- It then dropeed every month after January and has been stagnant at around 800 drivers reported every month

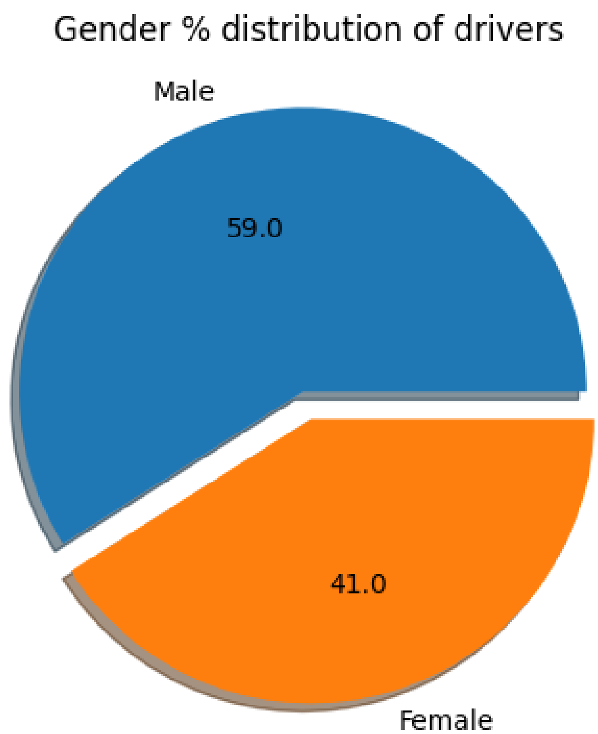
[91]:
```
fig, axs = plt.subplots(1,2,figsize=(10,4))
temp_df = df.groupby('Driver_ID').agg({'Age':'last'})['Age']
temp_df.plot(ax=axs[0], kind='hist', bins=35)
temp_df.plot(ax=axs[1], kind='box')
fig.suptitle('Age distribution of drivers')
plt.show()
```



1.1.2 Insight

- There are drivers from different **age** groups ranging from **21 to 58 years**
- **Most** of the drivers are in the age group of **30 to 35**
- The distribution is mostly **normal** with **little skewness** towards the **right**

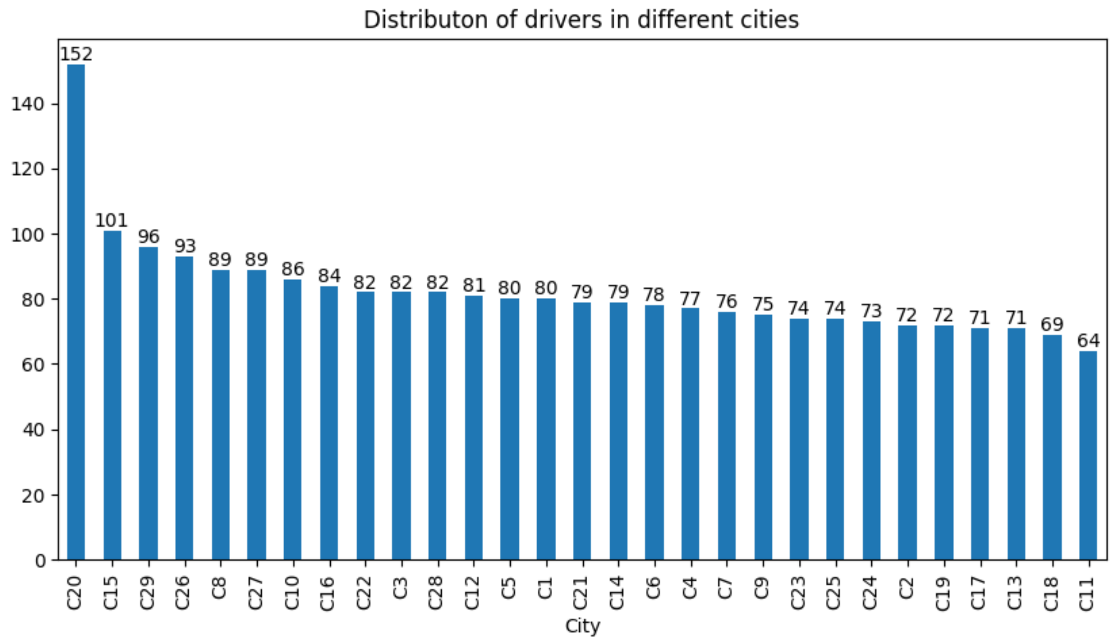
[92]:
```
temp_df = df.groupby('Driver_ID').agg({'Gender':'first'})
temp_df['Gender'].value_counts().plot(kind='pie', autopct='%.1f', shadow=True,
    explode=[.07,.03])
plt.title('Gender % distribution of drivers')
plt.ylabel('')
plt.show()
```



1.1.3 Insight

- **59%** of the drivers are **Male** and remaining **41%** are **Female**

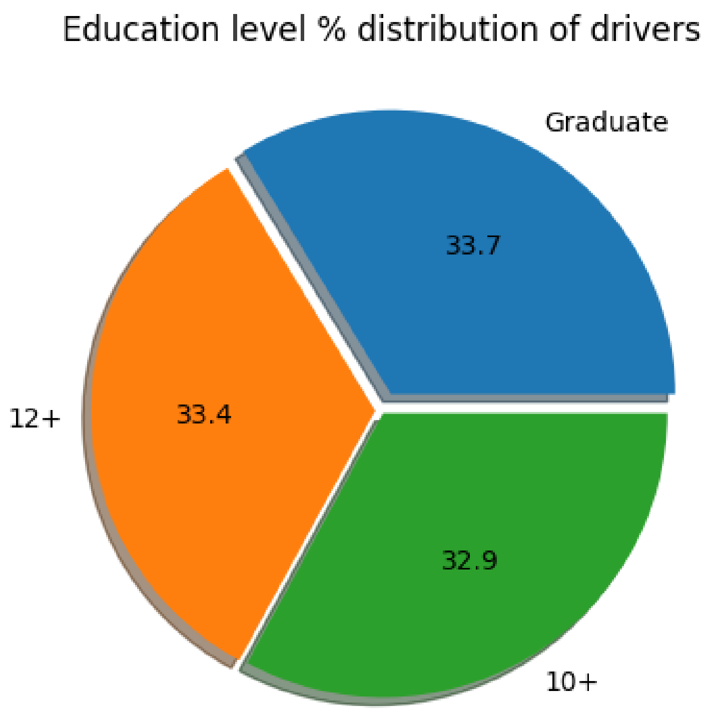
[93]:
```
plt.figure(figsize=(10,5))
temp_df = df.groupby('Driver_ID').agg({'City':'first'})
ax = temp_df['City'].value_counts().plot(kind='bar')
ax.bar_label(ax.containers[0])
plt.title('Distributon of drivers in different cities')
plt.show()
```



1.1.4 Insight

- City **C20** has the **maximum** number of **drivers** followed by city C15

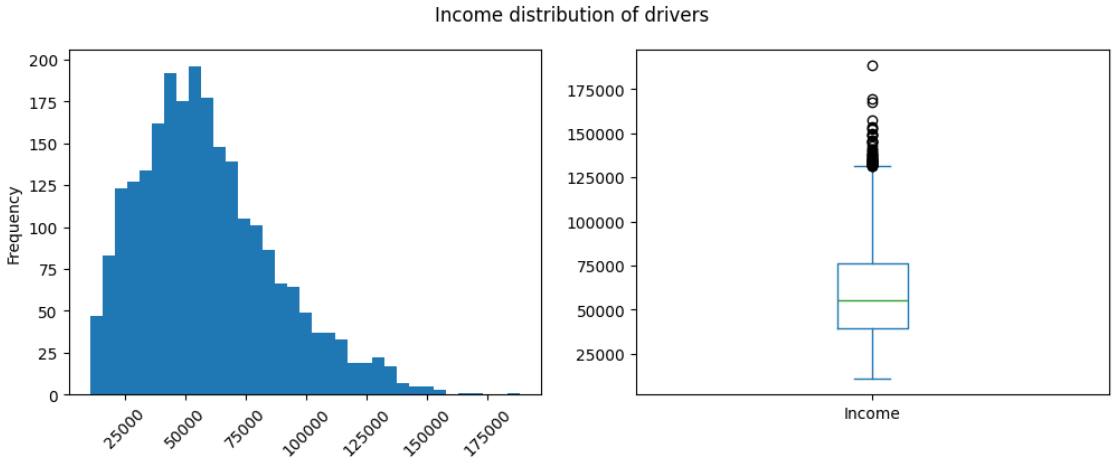
[94]:
```
temp_df = df.groupby('Driver_ID').agg({'Education_Level':'first'})
temp_df['Education_Level'].value_counts().plot(kind='pie', autopct='%.1f',␣
↪shadow=True, explode=[.067,.022,.011])
plt.ylabel('')
plt.title('Education level % distribution of drivers')
plt.show()
```



1.1.5 Insight

- Almost equal proportion of drivers are from the 3 different education level

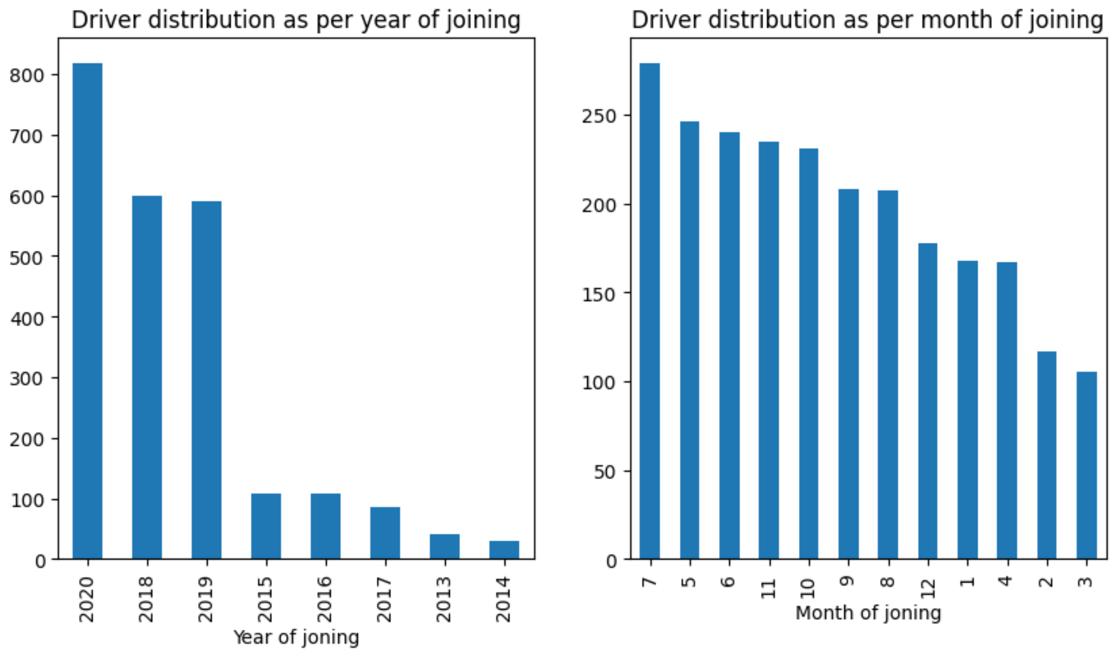
[95]:
```
fig, axs = plt.subplots(1,2,figsize=(12,4))
temp_df = df.groupby('Driver_ID').agg({'Income':'last'})['Income']
temp_df.plot(ax=axs[0], kind='hist', bins=35, rot=45)
temp_df.plot(ax=axs[1], kind='box')
fig.suptitle('Income distribution of drivers')
plt.show()
```



1.1.6 Insight

- Most of the drivers have an average monthly income of **40k to 75k**
- The distribution is **right skewed**

[96]:
```
temp_df = df.groupby('Driver_ID').agg({'Dateofjoining':
 ↪'first'})['Dateofjoining']
fig, axs = plt.subplots(1,2,figsize=(10,5))
temp_df.dt.year.value_counts().plot(kind='bar', ax=axs[0], xlabel='Year of␣
 ↪joning', title='Driver distribution as per year of joining')
temp_df.dt.month.value_counts().plot(kind='bar', ax=axs[1], xlabel='Month of␣
 ↪joning', title='Driver distribution as per month of joining')
plt.show()
```

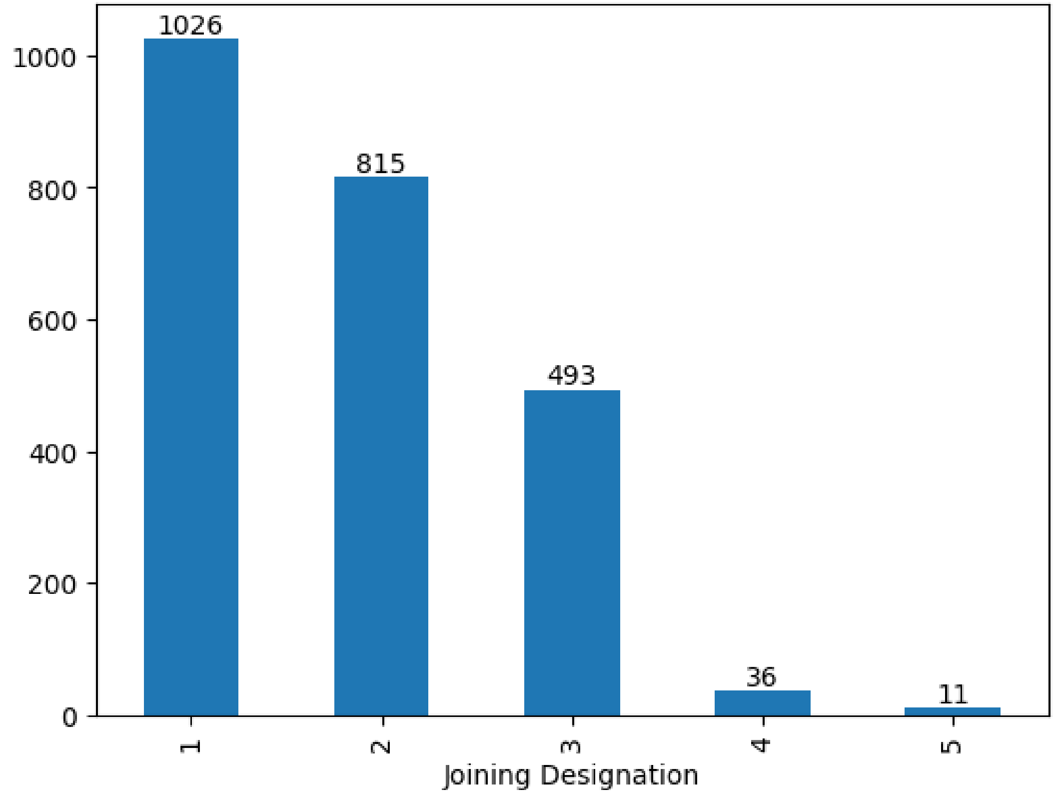


1.1.7 Insight

- Maximum number of drivers joined in the year **2020**
- Maximum number of drivers joined in the month of **July**

[97]:

```
ax = df.groupby('Driver_ID').agg({'Joining Designation':'first'})['Joining Designation'].value_counts().plot(kind='bar')
ax.bar_label(ax.containers[0])
plt.show()
```

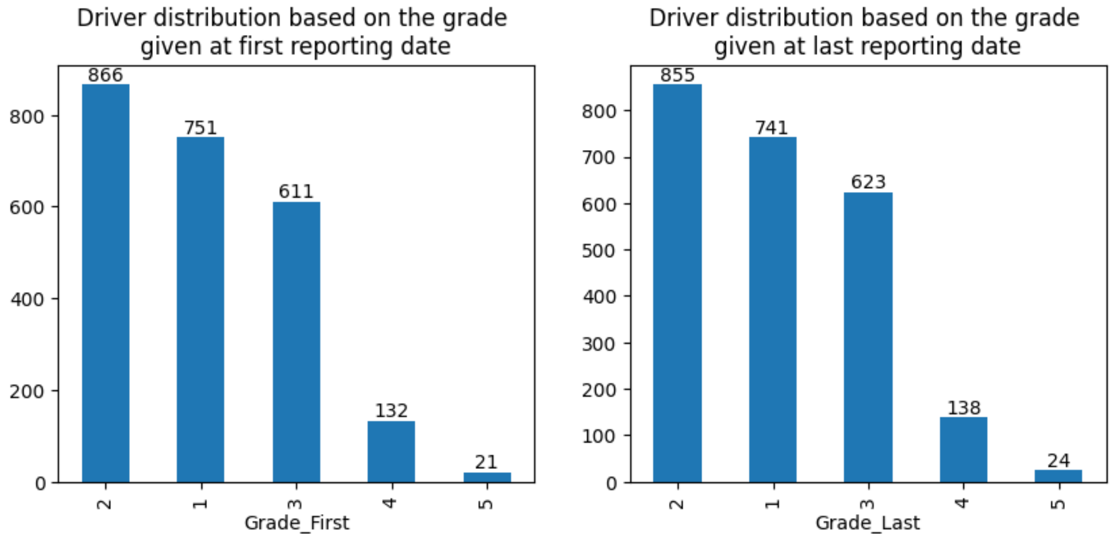


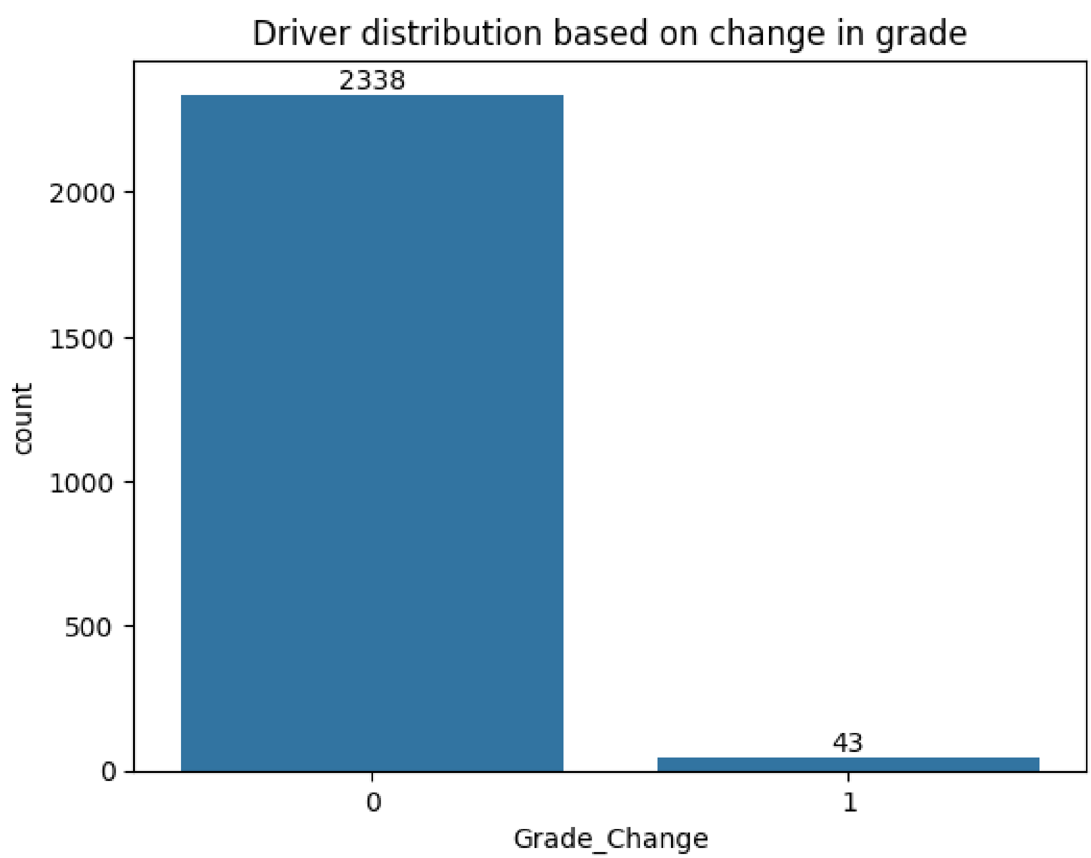
1.1.8 Insight

- Maximum number of drivers, 1026, have a **joining designation of 1**

[98]:
```
temp_df_1 = df.groupby('Driver_ID').agg({'Grade':'first'}).reset_index()
temp_df_1.rename(columns = {'Grade':'Grade_First'}, inplace=True)
temp_df_2 = df.groupby('Driver_ID').agg({'Grade':'last'}).reset_index()
temp_df_2.rename(columns = {'Grade':'Grade_Last'}, inplace=True)
temp_df = pd.merge(temp_df_1, temp_df_2, on='Driver_ID')
temp_df['Grade_Change'] = temp_df['Grade_Last'].astype('int') -␣
↪temp_df['Grade_First'].astype('int')
fig, axs = plt.subplots(1,2,figsize=(10,4))
ax = temp_df['Grade_First'].value_counts().plot(kind='bar', ax=axs[0],␣
↪title='Driver distribution based on the grade \ngiven at first reporting␣
↪date')
ax.bar_label(ax.containers[0])
ax = temp_df['Grade_Last'].value_counts().plot(kind='bar', ax=axs[1],␣
↪title='Driver distribution based on the grade \ngiven at last reporting␣
↪date')
ax.bar_label(ax.containers[0])
plt.show()
```

```python
ax = sns.countplot(data=temp_df, x = 'Grade_Change')
ax.set_title('Driver distribution based on change in grade')
ax.bar_label(ax.containers[0])
plt.show()
```

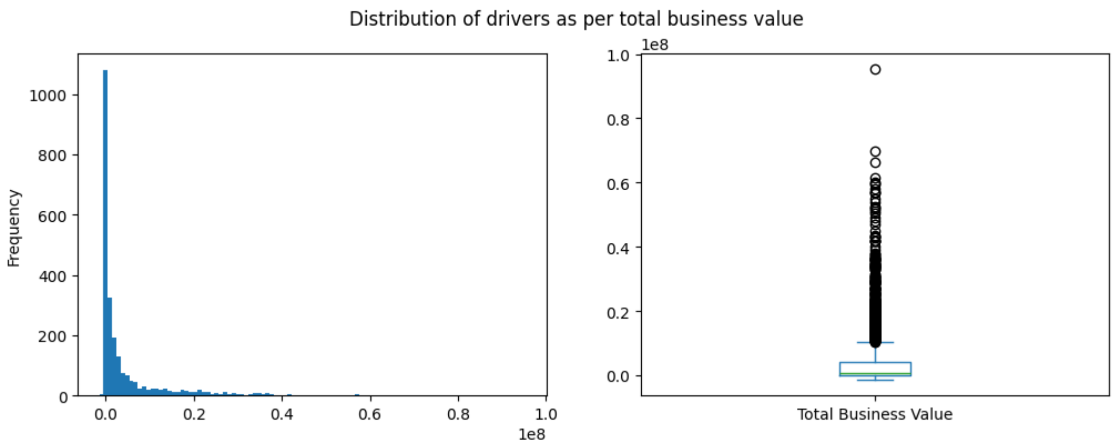




1.1.9 Insight

- Maximum number of drivers have a **grade of 2** and it doesnt change for the majority of the drivers

[99]:
```
fig, axs = plt.subplots(1,2,figsize=(12,4))
temp_df = df.groupby('Driver_ID').agg({'Total Business Value':'sum'})['Total␣
↪Business Value']
temp_df.plot(ax=axs[0], kind='hist', bins=100)
temp_df.plot(ax=axs[1], kind='box')
fig.suptitle('Distribution of drivers as per total business value')
plt.show()
```

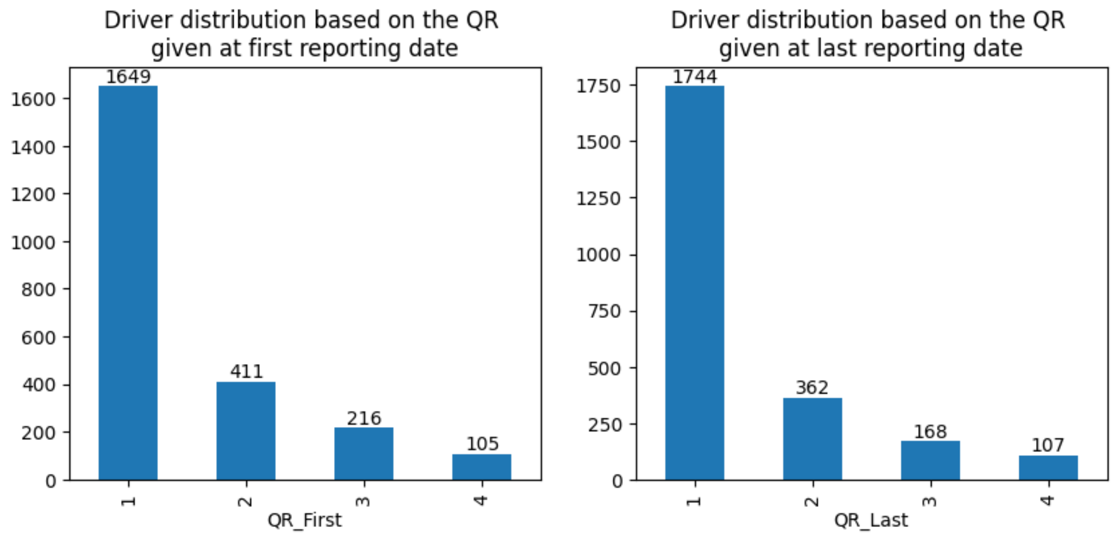


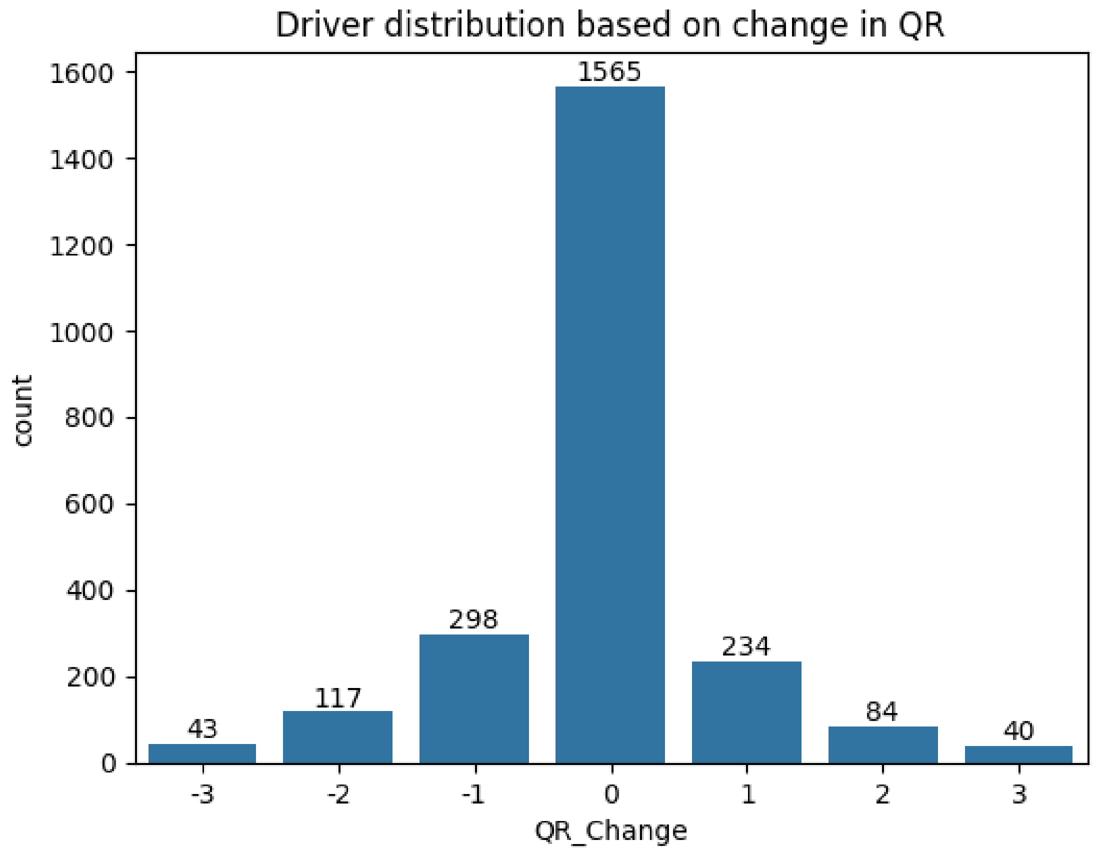
1.1.10 Insight

- It is very evident that **many drivers** have a **total business value of 0** and there are also a few drivers who have a -ve business value
- The distribution is extremely **right skewed**

[100]:
```
temp_df_1 = df.groupby('Driver_ID').agg({'Quarterly Rating':'first'}).
↪reset_index()
temp_df_1.rename(columns = {'Quarterly Rating':'QR_First'}, inplace=True)
temp_df_2 = df.groupby('Driver_ID').agg({'Quarterly Rating':'last'}).
↪reset_index()
temp_df_2.rename(columns = {'Quarterly Rating':'QR_Last'}, inplace=True)
temp_df = pd.merge(temp_df_1, temp_df_2, on='Driver_ID')
temp_df['QR_Change'] = temp_df['QR_Last'].astype('int') - temp_df['QR_First'].
↪astype('int')
fig, axs = plt.subplots(1,2,figsize=(10,4))
```

```
ax = temp_df['QR_First'].value_counts().plot(kind='bar', ax=axs[0],␣
 ↪title='Driver distribution based on the QR \ngiven at first reporting date')
ax.bar_label(ax.containers[0])
ax = temp_df['QR_Last'].value_counts().plot(kind='bar', ax=axs[1],␣
 ↪title='Driver distribution based on the QR \ngiven at last reporting date')
ax.bar_label(ax.containers[0])
plt.show()
ax = sns.countplot(data=temp_df, x = 'QR_Change')
ax.set_title('Driver distribution based on change in QR')
ax.bar_label(ax.containers[0])
plt.show()
```

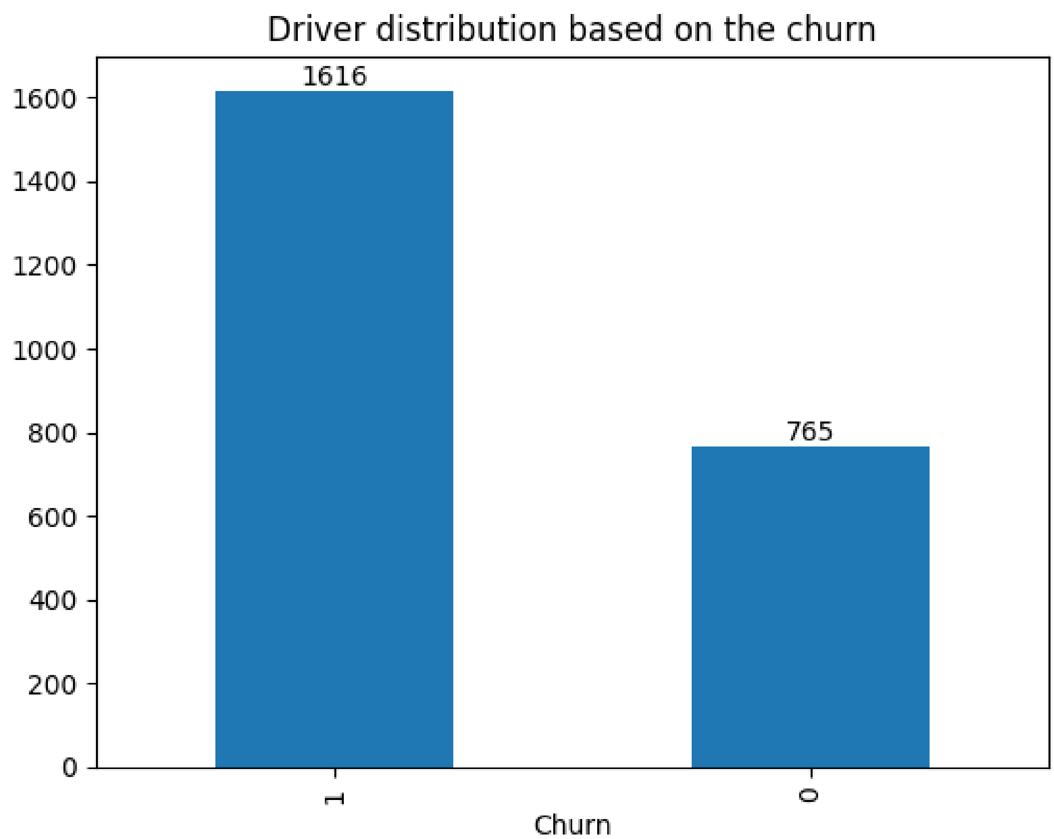




1.1.11 Insight

- **Majority** of the drivers have a very **low quarterly rating of 1**
- The change in QR plot shows that **majority** of the drivers **don't see a change in their QR** but there are **decent number** of drivers with **positive change in QR** and equally decent number of drivers with **negative change in QR**
- There are **no drivers** with QR of **5**

[101]:
```
temp_df = df.groupby('Driver_ID').agg({'Churn':'first'})['Churn']
ax = temp_df.value_counts().plot(kind='bar', title='Driver distribution based␣
↪on the churn')
ax.bar_label(ax.containers[0])
plt.show()
(temp_df.value_counts(normalize=True)*100).round(0)
```



[101]:

```
Churn
1    68.0
0    32.0
Name: proportion, dtype: float64
```

1.2 Bivariate analysis

[102]:

```python
df.Grade=df.Grade.astype('int')
driver_df = df.groupby('Driver_ID').agg({
    'ReportingMonthYear' : len,
    'Age' : 'last',
    'Gender' : 'first',
    'City' : 'first',
    'Education_Level' : 'first',
    'Income' : 'median',
    'Dateofjoining' : 'first',
    'LastWorkingDate' : 'last',
    'Joining Designation' : 'first',
    'Grade' : 'median',
    'Total Business Value' : 'sum',
```

```
    'Quarterly Rating' : 'median',
    'Churn':'last',
    'trend':'mean',
    'seasonal':'mean',
    'residual':'mean',
    #'sin_quarter':'median',
    #'cos_quarter':'median'
}).reset_index()
driver_df.rename(columns={'ReportingMonthYear': 'Months of Service'},␣
 ↪inplace=True)
driver_df.head(10)
```

[102]:

```
   Driver_ID  Months of Service   Age  Gender City Education_Level    Income  \
0          1                  3  28.0    Male  C23        Graduate   57387.0
1          2                  2  31.0    Male   C7        Graduate   67016.0
2          4                  5  43.0    Male  C13        Graduate   65603.0
3          5                  3  29.0    Male   C9             10+   46368.0
4          6                  5  31.0  Female  C11             12+   78728.0
5          8                  3  34.0    Male   C2             10+   70656.0
6         11                  1  28.0  Female  C19        Graduate   42172.0
7         12                  6  35.0    Male  C23        Graduate   28116.0
8         13                 23  31.0    Male  C19        Graduate  119227.0
9         14                  3  39.0  Female  C26             10+   19734.0

  Dateofjoining LastWorkingDate Joining Designation  Grade  \
0    2018-12-24      2019-11-03                   1    1.0
1    2020-06-11      2020-04-27                   2    2.0
2    2019-07-12      2020-04-27                   2    2.0
3    2019-09-01      2019-07-03                   1    1.0
4    2020-07-31      2020-11-15                   3    3.0
5    2020-09-19      2020-11-15                   3    3.0
6    2020-07-12      2019-12-21                   1    1.0
7    2019-06-29      2019-12-21                   1    1.0
8    2015-05-28      2020-11-25                   1    4.0
9    2020-10-16      2019-02-22                   3    3.0

   Total Business Value  Quarterly Rating  Churn       trend   seasonal  \
0               1715580               2.0      1  444.833333  -5.475694
1                     0               1.0      0  444.833333  21.753472
2                350000               1.0      1  416.016667  -0.217361
3                120360               1.0      1  540.041667  -5.475694
4               1265000               2.0      0  540.041667   1.340972
5                     0               1.0      1  540.041667   1.010417
6                     0               1.0      0  540.041667  25.565972
7               2607180               2.5      1  485.888889  -5.322917
8              10213040               1.0      1  485.750000  -1.111564
9                     0               1.0      0  540.041667   9.357639
```

```
   residual
0  3.600694
1  3.600694
2  3.600694
3  3.600694
4  3.600694
5  3.600694
6  3.600694
7  3.600694
8  3.600694
9  3.600694
```

[103]:
```python
drivers_with_2_year_service = driver_df[driver_df['Months of Service'] ==
 ↪24]['Driver_ID'].reset_index(drop=True)
```

[104]:
```python
def calculate_change(df, column_name):
    temp_df_1 = df.groupby('Driver_ID').agg({column_name:'first'}).reset_index()
    first_column_name = column_name+'_First'
    temp_df_1.rename(columns = {column_name:first_column_name}, inplace=True)
    temp_df_2 = df.groupby('Driver_ID').agg({column_name:'last'}).reset_index()
    last_column_name = column_name+'_Last'
    temp_df_2.rename(columns = {column_name:last_column_name}, inplace=True)
    temp_df = pd.merge(temp_df_1, temp_df_2, on='Driver_ID')
    temp_df[column_name+'_Change'] = temp_df[last_column_name].astype('int') -
 ↪temp_df[first_column_name].astype('int')
    temp_df.drop(columns=[first_column_name, last_column_name], inplace=True)
    return temp_df
```

[105]:
```python
column_name = 'Income'
temp_df1 = calculate_change(df, 'Income')
driver_df = pd.merge(driver_df, temp_df1, on='Driver_ID')
temp_df2 = calculate_change(df, 'Grade')
driver_df = pd.merge(driver_df, temp_df2, on='Driver_ID')
temp_df3 = calculate_change(df, 'Quarterly Rating')
driver_df = pd.merge(driver_df, temp_df3, on='Driver_ID')
driver_df['Quarterly Rating Improved'] = driver_df['Quarterly Rating_Change'].
 ↪apply(lambda x: 1 if x>0 else 0)
driver_df.head()
```

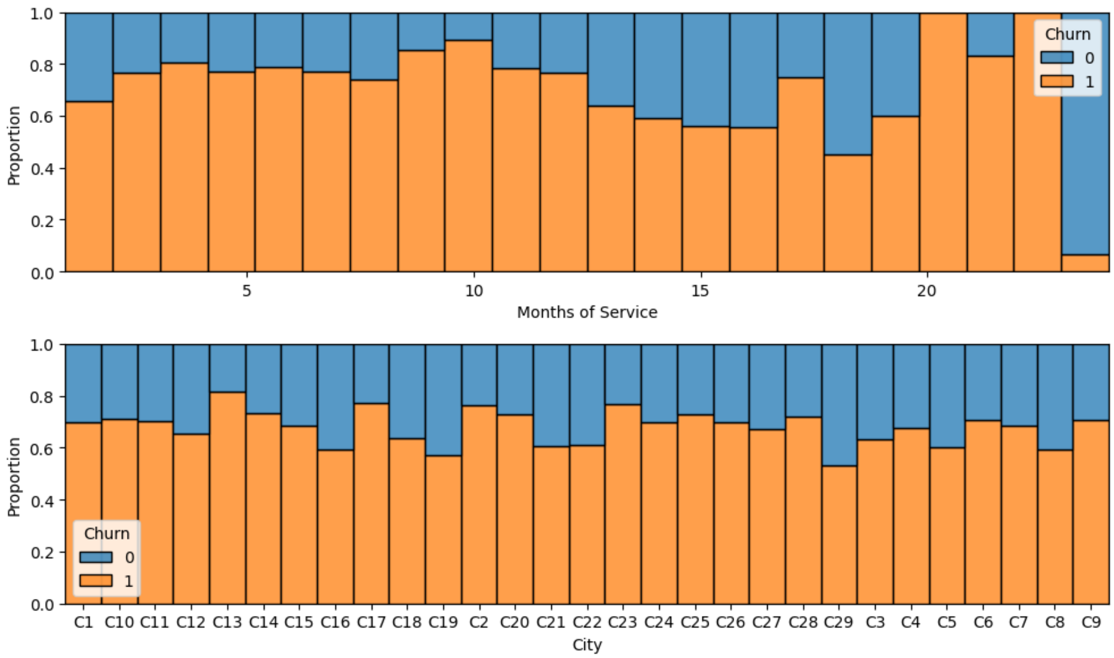
[105]:

	Driver_ID	Months of Service	Age	Gender	City	Education_Level	Income	\
0	1	3	28.0	Male	C23	Graduate	57387.0	
1	2	2	31.0	Male	C7	Graduate	67016.0	
2	4	5	43.0	Male	C13	Graduate	65603.0	
3	5	3	29.0	Male	C9	10+	46368.0	
4	6	5	31.0	Female	C11	12+	78728.0	

```
  Dateofjoining LastWorkingDate Joining Designation  …  \
0    2018-12-24      2019-11-03                   1  …
1    2020-06-11      2020-04-27                   2  …
2    2019-07-12      2020-04-27                   2  …
3    2019-09-01      2019-07-03                   1  …
4    2020-07-31      2020-11-15                   3  …

   Total Business Value  Quarterly Rating  Churn       trend   seasonal  \
0               1715580               2.0      1  444.833333  -5.475694
1                     0               1.0      0  444.833333  21.753472
2                350000               1.0      1  416.016667  -0.217361
3                120360               1.0      1  540.041667  -5.475694
4               1265000               2.0      0  540.041667   1.340972

   residual  Income_Change  Grade_Change  Quarterly Rating_Change  \
0  3.600694              0             0                        0
1  3.600694              0             0                        0
2  3.600694              0             0                        0
3  3.600694              0             0                        0
4  3.600694              0             0                        1

   Quarterly Rating Improved
0                          0
1                          0
2                          0
3                          0
4                          1

[5 rows x 21 columns]
```

[106]:
```
driver_df['Income_Raise'] = driver_df['Income_Change'].apply(lambda x: 1 if x>0␣
↪else 0)
```

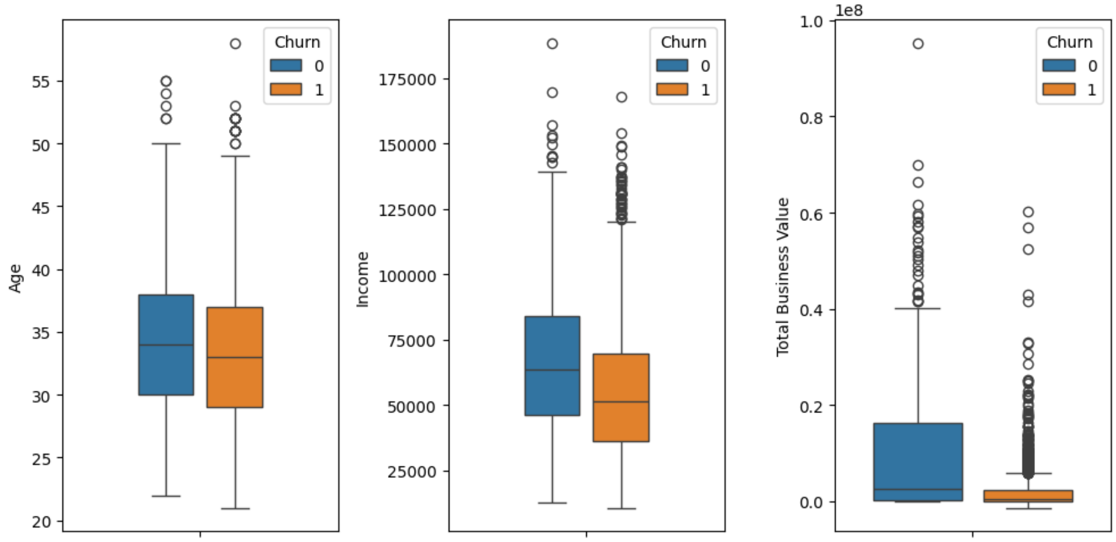
[107]:
```
fig, axs = plt.subplots(2,1,figsize=(10,6))
sns.histplot(ax = axs[0], data=driver_df, x='Months of Service', hue='Churn',␣
↪stat="proportion", multiple="fill")
sns.histplot(ax = axs[1], data=driver_df, x='City', hue='Churn',␣
↪stat="proportion", multiple="fill")
plt.tight_layout()
plt.show()
```



1.2.1 Insight

- The **churn** rate is generally **higher** in drivers with **less months of service** and low in drivers with longer months of service with exception for 21, 22 and 23 months of service where the churn rates seems to be very high
- The city **C13** has the **highest churn rate** and city **C29** has the **lowest churn rate**

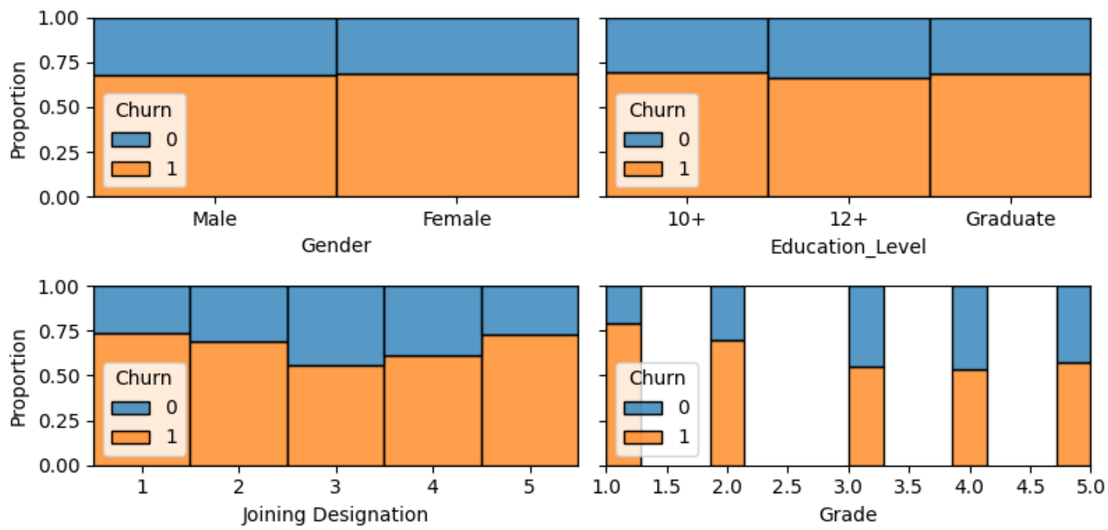
[108]:
```
fig, axs = plt.subplots(1,3,figsize=(10,5))
sns.boxplot(ax=axs[0], data=driver_df, y='Age', hue='Churn', width=0.5, gap=0.2)
sns.boxplot(ax=axs[1], data=driver_df, y='Income', hue='Churn', width=0.5,␣
 ↪gap=0.2)
sns.boxplot(ax=axs[2], data=driver_df, y='Total Business Value', hue='Churn',␣
 ↪gap=0.2)
plt.tight_layout()
plt.show()
```



1.2.2 Insight

- The **median age** of drivers who have **churned** is **slighly lesser** than that of the drivers who have not churned
- The **median income** of drivers who have **churned** is **lesser** than that of the drivers who have not churned
- The **median Total Bussiness Value** of drivers who have **churned** is **lesser** than that of the drivers who have not churned
- The drivers who have **churned** also had **-ve Total Bussiness Value**

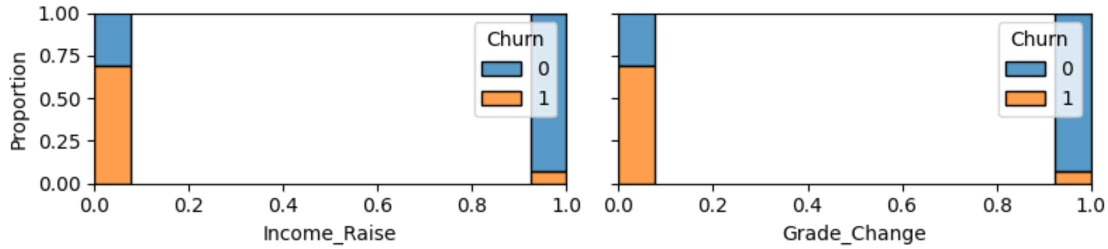
[109]:
```
fig, axs = plt.subplots(2,2,figsize=(8,4),sharey=True)
sns.histplot(ax=axs[0,0], data=driver_df, x='Gender', hue='Churn',␣
 ↪stat="proportion", multiple="fill")
sns.histplot(ax=axs[0,1], data=driver_df, x='Education_Level', hue='Churn',␣
 ↪stat='proportion', multiple='fill')
sns.histplot(ax=axs[1,0], data=driver_df, x='Joining Designation', hue='Churn',␣
 ↪stat='proportion', multiple='fill')
sns.histplot(ax=axs[1,1], data=driver_df, x='Grade', hue='Churn',␣
 ↪stat='proportion', multiple='fill')
plt.tight_layout()
plt.show()
```

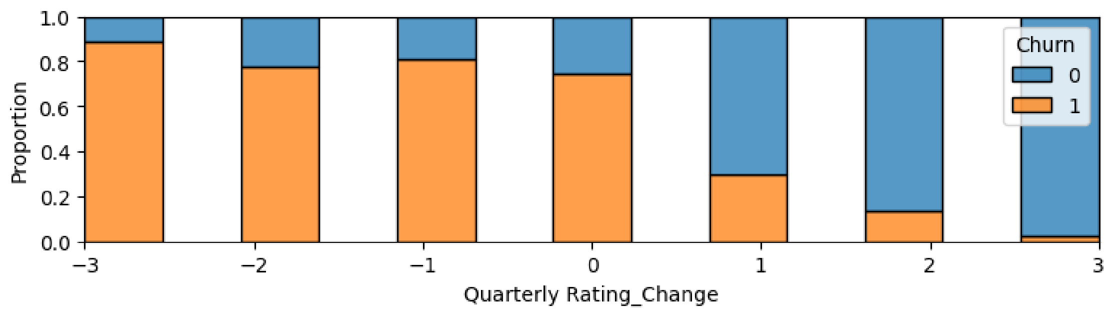


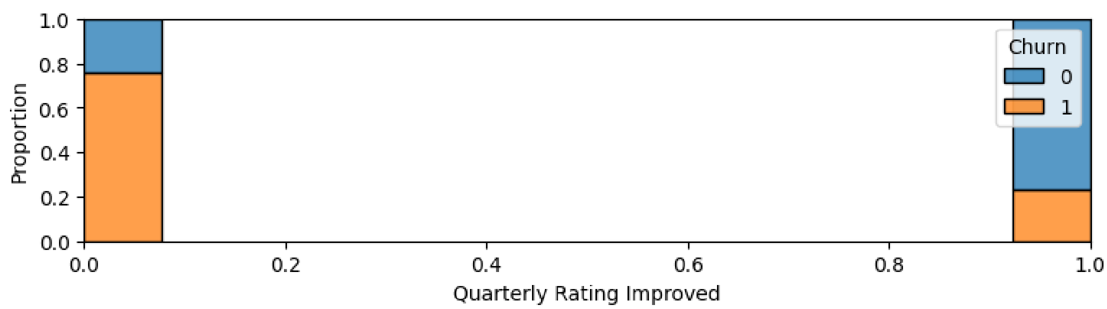
1.2.3 Insight

- The **churn** rate is **almost equal** in both **male and female** drivers
- The **churn** rate is **almost equal** in **10+ and Graduates** and slighly **lower** in **12+**
- The **churn** rate is **less** for **joining designation 3**
- The **churn** rate is **less** for **higher grades**

[110]:
```
fig, axs = plt.subplots(1,2,figsize=(8,2),sharey=True)
sns.histplot(ax=axs[0], data=driver_df, x='Income_Raise', hue='Churn',␣
  ↪stat='proportion', multiple='fill')
sns.histplot(ax=axs[1], data=driver_df, x='Grade_Change', hue='Churn',␣
  ↪stat='proportion', multiple='fill')
plt.tight_layout()
plt.show()
plt.figure(figsize=(9,2))
sns.histplot(data=driver_df, x='Quarterly Rating_Change', hue='Churn',␣
  ↪stat='proportion', multiple='fill')
plt.show()
plt.figure(figsize=(9,2))
sns.histplot(data=driver_df, x='Quarterly Rating Improved', hue='Churn',␣
  ↪stat='proportion', multiple='fill')
plt.show()
```





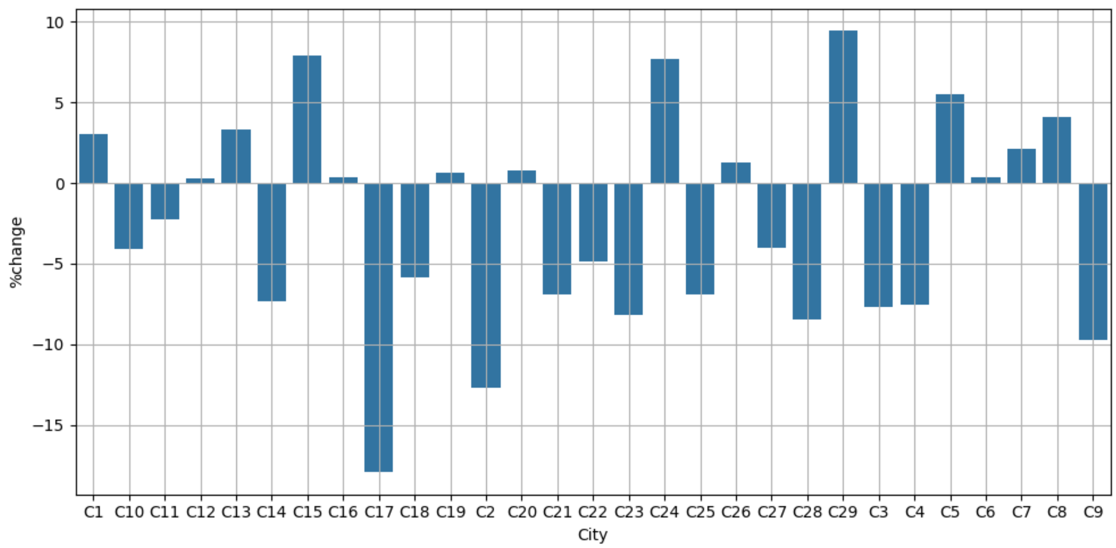


1.2.4 Insight

- The **churn rate** is **very less** in drivers whose **income has raised**
- The **churn rate** is **very less** in drivers whose **grade has raised**
- The **churn rate** is **very less** in drivers whose **Quarterly rating has increased**

[111]:
```
temp_df = df.groupby(['City', 'ReportingYear']).agg({'Quarterly Rating':␣
↪'mean'}).reset_index()
temp_df1 = pd.pivot_table(data=temp_df, index='City', columns='ReportingYear',␣
↪values='Quarterly Rating').reset_index()
temp_df1.rename(columns={'ReportingYear':'index', 2019:'2019', 2020:'2020'},␣
↪inplace=True)
```

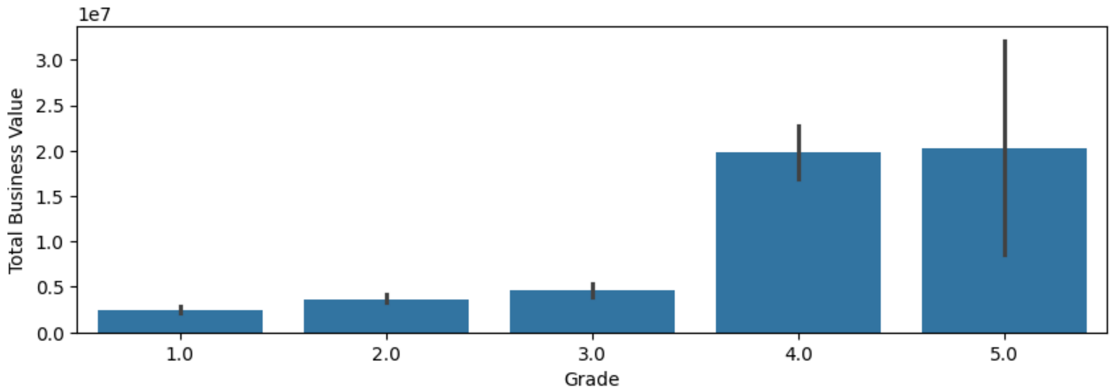
```
temp_df1['%change'] = (((temp_df1['2020'] - temp_df1['2019'])/
↪temp_df1['2019'])*100).round(2)
plt.figure(figsize=(10,5))
sns.barplot(data=temp_df1, x='City', y='%change')
plt.tight_layout()
plt.grid(True)
plt.show()
```



1.2.5 Insight

- The city **C29** shows most improvement in Quarterly Rating in 2020 compared to 2019

[112]:
```
plt.figure(figsize=(10,3))
sns.barplot(data=driver_df, x='Grade', y='Total Business Value',␣
↪estimator='mean')
plt.show()
print('Mean of Total Business Value of drivers with grade 5:',␣
↪driver_df[driver_df['Grade'] == 5]['Total Business Value'].sum())
```

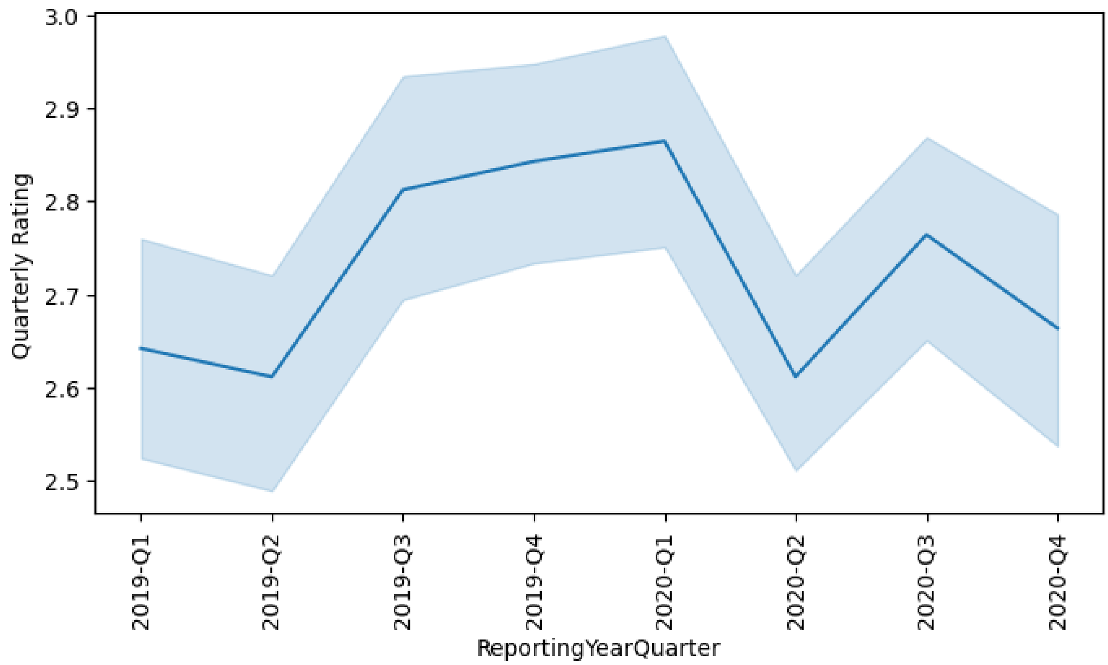



```
Mean of Total Business Value of drivers with grade 5: 424618700
```

1.2.6 Insight

- The mean of Total Business Value of drivers with grade 5 is higher than those with other grades

[113]:
```
def convert_to_year_quarter(x):
    year = str(x.year)
    month = x.month
    if(month >=1 and month <=3):
        return year+'-Q1'
    elif(month >=4 and month <=6):
        return year+'-Q2'
    elif(month >=7 and month <=9):
        return year+'-Q3'
    else:
        return year+'-Q4'

temp_df = df.copy()
temp_df['ReportingYearQuarter']=temp_df['ReportingMonthYear'].
 ↪apply(convert_to_year_quarter)
temp_df.head()
temp_driver_full_service_df = temp_df[temp_df['Driver_ID'].
 ↪isin(drivers_with_2_year_service)].groupby(['Driver_ID',␣
 ↪'ReportingYearQuarter']).agg({'Quarterly Rating':'last', 'Total Business␣
 ↪Value':'sum'}).reset_index()
plt.figure(figsize=(8,4))
sns.lineplot(data=temp_driver_full_service_df, x='ReportingYearQuarter',␣
 ↪y='Quarterly Rating')
plt.xticks(rotation=90)
plt.show()
```

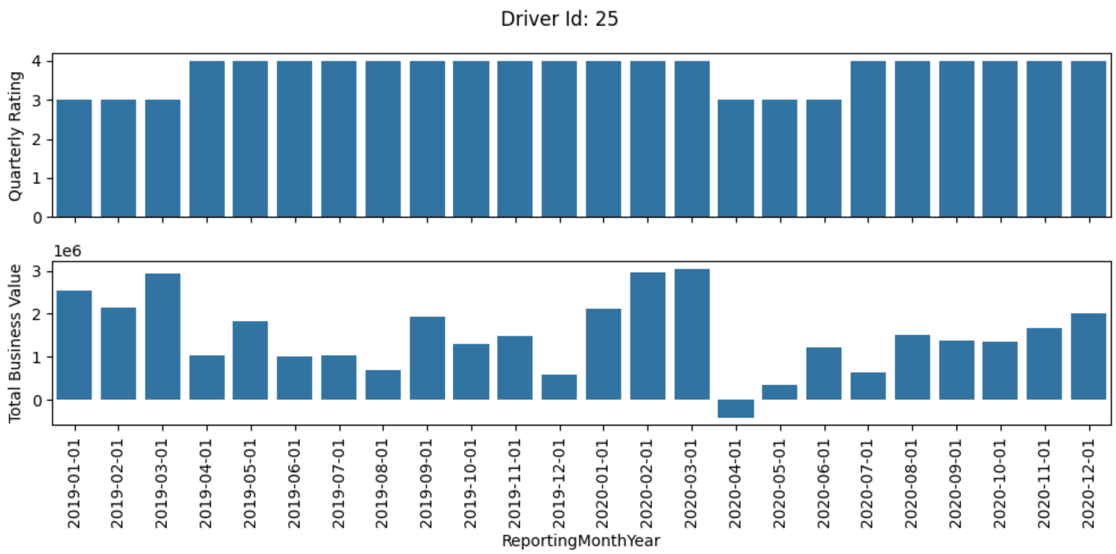


1.2.7 Insight

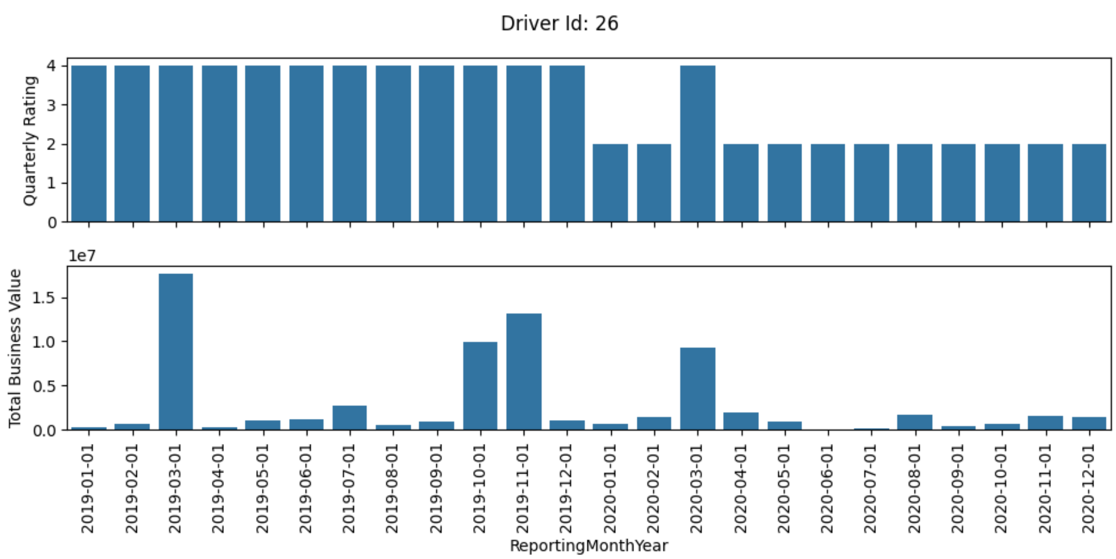
- There is a dip in the quarterly rating in Q2 and then it increases in Q3.
- This pattern can be osberved for both the years

[114]:
```
temp_driver_full_service_df = temp_df[temp_df['Driver_ID'].
 ↪isin(drivers_with_2_year_service)]
num_of_drivers = 20
count=0
for driver_id in temp_driver_full_service_df['Driver_ID'].unique():
    if(count < num_of_drivers):
        count = count + 1
        sample_df =␣
 ↪temp_driver_full_service_df[temp_driver_full_service_df['Driver_ID'] ==␣
 ↪driver_id]
        fig, axs = plt.subplots(2,1,figsize=(10, 5), sharex=True)
        sns.barplot(ax=axs[0], data=sample_df, x = 'ReportingMonthYear',␣
 ↪y='Quarterly Rating')
        axs[0].tick_params(axis='x', rotation=90)
        sns.barplot(ax=axs[1], data=sample_df, x = 'ReportingMonthYear',␣
 ↪y='Total Business Value')
        axs[1].tick_params(axis='x', rotation=90)
        fig.suptitle(f'Driver Id: {driver_id}')
        plt.tight_layout()
        plt.show()
```

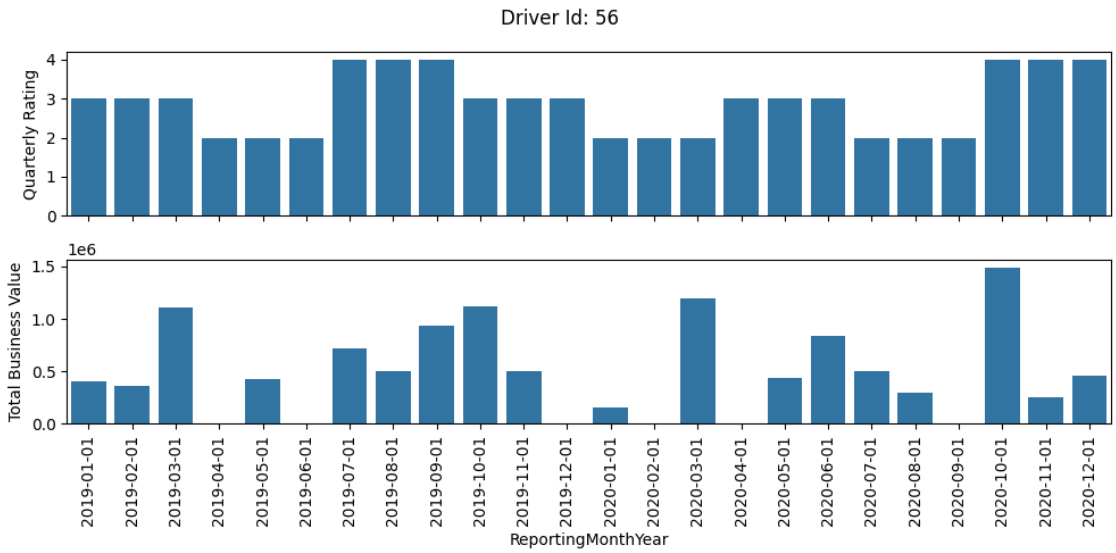
```
    else:
        break
```

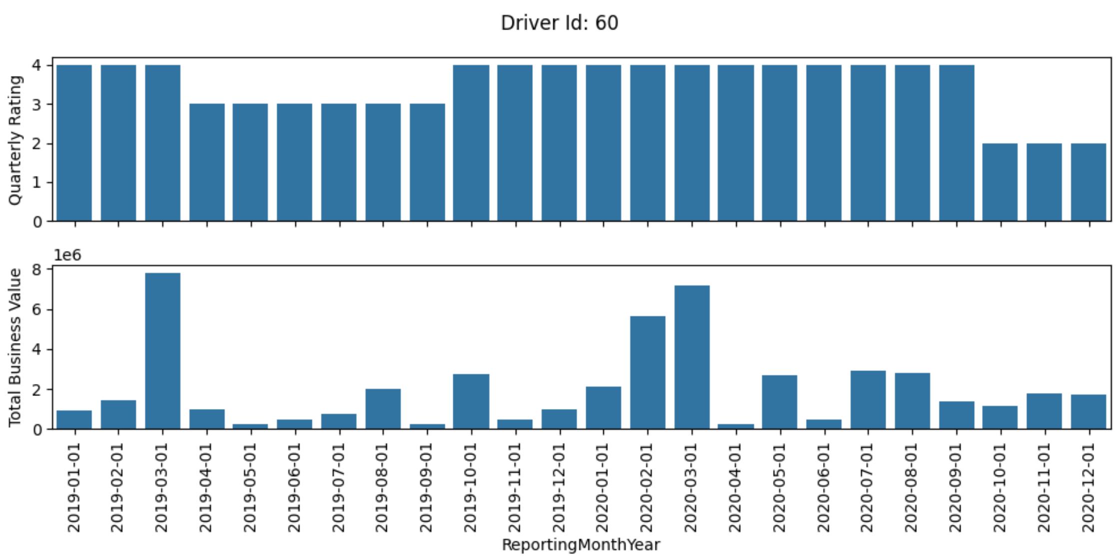


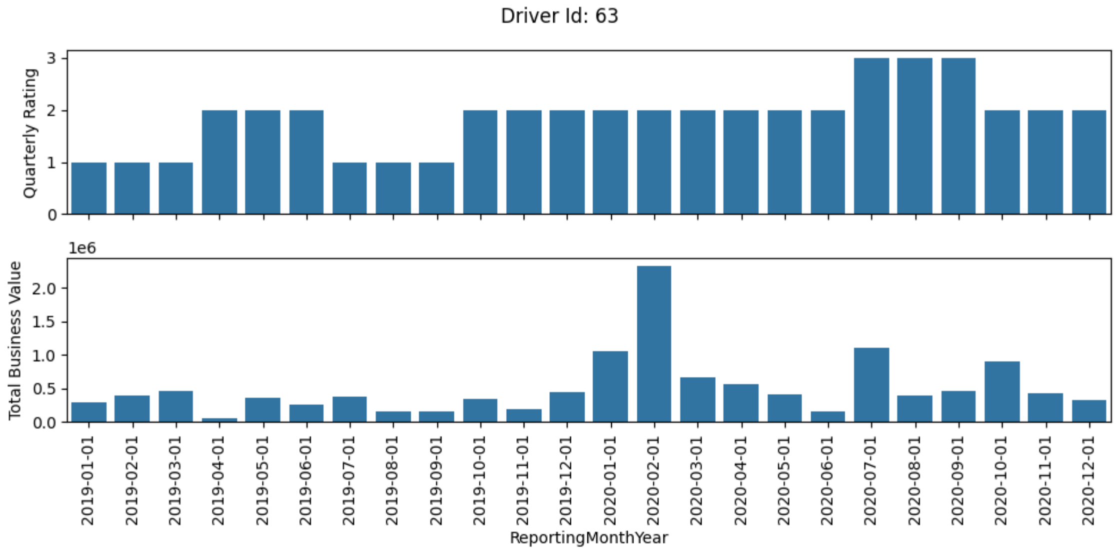
Driver Id: 25
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear

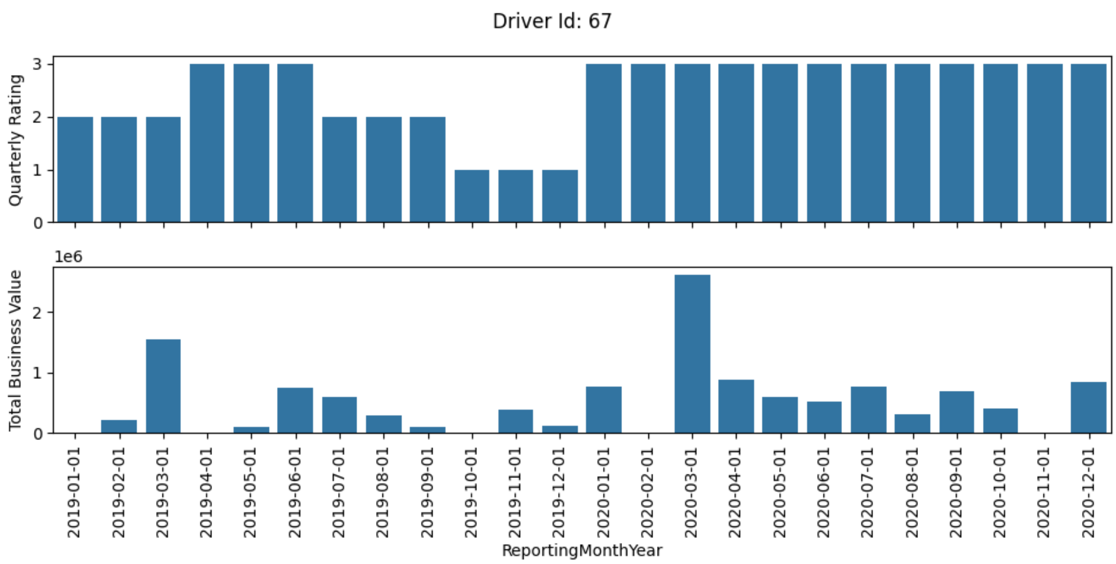


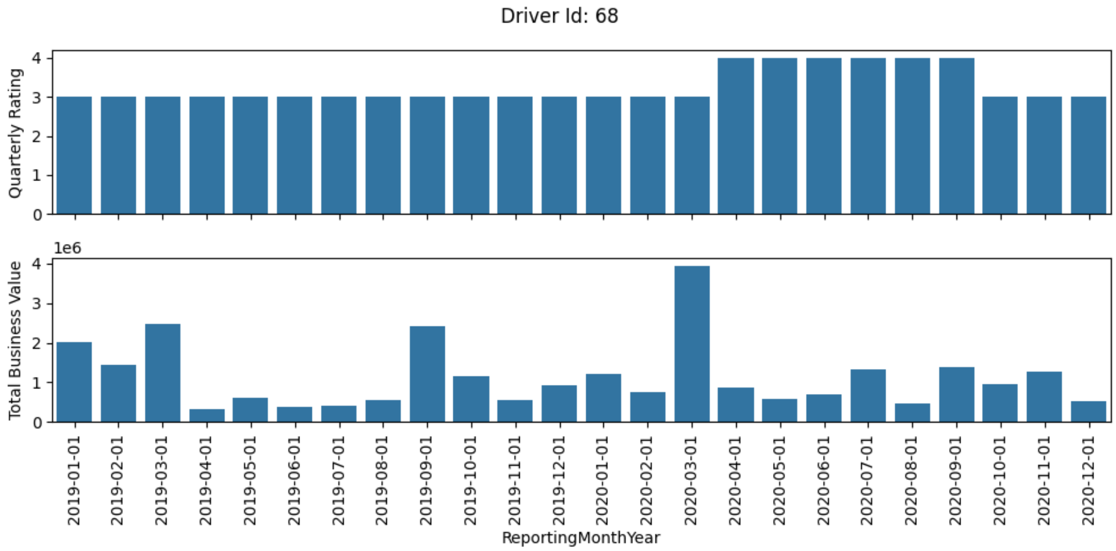
Driver Id: 26
Quarterly Rating
Total Business Value
1e7
ReportingMonthYear

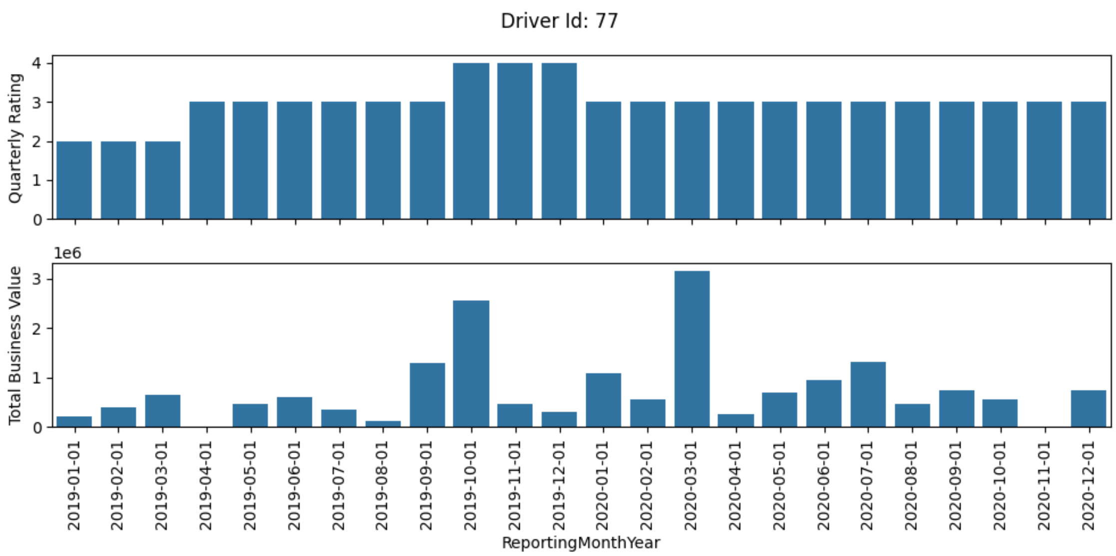

Driver Id: 56
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear
2019-01-01
2019-02-01
2019-03-01
2019-04-01
2019-05-01
2019-06-01
2019-07-01
2019-08-01
2019-09-01
2019-10-01
2019-11-01
2019-12-01
2020-01-01
2020-02-01
2020-03-01
2020-04-01
2020-05-01
2020-06-01
2020-07-01
2020-08-01
2020-09-01
2020-10-01
2020-11-01
2020-12-01

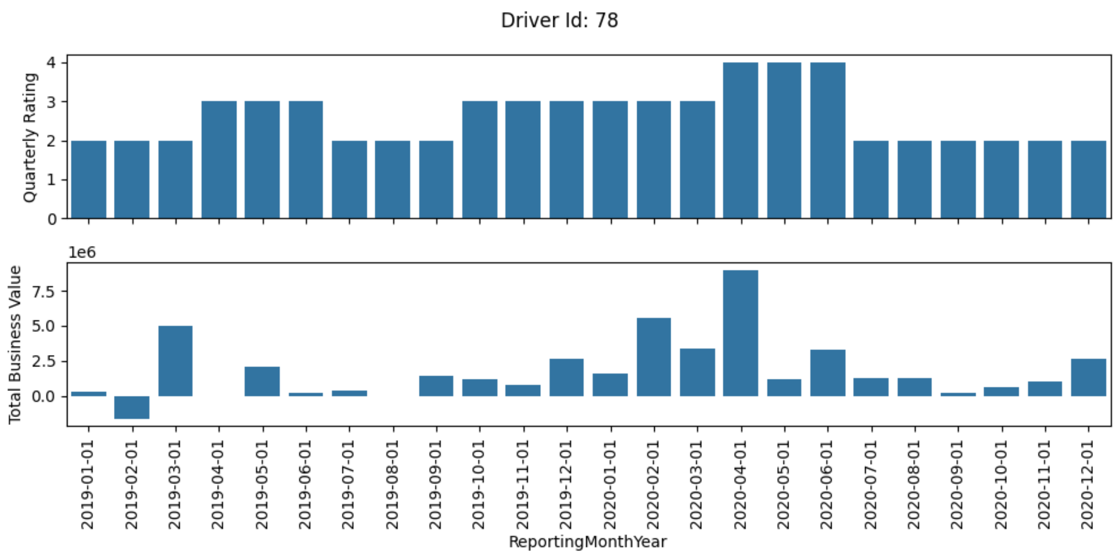

Driver Id: 60
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear
2019-01-01
2019-02-01
2019-03-01
2019-04-01
2019-05-01
2019-06-01
2019-07-01
2019-08-01
2019-09-01
2019-10-01
2019-11-01
2019-12-01
2020-01-01
2020-02-01
2020-03-01
2020-04-01
2020-05-01
2020-06-01
2020-07-01
2020-08-01
2020-09-01
2020-10-01
2020-11-01
2020-12-01

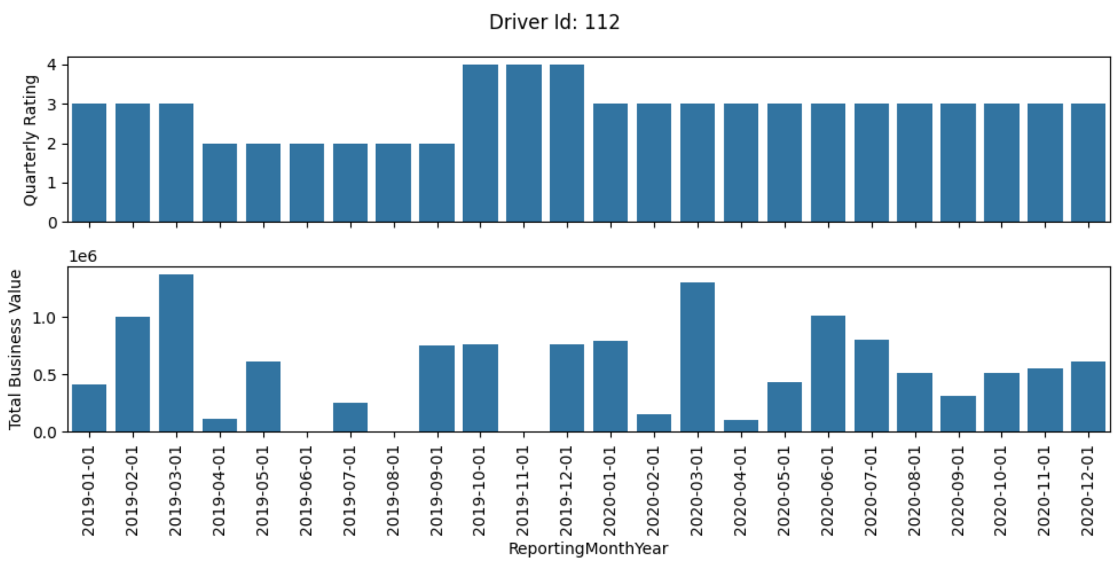

Driver Id: 63
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear
2019-01-01
2019-02-01
2019-03-01
2019-04-01
2019-05-01
2019-06-01
2019-07-01
2019-08-01
2019-09-01
2019-10-01
2019-11-01
2019-12-01
2020-01-01
2020-02-01
2020-03-01
2020-04-01
2020-05-01
2020-06-01
2020-07-01
2020-08-01
2020-09-01
2020-10-01
2020-11-01
2020-12-01

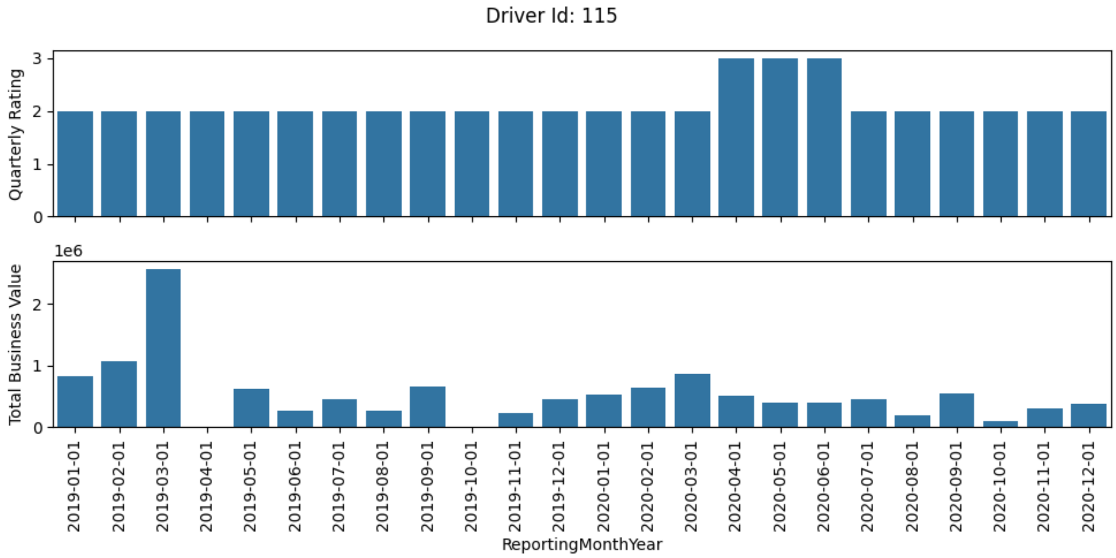

Driver Id: 67
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear
2019-01-01
2019-02-01
2019-03-01
2019-04-01
2019-05-01
2019-06-01
2019-07-01
2019-08-01
2019-09-01
2019-10-01
2019-11-01
2019-12-01
2020-01-01
2020-02-01
2020-03-01
2020-04-01
2020-05-01
2020-06-01
2020-07-01
2020-08-01
2020-09-01
2020-10-01
2020-11-01
2020-12-01

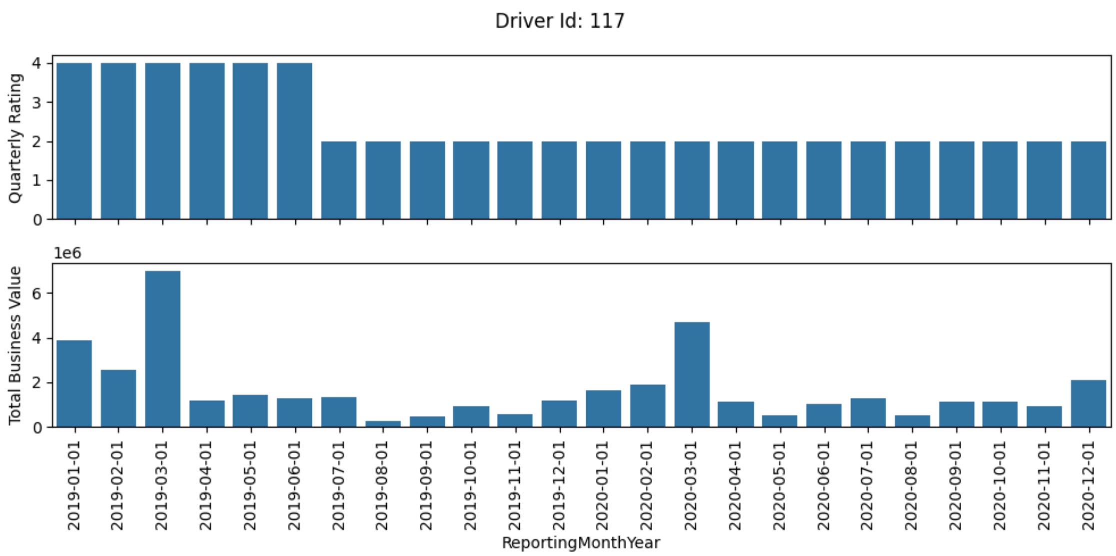

Driver Id: 68
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear

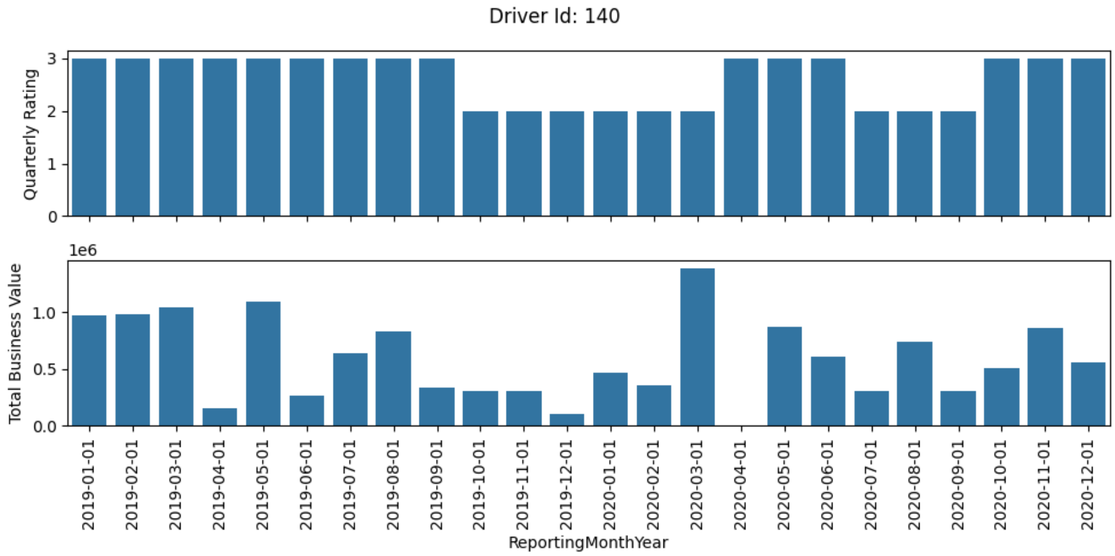

Driver Id: 77
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear

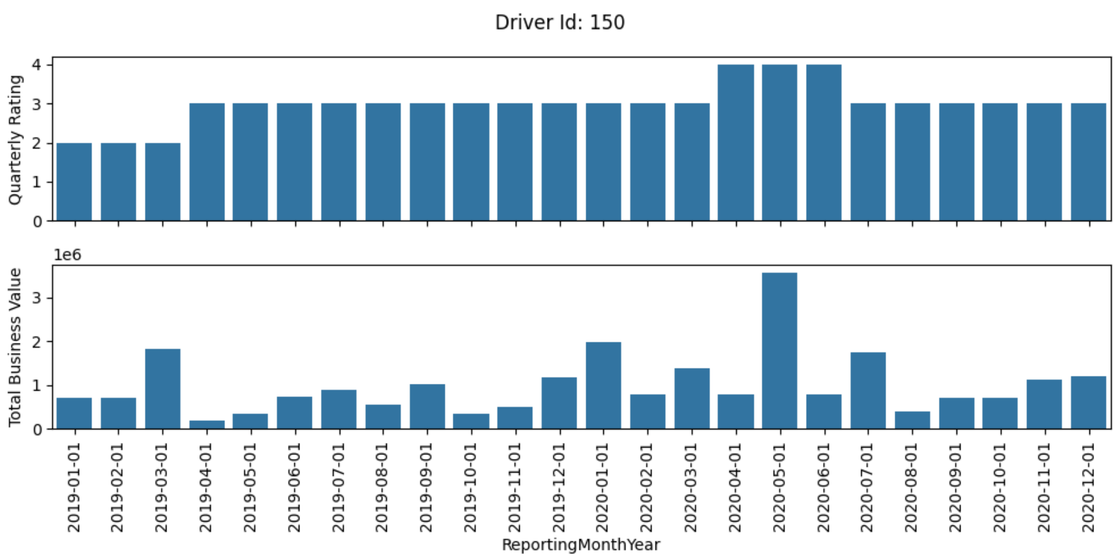

Driver Id: 78
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear

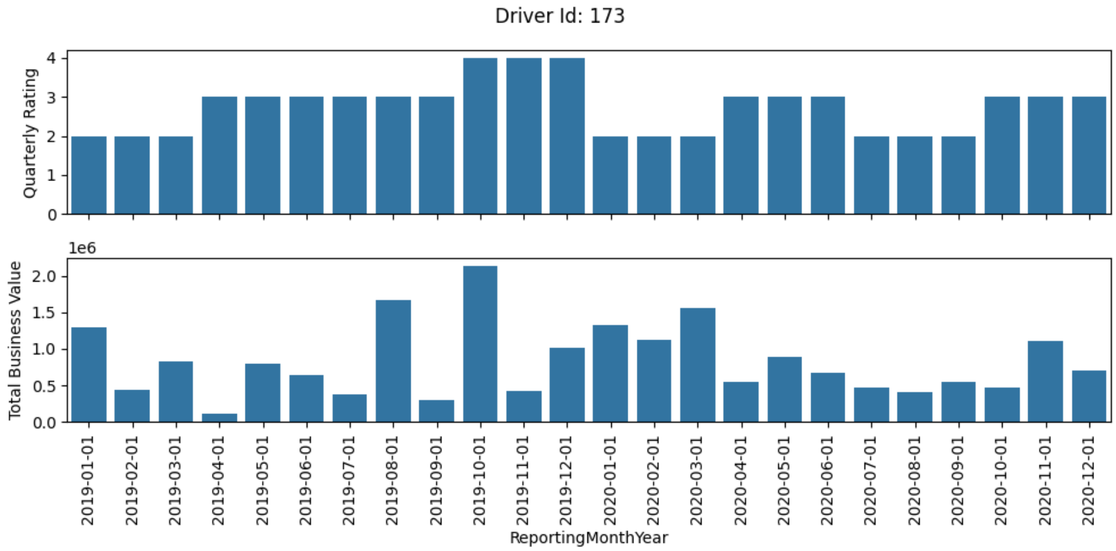

Driver Id: 112
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear

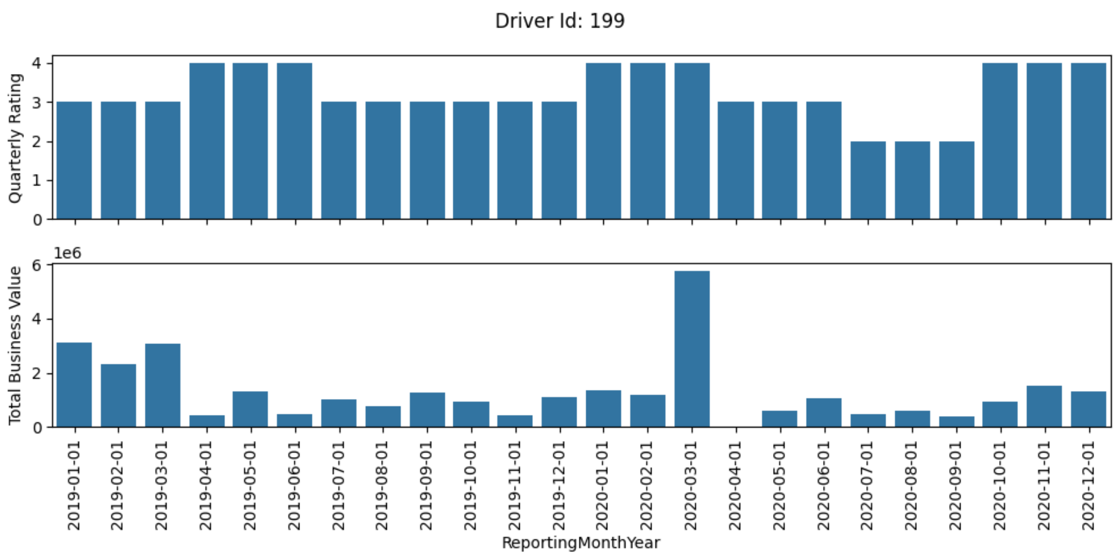

Driver Id: 115
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear
2019-01-01
2019-02-01
2019-03-01
2019-04-01
2019-05-01
2019-06-01
2019-07-01
2019-08-01
2019-09-01
2019-10-01
2019-11-01
2019-12-01
2020-01-01
2020-02-01
2020-03-01
2020-04-01
2020-05-01
2020-06-01
2020-07-01
2020-08-01
2020-09-01
2020-10-01
2020-11-01
2020-12-01

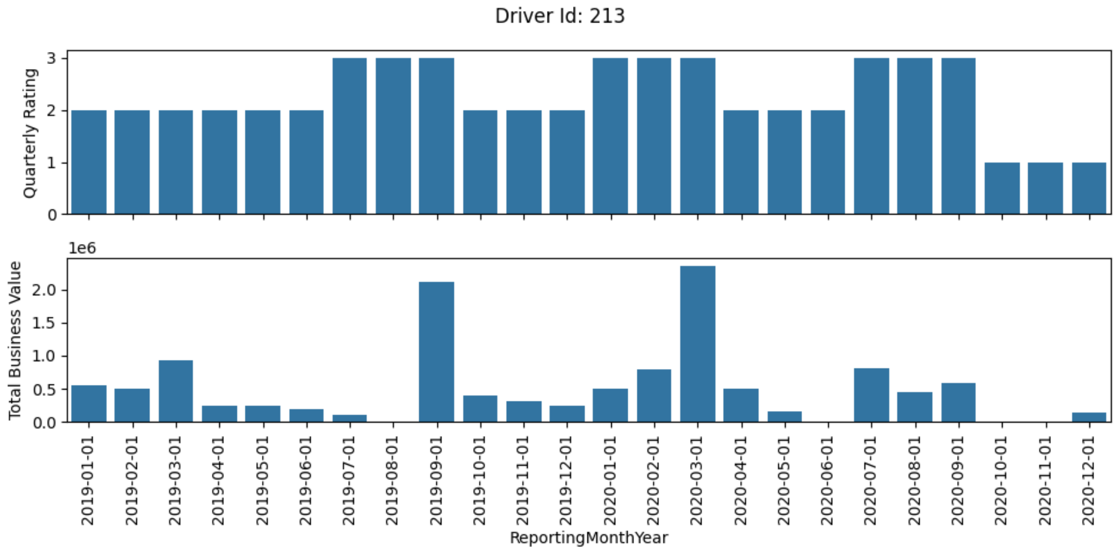

Driver Id: 117
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear
2019-01-01
2019-02-01
2019-03-01
2019-04-01
2019-05-01
2019-06-01
2019-07-01
2019-08-01
2019-09-01
2019-10-01
2019-11-01
2019-12-01
2020-01-01
2020-02-01
2020-03-01
2020-04-01
2020-05-01
2020-06-01
2020-07-01
2020-08-01
2020-09-01
2020-10-01
2020-11-01
2020-12-01

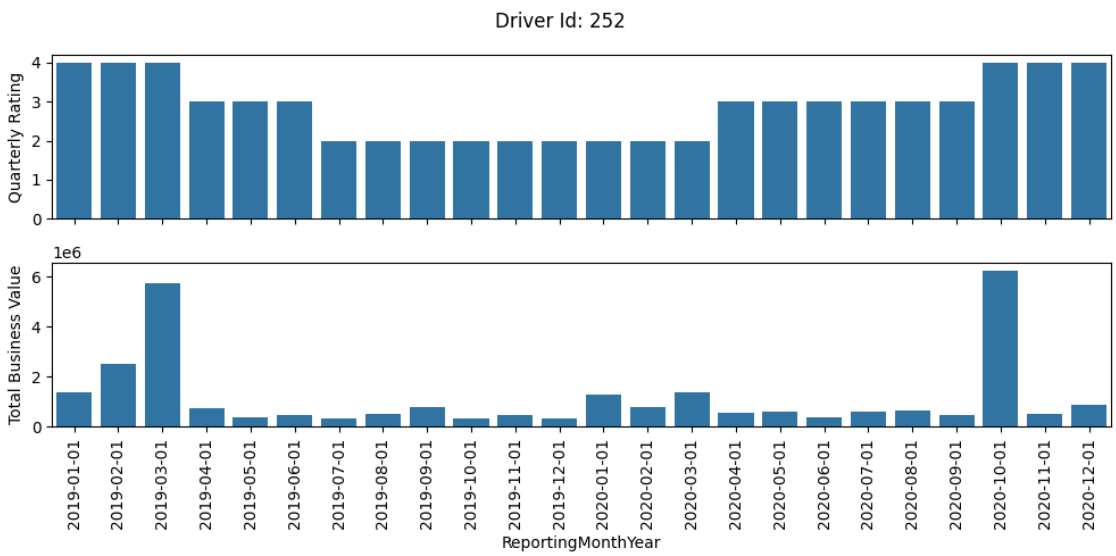

Driver Id: 140
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear

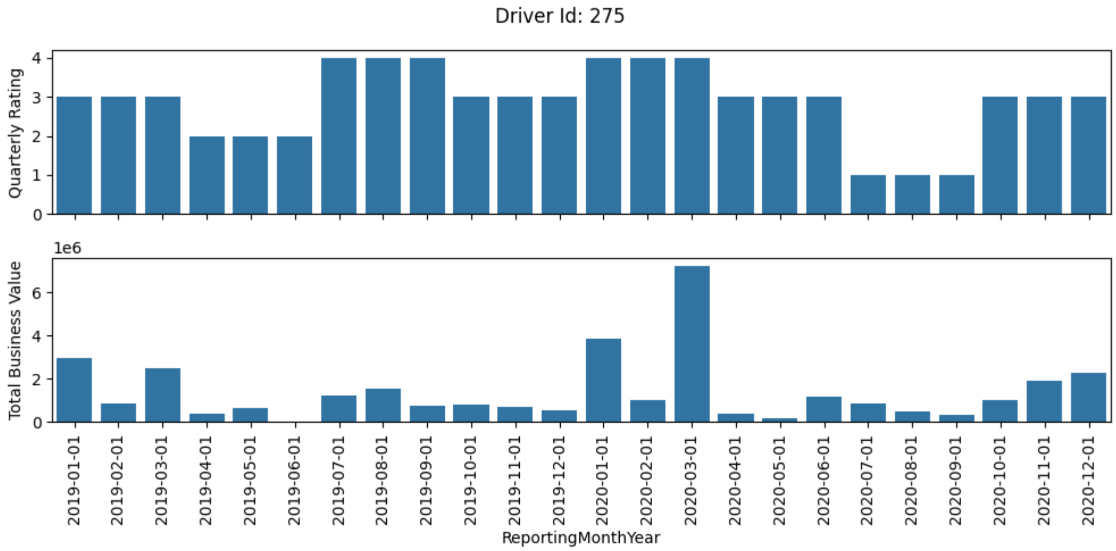

Driver Id: 150
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear

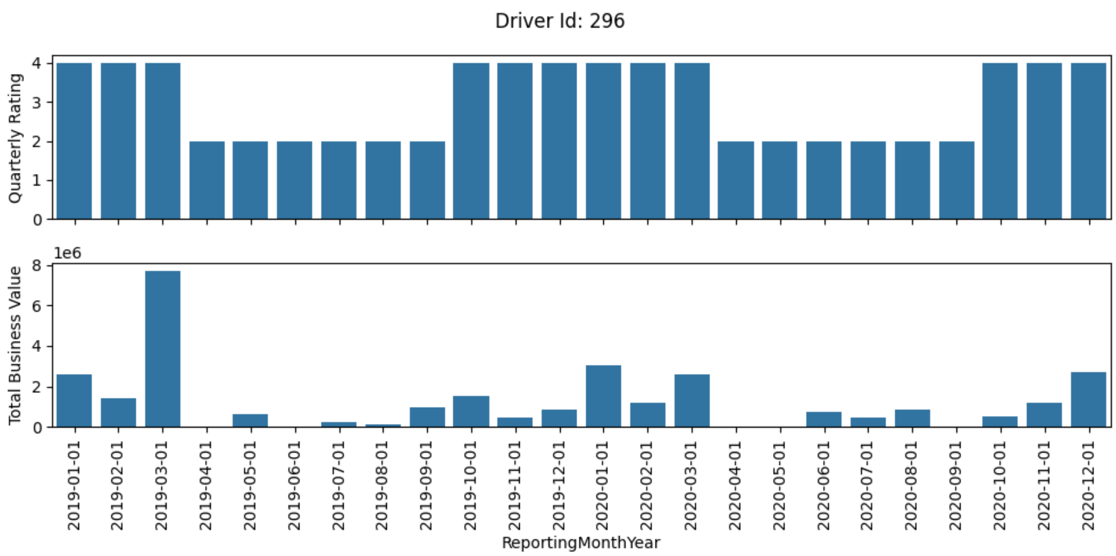

Driver Id: 173
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear


Driver Id: 199
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear


Driver Id: 213
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear


Driver Id: 252
Quarterly Rating
Total Business Value
1e6
ReportingMonthYear





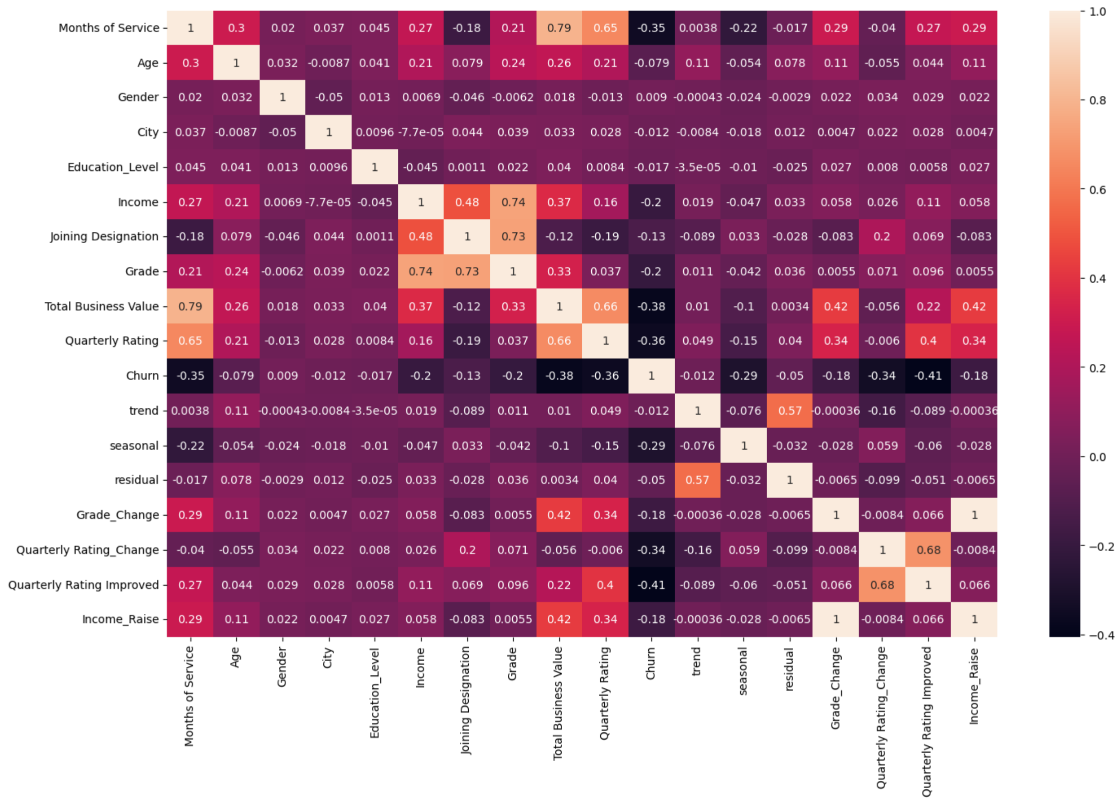
1.2.8 Insight

- It can be observed that a significant drop in rating impacts the Total Business Value. Drop in rating demotivates the drivers, leading to accepting only a few rides or in somecases not accepting any rides and hence impacting the Total Business Value

1.3 Multivariate analysis

[115]:
```
driver_df['Gender'].replace({'Male':0, 'Female':1}, inplace=True)
driver_df['Education_Level'].replace({'Graduate':0, '10+':1, '12+':2},␣
↪inplace=True)
driver_df['City'] = driver_df['City'].str[1:]
```

[116]:
```
plt.figure(figsize=(15,10))
sns.heatmap(driver_df.drop(columns=['Driver_ID', 'Dateofjoining',␣
↪'LastWorkingDate', 'Income_Change']).corr(), annot=True)
plt.tight_layout()
plt.show()
```



1.3.1 Insight

- **Months of Service** and **Total Business Value** are highly correlated
- **Income** and **Grade** are highly correlated
- **Joining Designation** and **Grade** are highly correlated
- **Quarterly Rating** and **Months of Service** are highly correlated
- **Chrun** is decently correlated with **Quarterly Rating, Total Business Value, Months of Service**

2 Data Preprocessing

[117]:
```
driver_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 22 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Driver_ID                  2381 non-null   int64
 1   Months of Service          2381 non-null   int64
 2   Age                        2381 non-null   float64
 3   Gender                     2381 non-null   category
 4   City                       2381 non-null   object
 5   Education_Level            2381 non-null   category
 6   Income                     2381 non-null   float64
 7   Dateofjoining              2381 non-null   datetime64[ns]
 8   LastWorkingDate            2380 non-null   datetime64[ns]
 9   Joining Designation        2381 non-null   category
 10  Grade                      2381 non-null   float64
 11  Total Business Value       2381 non-null   int64
 12  Quarterly Rating           2381 non-null   float64
 13  Churn                      2381 non-null   int64
 14  trend                      2381 non-null   float64
 15  seasonal                   2381 non-null   float64
 16  residual                   2381 non-null   float64
 17  Income_Change              2381 non-null   int32
 18  Grade_Change               2381 non-null   int32
 19  Quarterly Rating_Change    2381 non-null   int32
 20  Quarterly Rating Improved  2381 non-null   int64
 21  Income_Raise               2381 non-null   int64
dtypes: category(3), datetime64[ns](2), float64(7), int32(3), int64(6),
object(1)
memory usage: 333.1+ KB
```

2.0.1 Insight

- The columns **Driver_ID, Dateofjoining, LastWorkingDate** can be dropped as they do not contribute towards the driver churn rate

[118]:
```
driver_df.drop(columns=['Driver_ID', 'Dateofjoining', 'LastWorkingDate'],␣
↪inplace=True)
driver_df['Quarterly Rating'] = driver_df['Quarterly Rating'].astype('category')
driver_df['Churn'] = driver_df['Churn'].astype('category')
driver_df['Grade_Change'] = driver_df['Grade_Change'].astype('category')
driver_df['Quarterly Rating_Change'] = driver_df['Quarterly Rating_Change'].
↪astype('category')
driver_df['Income_Raise'] = driver_df['Income_Raise'].astype('category')
```

```python
driver_df.head()
```

[118]:
```
   Months of Service   Age Gender City Education_Level   Income  \
0                  3  28.0      0   23               0  57387.0
1                  2  31.0      0    7               0  67016.0
2                  5  43.0      0   13               0  65603.0
3                  3  29.0      0    9               1  46368.0
4                  5  31.0      1   11               2  78728.0

  Joining Designation  Grade  Total Business Value Quarterly Rating Churn  \
0                   1    1.0               1715580              2.0     1
1                   2    2.0                     0              1.0     0
2                   2    2.0                350000              1.0     1
3                   1    1.0                120360              1.0     1
4                   3    3.0               1265000              2.0     0

        trend   seasonal  residual  Income_Change Grade_Change  \
0  444.833333  -5.475694  3.600694              0            0
1  444.833333  21.753472  3.600694              0            0
2  416.016667  -0.217361  3.600694              0            0
3  540.041667  -5.475694  3.600694              0            0
4  540.041667   1.340972  3.600694              0            0

  Quarterly Rating_Change  Quarterly Rating Improved Income_Raise
0                       0                          0            0
1                       0                          0            0
2                       0                          0            0
3                       0                          0            0
4                       1                          1            0
```

[119]:
```python
driver_df.duplicated().value_counts()
```

[119]:
```
False    2381
Name: count, dtype: int64
```

2.0.2 Insight

- There are no duplicates

2.1 Handling null values

[120]:
```python
driver_df.isna().sum()
```

[120]:
```
Months of Service            0
Age                          0
Gender                       0
City                         0
Education_Level              0
```

```
Income                       0
Joining Designation          0
Grade                        0
Total Business Value         0
Quarterly Rating             0
Churn                        0
trend                        0
seasonal                     0
residual                     0
Income_Change                0
Grade_Change                 0
Quarterly Rating_Change      0
Quarterly Rating Improved    0
Income_Raise                 0
dtype: int64
```

2.1.1 Insight

- There are **no missing data or null values**

2.2 Outlier Treatment

[121]:
```python
# helper function to detect outliers using IQR method
def detectOutliers_iqr(df):
    q1 = df.quantile(0.25)
    q3 = df.quantile(0.75)
    iqr = q3-q1
    lower_outliers = df[df<(q1-1.5*iqr)]
    higher_outliers = df[df>(q3+1.5*iqr)]
    return lower_outliers, higher_outliers

# helper function to detect outliers using standard deviation method
def detectOutliers_std(df):
    mean = df.mean()
    std = df.std()
    upper_limit = mean+(3*std)
    lower_limit = mean-(3*std)
    lower_outliers = df[df<lower_limit]
    higher_outliers = df[df>upper_limit]
    return lower_outliers, higher_outliers
```

[122]:
```python
numerical_columns = driver_df.select_dtypes(include=np.number).columns
column_outlier_dictionary = {}
for column in numerical_columns:
    lower_outliers, higher_outliers = detectOutliers_iqr(driver_df[column])
    column_outlier_dictionary[column] = [lower_outliers, higher_outliers]
```

```
[123]: for key, value in column_outlier_dictionary.items():
           print(f'The column \'{key}\' has {len(value[0]) + len(value[1])} outliers')
```

```
The column 'Months of Service' has 249 outliers
The column 'Age' has 25 outliers
The column 'Income' has 48 outliers
The column 'Grade' has 0 outliers
The column 'Total Business Value' has 336 outliers
The column 'trend' has 0 outliers
The column 'seasonal' has 227 outliers
The column 'residual' has 0 outliers
The column 'Income_Change' has 43 outliers
The column 'Quarterly Rating Improved' has 358 outliers
```

```
[124]: numerical_columns = driver_df.select_dtypes(include=np.number).columns
       column_outlier_dictionary = {}
       for column in numerical_columns:
           lower_outliers, higher_outliers = detectOutliers_std(driver_df[column])
           column_outlier_dictionary[column] = [lower_outliers, higher_outliers]
```

```
[125]: for key, value in column_outlier_dictionary.items():
           print(f'The column \'{key}\' has {len(value[0]) + len(value[1])} outliers')
```

```
The column 'Months of Service' has 0 outliers
The column 'Age' has 14 outliers
The column 'Income' has 14 outliers
The column 'Grade' has 21 outliers
The column 'Total Business Value' has 64 outliers
The column 'trend' has 0 outliers
The column 'seasonal' has 18 outliers
The column 'residual' has 0 outliers
The column 'Income_Change' has 43 outliers
The column 'Quarterly Rating Improved' has 0 outliers
```

2.2.1 Insight

- I will **keep the outliers** in **Age** and **Income** columns as they are less in number
- I will **cap the outliers** in **Total Business Value** column as drivers with higher business value do not churn usually

```
[126]: mean = driver_df['Total Business Value'].mean()
       std = driver_df['Total Business Value'].std()
       upper_limit = mean+(3*std)
       driver_df['Total Business Value'] = driver_df['Total Business Value'].
        ↪apply(lambda x: x if x <= upper_limit else upper_limit)
```

2.3 Multicollinearity Check

[127]:
```
driver_df.Grade=driver_df.Grade.astype('category')
driver_df.City=driver_df.City.astype('category')
```

[128]:
```
features_df = driver_df.drop(columns=['Churn']) # Drop target column
features_df = features_df.drop(columns=features_df.
 ↪select_dtypes(include='category').columns)
#print(features_df.columns)
features_df = sm.add_constant(features_df)  # Adding a constant column for the␣
 ↪intercept
vif_df = pd.DataFrame()
vif_df['Features'] = features_df.columns
vif_df['VIF'] = [variance_inflation_factor(features_df.values, idx) for idx in␣
 ↪range(len(features_df.columns))]
vif_df['VIF'] = round(vif_df['VIF'], 2)
vif_df = vif_df.sort_values(by='VIF', ascending=False)
vif_df
```

[128]:

	Features	VIF
4	Total Business Value	4.62
1	Months of Service	4.14
8	Income_Change	1.25
3	Income	1.19
2	Age	1.14
9	Quarterly Rating Improved	1.10
6	seasonal	1.08
5	trend	1.03
0	const	0.00
7	residual	0.00

2.4 Insight

- Based on the above VIF scores, I can conclude that there are no multicolinear numerical features

2.5 Encode categorical variables

[129]:
```
df = driver_df.copy()
```

[130]:
```
X = df.drop(columns=['Churn'])
y = df['Churn']
X.shape, y.shape
```

[130]: ((2381, 18), (2381,))

[131]:
```
y = y.astype(int)
```

```
X[['Grade_Change','Quarterly Rating_Change', 'Income_Raise']] =␣
↪X[['Grade_Change','Quarterly Rating_Change', 'Income_Raise']].astype('int8')
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 18 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Months of Service          2381 non-null   int64
 1   Age                        2381 non-null   float64
 2   Gender                     2381 non-null   category
 3   City                       2381 non-null   category
 4   Education_Level            2381 non-null   category
 5   Income                     2381 non-null   float64
 6   Joining Designation        2381 non-null   category
 7   Grade                      2381 non-null   category
 8   Total Business Value       2381 non-null   float64
 9   Quarterly Rating           2381 non-null   category
 10  trend                      2381 non-null   float64
 11  seasonal                   2381 non-null   float64
 12  residual                   2381 non-null   float64
 13  Income_Change              2381 non-null   int32
 14  Grade_Change               2381 non-null   int8
 15  Quarterly Rating_Change    2381 non-null   int8
 16  Quarterly Rating Improved  2381 non-null   int64
 17  Income_Raise               2381 non-null   int8
dtypes: category(6), float64(6), int32(1), int64(2), int8(3)
memory usage: 181.5 KB
```

[132]:
```
categorical_columns = list(X.select_dtypes(include='category').columns)
categorical_columns
```

[132]:
```
['Gender',
 'City',
 'Education_Level',
 'Joining Designation',
 'Grade',
 'Quarterly Rating']
```

2.6 Rebalancing reduces F1 score so not rebalancing

[133]:
```
'''
#rebalance
from imblearn.over_sampling import RandomOverSampler

oversampler = RandomOverSampler(sampling_strategy='auto')
X, y = oversampler.fit_resample(X, y)
```

```
'''
```

[133]: "\n#rebalance\nfrom imblearn.over_sampling import RandomOverSampler\n\noversampler = RandomOverSampler(sampling_strategy='auto')\nX, y = oversampler.fit_resample(X, y)\n"

2.7 Splitting and Preprocessing

[134]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.
↪2,random_state=SEED)

te = TargetEncoder()

X_train[categorical_columns] = te.fit_transform(X_train[categorical_columns],␣
↪y_train)
X_test[categorical_columns] = te.transform(X_test[categorical_columns])
```

[135]:
```
scaler=StandardScaler()
#scaler=MinMaxScaler()
scols=['Months of Service','Income','Total Business Value','Quarterly␣
↪Rating_Change','Income_Change','Grade_Change','Quarterly␣
↪Rating_Change','trend','seasonal','residual']
X_train[scols]=scaler.fit_transform(X_train[scols])
X_test[scols]=scaler.transform(X_test[scols])
```

[136]:
```
X_train.head()
```

[136]:
```
      Months of Service   Age    Gender      City  Education_Level    Income  \
1877           1.462948  29.0  0.684143  0.676505         0.681470  0.075354
893           -0.451352  37.0  0.684143  0.748753         0.669841  1.164405
640           -0.893113  27.0  0.678253  0.603576         0.690821  1.605130
2199          -0.745859  28.0  0.678253  0.835067         0.669841 -0.336200
1903           1.462948  26.0  0.684143  0.760587         0.669841  0.472533

      Joining Designation     Grade  Total Business Value  Quarterly Rating  \
1877             0.734719  0.797980              0.382943          0.622685
893              0.555556  0.550403             -0.430901          0.793750
640              0.555556  0.550403             -0.558843          0.793750
2199             0.690184  0.699128             -0.558843          0.793750
1903             0.734719  0.699128              1.151720          0.412714

         trend  seasonal      residual  Income_Change  Grade_Change  \
1877  0.003221  0.013058 -4.440892e-16      -0.134364     -0.140776
893  -0.241182 -0.036426 -2.220446e-15      -0.134364     -0.140776
640  -1.971075  2.294558 -2.220446e-15      -0.134364     -0.140776
2199 -2.004277  1.651424 -4.440892e-16      -0.134364     -0.140776
```

```
1903  0.003221  0.013058 -4.440892e-16      -0.134364     -0.140776

      Quarterly Rating_Change  Quarterly Rating Improved  Income_Raise
1877                -1.015746                          0             0
893                  0.062882                          0             0
640                  0.062882                          0             0
2199                 0.062882                          0             0
1903                -2.094374                          0             0
```

3 Baseline Models

[137]:
```
logreg = LogisticRegression(max_iter=100000)
logreg.fit(X_train, y_train)

y_pred_logreg = logreg.predict(X_test)


f1_logreg = f1_score(y_test, y_pred_logreg, average='weighted')
print(f'Logistic Regression f1_score: {f1_logreg:.2f}')
```

```
Logistic Regression f1_score: 0.87
```

[138]:
```
svm_model = SVC()
svm_model.fit(X_train, y_train)

y_pred_svm = svm_model.predict(X_test)


f1_svm = f1_score(y_test, y_pred_svm, average='weighted')
print(f'SVM f1_score: {f1_svm:.2f}')
```

```
SVM f1_score: 0.74
```

[139]:
```
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

y_pred_nb = nb_model.predict(X_test)


f1_nb = f1_score(y_test, y_pred_nb, average='weighted')
print(f'Naive Bayes f1_score: {f1_nb:.2f}')
```

```
Naive Bayes f1_score: 0.79
```

[140]:
```
'''
nbm_model = MultinomialNB()
nbm_model.fit(X_train, y_train)

```

```
y_pred_nbm = nbm_model.predict(X_test)

f1_nb = accuracy_score(y_test, y_pred_nbm, average='weighted')
print(f'Multinomial Naive Bayes f1_score: {f1_nb:.2f}')
'''
```

[140]: "\nnbm_model = MultinomialNB()\nnbm_model.fit(X_train, y_train)\n\ny_pred_nbm = nbm_model.predict(X_test)\n\nf1_nb = accuracy_score(y_test, y_pred_nbm, average='weighted')\nprint(f'Multinomial Naive Bayes f1_score: {f1_nb:.2f}')\n"

4 Random Forest Model [Bagging]

[141]:
```
'''
def rf_objective(trial):
    n_estimators = trial.suggest_int("n_estimators", 50, 1000, log=True)
    max_depth = trial.suggest_int("max_depth", 4, 128)
    min_samples_split = trial.suggest_int("min_samples_split", 2, 10)
    min_samples_leaf = trial.suggest_int("min_samples_leaf", 1, 10)

    model = RandomForestClassifier(
    n_estimators=n_estimators,
    max_depth=max_depth,
    min_samples_split=min_samples_split,
    min_samples_leaf=min_samples_leaf,
    random_state=SEED,
    )
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    f1 = f1_score(y_test, y_pred, average='weighted')

    return f1

study_Rf = optuna.create_study(study_name="Rf_Scaler",direction='maximize')
optuna.logging.set_verbosity(optuna.logging.WARNING)
study_Rf.optimize(rf_objective, n_trials=200, show_progress_bar=True)
# Best parameters and score
print("Best Trial:")
print(f"  Value: {study_Rf.best_trial.value:.4f}")
print("  Params: ")
for key, value in study_Rf.best_trial.params.items():
    print(f"    {key}: {value}")


rf_final = RandomForestClassifier(**study_Rf.best_params,random_state=SEED)
rf_final.fit(X_train,y_train)
```

```
print(f1_score(y_test,rf_final.predict(X_test), average='weighted'))'''
```

[141]:
```
'\ndef rf_objective(trial):\n    n_estimators =
trial.suggest_int("n_estimators", 50, 1000, log=True)\n    max_depth =
trial.suggest_int("max_depth", 4, 128)\n    min_samples_split =
trial.suggest_int("min_samples_split", 2, 10)\n    min_samples_leaf =
trial.suggest_int("min_samples_leaf", 1, 10)\n\n    model =
RandomForestClassifier(\n    n_estimators=n_estimators,\n
max_depth=max_depth,\n    min_samples_split=min_samples_split,\n
min_samples_leaf=min_samples_leaf,\n    random_state=SEED,\n    )\n
model.fit(X_train, y_train)\n\n    y_pred = model.predict(X_test)\n    f1 =
f1_score(y_test, y_pred, average=\'weighted\')\n\n    return f1\n\nstudy_Rf =
optuna.create_study(study_name="Rf_Scaler",direction=\'maximize\')  \noptuna.log
ging.set_verbosity(optuna.logging.WARNING)\nstudy_Rf.optimize(rf_objective,
n_trials=200, show_progress_bar=True)  \n# Best parameters and
score\nprint("Best Trial:")\nprint(f"  Value:
{study_Rf.best_trial.value:.4f}")\nprint("  Params: ")\nfor key, value in
study_Rf.best_trial.params.items():\n    print(f"    {key}:
{value}")\n\n\nrf_final = RandomForestClassifier(**study_Rf.best_params,random_s
tate=SEED)\nrf_final.fit(X_train,y_train)\nprint(f1_score(y_test,rf_final.predic
t(X_test), average=\'weighted\'))'
```

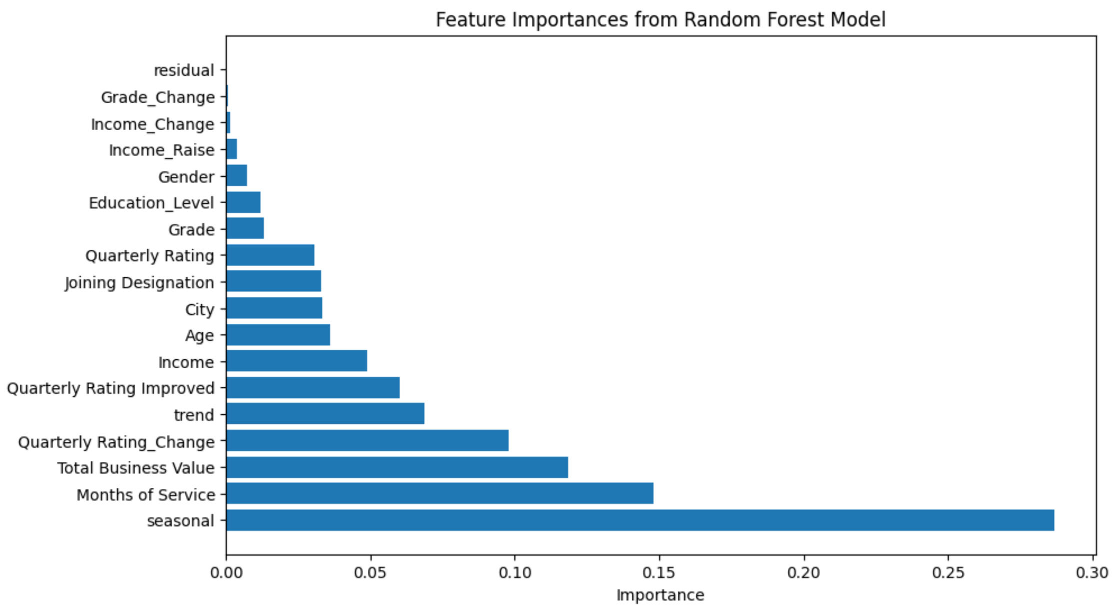
[142]:
```
bestparamsrf={'n_estimators': 68, 'max_depth': 73, 'min_samples_split': 7,␣
 ↪'min_samples_leaf': 3}
rf_final = RandomForestClassifier(**bestparamsrf,random_state=SEED)
rf_final.fit(X_train,y_train)
print(f1_score(y_test,rf_final.predict(X_test),average='weighted'))
```

```
0.8976877892975846
```

[143]:
```
importances = rf_final.feature_importances_

feature_names = list(X.columns)
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':␣
 ↪importances})

importance_df = importance_df.sort_values(by='Importance', ascending=False)
print(importance_df)



plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.xlabel('Importance')
plt.title('Feature Importances from Random Forest Model')
plt.show()
```

```
                    Feature  Importance
```

```
11                  seasonal    0.287094
0          Months of Service    0.148162
8       Total Business Value    0.118573
15   Quarterly Rating_Change    0.097784
10                     trend    0.068721
16 Quarterly Rating Improved    0.060126
5                     Income    0.048857
1                        Age    0.035908
3                       City    0.033266
6        Joining Designation    0.032771
9           Quarterly Rating    0.030668
7                      Grade    0.013060
4            Education_Level    0.012027
2                     Gender    0.007208
17              Income_Raise    0.003778
13             Income_Change    0.001465
14              Grade_Change    0.000534
12                  residual    0.000000
```



5 LightGBM model

[144]:
```
'''
def lgbm_objective(trial):

    lgbm_params = {
        "n_estimators": 2000,
```

```
        "subsample": trial.suggest_float("subsample", 0.3, 0.9),
        "min_child_samples": trial.suggest_int("min_child_samples", 60, 100),
        "max_depth": trial.suggest_int("max_depth", 4, 25),
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.1),
        "lambda_l1": trial.suggest_float("lambda_l1", 0.001, 0.1),
        "lambda_l2": trial.suggest_float("lambda_l2", 0.001, 0.1),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.3, 1.0)

    }

    lgbm_model = LGBMClassifier(**lgbm_params, random_state=SEED, verbose=-1)

    lgbm_model.fit(X_train, y_train)
    y_pred = lgbm_model.predict(X_test)
    return f1_score(y_test, y_pred, average='weighted')

study_LGBM = optuna.create_study(study_name="LGBM_Scaler", direction="maximize")
optuna.logging.set_verbosity(optuna.logging.WARNING)
study_LGBM.optimize(lgbm_objective, n_trials=200, show_progress_bar=True)

print("Best trial:", study_LGBM.best_trial)
print("Best parameters:", study_LGBM.best_params)

lgbm_final = LGBMClassifier(**study_LGBM.
 ↪best_params,n_estimators=1900,random_state=SEED,verbose=-1)
lgbm_final.fit(X_train, y_train)
y_pred = lgbm_final.predict(X_test)
print("f1:",f1_score(y_test, y_pred, average='weighted'))'''
```

[144]:

```
'\ndef lgbm_objective(trial):\n\n    lgbm_params = {\n        "n_estimators":
2000,\n        "subsample": trial.suggest_float("subsample", 0.3, 0.9),\n
"min_child_samples": trial.suggest_int("min_child_samples", 60, 100),\n
"max_depth": trial.suggest_int("max_depth", 4, 25),\n        "learning_rate":
trial.suggest_float("learning_rate", 0.01, 0.1),\n        "lambda_l1":
trial.suggest_float("lambda_l1", 0.001, 0.1),\n        "lambda_l2":
trial.suggest_float("lambda_l2", 0.001, 0.1),\n        \'colsample_bytree\':
trial.suggest_float(\'colsample_bytree\', 0.3, 1.0)\n\n    }\n\n    lgbm_model =
LGBMClassifier(**lgbm_params, random_state=SEED, verbose=-1)\n\n
lgbm_model.fit(X_train, y_train)\n    y_pred = lgbm_model.predict(X_test)\n
return f1_score(y_test, y_pred, average=\'weighted\')\n\nstudy_LGBM =
optuna.create_study(study_name="LGBM_Scaler", direction="maximize")\noptuna.logg
ing.set_verbosity(optuna.logging.WARNING)\nstudy_LGBM.optimize(lgbm_objective,
n_trials=200, show_progress_bar=True)\n\nprint("Best trial:",
study_LGBM.best_trial)\nprint("Best parameters:",
study_LGBM.best_params)\n\nlgbm_final = LGBMClassifier(**study_LGBM.best_params,
n_estimators=1900,random_state=SEED,verbose=-1)\nlgbm_final.fit(X_train,
y_train)\ny_pred = lgbm_final.predict(X_test)\nprint("f1:",f1_score(y_test,
```

```
y_pred, average=\'weighted\'))'
```

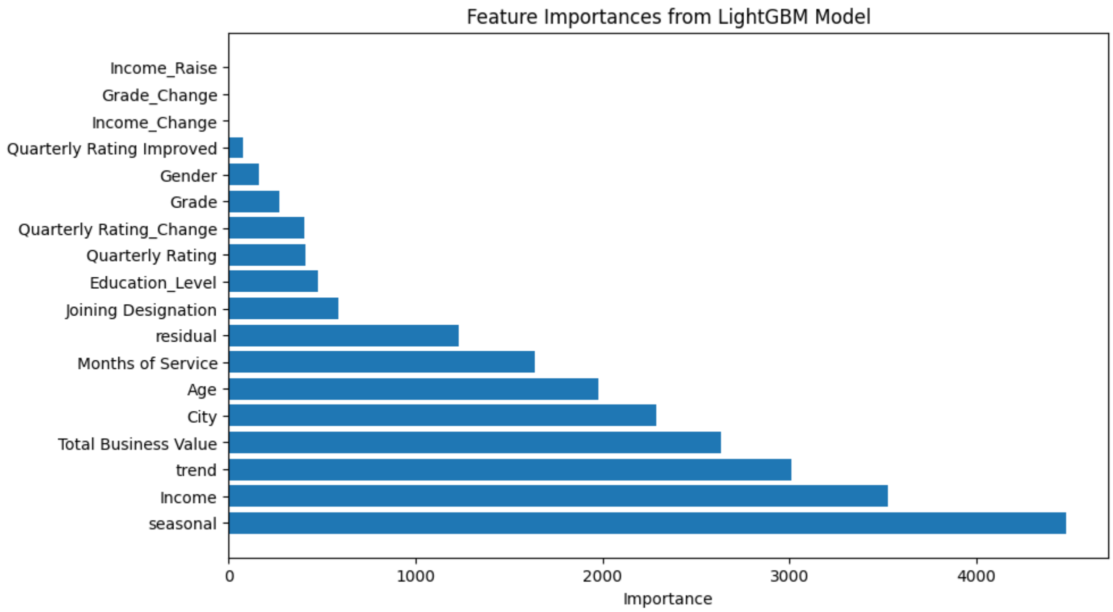
[145]:
```python
bestparamslgbm={'subsample': 0.7185412234228604, 'min_child_samples': 92,␣
 ↪'max_depth': 7, 'learning_rate': 0.04249913492277367, 'lambda_l1': 0.
 ↪09413308734404659, 'lambda_l2': 0.05501380657087828, 'colsample_bytree': 0.
 ↪6779843680702}
lgbm_final = LGBMClassifier(**bestparamslgbm,n_estimators=␣
 ↪1900,random_state=SEED,verbose=-1)
lgbm_final.fit(X_train, y_train)
y_pred = lgbm_final.predict(X_test)
print("f1:",f1_score(y_test, y_pred, average='weighted'))
```

```
f1: 0.9412997903563941
```

[146]:
```python
importances = lgbm_final.feature_importances_

feature_names = list(X.columns)
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':␣
 ↪importances})

importance_df = importance_df.sort_values(by='Importance', ascending=False)
print(importance_df)



plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.xlabel('Importance')
plt.title('Feature Importances from LightGBM Model')
plt.show()
```

```
                     Feature  Importance
11                  seasonal        4482
5                     Income        3523
10                     trend        3008
8       Total Business Value        2633
3                       City        2290
1                        Age        1978
0          Months of Service        1636
12                  residual        1228
6        Joining Designation         587
4            Education_Level         480
9          Quarterly Rating          413
15   Quarterly Rating_Change         405
7                      Grade         273
2                     Gender         163
16 Quarterly Rating Improved          76
13             Income_Change           0
```

```
14                Grade_Change          0
17                Income_Raise          0
```



6 XGBoost model [Boosting]

[147]:
```
'''
def xgb_objective(trial):
    param = {
        'booster': trial.suggest_categorical('booster', ['gbtree', 'dart']),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3,␣
 ↪log=True),
        'n_estimators': trial.suggest_int('n_estimators', 50, 500),
        'max_depth': trial.suggest_int('max_depth', 3, 12),
        'min_child_weight': trial.suggest_float('min_child_weight', 1, 10),
        'gamma': trial.suggest_float('gamma', 0, 5),
        'subsample': trial.suggest_float('subsample', 0.5, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.5, 1.0),
        'lambda': trial.suggest_float('lambda', 1e-3, 10.0, log=True),
        'alpha': trial.suggest_float('alpha', 1e-3, 10.0, log=True),
    }

    model = XGBClassifier(**param, random_state=SEED)
    model.fit(X_train, y_train, eval_set=[(X_test, y_test)], verbose=False)

    preds = model.predict(X_test)
    f1 = f1_score(y_test, preds, average='weighted')
```

```
    return f1


study_XGB = optuna.create_study(study_name='XGB_Scaler',direction='maximize')
optuna.logging.set_verbosity(optuna.logging.WARNING)
study_XGB.optimize(xgb_objective, n_trials=200, show_progress_bar=True)

# Best parameters and score
print("Best Trial:")
print(f"  Value: {study_XGB.best_trial.value:.4f}")
print("  Params: ")
for key, value in study_XGB.best_trial.params.items():
    print(f"    {key}: {value}")

xgb_final = XGBClassifier(**study_XGB.best_params,random_state=SEED,verbose=-1)
xgb_final.fit(X_train,y_train)
print(f1_score(y_test,xgb_final.predict(X_test), average='weighted'))'''
```

[147]:

```
'\ndef xgb_objective(trial):\n    param = {\n        \'booster\':
trial.suggest_categorical(\'booster\', [\'gbtree\', \'dart\']),\n
\'learning_rate\': trial.suggest_float(\'learning_rate\', 0.01, 0.3,
log=True),\n        \'n_estimators\': trial.suggest_int(\'n_estimators\', 50,
500),\n        \'max_depth\': trial.suggest_int(\'max_depth\', 3, 12),\n
\'min_child_weight\': trial.suggest_float(\'min_child_weight\', 1, 10),\n
\'gamma\': trial.suggest_float(\'gamma\', 0, 5),\n        \'subsample\':
trial.suggest_float(\'subsample\', 0.5, 1.0),\n        \'colsample_bytree\':
trial.suggest_float(\'colsample_bytree\', 0.5, 1.0),\n        \'lambda\':
trial.suggest_float(\'lambda\', 1e-3, 10.0, log=True),\n        \'alpha\':
trial.suggest_float(\'alpha\', 1e-3, 10.0, log=True),\n    }\n\n    model =
XGBClassifier(**param, random_state=SEED)\n    model.fit(X_train, y_train,
eval_set=[(X_test, y_test)], verbose=False)\n\n    preds =
model.predict(X_test)\n    f1 = f1_score(y_test, preds, average=\'weighted\')\n
return f1\n\n\nstudy_XGB =
optuna.create_study(study_name=\'XGB_Scaler\',direction=\'maximize\')  \noptuna.
logging.set_verbosity(optuna.logging.WARNING)\nstudy_XGB.optimize(xgb_objective,
n_trials=200, show_progress_bar=True)  \n\n# Best parameters and
score\nprint("Best Trial:")\nprint(f"  Value:
{study_XGB.best_trial.value:.4f}")\nprint("  Params: ")\nfor key, value in
study_XGB.best_trial.params.items():\n    print(f"    {key}:
{value}")\n\nxgb_final = XGBClassifier(**study_XGB.best_params,random_state=SEED
,verbose=-
1)\nxgb_final.fit(X_train,y_train)\nprint(f1_score(y_test,xgb_final.predict(X_te
st), average=\'weighted\'))'
```

[148]:

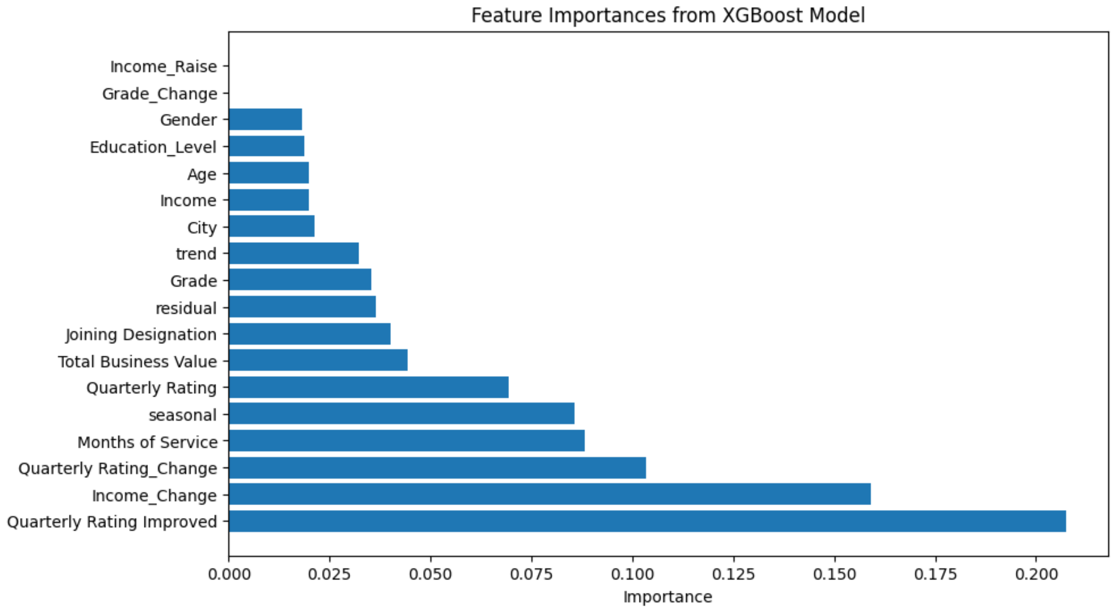
```
bestparamsxgb={"booster": "dart","learning_rate": 0.
 ↪041051028714836436,"n_estimators": 416,"max_depth": 11,"min_child_weight": 1.
 ↪4156268636710905,"gamma": 0.6533322783837707,"subsample": 0.
 ↪7850542756167057,"colsample_bytree": 0.5473747833100527,"lambda": 0.
 ↪26623207096129603,"alpha": 0.0011564970537568634}
xgb_final = XGBClassifier(**bestparamsxgb,random_state=SEED,verbose=-1)
xgb_final.fit(X_train,y_train)
print(f1_score(y_test,xgb_final.predict(X_test), average='weighted'))
```

```
0.9434417827721043
```

[149]:

```
importances = xgb_final.feature_importances_

feature_names = list(X.columns)
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':␣
 ↪importances})

importance_df = importance_df.sort_values(by='Importance', ascending=False)
print(importance_df)



plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.xlabel('Importance')
plt.title('Feature Importances from XGBoost Model')
plt.show()
```

```
                    Feature  Importance
16  Quarterly Rating Improved    0.207589
13              Income_Change    0.159164
15    Quarterly Rating_Change    0.103348
0           Months of Service    0.088132
11                   seasonal    0.085610
9            Quarterly Rating    0.069468
8        Total Business Value    0.044474
6         Joining Designation    0.040133
12                   residual    0.036437
7                       Grade    0.035283
10                      trend    0.032269
3                        City    0.021381
5                      Income    0.019978
1                         Age    0.019762
4             Education_Level    0.018719
2                      Gender    0.018252
14               Grade_Change    0.000000
17               Income_Raise    0.000000
```



[150]:
```
def plot_feature_importance(estimator, features, model_type):
    # Extract feature importances
    importances = estimator.feature_importances_

    # Create a DatafRame for plotting
    feature_importance_df = pd.DataFrame({'Feature':features, 'Importance':
↪importances})
    feature_importance_df = feature_importance_df.sort_values(by='Importance',␣
↪ascending=False)

    # Plot feature importance
    plt.figure(figsize=(8,5))
    sns.barplot(data=feature_importance_df, x='Importance', y='Feature')
    plt.title(f'Feature Importance of Final {model_type} model')
    plt.show()


def display_confusion_matrix(y_test, y_pred):
    # Compute confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Plot confusion matrix
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.show()
```

```
def plot_roc_curve(estimator, X_train, X_test, y_train, y_test):
    # Binarize the output
    y_test_binarized = label_binarize(y_test, classes=[0, 1, 2])
    n_classes = y_test_binarized.shape[1]-1

    # Compute ROC curve and ROC area for each class
    classifier = OneVsRestClassifier(estimator)
    y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plot ROC curve for each class
    plt.figure(figsize=(5, 5))
    for i in range(n_classes):
        plt.plot(fpr[i], tpr[i], label='ROC curve of class {0} (area = {1:0.
↪2f})'.format(i, roc_auc[i]))
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc='lower right')
    plt.show()


def plot_pr_curve(estimator, X_train, X_test, y_train, y_test):
    # Binarize the output
    y_test_binarized = label_binarize(y_test, classes=[0, 1, 2])
    n_classes = y_test_binarized.shape[1]-1

    # Compute ROC curve and ROC area for each class
    classifier = OneVsRestClassifier(estimator)
    y_score = classifier.fit(X_train, y_train).predict_proba(X_test)

    # For each class
    precision = dict()
    recall = dict()
    average_precision = dict()
    for i in range(n_classes):
```

```
        precision[i], recall[i], _ = precision_recall_curve(y_test_binarized[:,␣
↪i], y_score[:, i])
        average_precision[i] = average_precision_score(y_test_binarized[:, i],␣
↪y_score[:, i])

    # Plot Precision-Recall curve for each class
    plt.figure(figsize=(5, 5))
    for i in range(n_classes):
        plt.plot(recall[i], precision[i], label='PR curve of class {0} (area =␣
↪{1:0.2f})'.format(i, average_precision[i]))

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve')
    plt.legend(loc='lower right')
    plt.show()
```

7 Voting Model [further Ensembling]

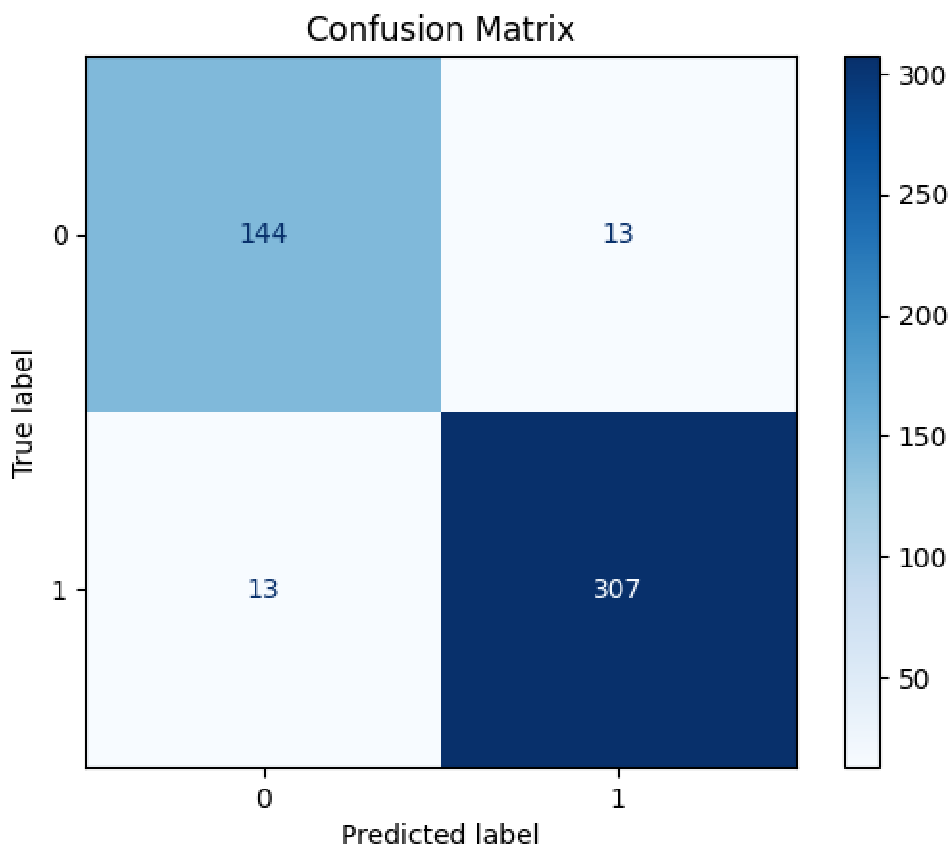
[151]:
```
#nbm_weight=f1_score(y_test,nbm_model.predict(X_test), average='weighted')
#logreg_weight=f1_score(y_test,logreg.predict(X_test), average='weighted')
#svm_weight=f1_score(y_test,svm_model.predict(X_test), average='weighted')
#nb_weight=f1_score(y_test,nb_model.predict(X_test), average='weighted')
rf_weight=f1_score(y_test,rf_final.predict(X_test), average='weighted')
xgb_weight=f1_score(y_test,xgb_final.predict(X_test), average='weighted')
lgbm_weight=f1_score(y_test,lgbm_final.predict(X_test), average='weighted')

voting_clf = VotingClassifier(estimators=[
    #('nbm',nbm_model),
    #('logreg', logreg),
    #('svm', svm_model),
    #('nb', nb_model),
    ('rf', rf_final),
    ('xgb', xgb_final),
    ('lgbm', lgbm_final)
], voting='soft',weights=[rf_weight,8*xgb_weight,4*lgbm_weight])
#[logreg_weight,rf_weight,xgb_weight,lgbm_weight])

voting_clf.fit(X_train, y_train)

y_pred = voting_clf.predict(X_test)
f1 = f1_score(y_test, y_pred, average='weighted')

print(f'F1 Score: {f1}')
```

```
F1 Score: 0.9454926624737946
```

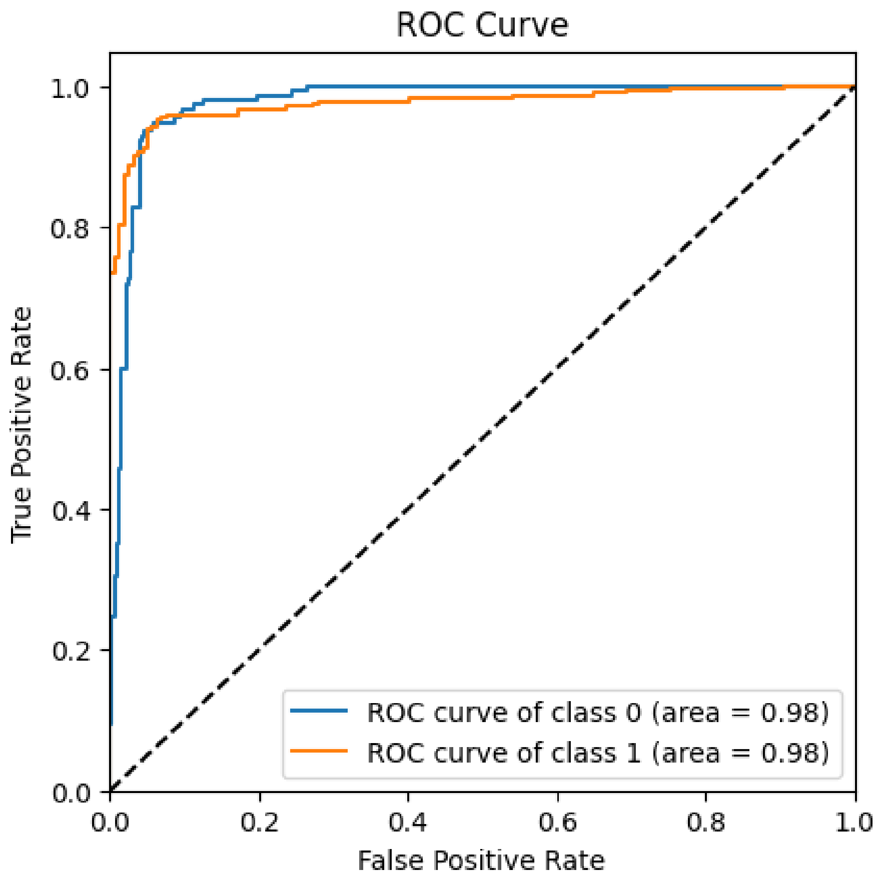
[152]:
```
color = '\033[91m'
bold = '\033[1m'
end = '\033[0m'
# Predict and evaluate performance
y_true = y_train
y_pred = voting_clf.predict(X_train)

print(color + bold + "Train data:" + color + end)
print("f1_score: ", f1_score(y_true, y_pred, average='weighted'))
print("Classification Report:\n", classification_report(y_true, y_pred))

y_true = y_test
y_pred = voting_clf.predict(X_test)

print(color + bold + "Test data:" + color + end)
print("f1_score: ", f1_score(y_true, y_pred, average='weighted'))
print("Classification Report:\n", classification_report(y_true, y_pred))
```

```
Train data:
f1_score:  0.9957964780747167
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.99      0.99       608
           1       1.00      1.00      1.00      1296

    accuracy                           1.00      1904
   macro avg       1.00      0.99      1.00      1904
weighted avg       1.00      1.00      1.00      1904

Test data:
f1_score:  0.9454926624737946
Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.92      0.92       157
           1       0.96      0.96      0.96       320

    accuracy                           0.95       477
   macro avg       0.94      0.94      0.94       477
weighted avg       0.95      0.95      0.95       477
```

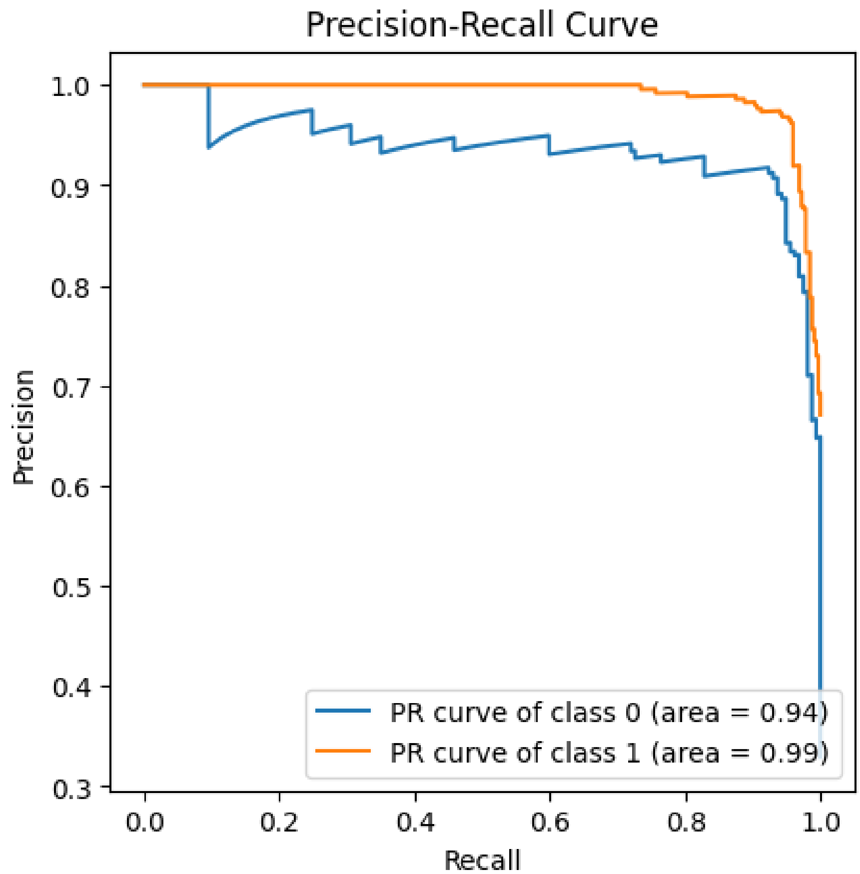
[153]:
```
display_confusion_matrix(y_test, y_pred)
```



[154]:
```
plot_roc_curve(voting_clf, X_train, X_test, y_train, y_test)
```



[155]:
```
plot_pr_curve(voting_clf, X_train, X_test, y_train, y_test)
```



8 Best Individual Ensembling Model

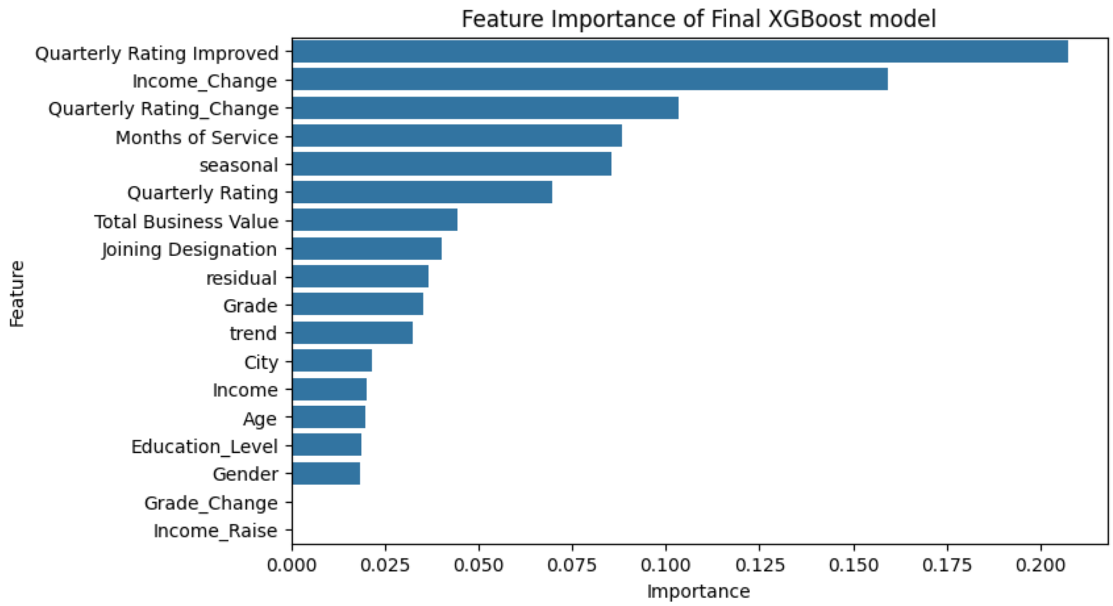
[156]:
```
model=xgb_final
model_type='XGBoost'

if rf_weight>xgb_weight and rf_weight>lgbm_weight:
    model=rf_final
    model_type='Random Forest'

elif lgbm_weight>xgb_weight and lgbm_weight>rf_weight:
    model=lgbm_final
    model_type='LightGBM'
```

[157]:
```
color = '\033[91m'
bold = '\033[1m'
end = '\033[0m'
# Predict and evaluate performance
y_true = y_train
```

```
y_pred = model.predict(X_train)

print(color + bold + "Train data:" + color + end)
print("f1_score: ", f1_score(y_true, y_pred, average='weighted'))
print("Classification Report:\n", classification_report(y_true, y_pred))

y_true = y_test
y_pred = model.predict(X_test)

print(color + bold + "Test data:" + color + end)
print("f1_score: ", f1_score(y_true, y_pred, average='weighted'))
print("Classification Report:\n", classification_report(y_true, y_pred))
```

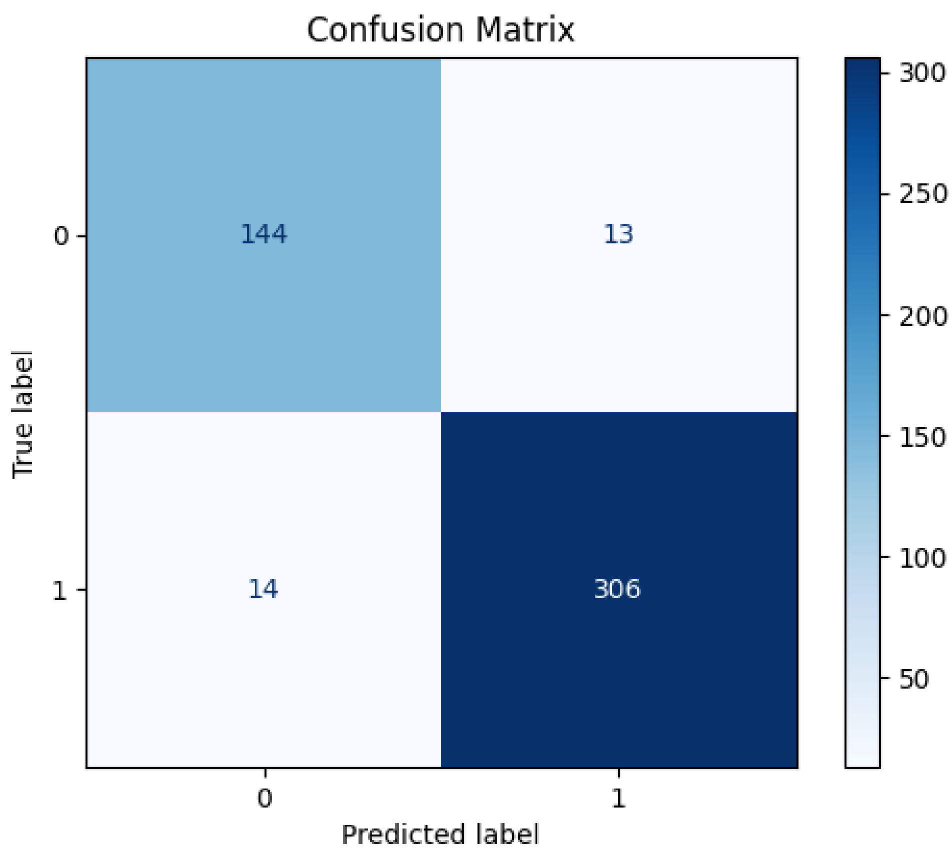
```
Train data:
f1_score:  0.9889681761493502
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.98      0.98       608
           1       0.99      0.99      0.99      1296

    accuracy                           0.99      1904
   macro avg       0.99      0.99      0.99      1904
weighted avg       0.99      0.99      0.99      1904

Test data:
f1_score:  0.9434417827721043
Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.92      0.91       157
           1       0.96      0.96      0.96       320

    accuracy                           0.94       477
   macro avg       0.94      0.94      0.94       477
weighted avg       0.94      0.94      0.94       477
```

[158]:
```
plot_feature_importance(model, X_train.columns, model_type)
```



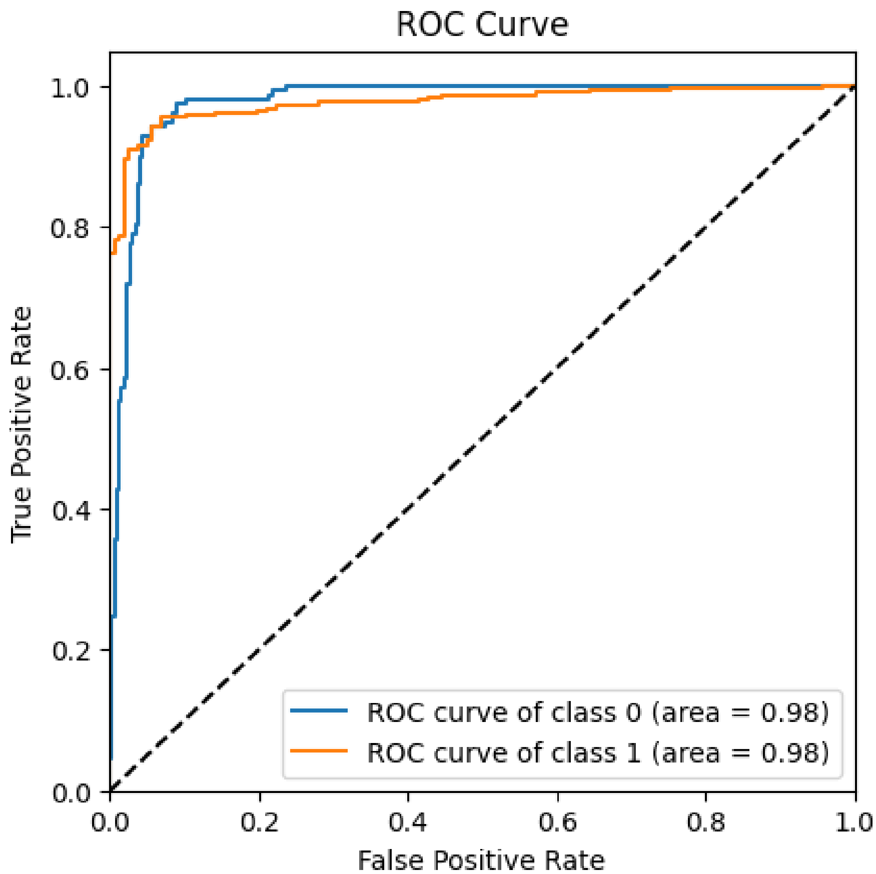
[159]:

```
display_confusion_matrix(y_test, y_pred)
```



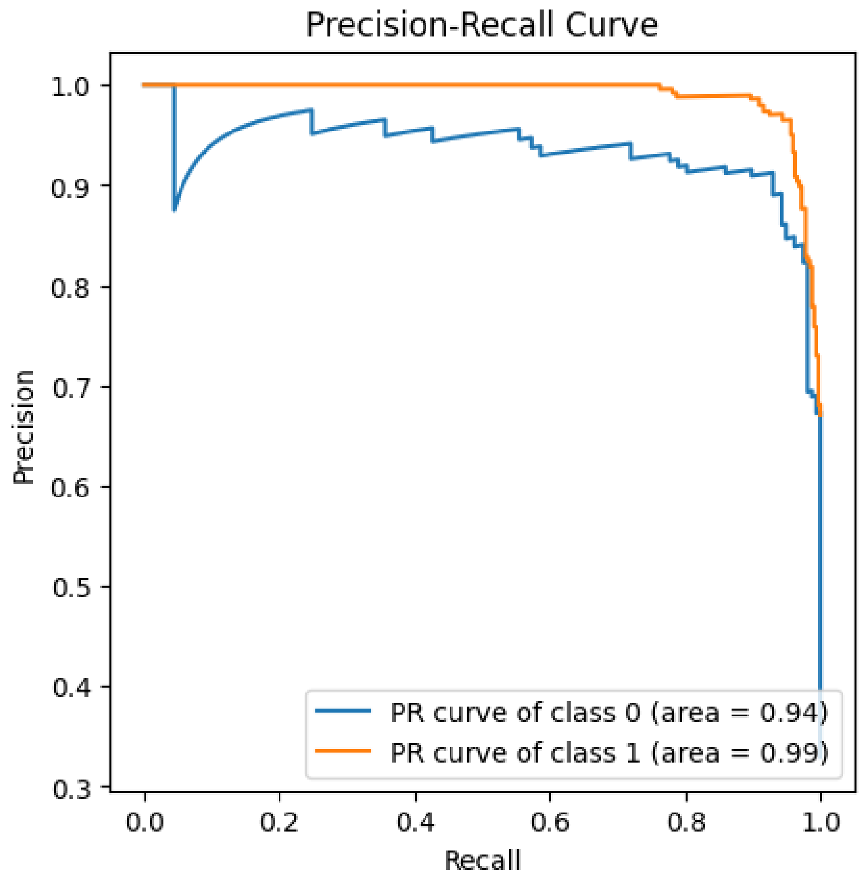
[160]:

```
plot_roc_curve(model, X_train, X_test, y_train, y_test)
```



[161]:

```
plot_pr_curve(model, X_train, X_test, y_train, y_test)
```



9 Insights

- **Most** of the drivers are in the age group of **30 to 35**
- **59%** of the drivers are **Male** and remaining **41%** are **Female**
- City **C20** has the **maximum** number of **drivers**
- Maximum number of drivers joined in the year **2020** and in the month of **July**
- 1026 drivers have a **joining designation of 1**
- Maximum number of drivers have a **grade of 2**
- **Majority** of the drivers have a very **low quarterly rating of 1**
- There are **no drivers** with quarterly rating of **5**
- **1616** drivers have **churned**, which is around **68%**
- The **median income** of drivers who have **churned** is **lesser** than that of the drivers who have not churned
- The **churn rate** is **very less** in drivers whose **income has raised**
- The **churn rate** is **very less** in drivers whose **grade has raised**
- The **churn rate** is **very less** in drivers whose **Quarterly rating has increased**

10 Recommendation

- The quartely rating has been the top contibutor on deciding if a driver will churn or not. As the ratings are given by the customers to the driver, Ola should urge all customers to rate the drivers on time. Ola should provide incentives/points to the customers to encourage timely rating.
- Ola should make sure that the income of deserving drivers should be increased every 6 months, if not every quarter, to encourage drivers to stay
- Long service awards/bonuses should be given to drivers to keep them motivated
- Special trainings should be given to drivers on how to handle different customers and different situations so that the customers always provide positive ratings

11 Questions

11.1 1 What percentage of drivers have received a quarterly rating of 5?

Ans: No drivers have received a quarterly rating of 5.

11.2 2 Comment on the correlation between Age and Quarterly Rating.

Ans: Age and Quarterly rating do not have much correlation. They have a small correlation value of 0.15.

11.3 3 Name the city which showed the most improvement in Quarterly Rating over the past year

Ans: The city C29 shows most improvement in Quarterly Rating in 2020 compared to 2019.

11.4 4 Drivers with a Grade of 'A' are more likely to have a higher Total Business Value. (T/F)

Ans: Yes, the mean of Total Business Value of drivers with grade 5(or A) is higher than those with other grades.

11.5 5 If a driver's Quarterly Rating drops significantly, how does it impact their Total Business Value in the subsequent period?

Ans: A significant drop in rating leads to dip in the Total Business Value in the subsequesnt period. Drop in rating demotivates the drivers, leading to accepting only a few rides or in somecases not accepting any rides and hence impacting the Total Business Value.

11.6 6 From Ola's perspective, which metric should be the primary focus for driver retention? 1. ROC AUC, 2. Precision, 3. Recall, 4. F1 Score

Ans: Recall. It is ok if the model predicts most drivers as **Churn** but it should not predict **Churn** drivers as **Not Churn**.

11.7 7 How does the gap in precision and recall affect Ola's relationship with its drivers and customers?

Ans: Gap in the precision and recall implies that the False Negatives and False Positives values are very different. If more instances of Churn are misclassified as Not Churn, then the customers may get drives who are not-motived/unsatisfied leading to bad customer experience. On the other hand if more instances of Not Churn are misclassified as Churn, then the good performing drivers will be neglected leading to driver dissatification.

11.8 8 Besides the obvious features like "Number of Rides", which lesser-discussed features might have a strong impact on a driver's Quarterly Rating?

Ans: 1) Customers not providing timely rating or providing false rating has a strong impact on high performing drivers and their quarterly rating.
2) Lack of training to the driver on handling different situation can also impact their quarterly rating. Not all customers are same, so the driver needs to adapt his behaviour as per the customer.

11.9 9 Will the driver's performance be affected by the City they operate in? (Yes/No)

Ans: Yes, it might be the case that the people(customers) of a city are of a particular mindset. The people of a city could be more accomodative and provide good ratings always and people of a different city could get irriated easily and provide bad ratings.

11.10 10 Analyze any seasonality in the driver's ratings. Do certain times of the year correspond to higher or lower ratings, and why might that be?

Ans: Yes, there is a seasonality in the driver's rating. The ratings dip in Q2 and then shoot up in Q3. This could be becuase of the holiday season in Q2 when many people move out of the cities for vacation and hence less usage of cabs.

[ ]:

```

```