Business Case: OLA - Ensemble Learning

Problem Statement

Recruiting and retaining drivers is seen by industry watchers as a tough battle for Ola. Churn among drivers is high and it's very easy for drivers to stop working for the service on the fly or jump to Uber depending on the rates.

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

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

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

In [2]:
```
# libraries to analyze data
import numpy as np
import pandas as pd

# libraries to visualize data
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 OneHotEncoder, MinMaxScaler, label_binarize
from sklearn.model_selection import train_test_split, RandomizedSearchCV, GridSearchCV, validation_curve
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, ConfusionMatrixDisplay, roc_curve, auc, p
from sklearn.multiclass import OneVsRestClassifier

from imblearn.over_sampling import SMOTE

from scipy.stats import randint

from xgboost import XGBClassifier

from lightgbm import LGBMClassifier
```

In [3]:
```
df=pd.read_csv('https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/002/492/original/ola_driver_scaler.csv')
df
```

```
/usr/local/lib/python3.12/dist-packages/google/colab/_dataframe_summarizer.py:88: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  cast_date_col = pd.to_datetime(column, errors="coerce")
```

Out[3]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	03/11/19	1	1
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2
...	...	...	...	...	...	...	...	...	...	...	...	...
19099	19099	08/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2
19100	19100	09/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2
19101	19101	10/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2
19102	19102	11/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2
19103	19103	12/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2

19104 rows × 14 columns

In [4]:
```
df.info()
```

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

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

Out[5]:

	Unnamed: 0	Driver_ID	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value
count	19104.000000	19104.000000	19043.000000	19052.000000	19104.000000	19104.000000	19104.000000	19104.000000	1.910400e+04
mean	9551.500000	1415.591133	34.668435	0.418749	1.021671	65652.025126	1.690536	2.252670	5.716621e+05
std	5514.994107	810.705321	6.257912	0.493367	0.800167	30914.515344	0.836984	1.026512	1.128312e+06
min	0.000000	1.000000	21.000000	0.000000	0.000000	10747.000000	1.000000	1.000000	-6.000000e+06
25%	4775.750000	710.000000	30.000000	0.000000	0.000000	42383.000000	1.000000	1.000000	0.000000e+00
50%	9551.500000	1417.000000	34.000000	0.000000	1.000000	60087.000000	1.000000	2.000000	2.500000e+05
75%	14327.250000	2137.000000	39.000000	1.000000	2.000000	83969.000000	2.000000	3.000000	6.997000e+05
max	19103.000000	2788.000000	58.000000	1.000000	2.000000	188418.000000	5.000000	5.000000	3.374772e+07

In [6]:
```
df.shape
```

Out[6]:
```
(19104, 14)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:

	0
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

- There are missing values in Age (61), Gender (52), and a very high count in LastWorkingDate (17,488) because only some employees have left the company, while all other columns have no missing data.

Drop unwanted columns

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

In [9]:
```
# 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
/tmp/ipython-input-2093539267.py:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chain
ed assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  df['Gender'].replace({0.0:'Male', 1.0: 'Female'}, inplace=True)
/tmp/ipython-input-2093539267.py:4: FutureWarning: The behavior of Series.replace (and DataFrame.replace) with CategoricalDtype
is deprecated. In a future version, replace will only be used for cases that preserve the categories. To change the categories,
use ser.cat.rename_categories instead.
  df['Gender'].replace({0.0:'Male', 1.0: 'Female'}, inplace=True)
/tmp/ipython-input-2093539267.py:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chain
ed assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  df['Education_Level'].replace({0:'10+', 1:'12+', 2:'Graduate'}, inplace=True)
/tmp/ipython-input-2093539267.py:5: FutureWarning: The behavior of Series.replace (and DataFrame.replace) with CategoricalDtype
is deprecated. In a future version, replace will only be used for cases that preserve the categories. To change the categories,
use ser.cat.rename_categories instead.
  df['Education_Level'].replace({0:'10+', 1:'12+', 2:'Graduate'}, inplace=True)
```

In [10]:
```
df.head(5)
```

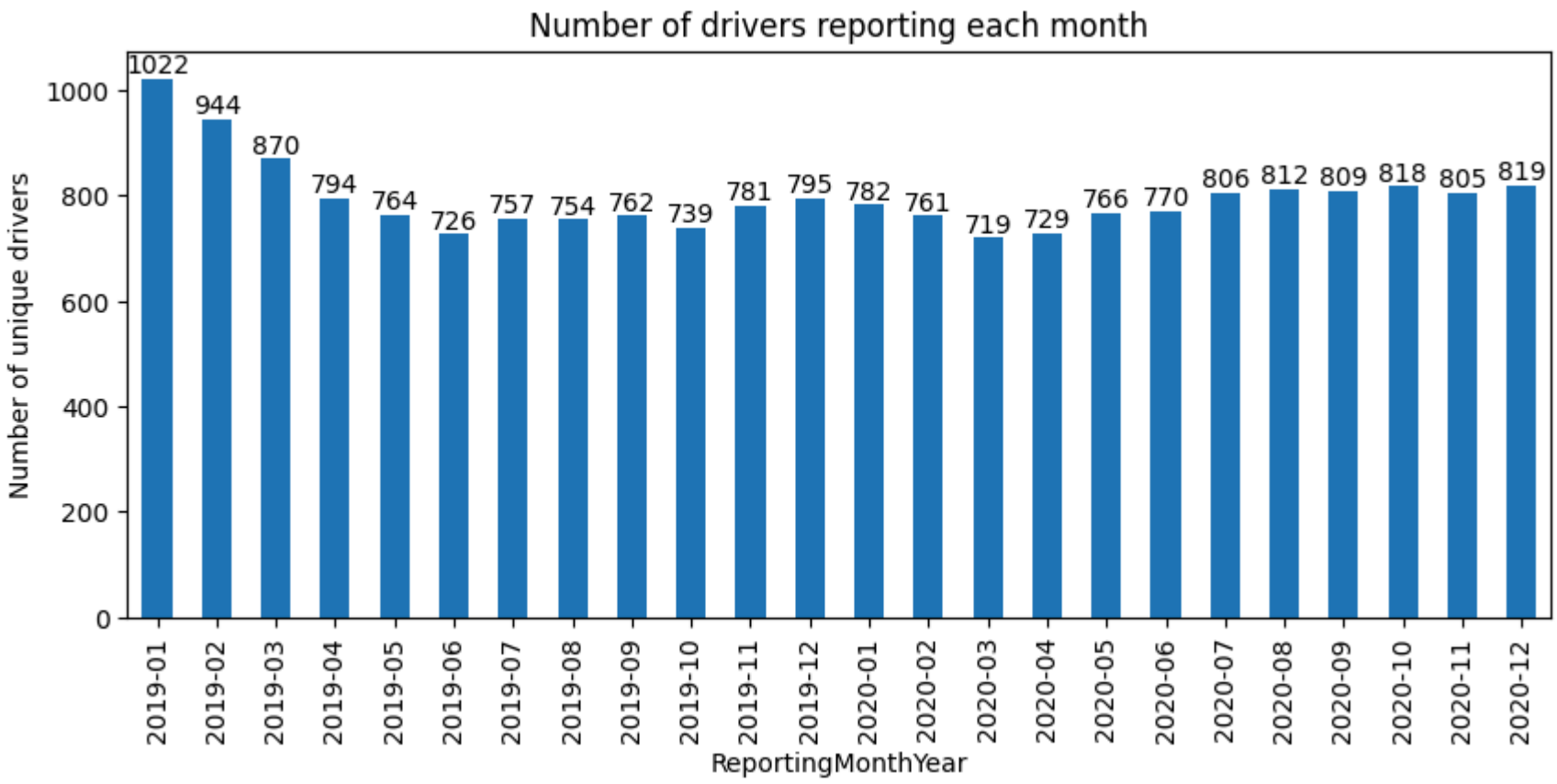
Out[10]:

	ReportingMonthYear	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Bu
0	2019-01	1	28.0	Male	C23	Graduate	57387	2018-12-24	NaT	1	1	23
1	2019-02	1	28.0	Male	C23	Graduate	57387	2018-12-24	NaT	1	1	-6
2	2019-03	1	28.0	Male	C23	Graduate	57387	2018-12-24	2019-11-03	1	1	
3	2020-11	2	31.0	Male	C7	Graduate	67016	2020-06-11	NaT	2	2	
4	2020-12	2	31.0	Male	C7	Graduate	67016	2020-06-11	NaT	2	2	

Exploratory Data Analysis

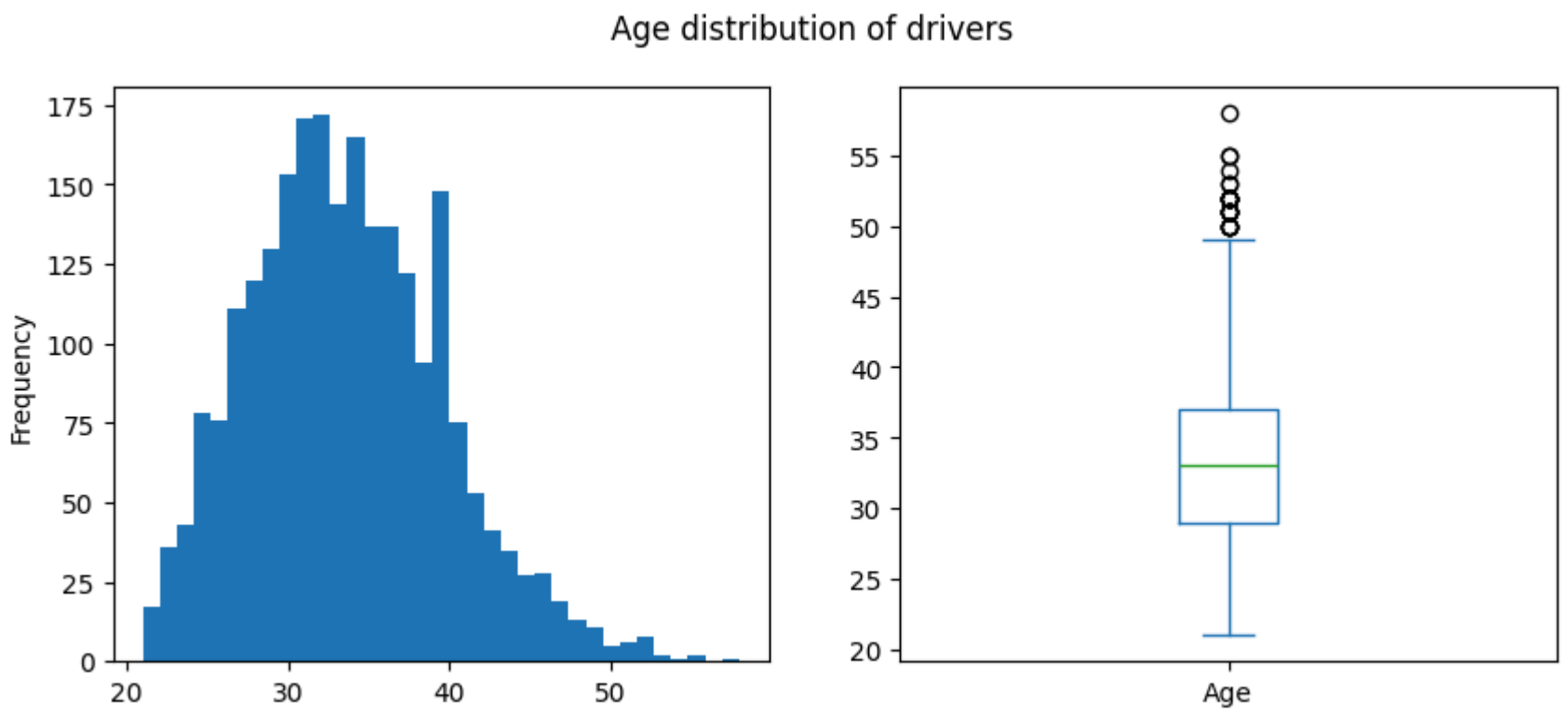
1-Univariate analysis

In [11]:
```
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()
```



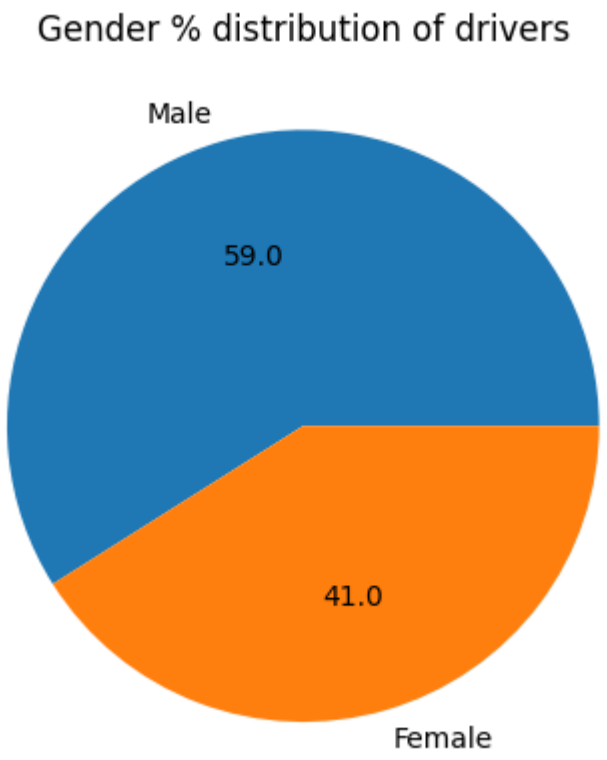
- The month during which maximum number of drivers reported is January 2019. A total of 1022 drivers reported on January 2019
- It then dropeed every month after January and has been stagnant at around 800 drivers reported every month

In [12]:
```
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()
```



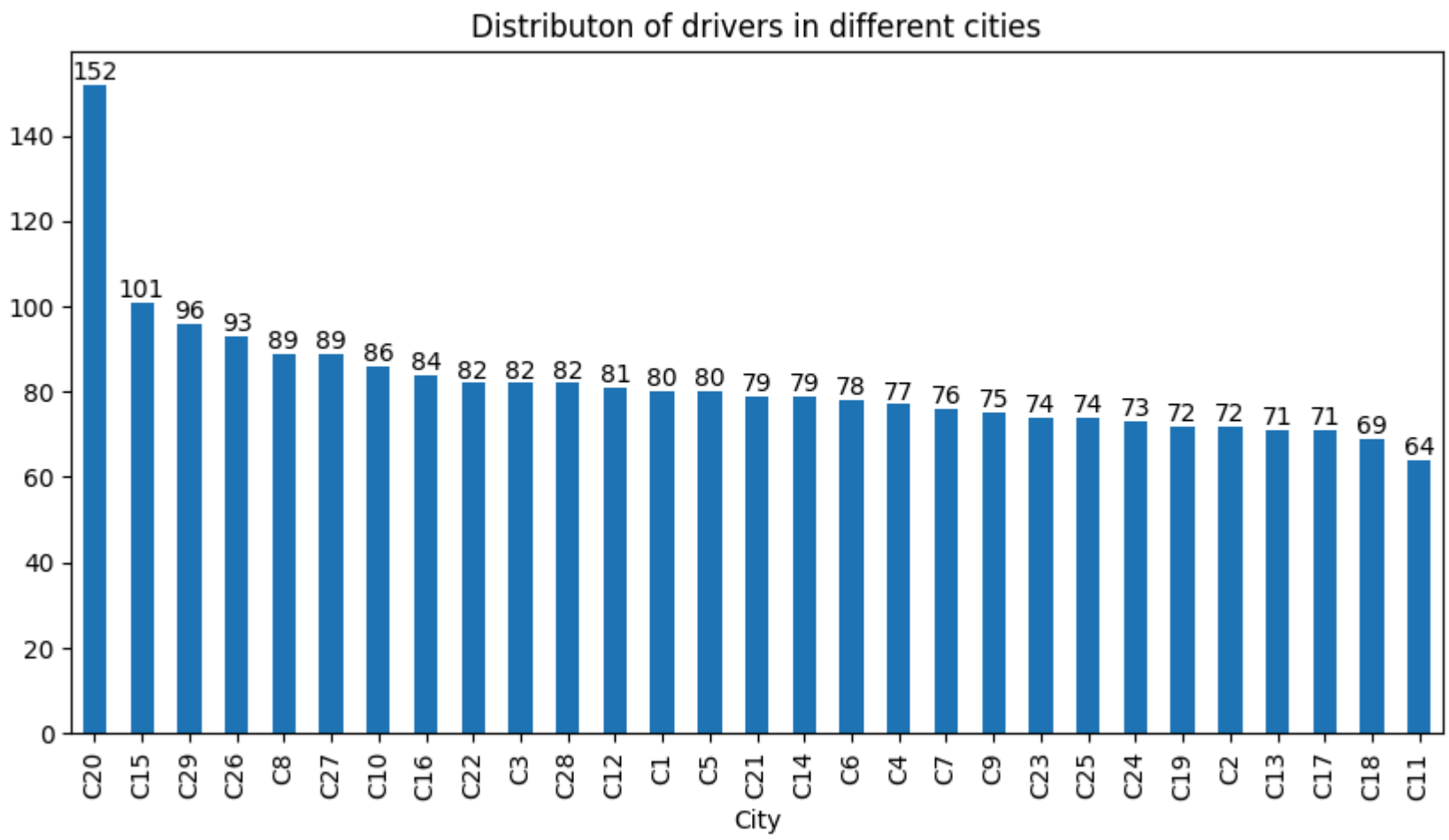
- There are drivers from different age groups ranging from 21 to 58 years
- Most of the drivers are in the age group of 30 to 35
- The distribution is mostly normal with little skewness towards the right

In [13]:
```
temp_df = df.groupby('Driver_ID').agg({'Gender':'first'})
temp_df['Gender'].value_counts().plot(kind='pie', autopct='%.1f')
plt.title('Gender % distribution of drivers')
plt.ylabel('')
plt.show()
```



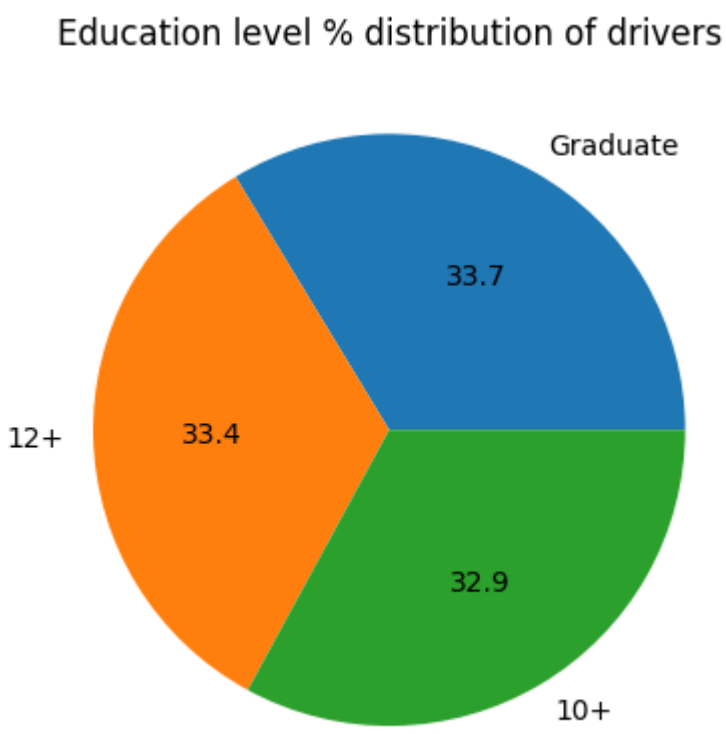
- 59% of the drivers are Male and remaining 41% are Female

In [14]:
```
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()
```



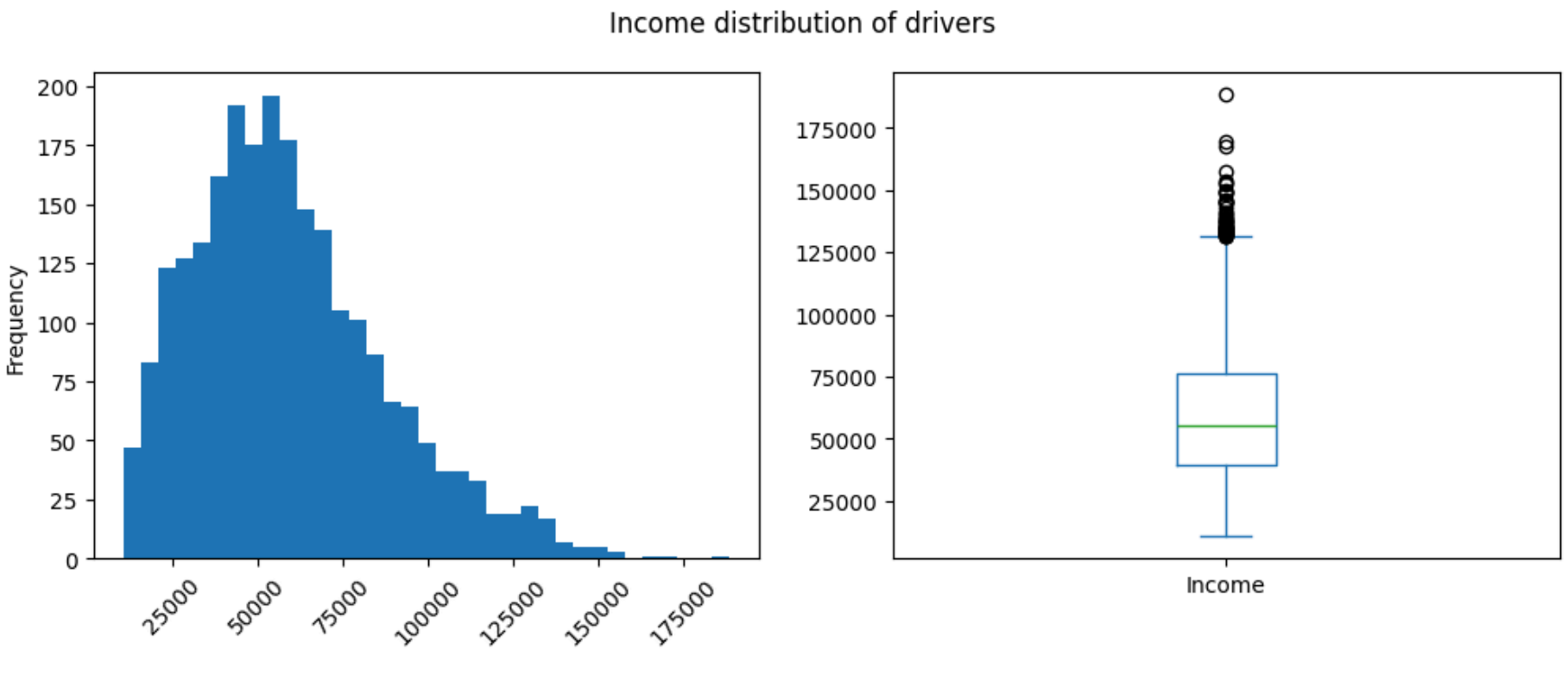
- City C20 has the maximum number of drivers followed by city C15

In [15]:
```
temp_df = df.groupby('Driver_ID').agg({'Education_Level':'first'})
temp_df['Education_Level'].value_counts().plot(kind='pie', autopct='%.1f')
plt.ylabel('')
plt.title('Education level % distribution of drivers')
plt.show()
```



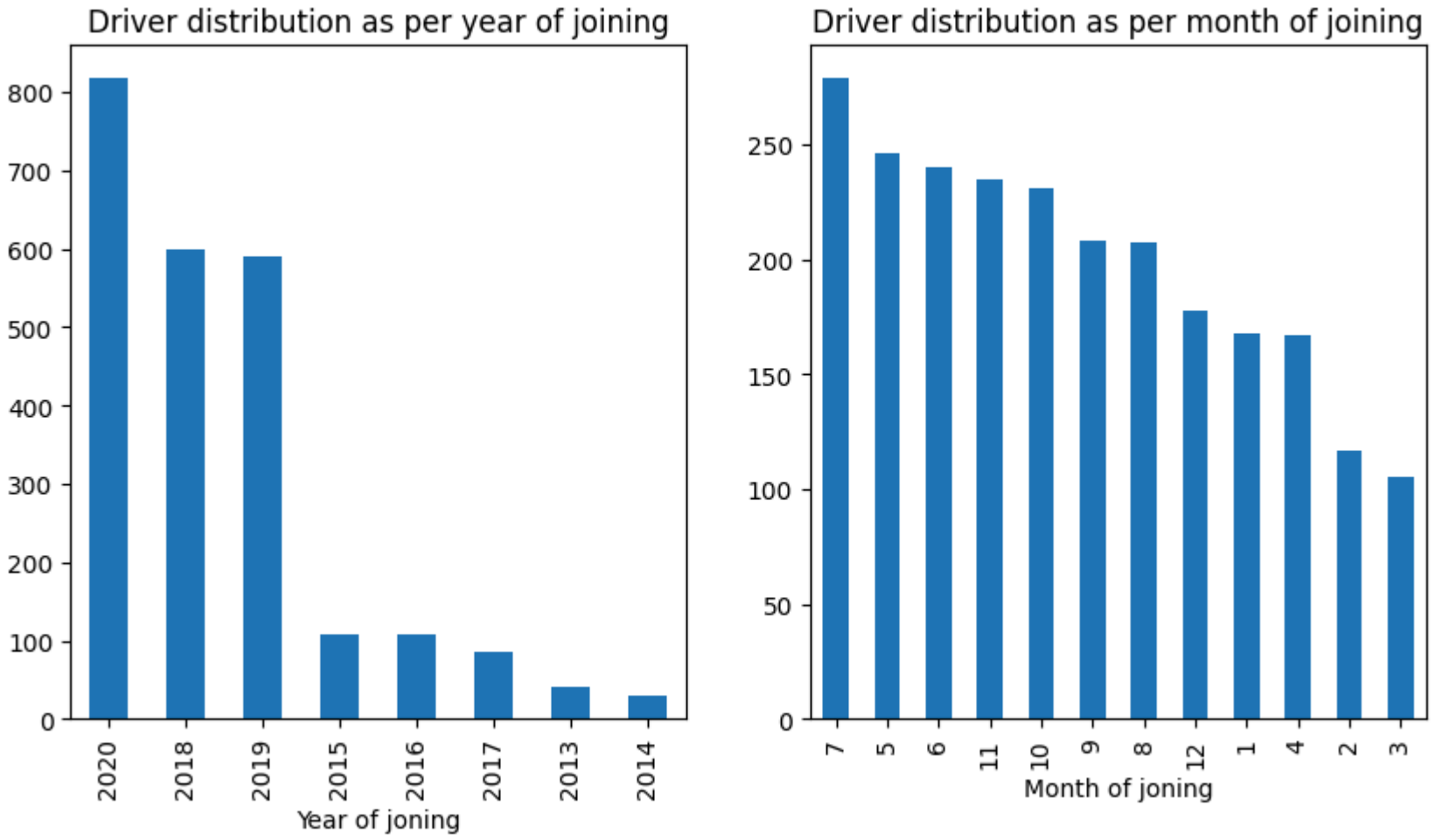
- Almost equal proportion of drivers are from the 3 different education level

In [16]:
```
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()
```



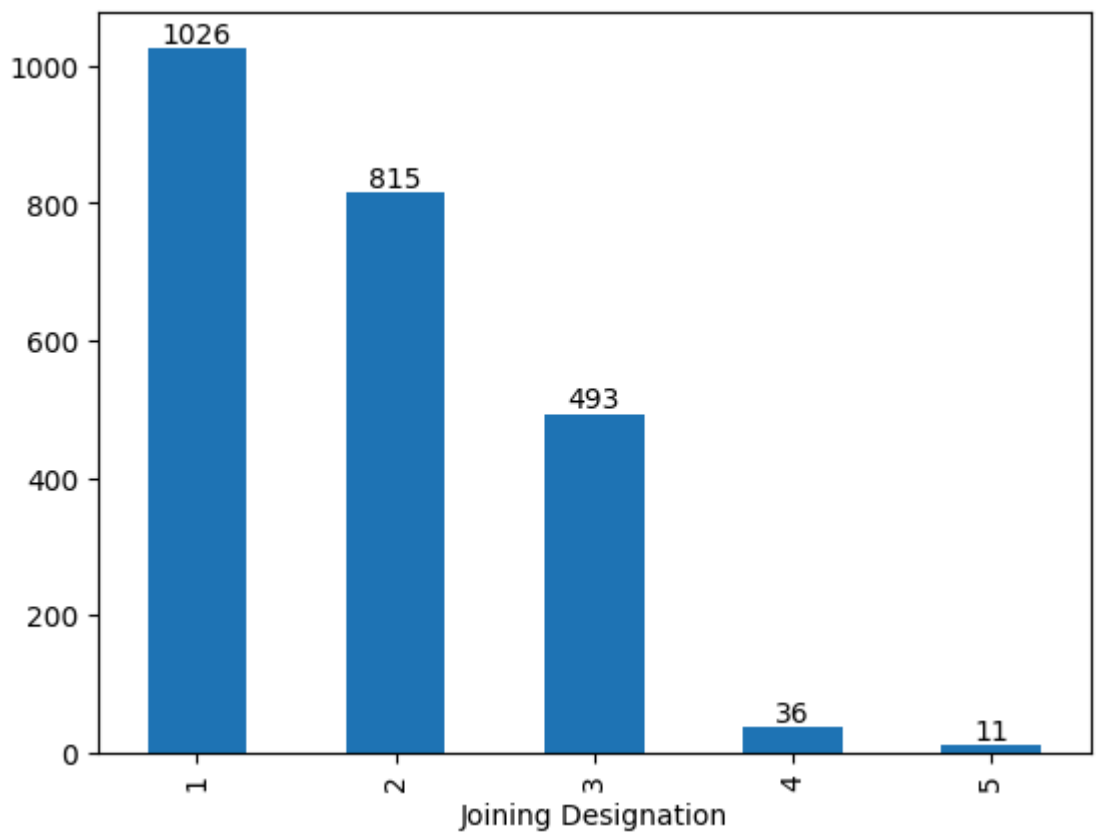
- Most of the drivers have an average monthly income of 40k to 75k
- The distribution is right skewed

In [17]:
```
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
temp_df.dt.month.value_counts().plot(kind='bar', ax=axs[1], xlabel='Month of joning', title='Driver distribution as per month
plt.show()
```



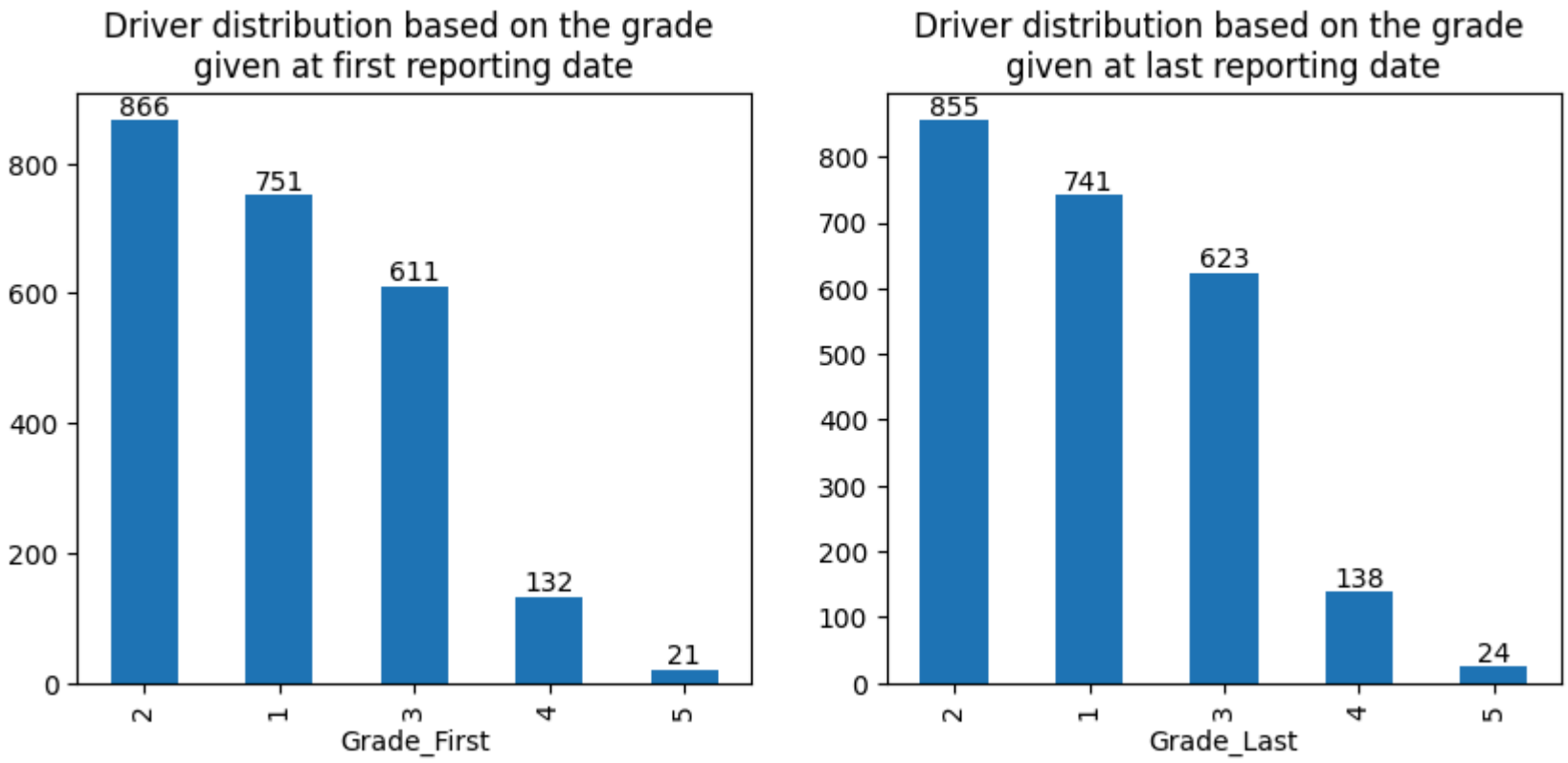
- Maximum number of drivers joined in the year 2020
- Maximum number of drivers joined in the month of July

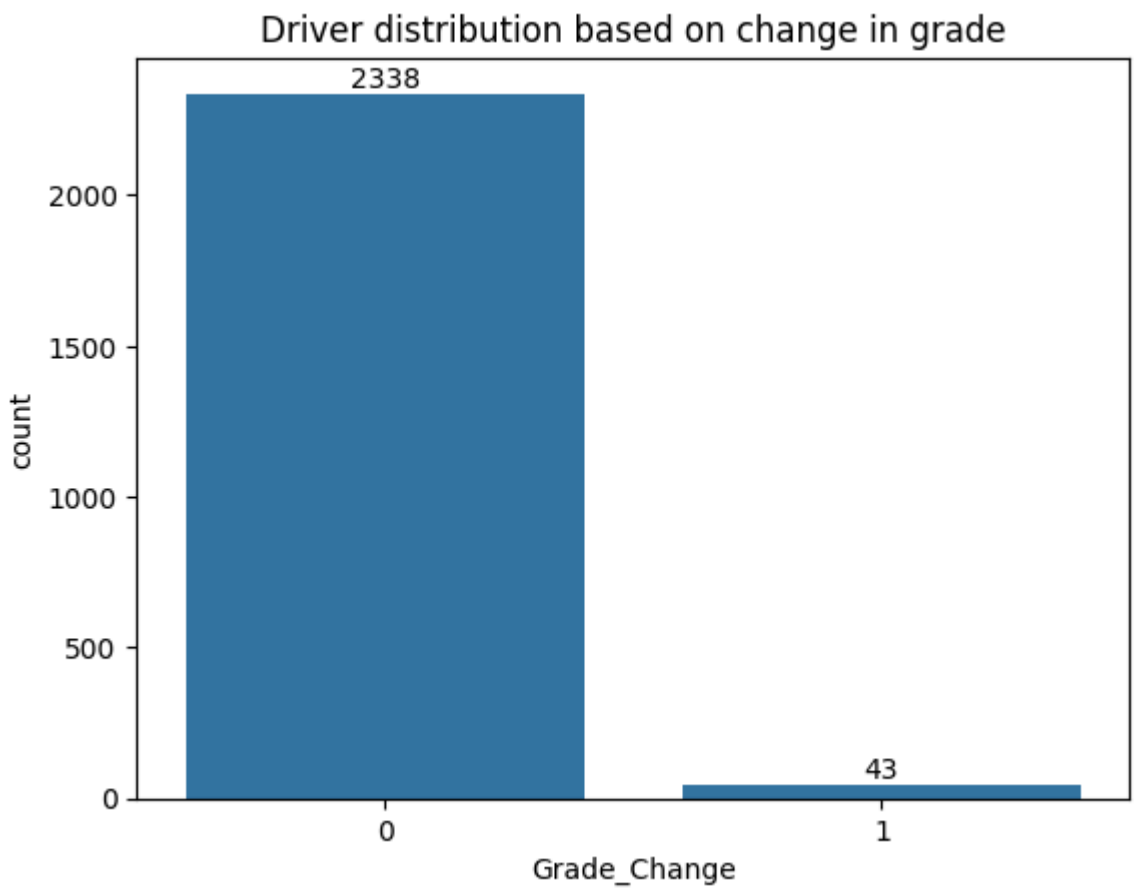
In [18]:
```
ax = df.groupby('Driver_ID').agg({'Joining Designation':'first'})['Joining Designation'].value_counts().plot(kind='bar')
ax.bar_label(ax.containers[0])
plt.show()
```



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

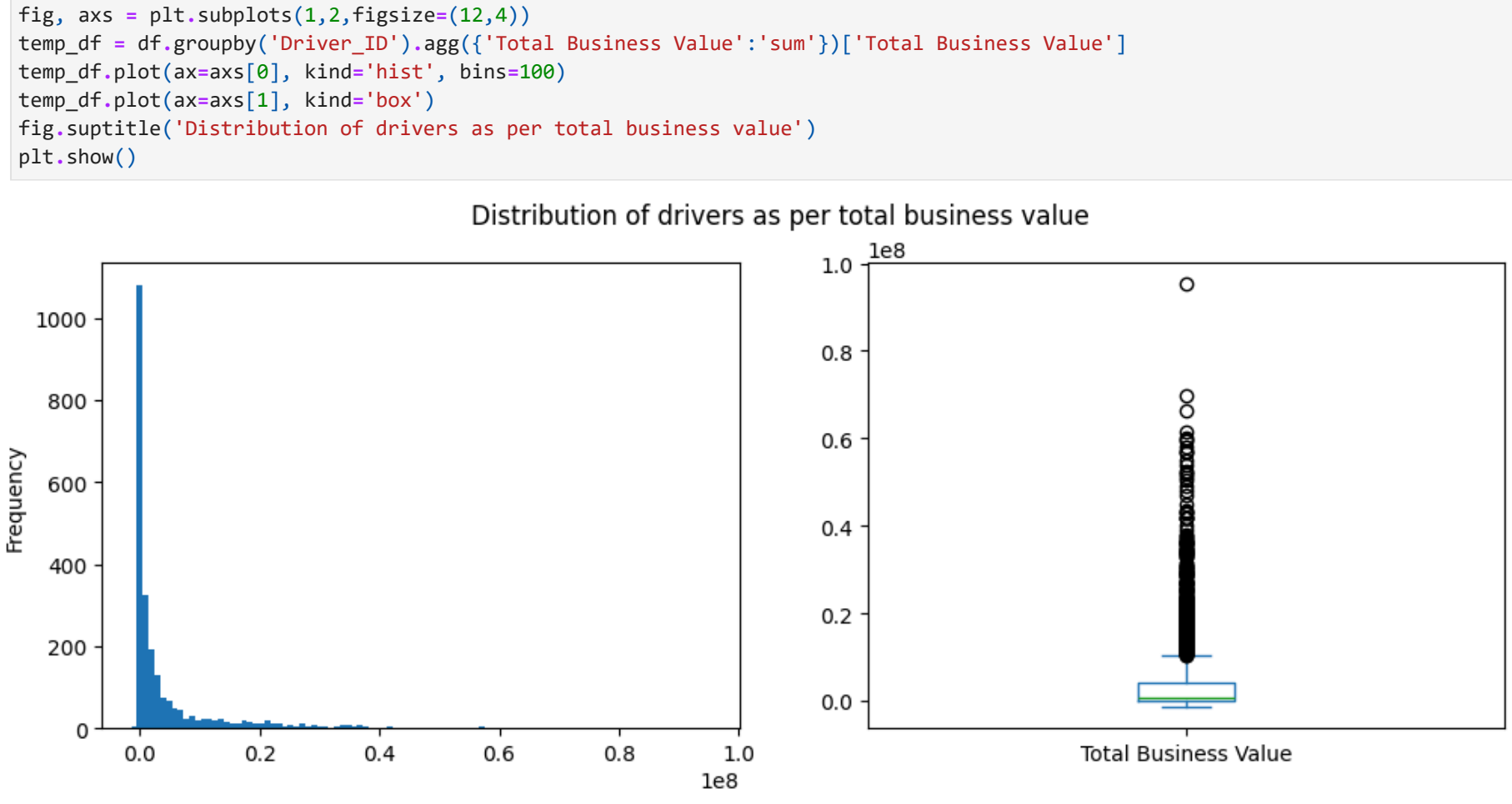
In [19]:
```
temp_df_1 = df.groupby('Driver_ID').agg({'Grade':'first'}).reset_index()
temp_df_1.rename(columns = {'Grade':'Grade_First'}, inplace=True)
temp_df_2 = df.groupby('Driver_ID').agg({'Grade':'last'}).reset_index()
temp_df_2.rename(columns = {'Grade':'Grade_Last'}, inplace=True)
temp_df = pd.merge(temp_df_1, 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 a
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
ax.bar_label(ax.containers[0])
plt.show()
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()
```





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

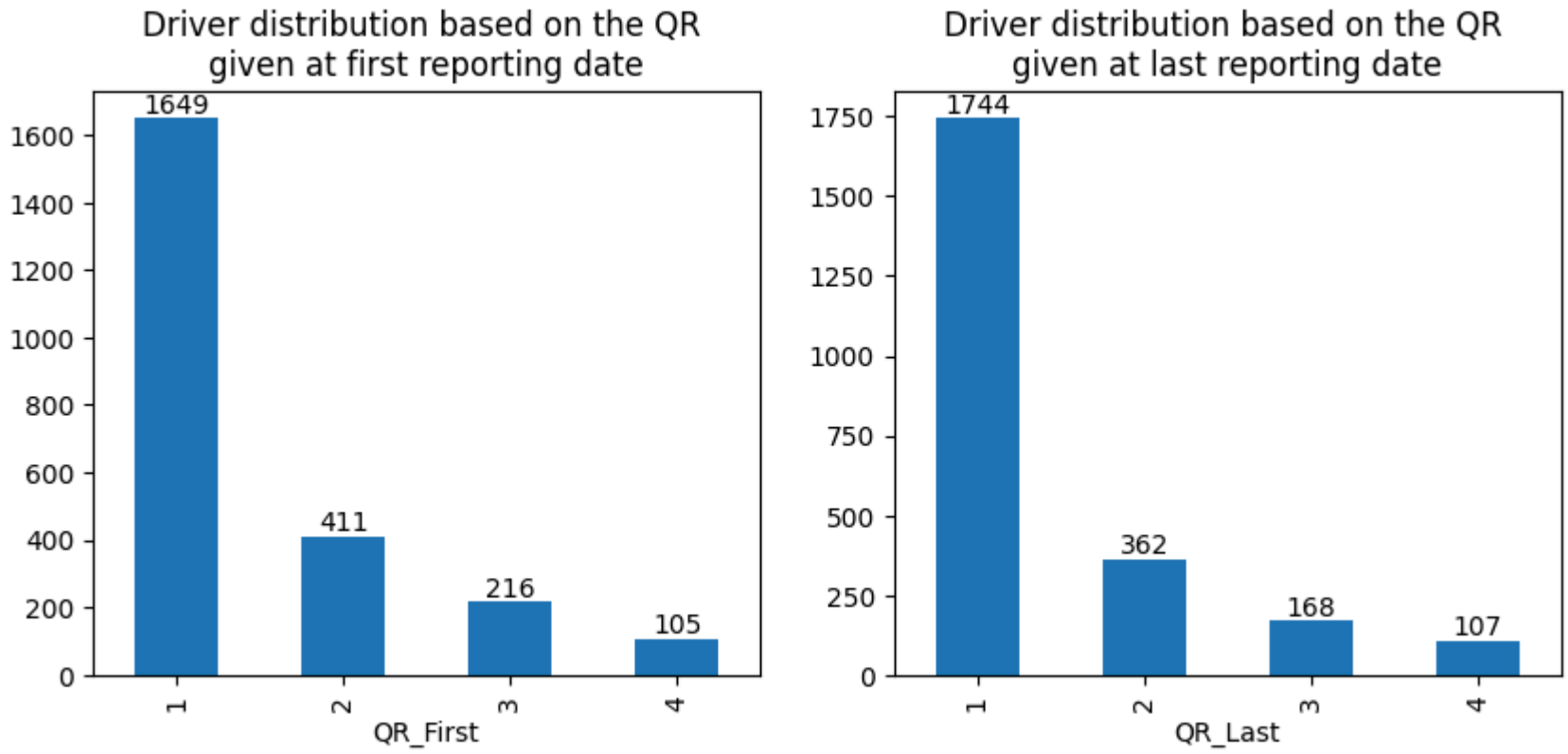
In [20]:
```
fig, axs = plt.subplots(1,2,figsize=(12,4))
temp_df = df.groupby('Driver_ID').agg({'Total Business Value':'sum'})['Total Business Value']
temp_df.plot(ax=axs[0], kind='hist', bins=100)
temp_df.plot(ax=axs[1], kind='box')
fig.suptitle('Distribution of drivers as per total business value')
plt.show()
```

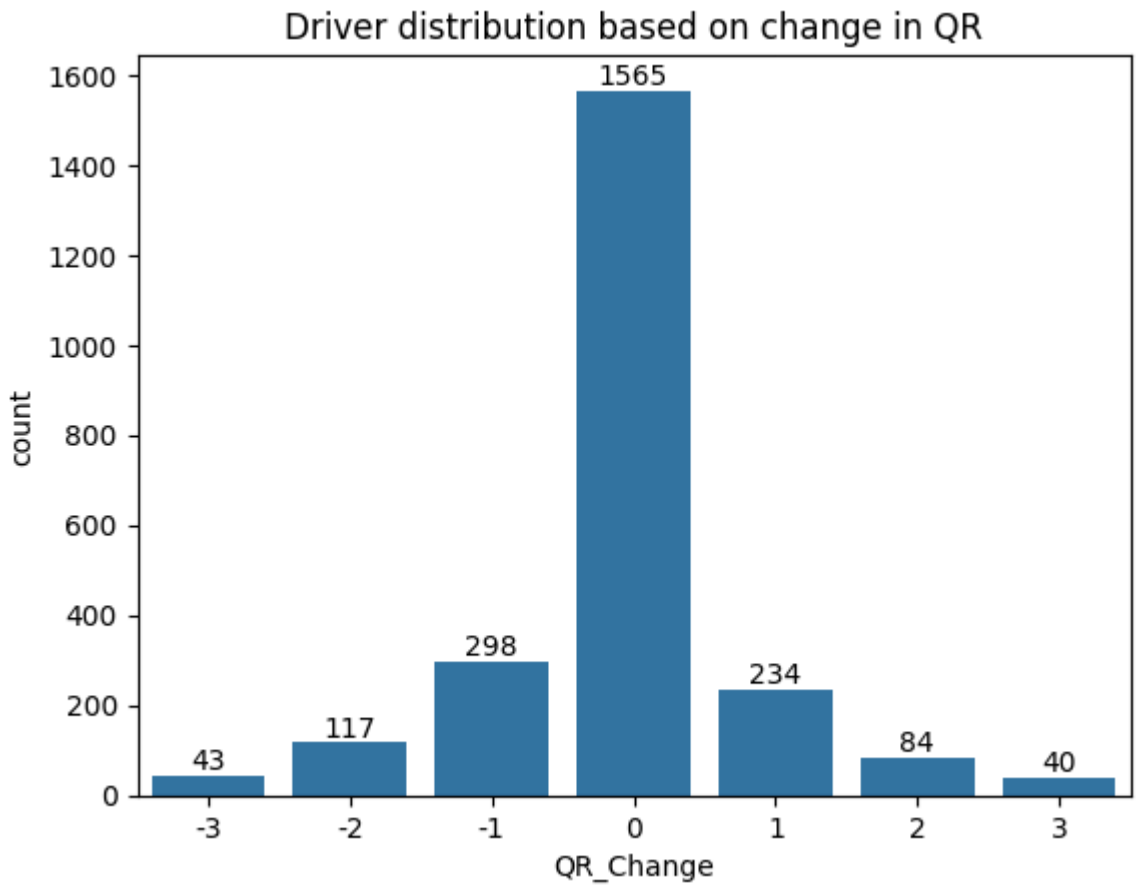


- It is very evident that many drivers have a total business value of 0 and there are also a few drivers who have a -ve business value

- The distribution is extremely right skewed

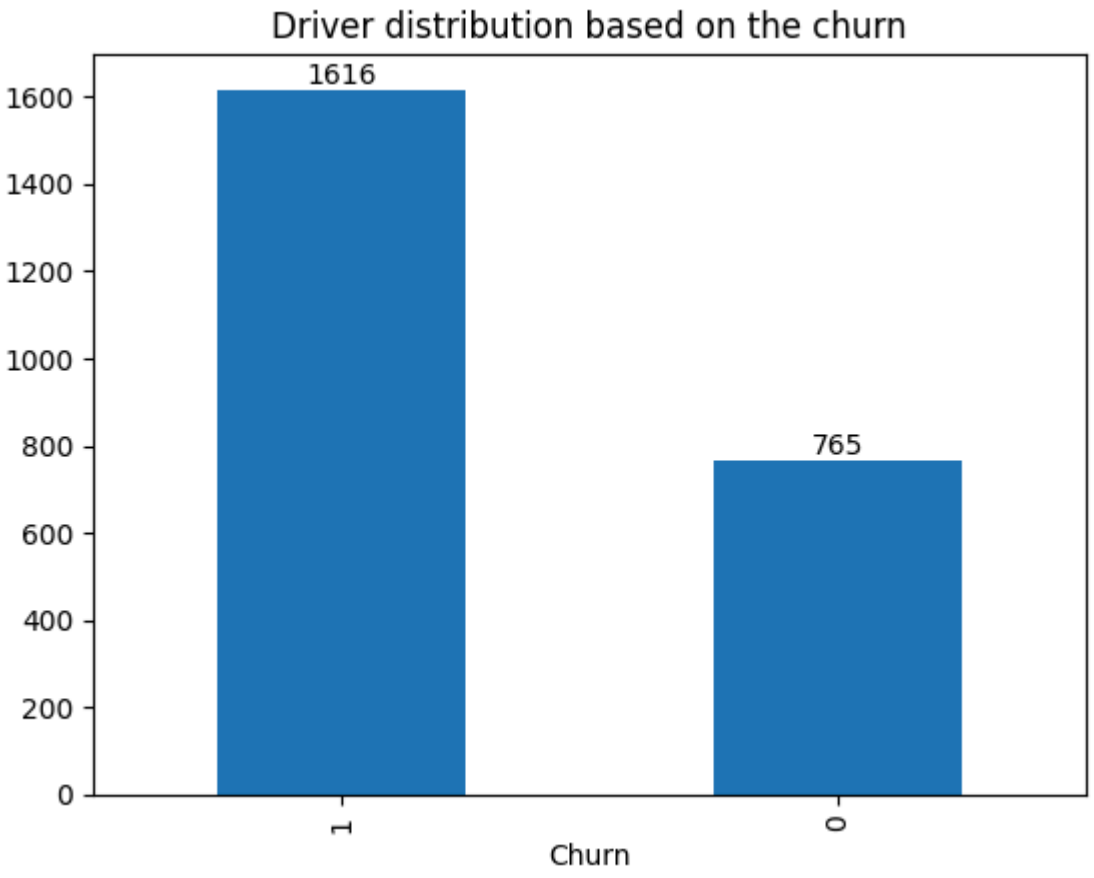
In [21]:
```
temp_df_1 = df.groupby('Driver_ID').agg({'Quarterly Rating':'first'}).reset_index()
temp_df_1.rename(columns = {'Quarterly Rating':'QR_First'}, inplace=True)
temp_df_2 = df.groupby('Driver_ID').agg({'Quarterly Rating':'last'}).reset_index()
temp_df_2.rename(columns = {'Quarterly Rating':'QR_Last'}, 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 firs
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
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()
```





- 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

In [22]:
```
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)
```



Out[22]:

proportion

Churn	
1	68.0
0	32.0

dtype: float64

- 1616 drivers have churned, which is around 68%

2-Bivariate analysis

In [23]:
```
driver_df = df.groupby('Driver_ID').agg({
    'ReportingMonthYear' : len,
    'Age' : 'last',
    'Gender' : 'first',
    'City' : 'first',
    'Education_Level' : 'first',
    'Income' : 'last',
    'Dateofjoining' : 'first',
    'LastWorkingDate' : 'last',
    'Joining Designation' : 'first',
    'Grade' : 'last',
    'Total Business Value' : 'sum',
    'Quarterly Rating' : 'last',
    'Churn':'last'
}).reset_index()
driver_df.rename(columns={'ReportingMonthYear': 'Months of Service'}, inplace=True)
driver_df.head(10)
```

Out[23]:

	Driver_ID	Months of Service	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quar R
0	1	3	28.0	Male	C23	Graduate	57387	2018-12-24	2019-11-03	1	1	1715580	
1	2	2	31.0	Male	C7	Graduate	67016	2020-06-11	NaT	2	2	0	
2	4	5	43.0	Male	C13	Graduate	65603	2019-07-12	2020-04-27	2	2	350000	
3	5	3	29.0	Male	C9	10+	46368	2019-09-01	2019-07-03	1	1	120360	
4	6	5	31.0	Female	C11	12+	78728	2020-07-31	NaT	3	3	1265000	
5	8	3	34.0	Male	C2	10+	70656	2020-09-19	2020-11-15	3	3	0	
6	11	1	28.0	Female	C19	Graduate	42172	2020-07-12	NaT	1	1	0	
7	12	6	35.0	Male	C23	Graduate	28116	2019-06-29	2019-12-21	1	1	2607180	
8	13	23	31.0	Male	C19	Graduate	119227	2015-05-28	2020-11-25	1	4	10213040	
9	14	3	39.0	Female	C26	10+	19734	2020-10-16	NaT	3	3	0	

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

In [25]:
```
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
```

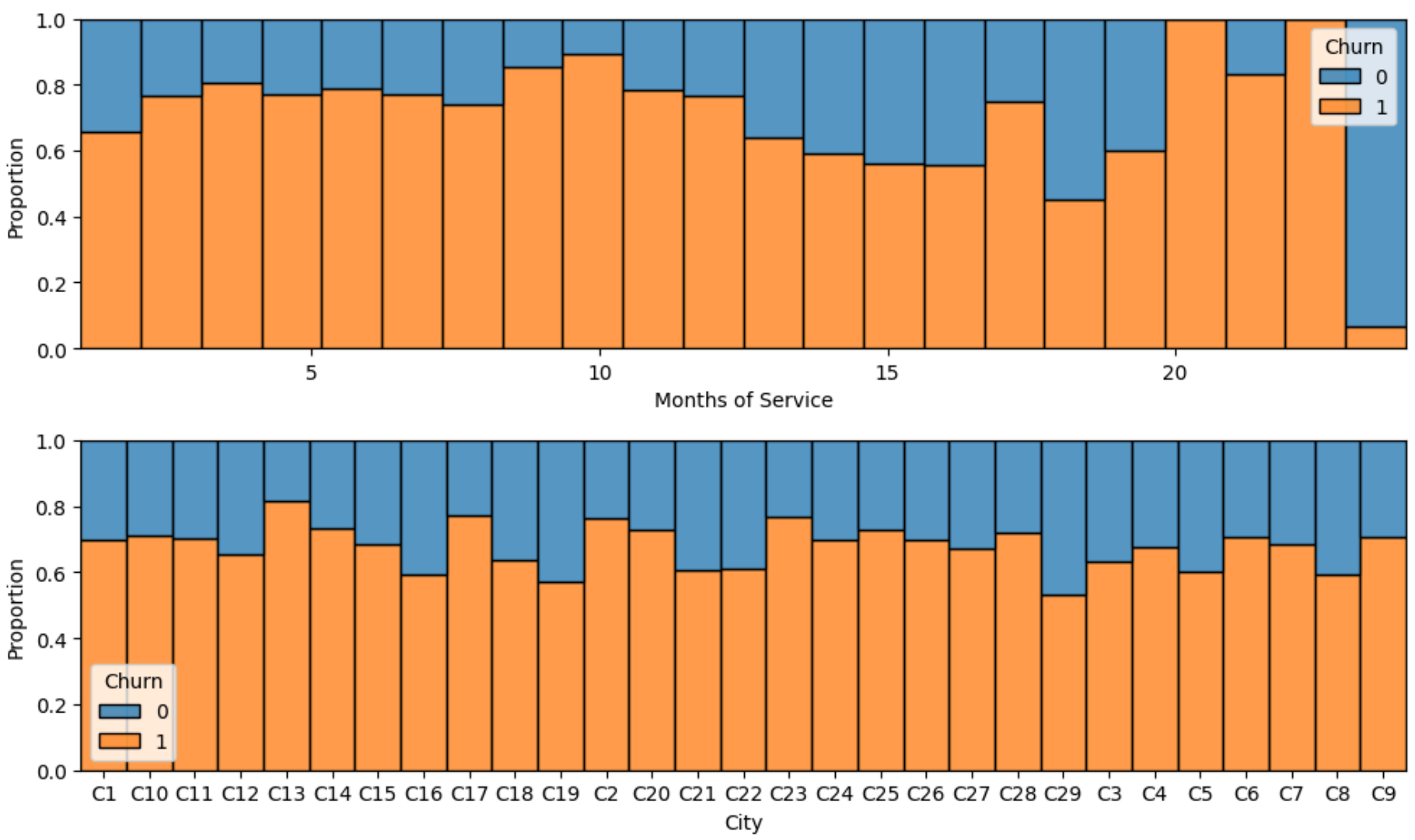
In [26]:
```
column_name = 'Income'
temp_df1 = calculate_change(df, 'Income')
driver_df = pd.merge(driver_df, temp_df1, on='Driver_ID')
temp_df2 = calculate_change(df, 'Grade')
driver_df = pd.merge(driver_df, temp_df2, on='Driver_ID')
temp_df3 = calculate_change(df, 'Quarterly Rating')
driver_df = pd.merge(driver_df, 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()
```

Out[26]:

	Driver_ID	Months of Service	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quar Ra
0	1	3	28.0	Male	C23	Graduate	57387	2018-12-24	2019-11-03	1	1	1715580	
1	2	2	31.0	Male	C7	Graduate	67016	2020-06-11	NaT	2	2	0	
2	4	5	43.0	Male	C13	Graduate	65603	2019-07-12	2020-04-27	2	2	350000	
3	5	3	29.0	Male	C9	10+	46368	2019-09-01	2019-07-03	1	1	120360	
4	6	5	31.0	Female	C11	12+	78728	2020-07-31	NaT	3	3	1265000	

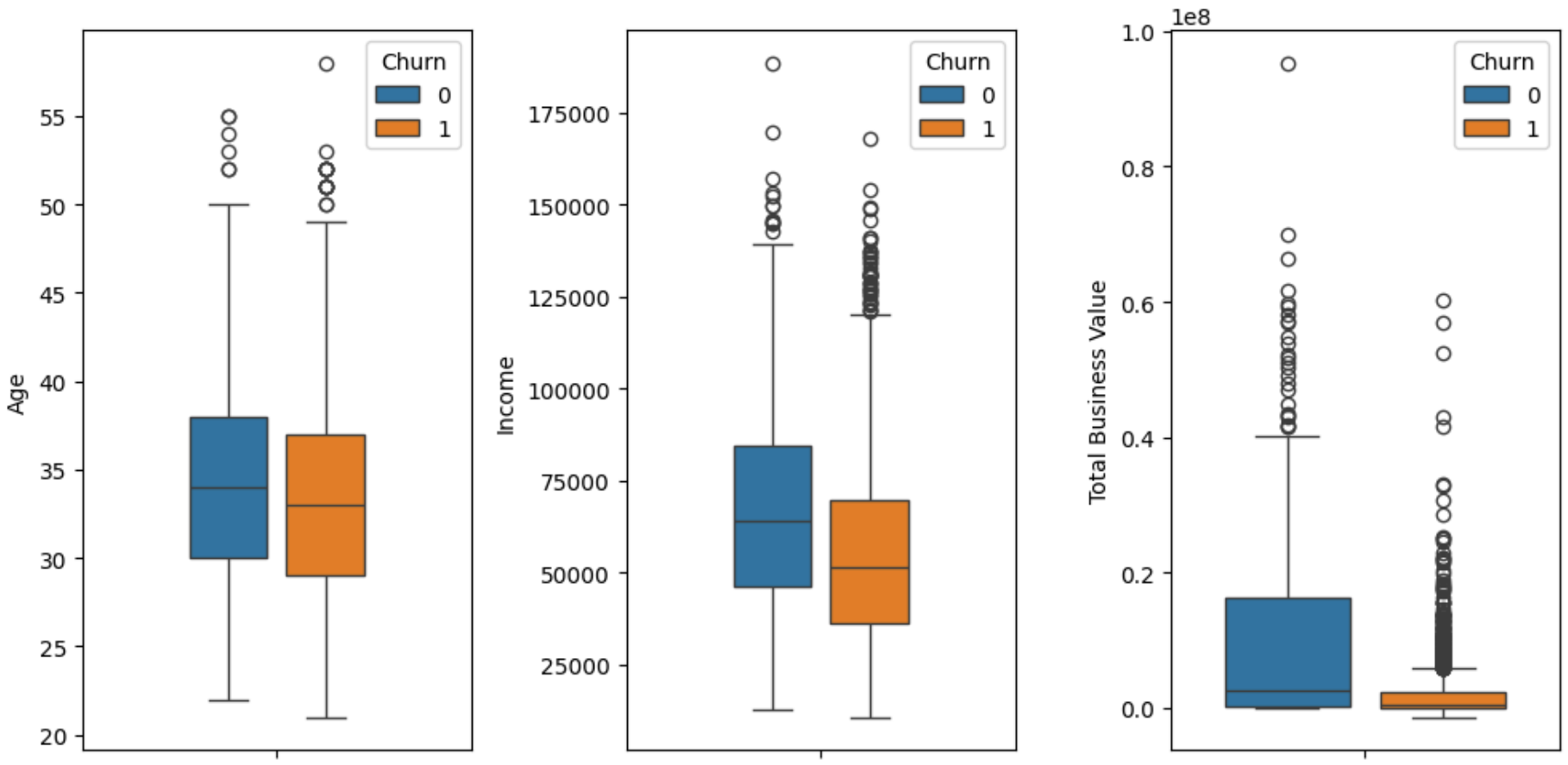
In [27]:
```
driver_df['Income_Raise'] = driver_df['Income_Change'].apply(lambda x: 1 if x>0 else 0)
```

In [28]:
```
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()
```



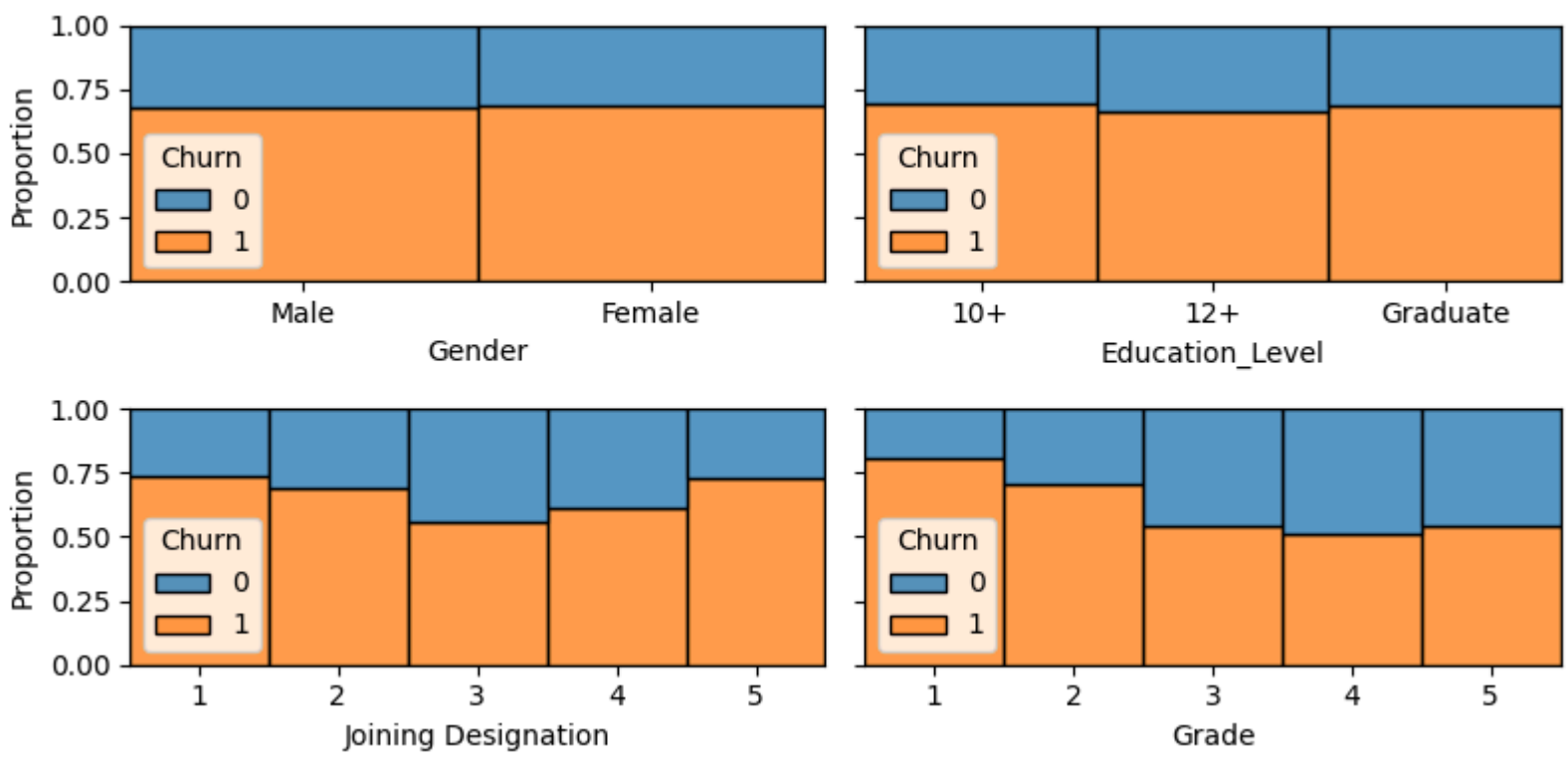
- The churn rate is generally higher in drivers with less months of service and low in drivers with longer months of service with exception for 21, 22 and 23 months of service where the churn rates seems to be very high
- The city C13 has the highest churn rate and city C29 has the lowest churn rate

In [29]:
```
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()
```



- 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

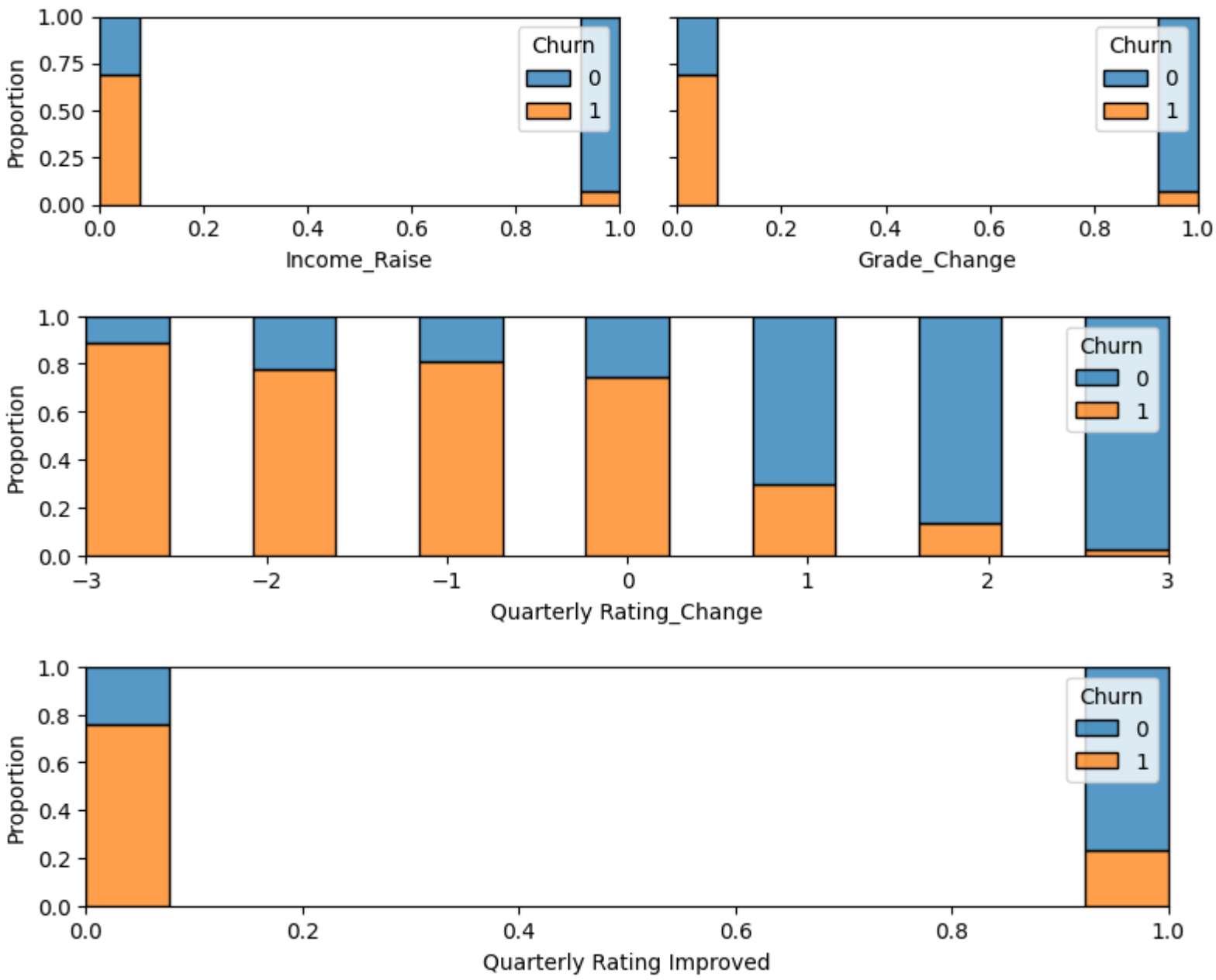
In [30]:
```
fig, axs = plt.subplots(2,2,figsize=(8,4),sharey=True)
sns.histplot(ax=axs[0,0], data=driver_df, x='Gender', hue='Churn', stat="proportion", multiple="fill")
sns.histplot(ax=axs[0,1], data=driver_df, x='Education_Level', hue='Churn', stat='proportion', multiple='fill')
sns.histplot(ax=axs[1,0], 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()
```



- 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

In [31]:
```
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))
```

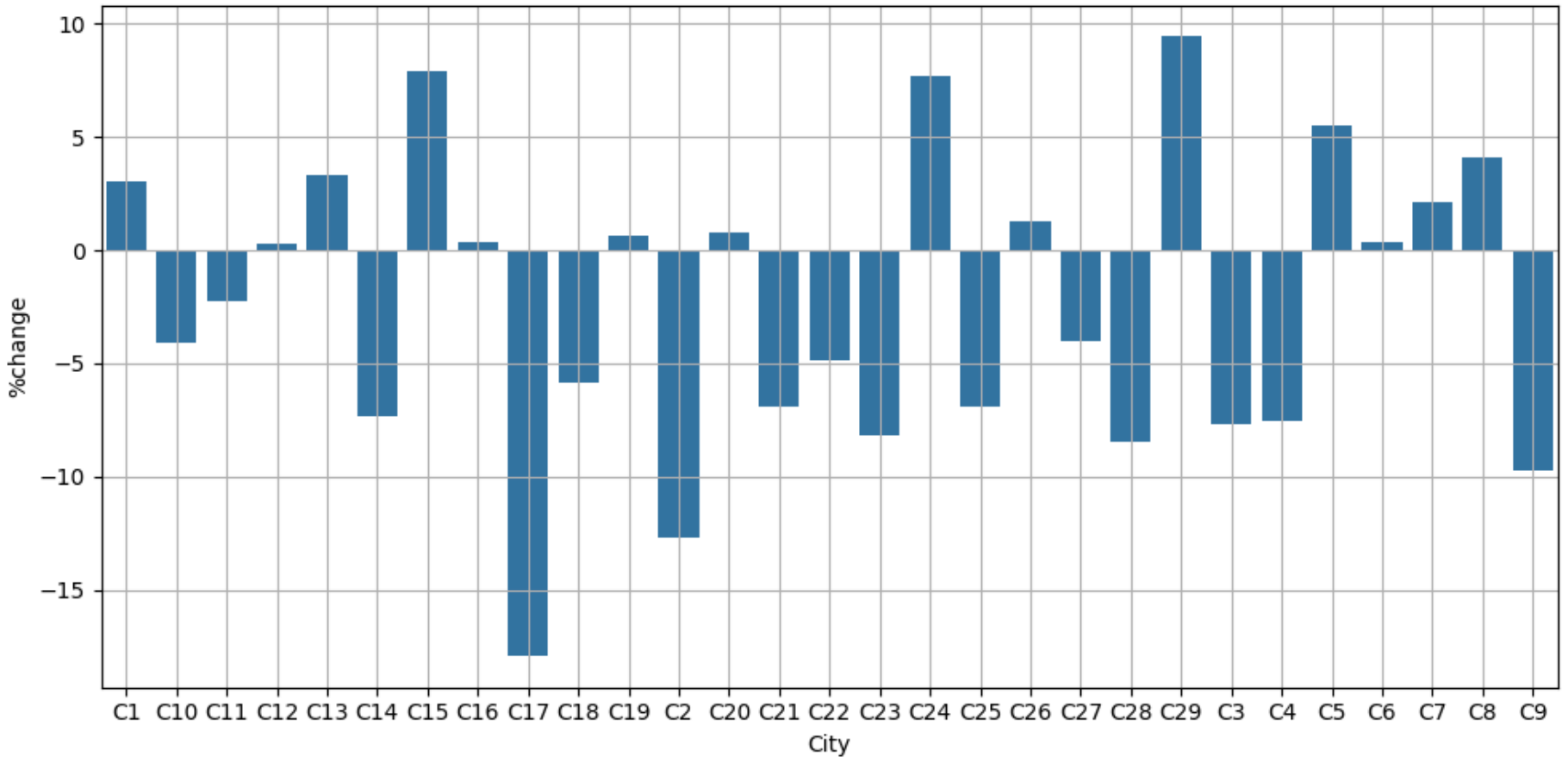
```
sns.histplot(data=driver_df, x='Quarterly Rating_Change', hue='Churn', stat='proportion', multiple='fill')
plt.show()
plt.figure(figsize=(9,2))
sns.histplot(data=driver_df, x='Quarterly Rating Improved', hue='Churn', stat='proportion', multiple='fill')
plt.show()
```



- 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

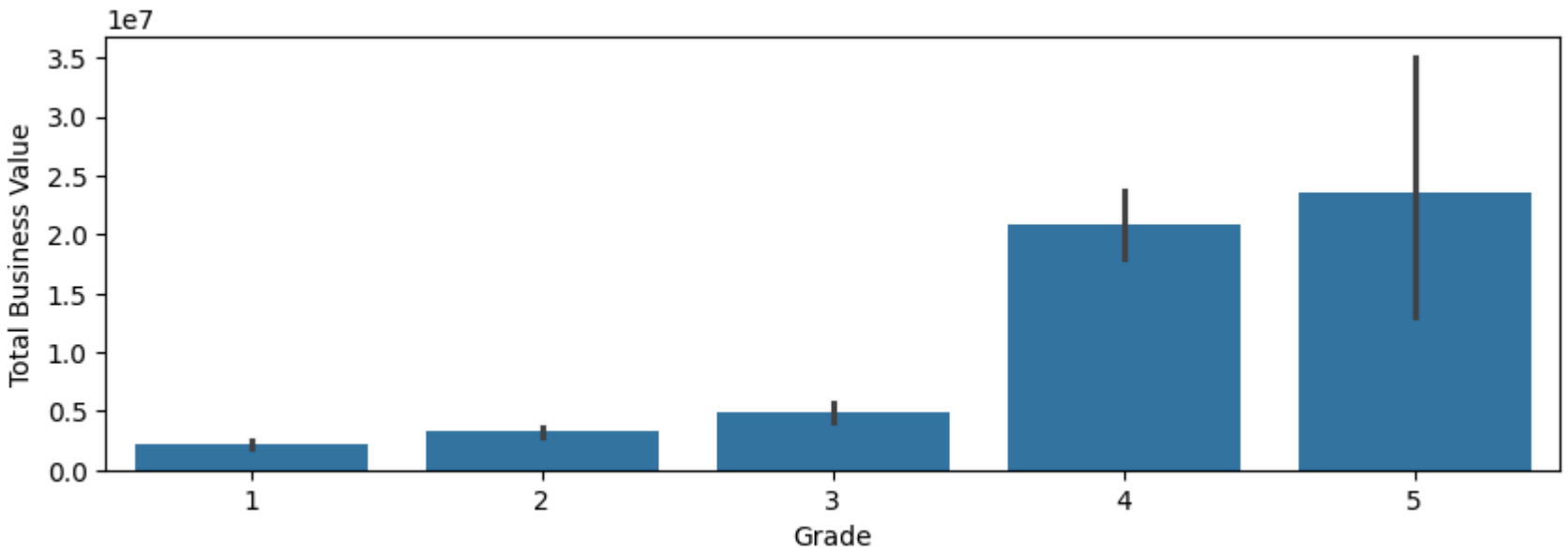
In [32]:
```
temp_df = df.groupby(['City', 'ReportingYear']).agg({'Quarterly Rating': 'mean'}).reset_index()
temp_df1 = pd.pivot_table(data=temp_df, index='City', columns='ReportingYear', values='Quarterly Rating').reset_index()
temp_df1.rename(columns={'ReportingYear':'index', 2019:'2019', 2020:'2020'}, 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()
```

```
/tmp/ipython-input-2090613480.py:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  temp_df = df.groupby(['City', 'ReportingYear']).agg({'Quarterly Rating': 'mean'}).reset_index()
/tmp/ipython-input-2090613480.py:2: FutureWarning: The default value of observed=False is deprecated and will change to observed=True in a future version of pandas. Specify observed=False to silence this warning and retain the current behavior
  temp_df1 = pd.pivot_table(data=temp_df, index='City', columns='ReportingYear', values='Quarterly Rating').reset_index()
```



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

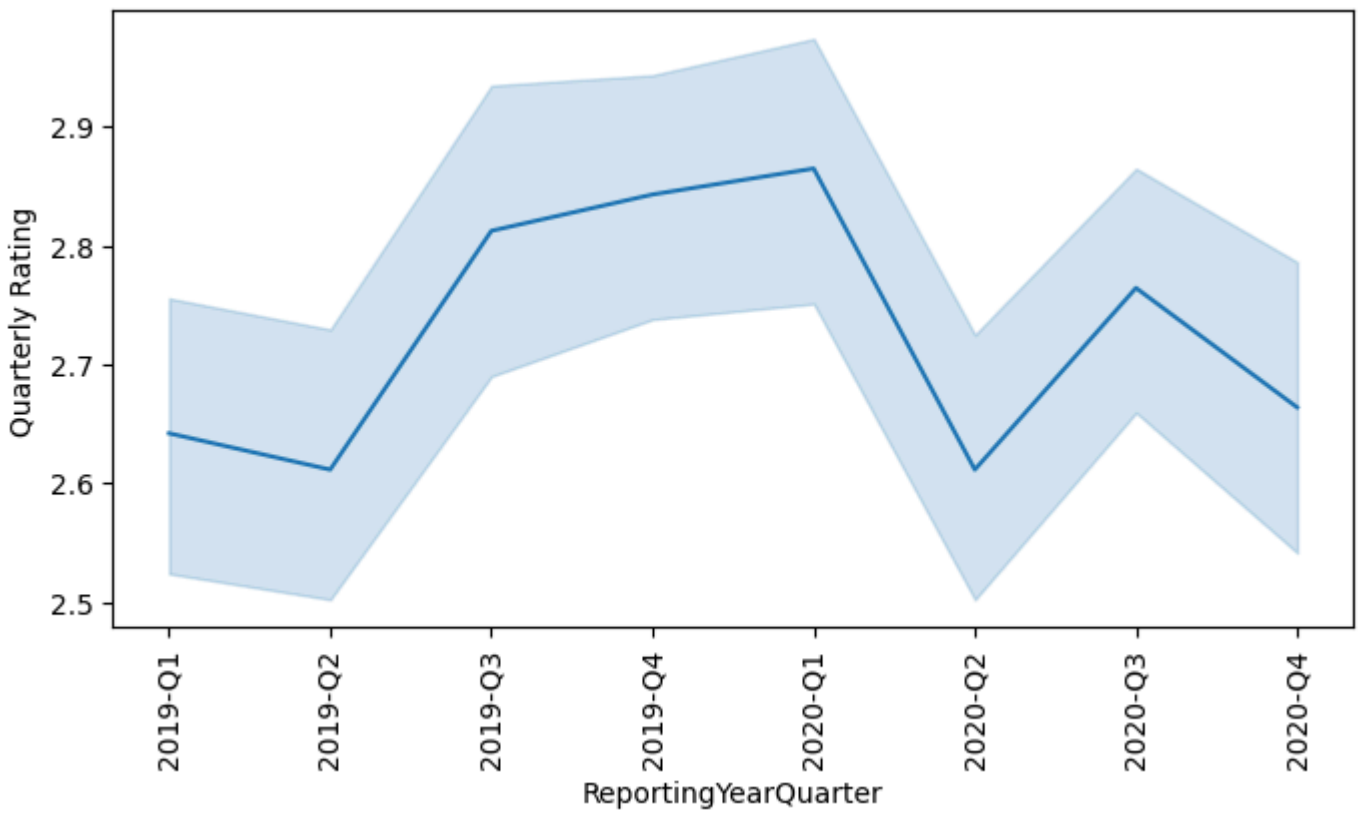
In [33]:
```
plt.figure(figsize=(10,3))
sns.barplot(data=driver_df, x='Grade', y='Total Business Value', estimator='mean')
plt.show()
print('Mean of Total Business Value of drivers with grade 5:', driver_df[driver_df['Grade'] == 5]['Total Business Value'].sum(
```




```
Mean of Total Business Value of drivers with grade 5: 565760460
```

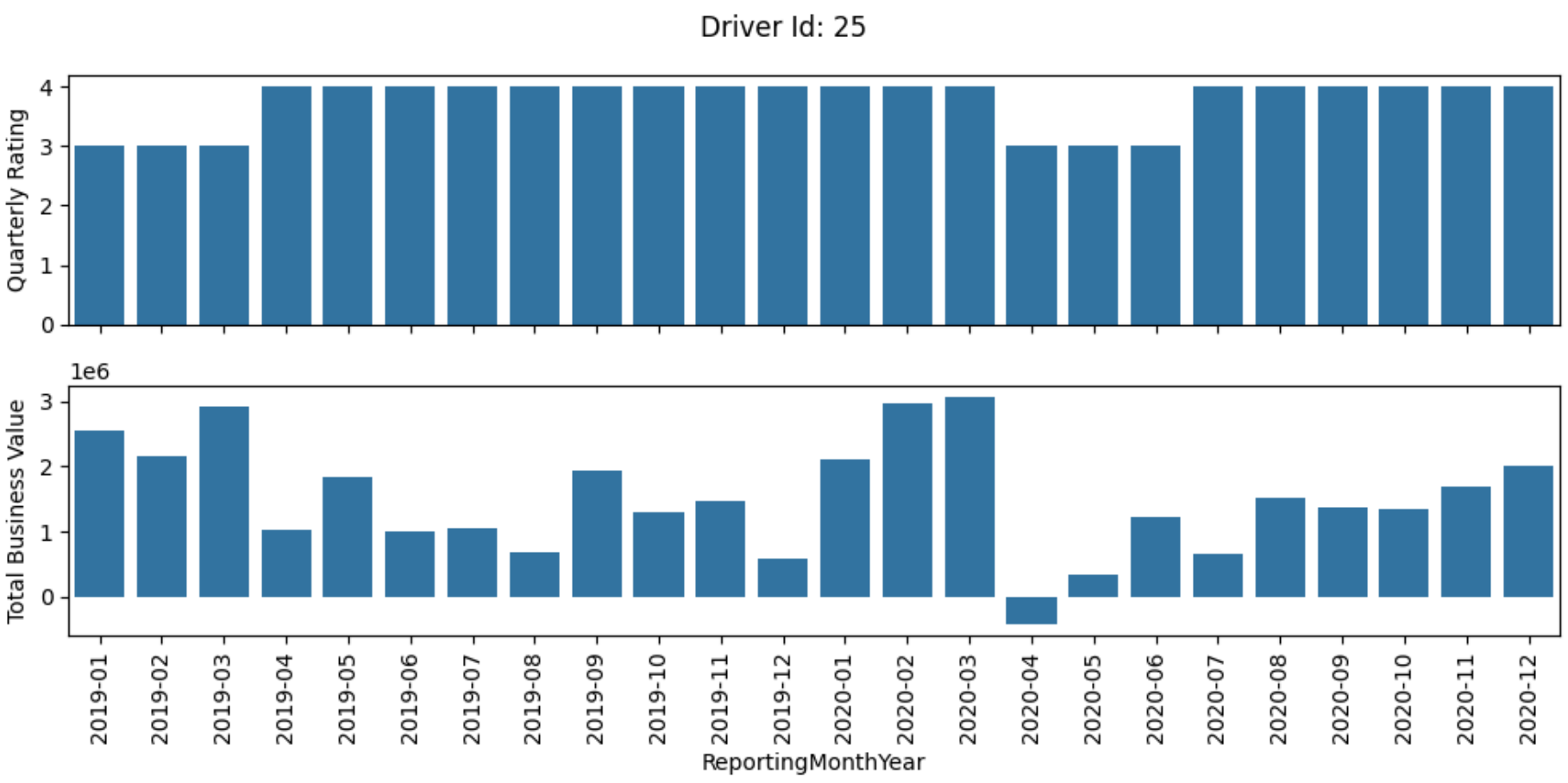
- The mean of Total Business Value of drivers with grade 5 is higher than those with other grades

In [34]:
```
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', 'Reporting
plt.figure(figsize=(8,4))
sns.lineplot(data=temp_driver_full_service_df, x='ReportingYearQuarter', y='Quarterly Rating')
plt.xticks(rotation=90)
plt.show()
```

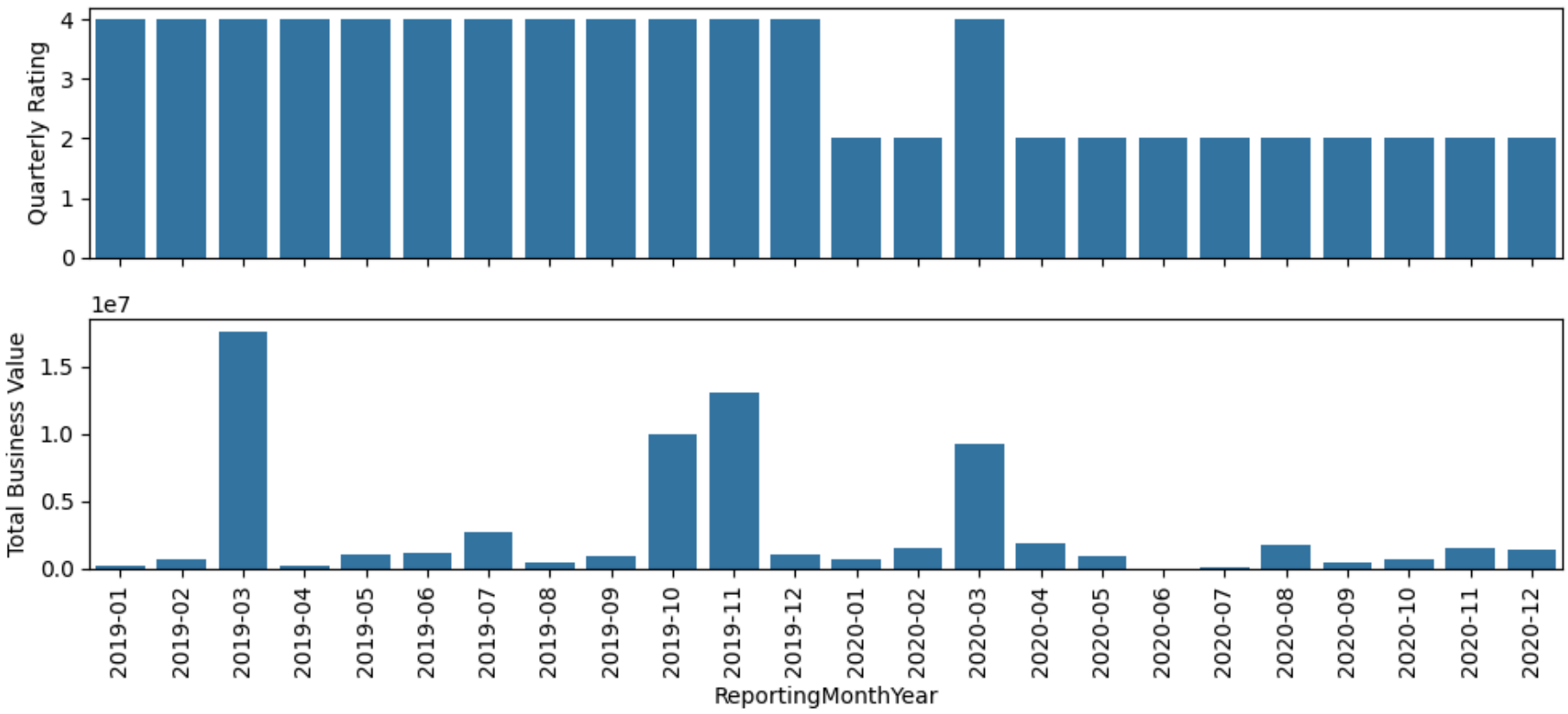


- There is a dip in the quarterly rating in Q2 and then it increases in Q3.
- This pattern can be osberved for both the years

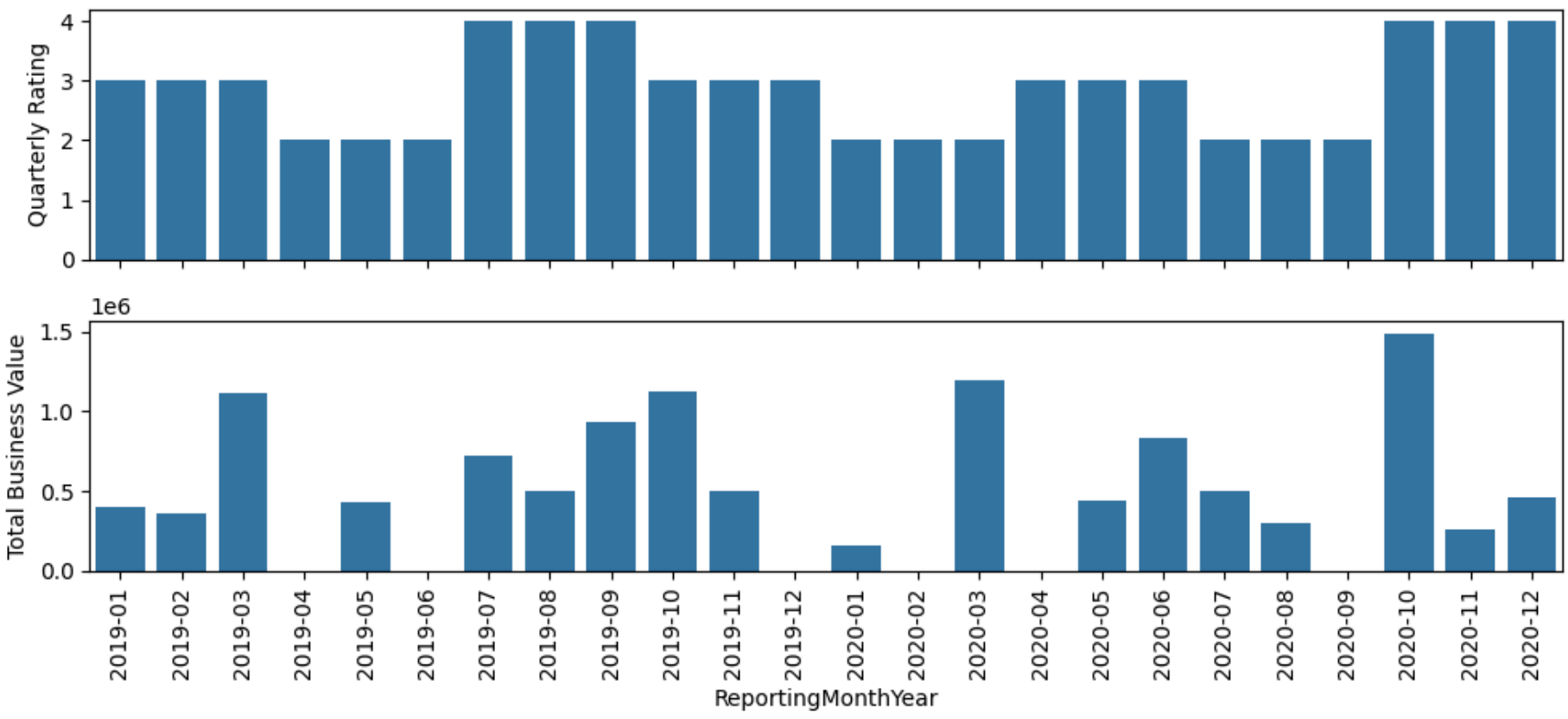
In [35]:
```
temp_driver_full_service_df = temp_df[temp_df['Driver_ID'].isin(drivers_with_2_year_service)]
num_of_drivers = 20
count=0
for driver_id in temp_driver_full_service_df['Driver_ID'].unique():
    if(count < num_of_drivers):
        count = count + 1
        sample_df = temp_driver_full_service_df[temp_driver_full_service_df['Driver_ID'] == 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
```



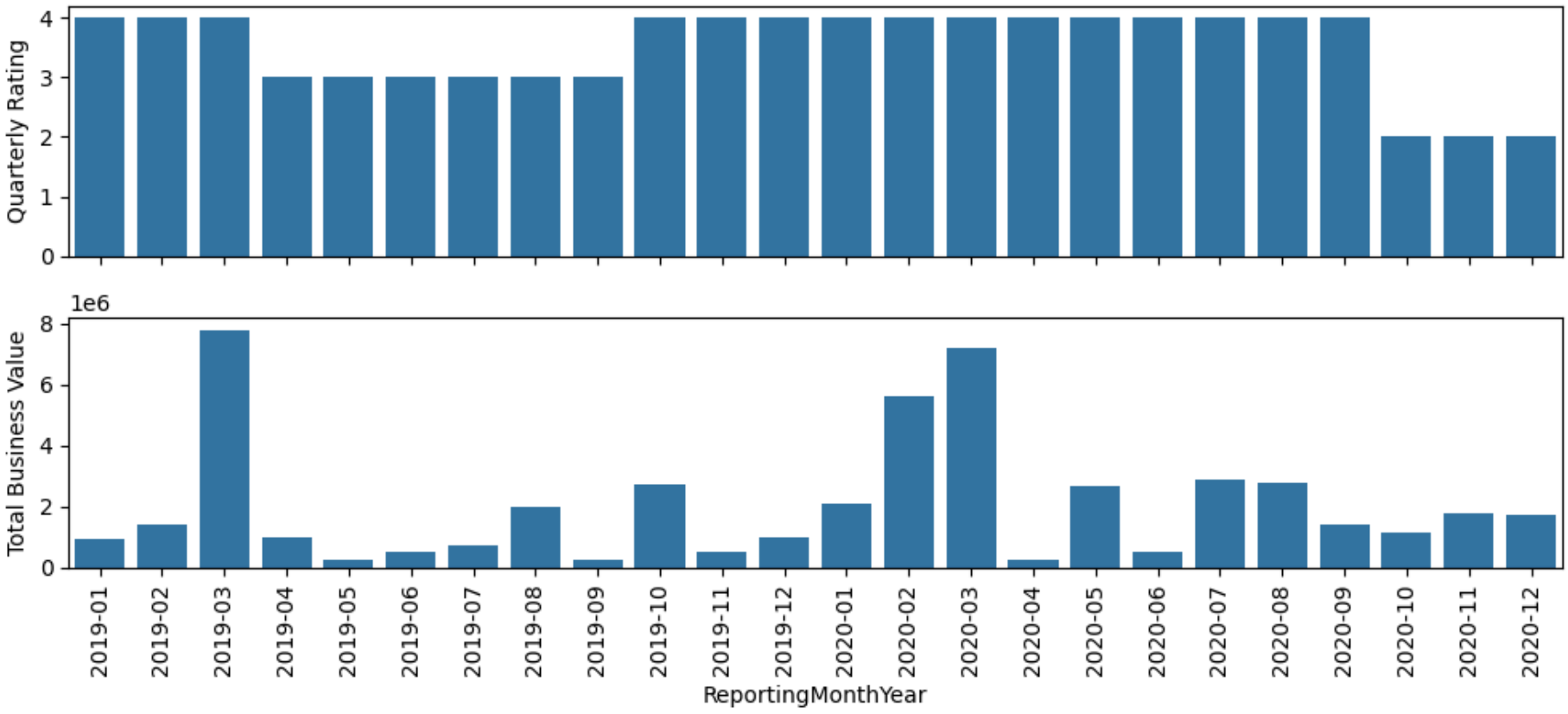
Driver Id: 26



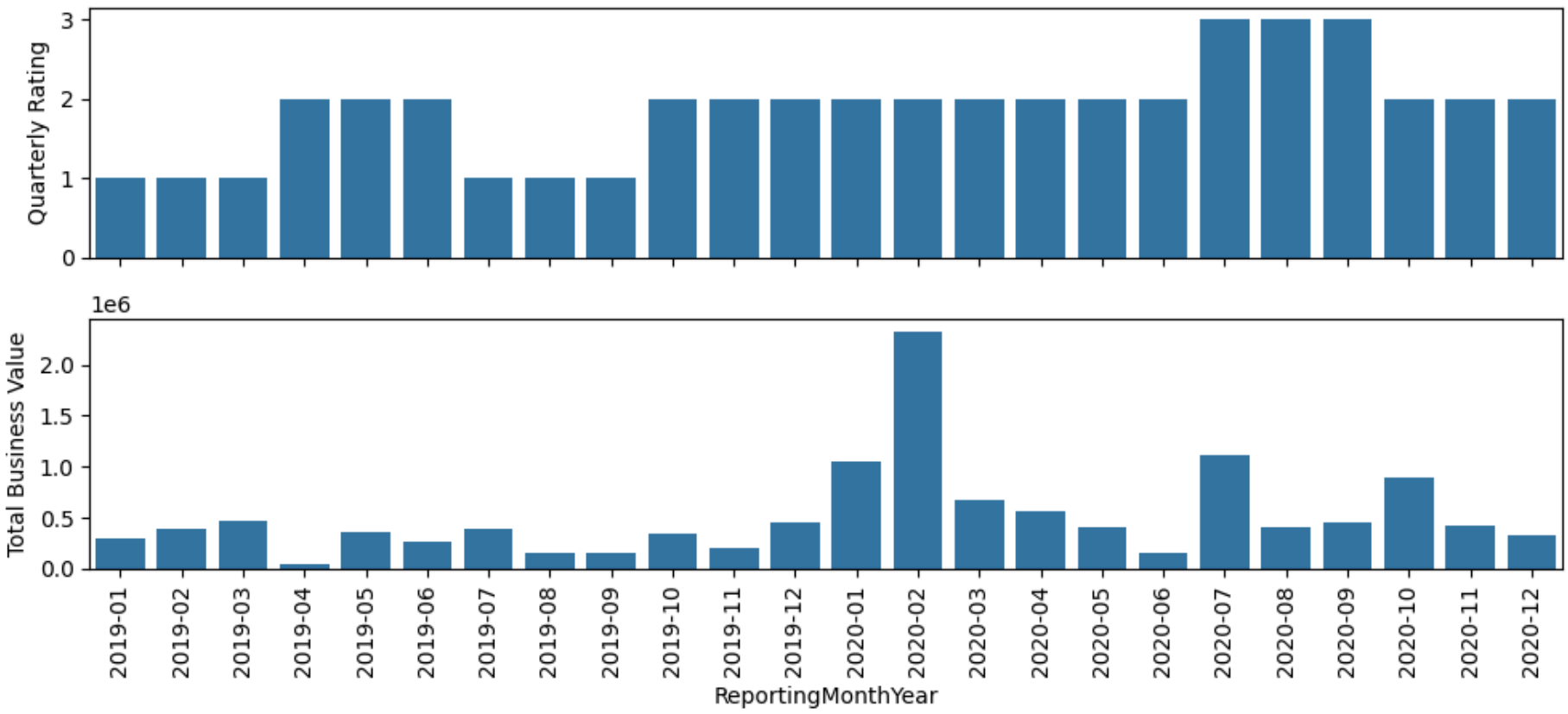
Driver Id: 56



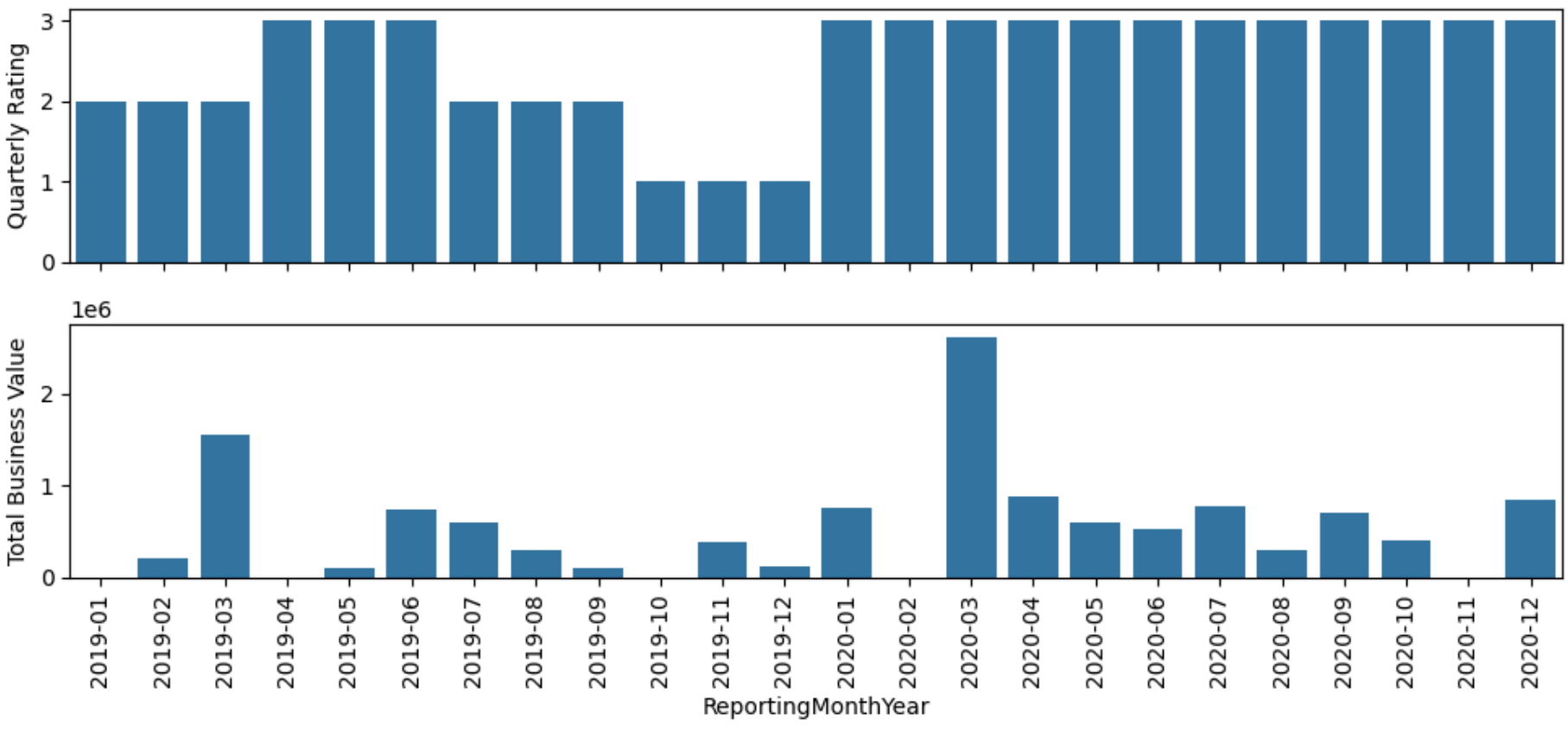
Driver Id: 60



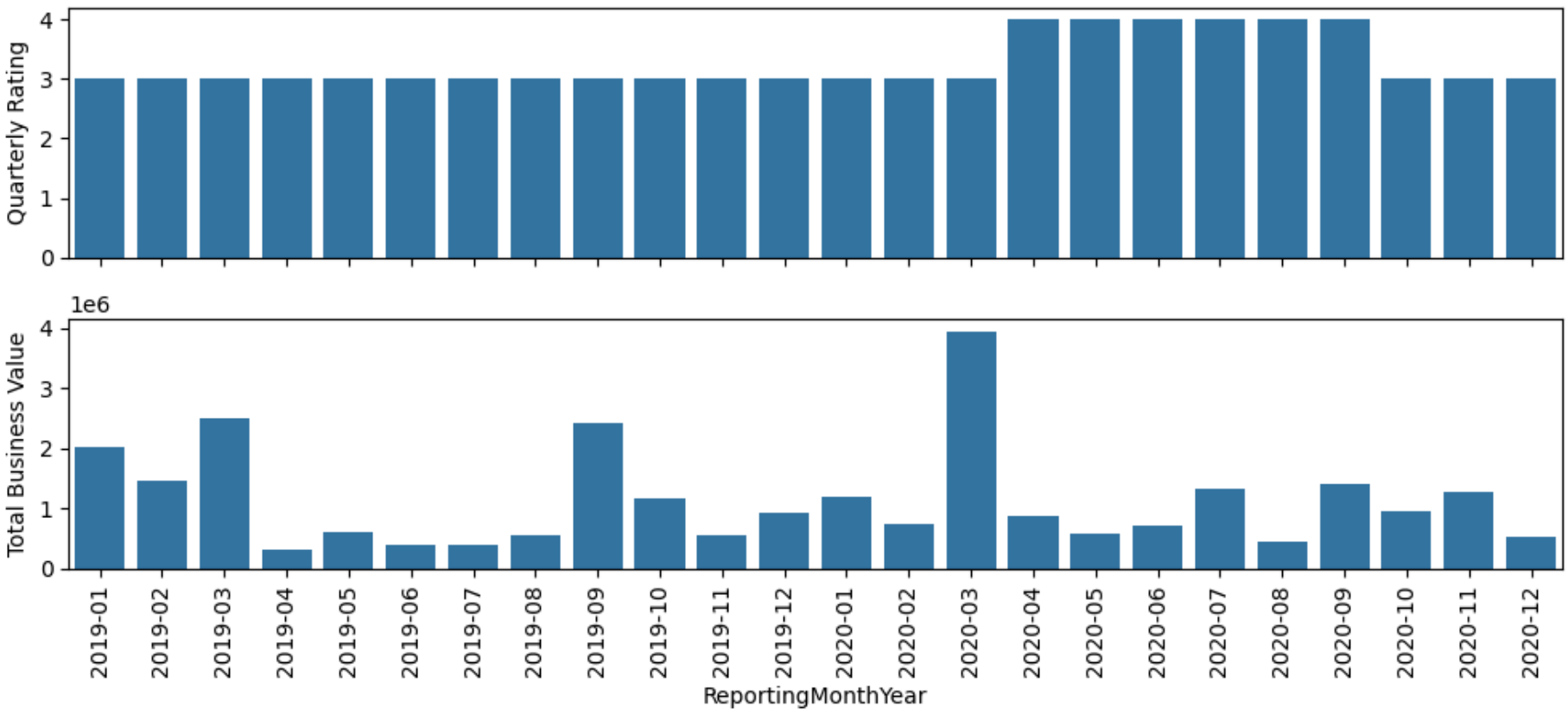
Driver Id: 63



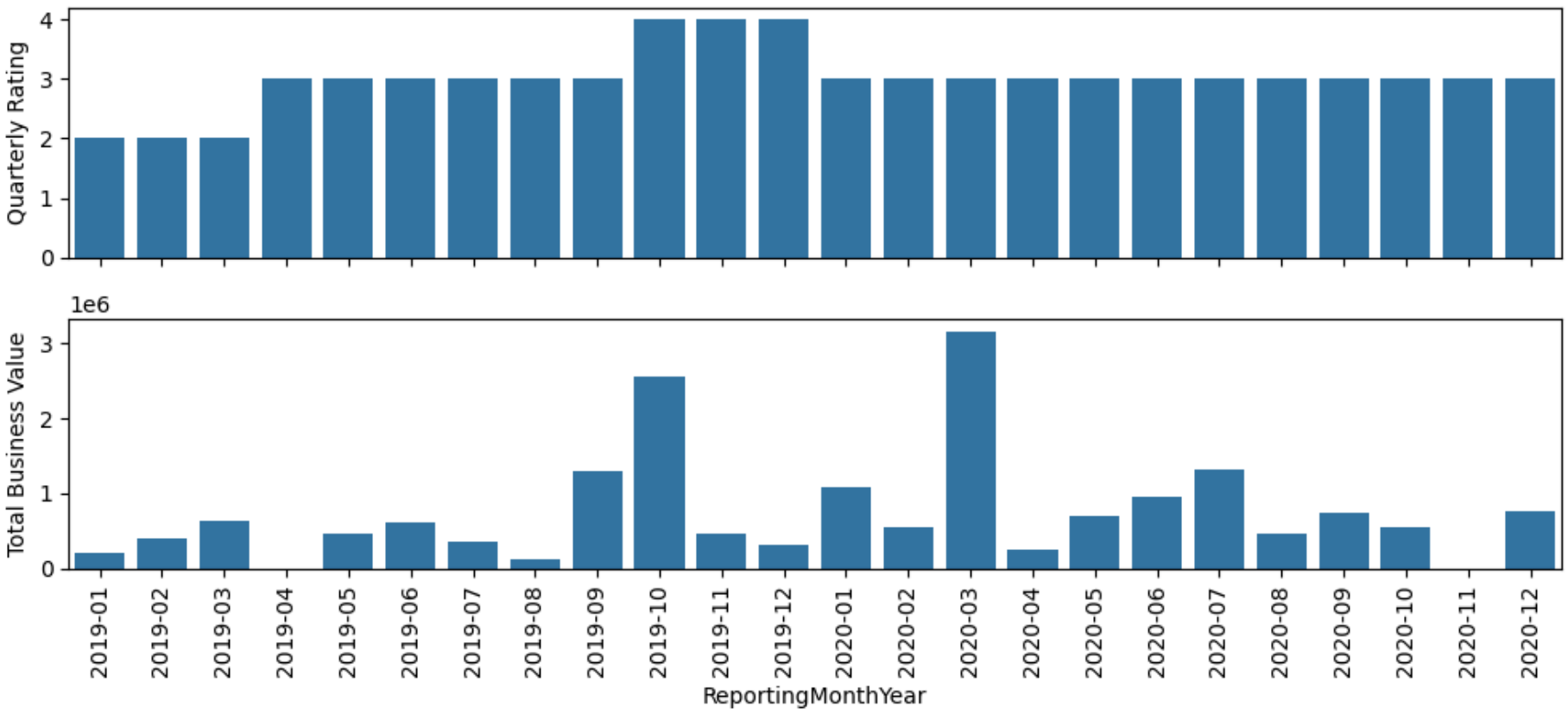
Driver Id: 67



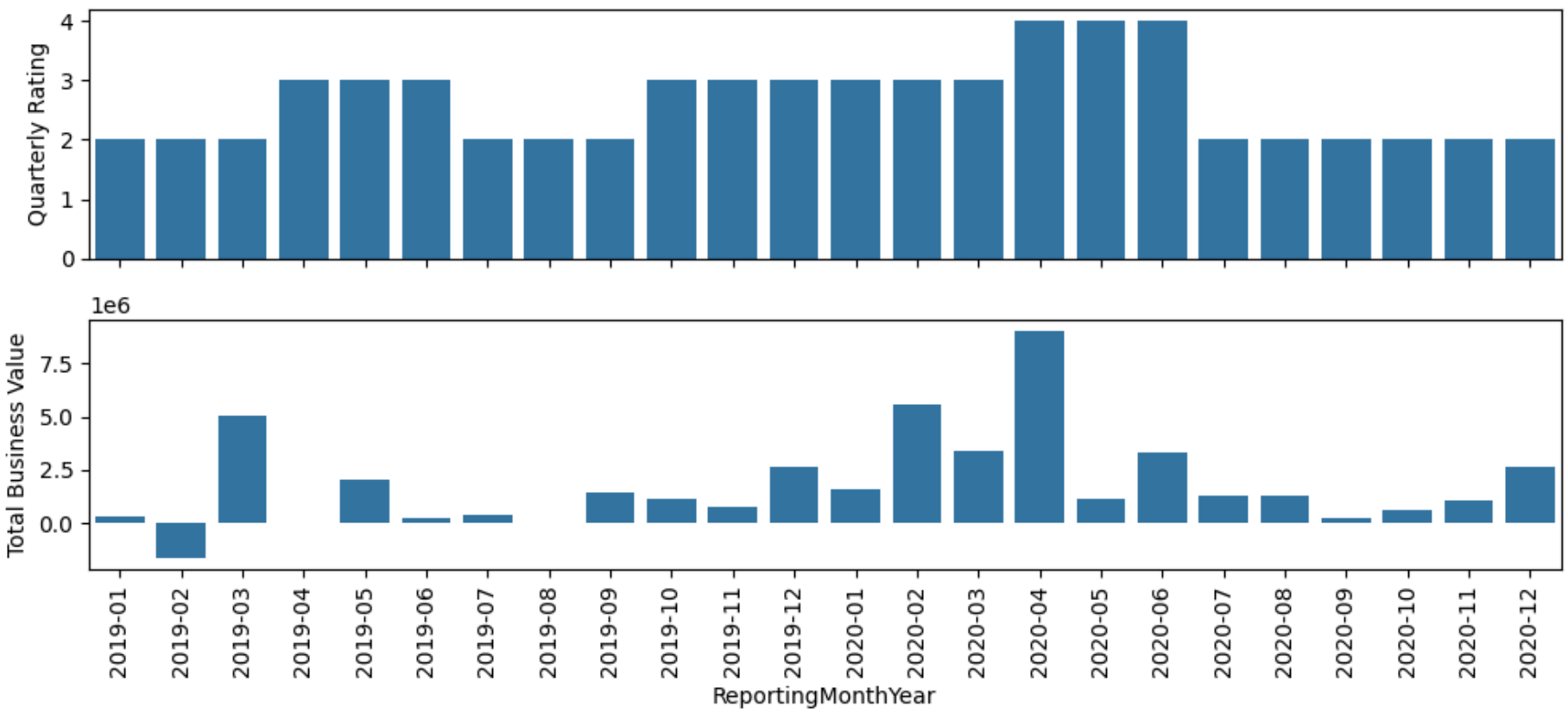
Driver Id: 68



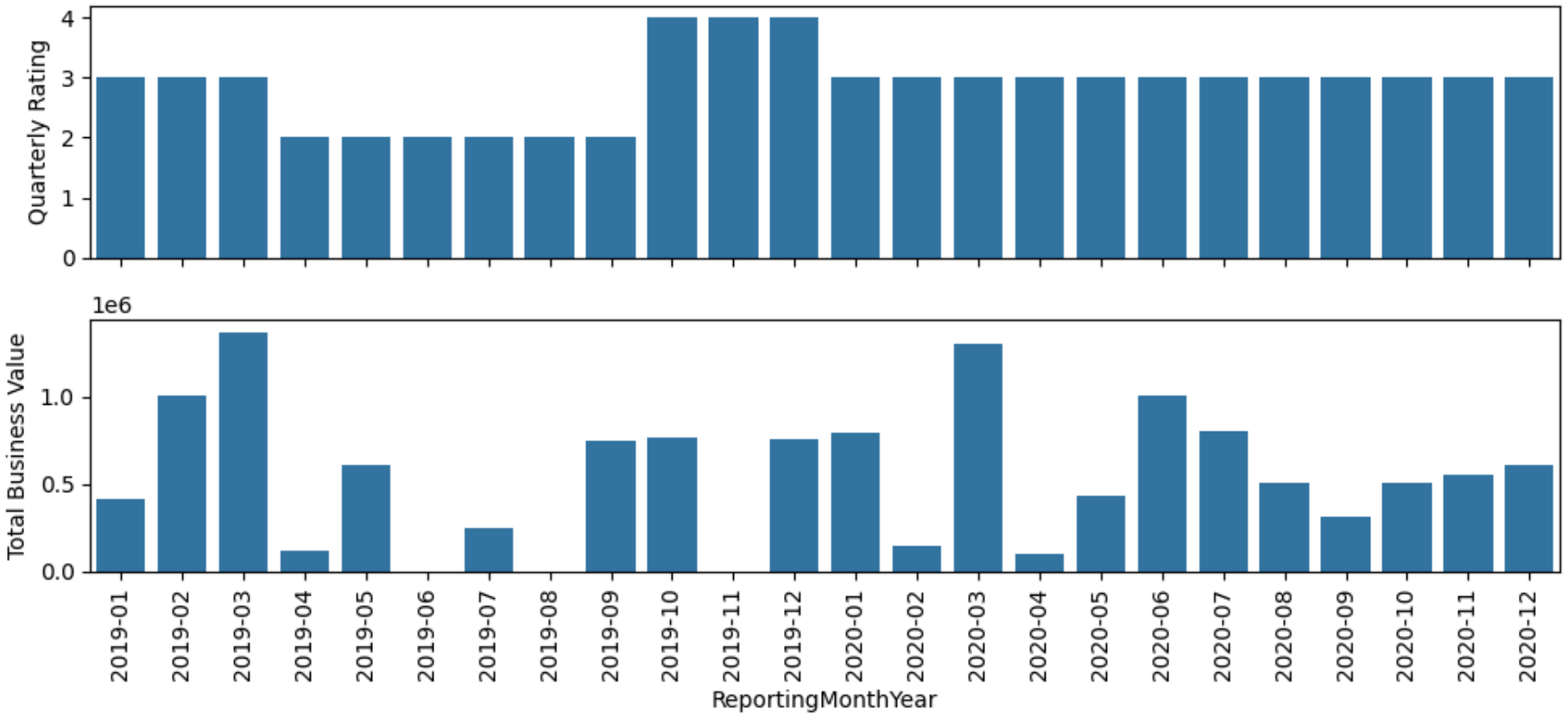
Driver Id: 77



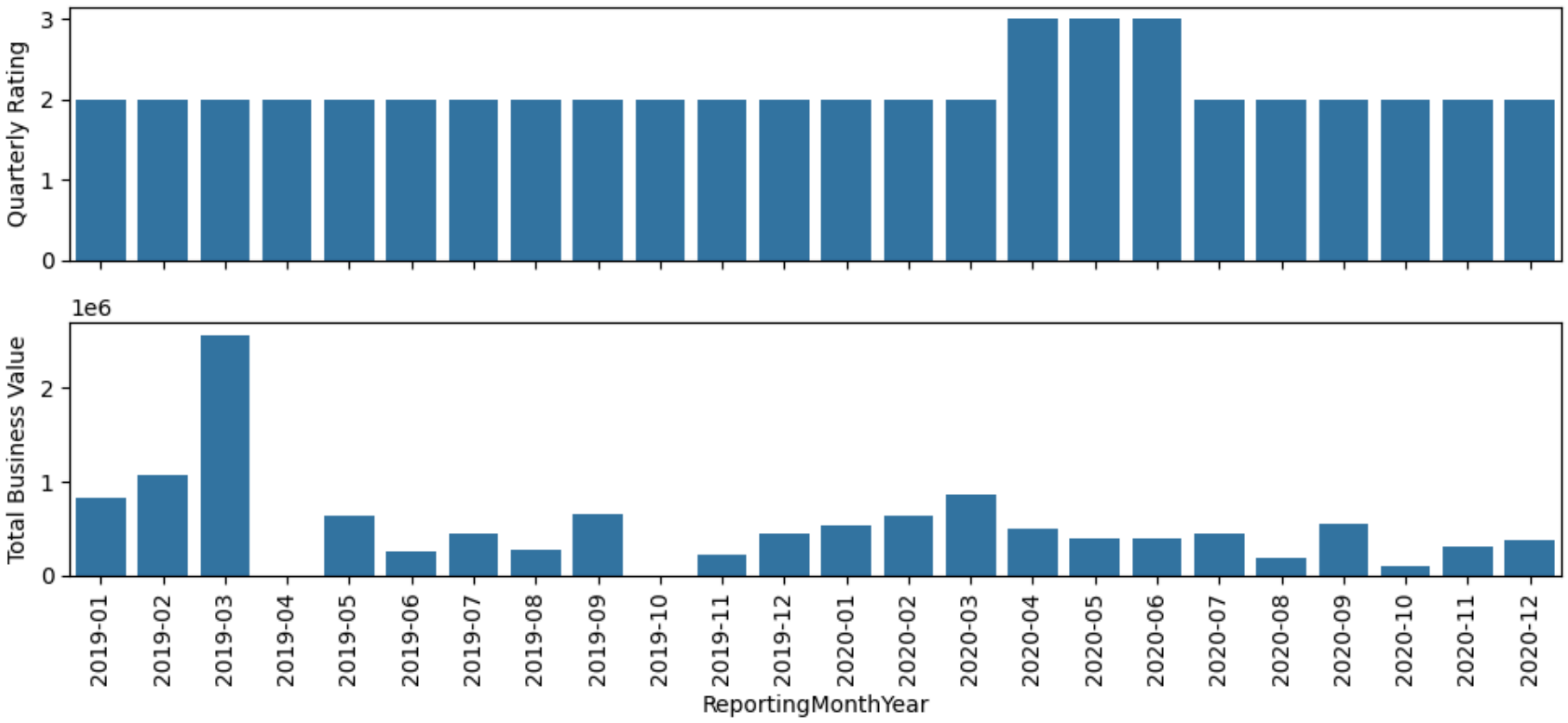
Driver Id: 78



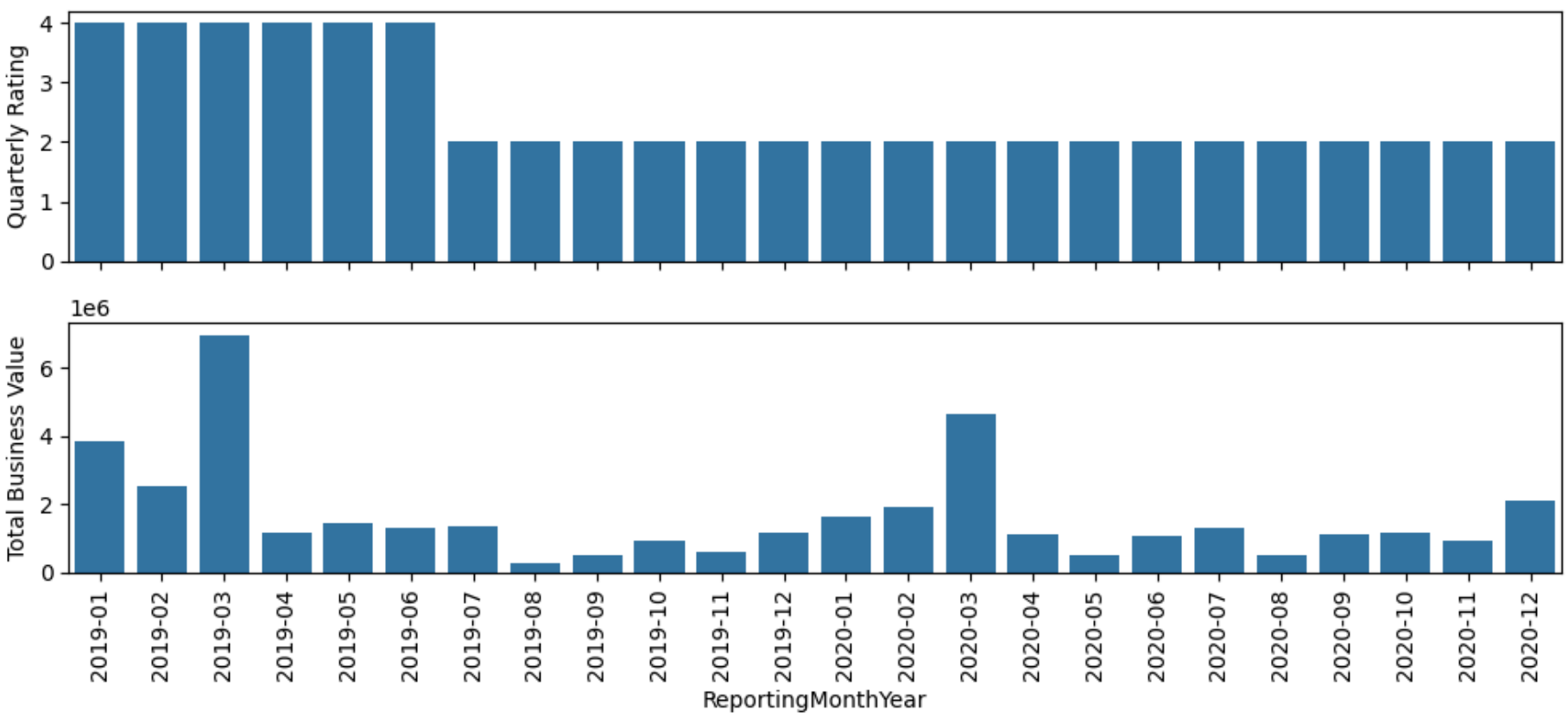
Driver Id: 112



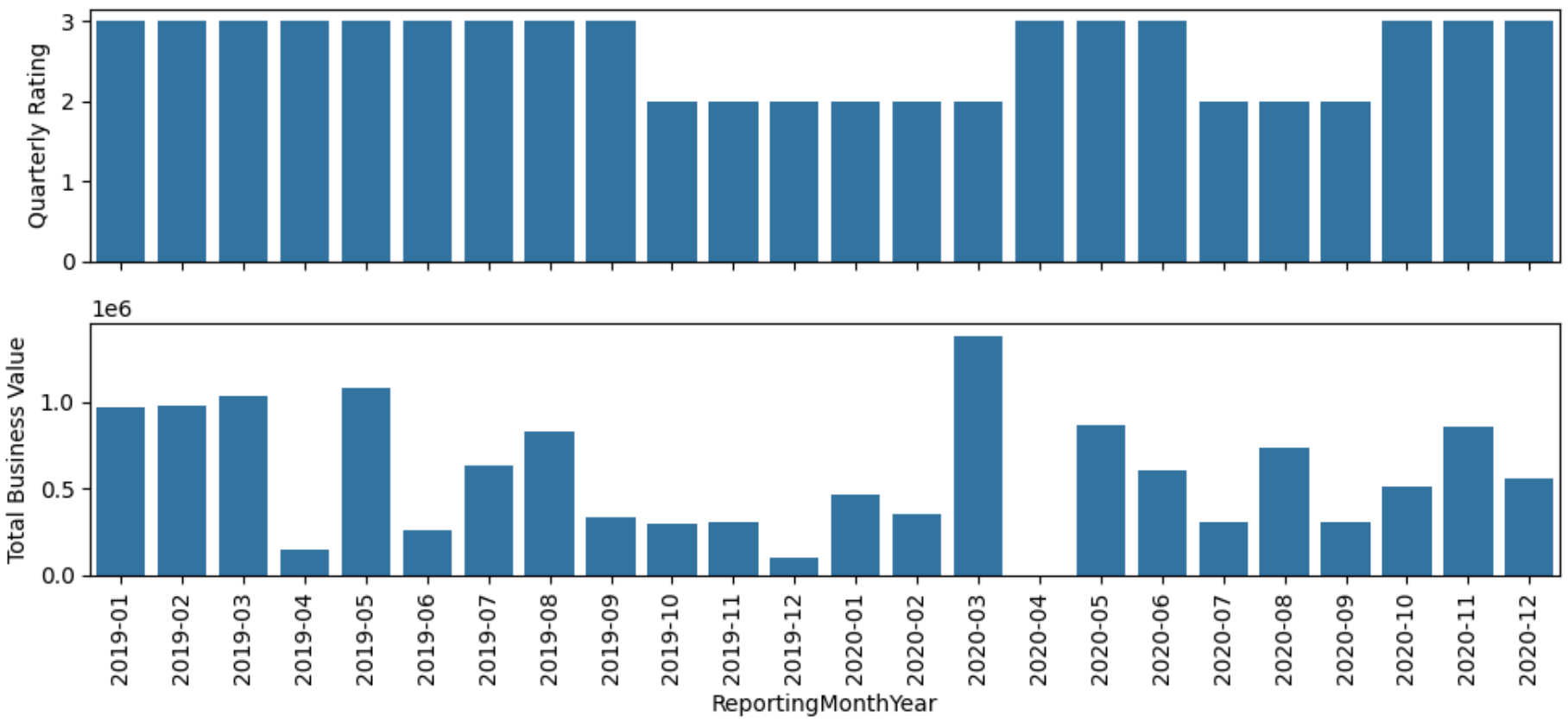
Driver Id: 115



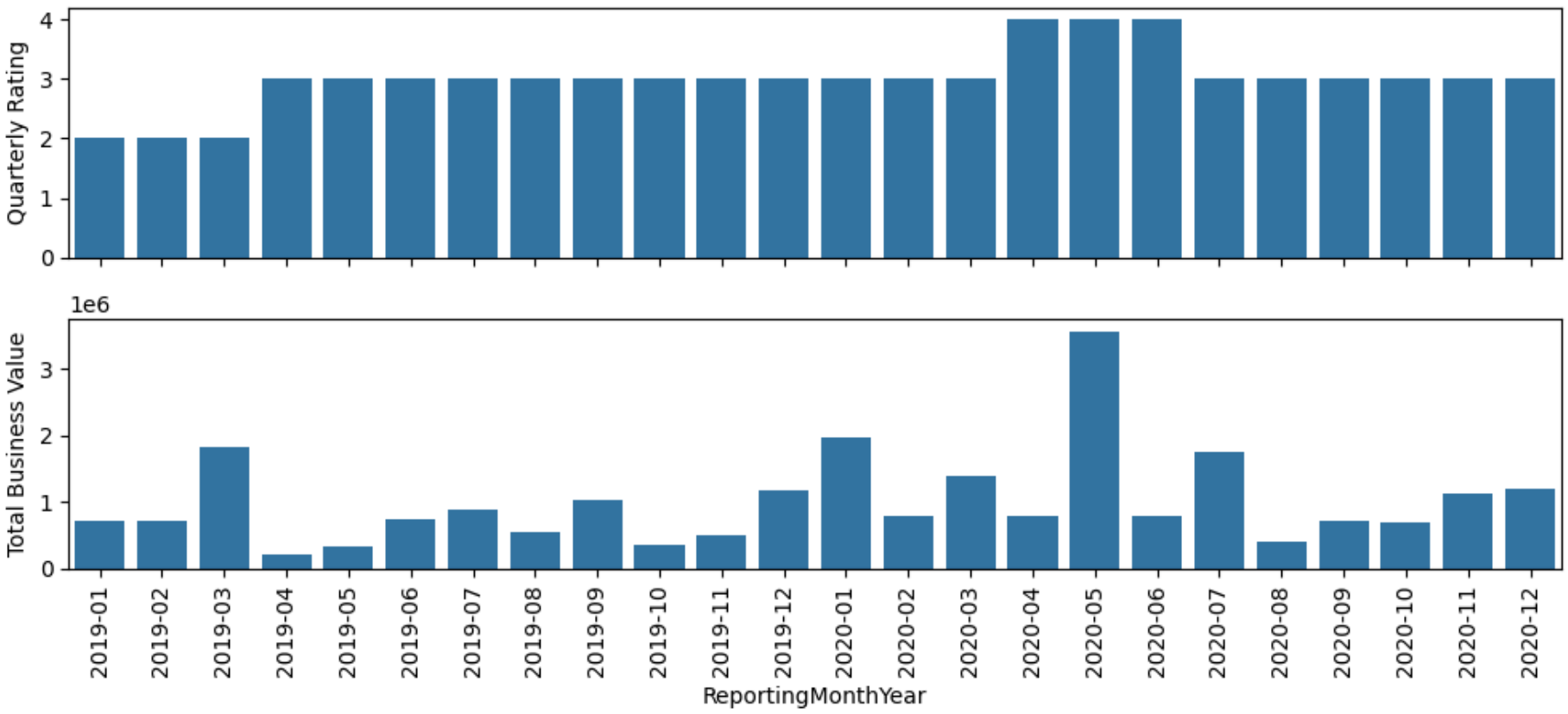
Driver Id: 117



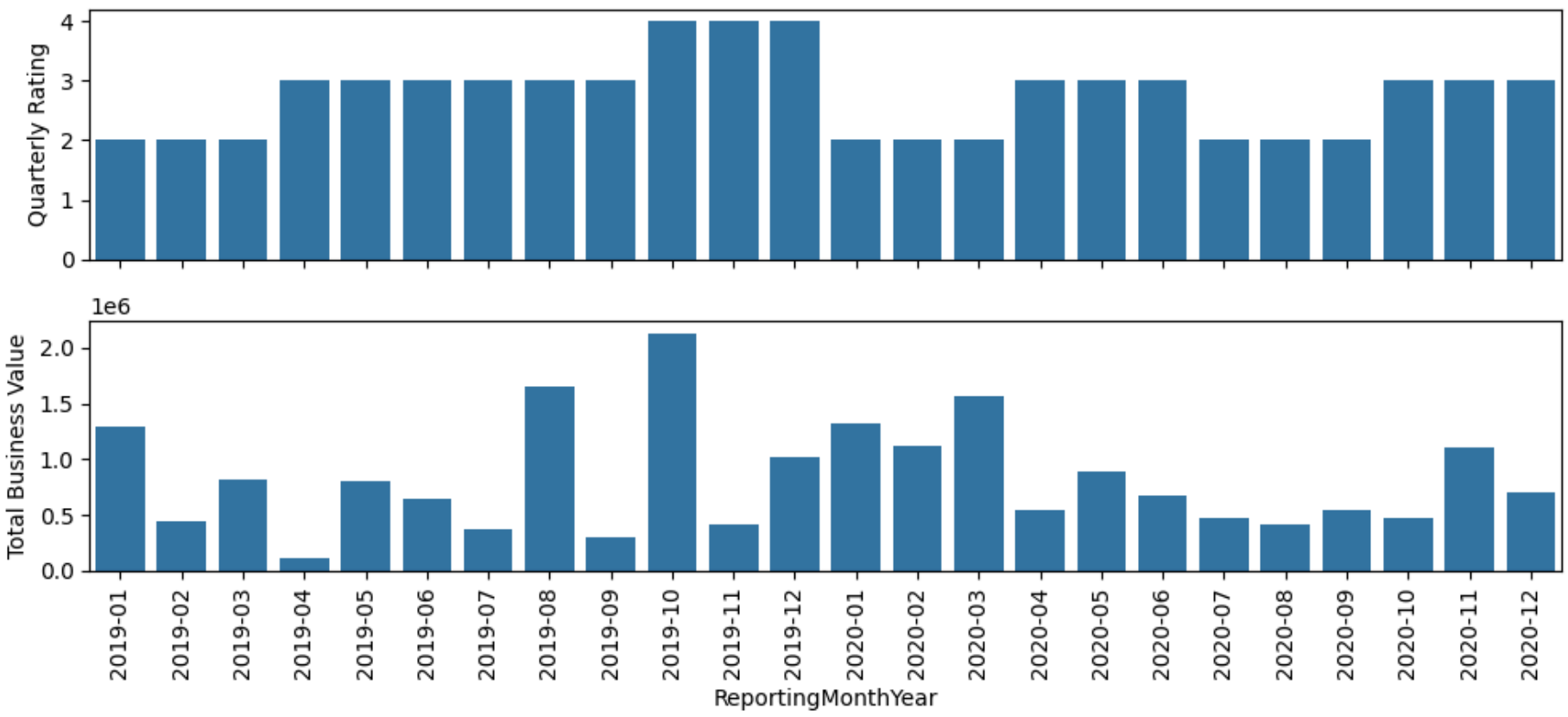
Driver Id: 140



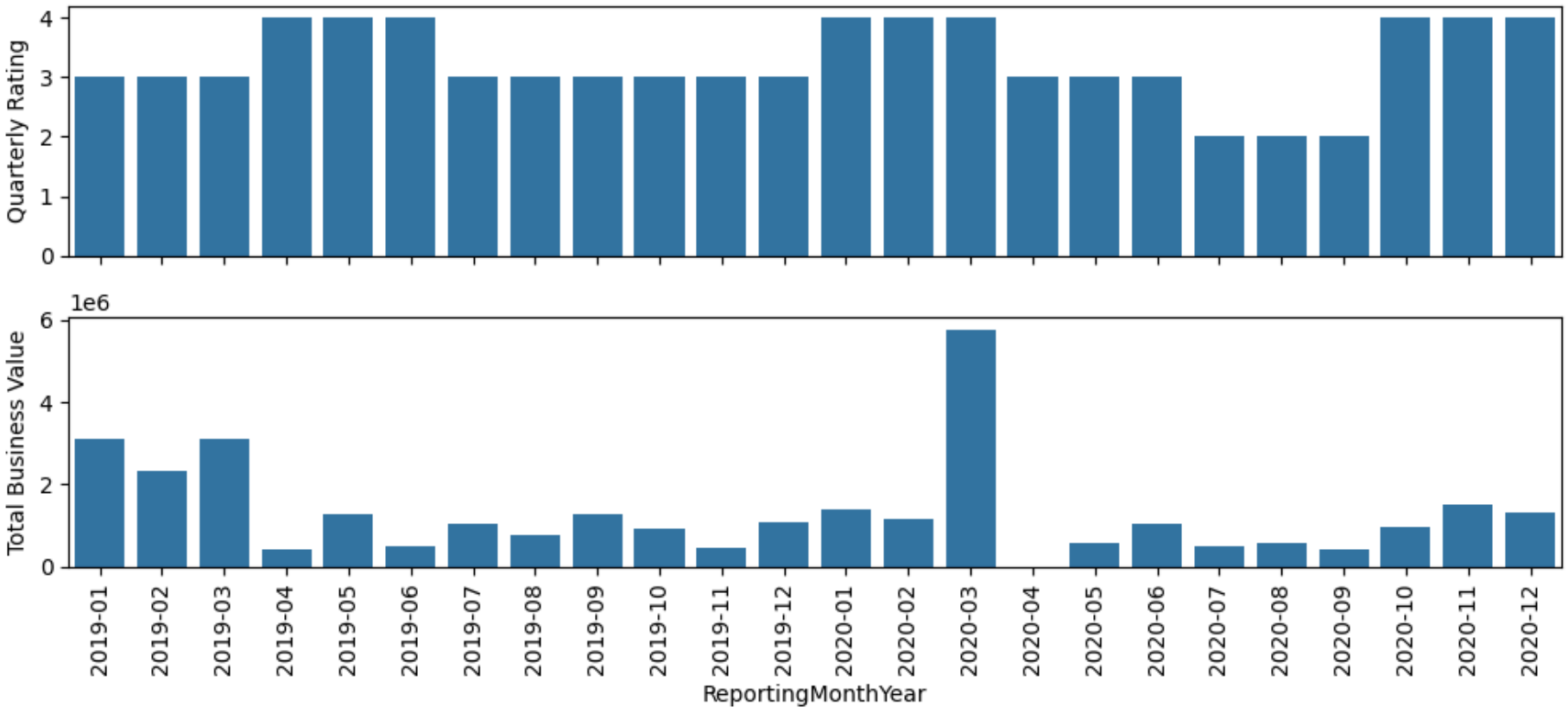
Driver Id: 150



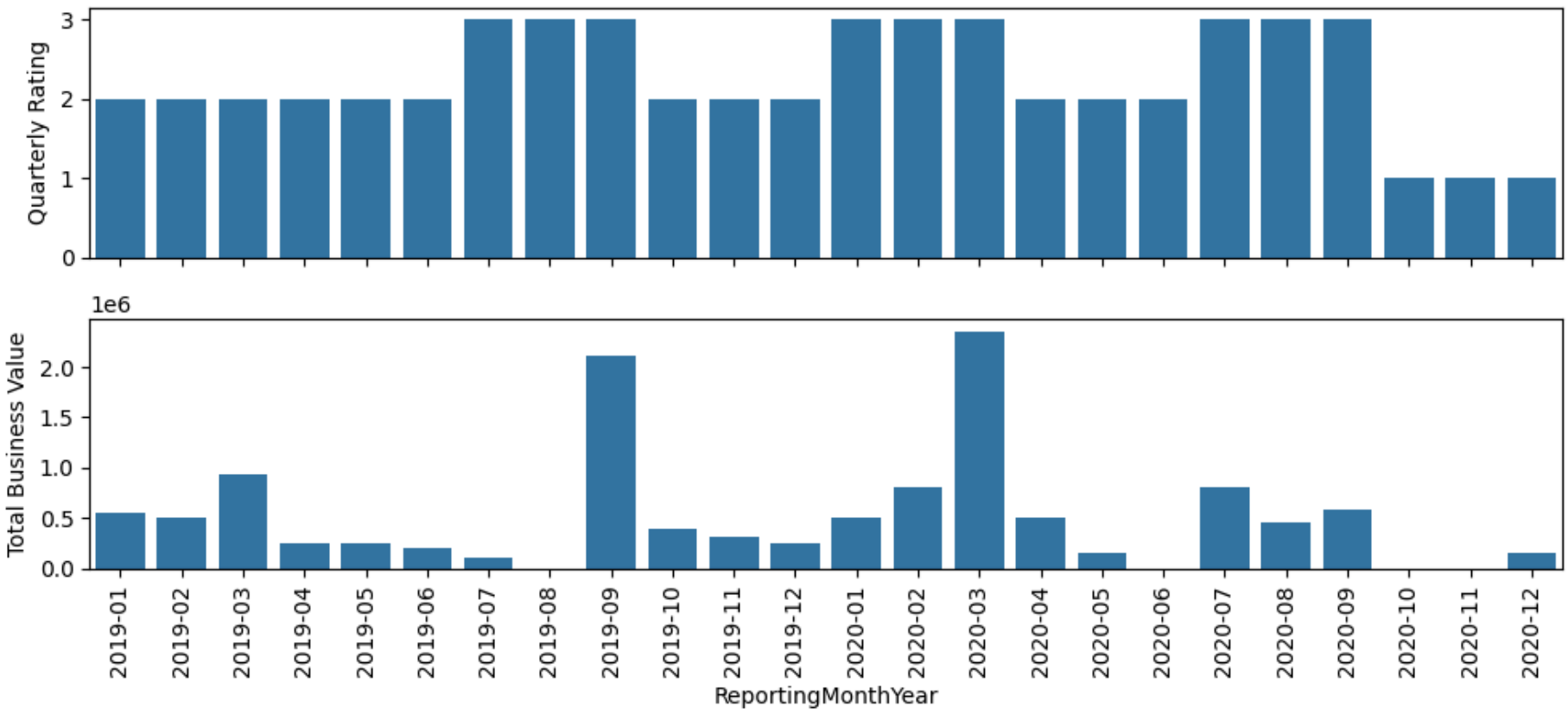
Driver Id: 173



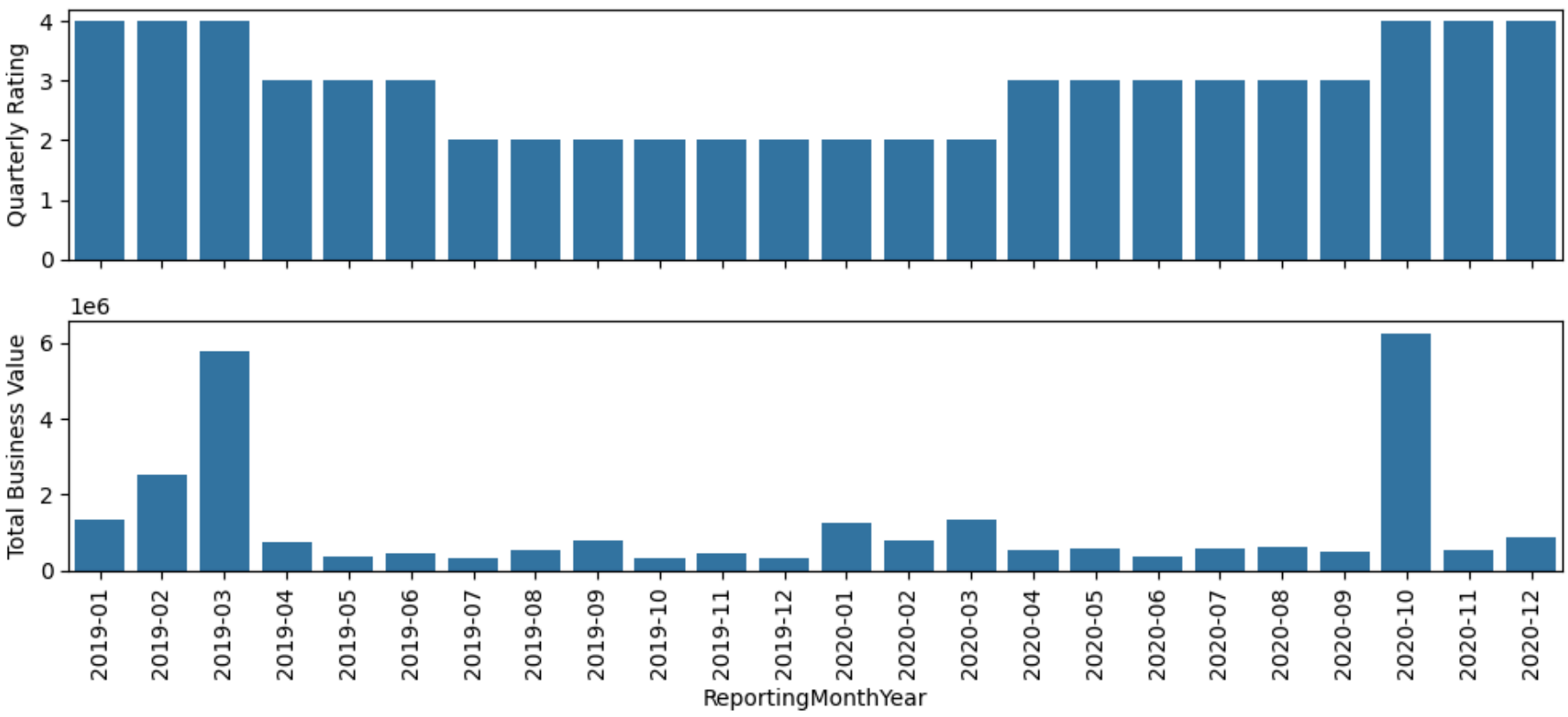
Driver Id: 199



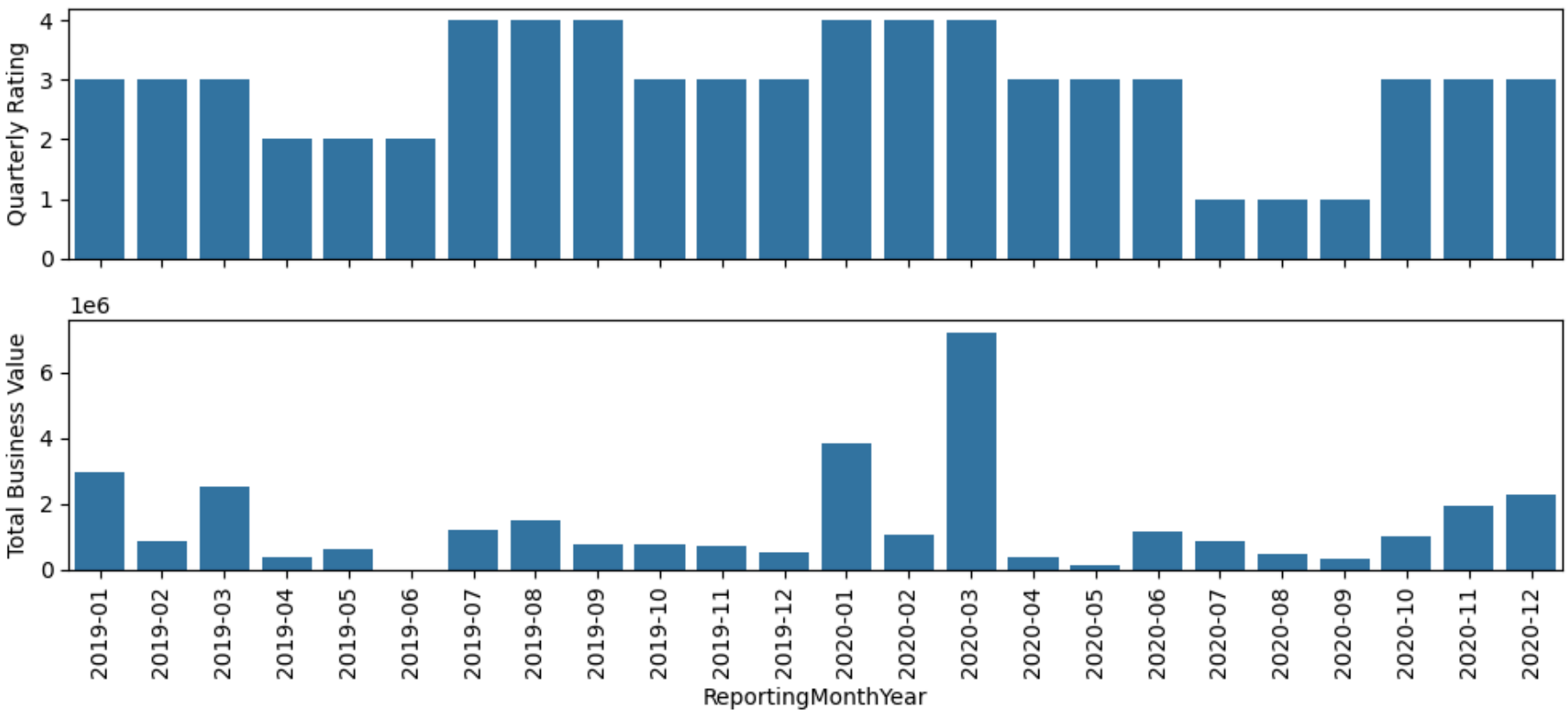
Driver Id: 213

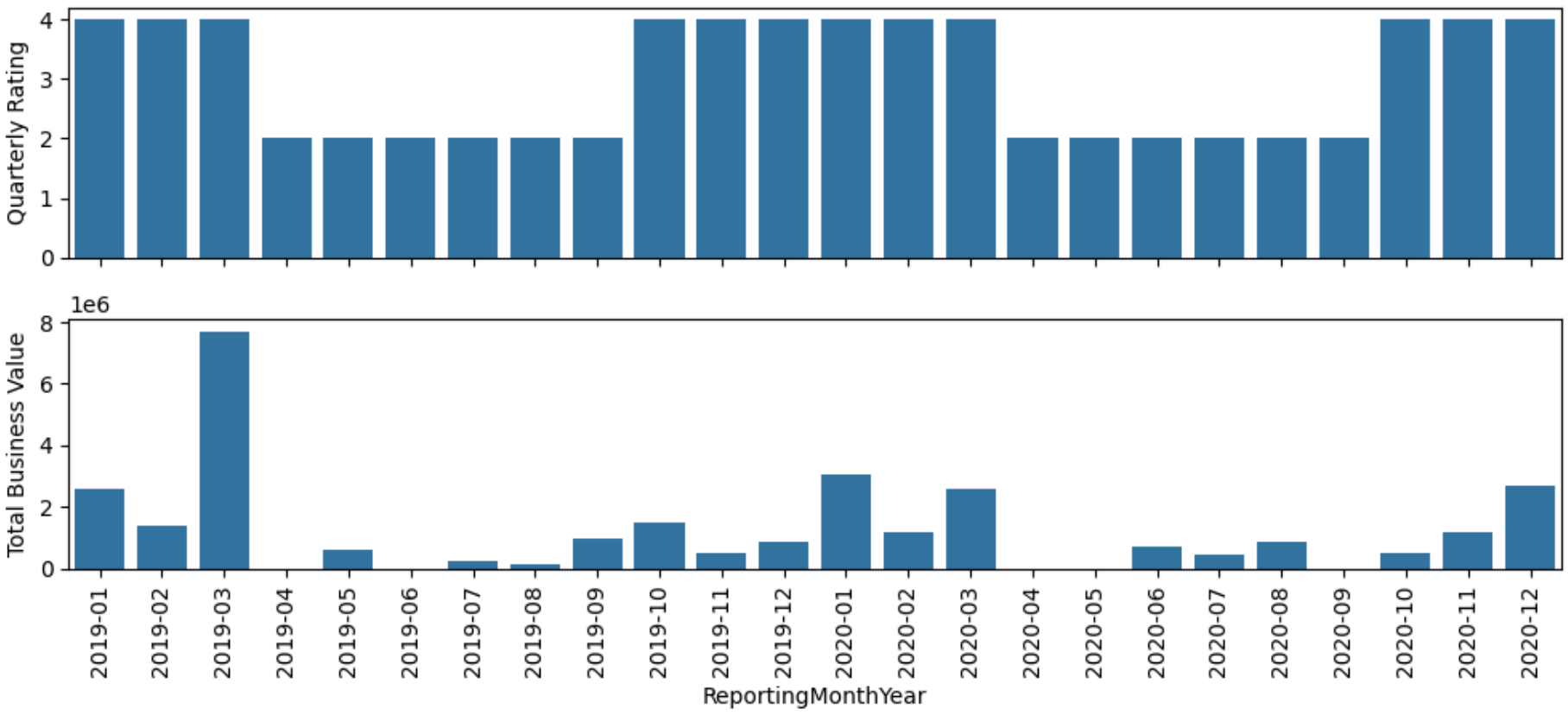


Driver Id: 252



Driver Id: 275





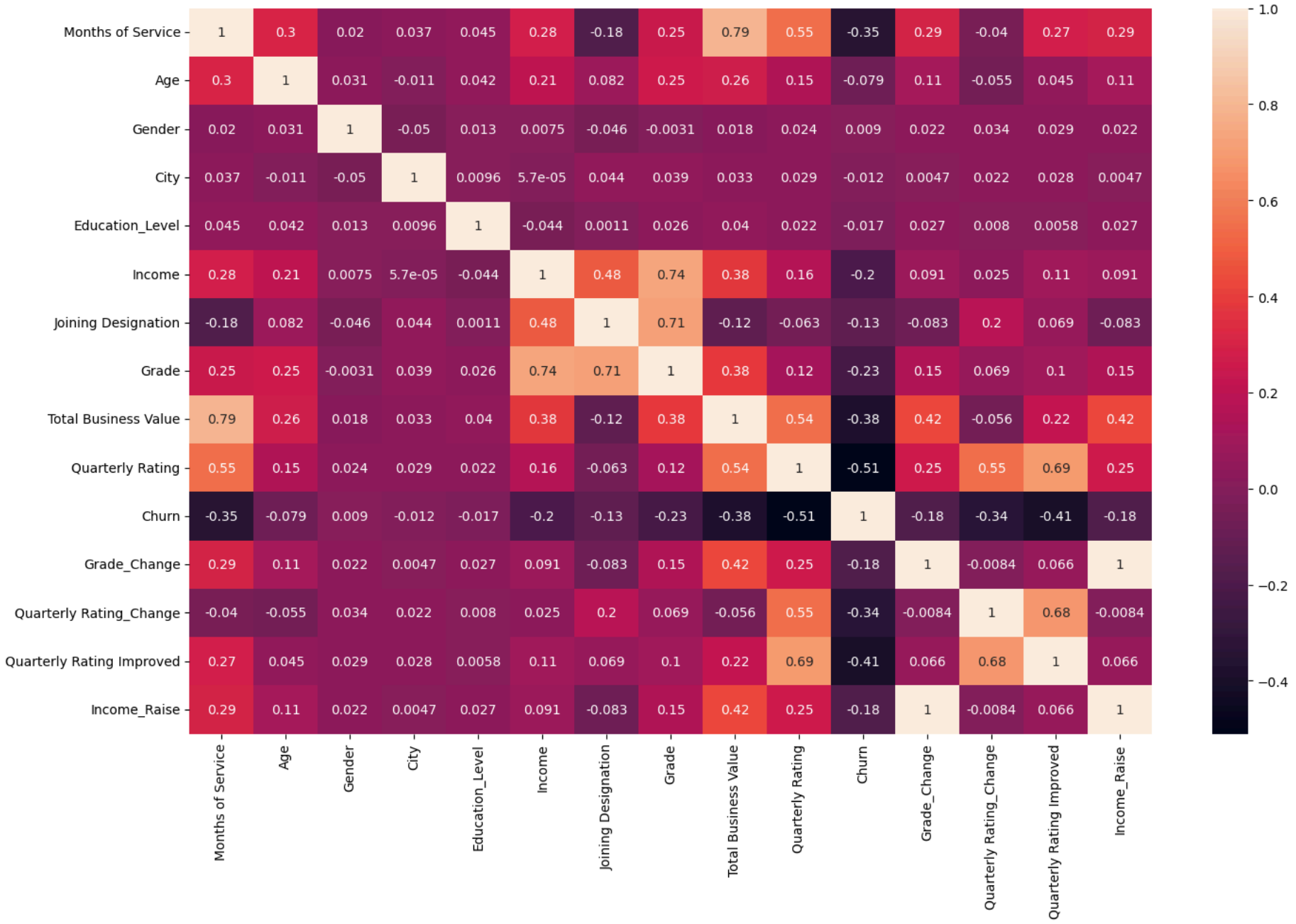
- It can be observed that a significant drop in rating impacts the Total Business Value. Drop in rating demotivates the drivers, leading to accepting only a few rides or in somecases not accepting any rides and hence impacting the Total Business Value

3-Multivariate analysis

In [36]:
```
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:]
```

```
/tmp/ipython-input-2453968839.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chain
ed assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  driver_df['Gender'].replace({'Male':0, 'Female':1}, inplace=True)
/tmp/ipython-input-2453968839.py:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a fut
ure version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior,
set `pd.set_option('future.no_silent_downcasting', True)`
  driver_df['Gender'].replace({'Male':0, 'Female':1}, inplace=True)
/tmp/ipython-input-2453968839.py:1: FutureWarning: The behavior of Series.replace (and DataFrame.replace) with CategoricalDtype
is deprecated. In a future version, replace will only be used for cases that preserve the categories. To change the categories,
use ser.cat.rename_categories instead.
  driver_df['Gender'].replace({'Male':0, 'Female':1}, inplace=True)
/tmp/ipython-input-2453968839.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chain
ed assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  driver_df['Education_Level'].replace({'Graduate':0, '10+':1, '12+':2}, inplace=True)
/tmp/ipython-input-2453968839.py:2: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a fut
ure version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior,
set `pd.set_option('future.no_silent_downcasting', True)`
  driver_df['Education_Level'].replace({'Graduate':0, '10+':1, '12+':2}, inplace=True)
/tmp/ipython-input-2453968839.py:2: FutureWarning: The behavior of Series.replace (and DataFrame.replace) with CategoricalDtype
is deprecated. In a future version, replace will only be used for cases that preserve the categories. To change the categories,
use ser.cat.rename_categories instead.
  driver_df['Education_Level'].replace({'Graduate':0, '10+':1, '12+':2}, inplace=True)
```

In [37]:
```
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()
```



- 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

Data Preprocessing

In [38]:
```
driver_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 19 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   int64         
 7   Dateofjoining              2381 non-null   datetime64[ns]
 8   LastWorkingDate            1616 non-null   datetime64[ns]
 9   Joining Designation        2381 non-null   category      
 10  Grade                      2381 non-null   category      
 11  Total Business Value       2381 non-null   int64         
 12  Quarterly Rating           2381 non-null   int64         
 13  Churn                      2381 non-null   int64         
 14  Income_Change              2381 non-null   int64         
 15  Grade_Change               2381 non-null   int64         
 16  Quarterly Rating_Change    2381 non-null   int64         
 17  Quarterly Rating Improved  2381 non-null   int64         
 18  Income_Raise               2381 non-null   int64         
dtypes: category(4), datetime64[ns](2), float64(1), int64(11), object(1)
memory usage: 289.1+ KB
```

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

In [39]:
```
driver_df.drop(columns=['Driver_ID', 'Gender', 'City', 'Education_Level', 'Dateofjoining', 'LastWorkingDate', 'Income_Change']
driver_df['Quarterly Rating'] = driver_df['Quarterly Rating'].astype('category')
```

```
In [39]:
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')
driver_df.head()
```

Out[39]:

	Months of Service	Age	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	Churn	Grade_Change	Quarterly Rating_Change	Quarterly Rating Improved	Income_Raise
0	3	28.0	57387	1	1	1715580	2	1	0	0	0	0
1	2	31.0	67016	2	2	0	1	0	0	0	0	0
2	5	43.0	65603	2	2	350000	1	1	0	0	0	0
3	3	29.0	46368	1	1	120360	1	1	0	0	0	0
4	5	31.0	78728	3	3	1265000	2	0	0	1	1	0

```
In [40]:
driver_df.duplicated().value_counts()
```

Out[40]:

	count
False	2381

dtype: int64

Handling null values

```
In [41]:
driver_df.isna().sum()
```

Out[41]:

	0
Months of Service	0
Age	0
Income	0
Joining Designation	0
Grade	0
Total Business Value	0
Quarterly Rating	0
Churn	0
Grade_Change	0
Quarterly Rating_Change	0
Quarterly Rating Improved	0
Income_Raise	0

dtype: int64

Outlier Treatment

```
In [42]:
# 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
```

In [43]:
```
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]
    #print('*'*50)
    #print(f'Outliers of \'{column}\' column are:')
    #print("Lower outliers:\n", lower_outliers)
    #print("Higher outliers:\n", higher_outliers)
    #print('*'*50, end="\n")
```

In [44]:
```
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 'Total Business Value' has 336 outliers
The column 'Quarterly Rating Improved' has 358 outliers
```

In [45]:
```
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]
    #print('*'*50)
    #print(f'Outliers of \'{column}\' column are:')
    #print("Lower outliers:\n", lower_outliers)
    #print("Higher outliers:\n", higher_outliers)
    #print('*'*50, end="\n")
```

In [46]:
```
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 16 outliers
The column 'Total Business Value' has 64 outliers
The column 'Quarterly Rating Improved' has 0 outliers
```

- 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

In [47]:
```
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)
```

Multicollinearity Check

In [48]:
```
features_df = driver_df.drop(columns=['Churn']) # Drop target column
features_df = features_df.drop(columns=features_df.select_dtypes(include='category').columns) # Drop category 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
```

Out[48]:

	Features	VIF
0	const	36.42
4	Total Business Value	3.99
1	Months of Service	3.83
3	Income	1.19
2	Age	1.12
5	Quarterly Rating Improved	1.09

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

Encode categorical variables

In [49]:
```
final_df = driver_df.copy()
```

In [50]:
```
# Sepearte out target and feature columns
X = final_df.drop(columns=['Churn'])
y = final_df['Churn']
X.shape, y.shape
```

Out[50]: ((2381, 11), (2381,))

In [51]:
```
#Encode target variable
y = y.astype(int)
```

In [52]:
```
#Encode features with just 2 classes as 0 or 1
X[['Grade_Change','Quarterly Rating_Change', 'Income_Raise']] = X[['Grade_Change','Quarterly Rating_Change', 'Income_Raise']].
```

One-Hot-Encoding for remaining categorical features

In [53]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 11 columns):
 #   Column                     Non-Null Count  Dtype   
---  ------                     --------------  -----   
 0   Months of Service          2381 non-null   int64   
 1   Age                        2381 non-null   float64 
 2   Income                     2381 non-null   int64   
 3   Joining Designation        2381 non-null   category
 4   Grade                      2381 non-null   category
 5   Total Business Value       2381 non-null   float64 
 6   Quarterly Rating           2381 non-null   category
 7   Grade_Change               2381 non-null   int8    
 8   Quarterly Rating_Change    2381 non-null   int8    
 9   Quarterly Rating Improved  2381 non-null   int64   
 10  Income_Raise               2381 non-null   int8    
dtypes: category(3), float64(2), int64(3), int8(3)
memory usage: 107.7 KB
```

In [54]:
```
categorical_columns = X.select_dtypes(include='category').columns
categorical_columns
```

Out[54]: Index(['Joining Designation', 'Grade', 'Quarterly Rating'], dtype='object')

In [55]:
```
encoder = OneHotEncoder(sparse_output=False)
encoded_data = encoder.fit_transform(X[categorical_columns])
encoded_df = pd.DataFrame(encoded_data, columns = encoder.get_feature_names_out(categorical_columns))
X = pd.concat([X, encoded_df], axis=1)
X.drop(columns = categorical_columns, inplace=True)
X.head()
```

Out[55]:

	Months of Service	Age	Income	Total Business Value	Grade_Change	Quarterly Rating_Change	Quarterly Rating Improved	Income_Raise	Joining Designation_1	Joining Designation_2	...	Jo Designati
0	3	28.0	57387	1715580.0	0	0	0	0	1.0	0.0	...	
1	2	31.0	67016	0.0	0	0	0	0	0.0	1.0	...	
2	5	43.0	65603	350000.0	0	0	0	0	0.0	1.0	...	
3	3	29.0	46368	120360.0	0	0	0	0	1.0	0.0	...	
4	5	31.0	78728	1265000.0	0	1	1	0	0.0	0.0	...	

5 rows × 22 columns

Model building

In [56]:
```
#Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[56]: ((1904, 22), (477, 22), (1904,), (477,))

Perform data normalization/standardization

Data normalization/standardization is required so that features with higher scales do not dominate the model's performance. Hence all features should have same scale

In [57]:
```
X_train.head()
```

Out[57]:

	Months of Service	Age	Income	Total Business Value	Grade_Change	Quarterly Rating_Change	Quarterly Rating Improved	Income_Raise	Joining Designation_1	Joining Designation_2	...	Desig
2236	7	28.0	57164	1092560.0	0	0	0	0	0.0	1.0	...	
6	1	28.0	42172	0.0	0	0	0	0	1.0	0.0	...	
1818	1	29.0	43989	0.0	0	0	0	0	1.0	0.0	...	
1534	7	40.0	59636	2589640.0	0	0	0	0	0.0	0.0	...	
2123	6	25.0	29052	2172260.0	0	0	0	0	1.0	0.0	...	

5 rows × 22 columns

In [58]:
```
min_max_scaler = MinMaxScaler()
# Fit min_max_scaler to training data
min_max_scaler.fit(X_train)
# Scale the training and testing data
X_train = pd.DataFrame(min_max_scaler.transform(X_train), columns=X_train.columns)
X_test = pd.DataFrame(min_max_scaler.transform(X_test), columns=X_test.columns)
```

In [59]:
```
# Data after scaling
X_train.head()
```

Out[59]:

	Months of Service	Age	Income	Total Business Value	Grade_Change	Quarterly Rating_Change	Quarterly Rating Improved	Income_Raise	Joining Designation_1	Joining Designation_2	...	Desi
0	0.260870	0.205882	0.261253	0.074297	0.0	0.5	0.0	0.0	0.0	1.0	...	
1	0.000000	0.205882	0.176872	0.041541	0.0	0.5	0.0	0.0	1.0	0.0	...	
2	0.000000	0.235294	0.187099	0.041541	0.0	0.5	0.0	0.0	1.0	0.0	...	
3	0.260870	0.558824	0.275166	0.119183	0.0	0.5	0.0	0.0	0.0	0.0	...	
4	0.217391	0.117647	0.103028	0.106669	0.0	0.5	0.0	0.0	1.0	0.0	...	

5 rows × 22 columns

In [60]:
```
#Check for imbalance in target class
y_train.value_counts(normalize=True)*100
```

Out[60]:

Churn	proportion
1	68.644958
0	31.355042

dtype: float64

- We can see a clear imbalance in the target class with 1 being ~69% and 0 being ~31%. Hence, I will use SMOTE to fix this imbalance

In [61]:
```
sm = SMOTE(random_state=0)
X_train, y_train = sm.fit_resample(X_train, y_train)
y_train.value_counts(normalize=True)*100
```

Out[61]:

Churn	proportion
1	50.0
0	50.0

dtype: float64

Ensemble Learning: Bagging - RandomForestClassifier

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The hyper-parameters of the random forest classifier will be selected using grid

search cross validation

In [62]:
```
# Define a smaller and faster parameter grid
param_grid = {
    'n_estimators': [100, 300, 500],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [10, 30, 50],
    'min_samples_split': [2, 4, 6]
}

# Initialize classifier
rf = RandomForestClassifier()

# Use RandomizedSearchCV for speed (same output style)
rf_random = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_grid,
    n_iter=10,
    cv=3,
    verbose=1,
    n_jobs=-1,
    random_state=42
)

# Fit the model
rf_random.fit(X_train, y_train)

# Best parameters
print("Best parameters found: ", rf_random.best_params_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best parameters found:  {'n_estimators': 300, 'min_samples_split': 2, 'max_features': 'sqrt', 'max_depth': 30}
```

In [63]:
```
color = '\033[91m'
bold = '\033[1m'
end = '\033[0m'
# Predict and evaluate performance
y_true = y_train
y_pred = rf_random.predict(X_train)
print(color + bold + "Train data:" + color + end)
print("Accuracy: ", accuracy_score(y_true, y_pred))
print("Classification Report:\n", classification_report(y_true, y_pred))
y_true = y_test
y_pred = rf_random.predict(X_test)
print(color + bold + "Test data:" + color + end)
print("Accuracy: ", accuracy_score(y_true, y_pred))
print("Classification Report:\n", classification_report(y_true, y_pred))
```

```
Train data:
Accuracy:  1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1307
           1       1.00      1.00      1.00      1307

    accuracy                           1.00      2614
   macro avg       1.00      1.00      1.00      2614
weighted avg       1.00      1.00      1.00      2614

Test data:
Accuracy:  0.7819706498951782
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.64      0.67       168
           1       0.81      0.86      0.84       309

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

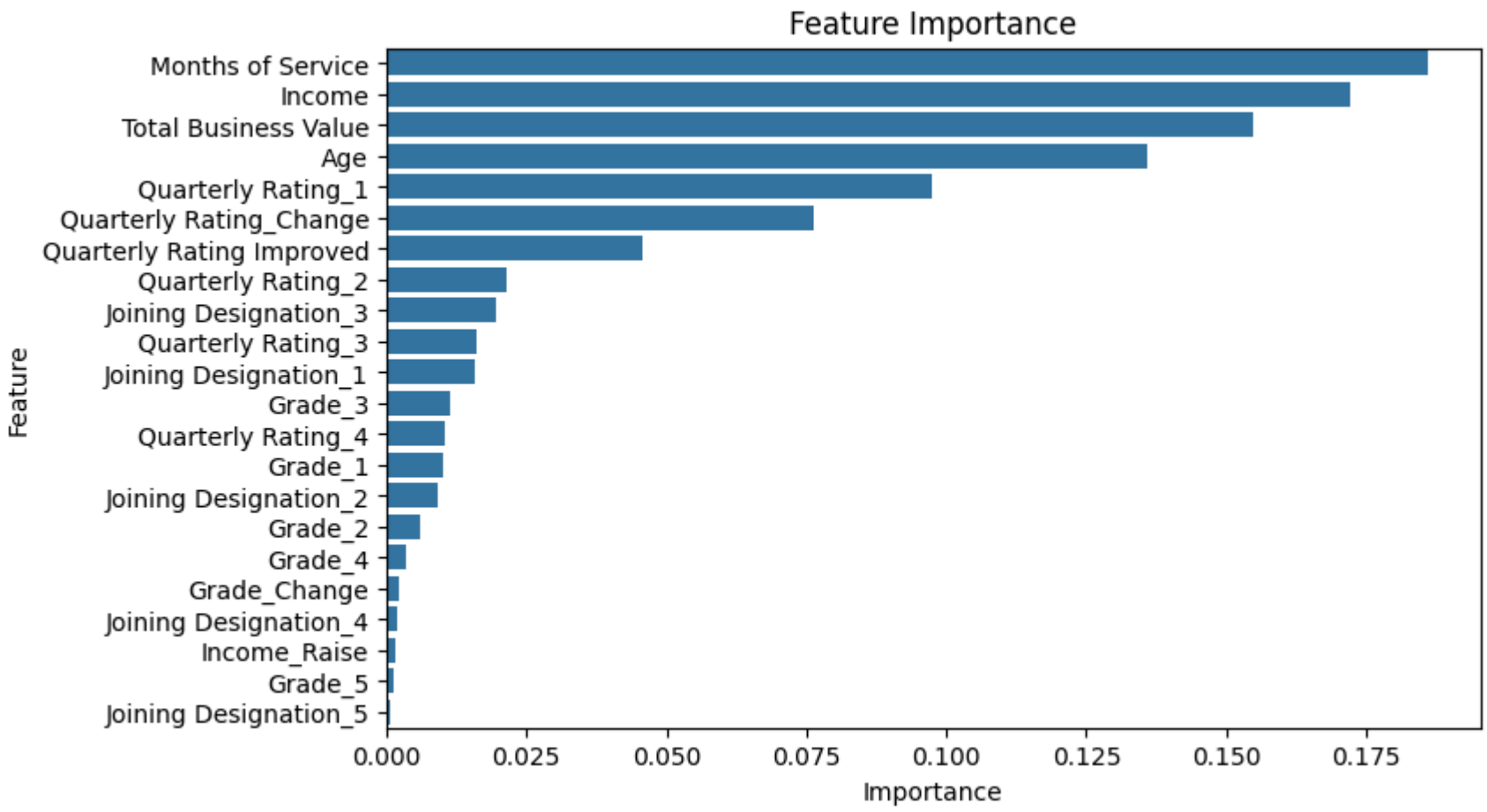
- The model achieves a perfect training accuracy of 1.0, but the testing accuracy drops to 0.77, indicating strong overfitting. The classifier has memorized the training data very well but fails to generalize effectively to unseen data. Despite this, the best parameters selected through tuning fall within the expected and predefined hyperparameter ranges.

Feature Importance Plot

In [64]:
```
def plot_feature_importance(estimator, features):
    # 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('Feature Importance')
    plt.show()
```

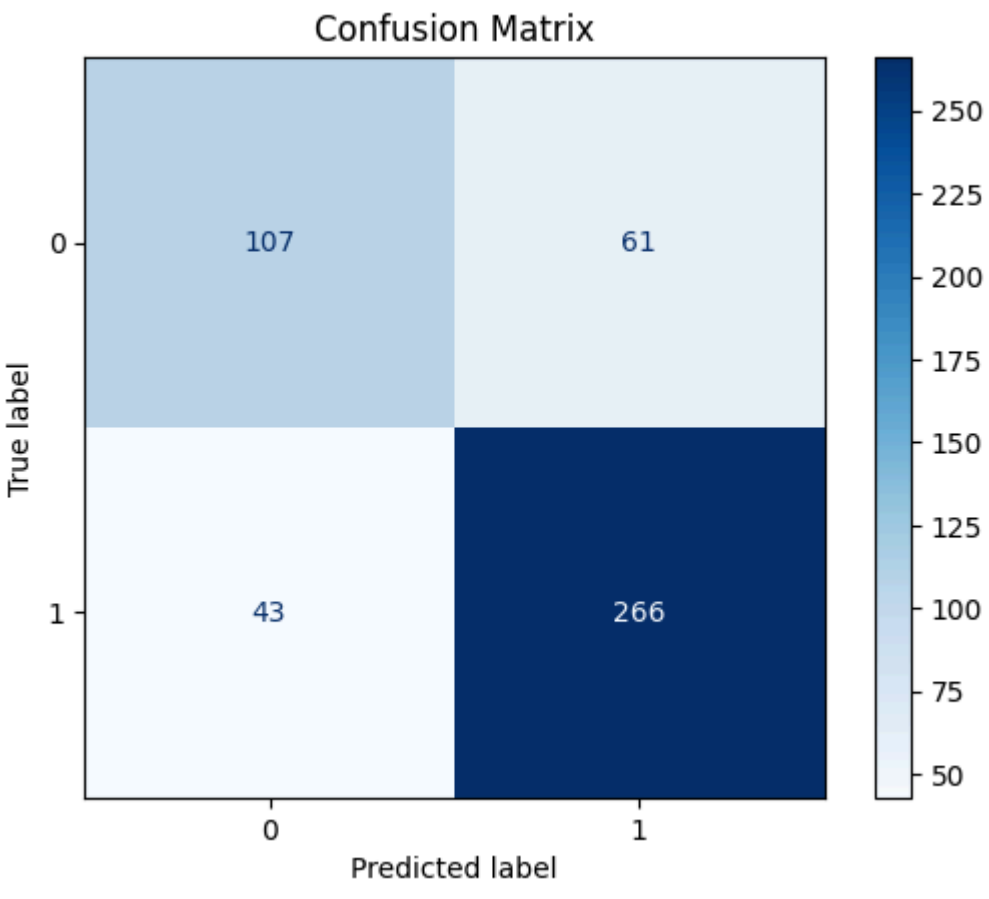
In [65]:
```
plot_feature_importance(rf_random.best_estimator_, X_train.columns)
```



Confusion Matrix

In [66]:
```
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()
```

In [67]:
```
display_confusion_matrix(y_test, y_pred)
```

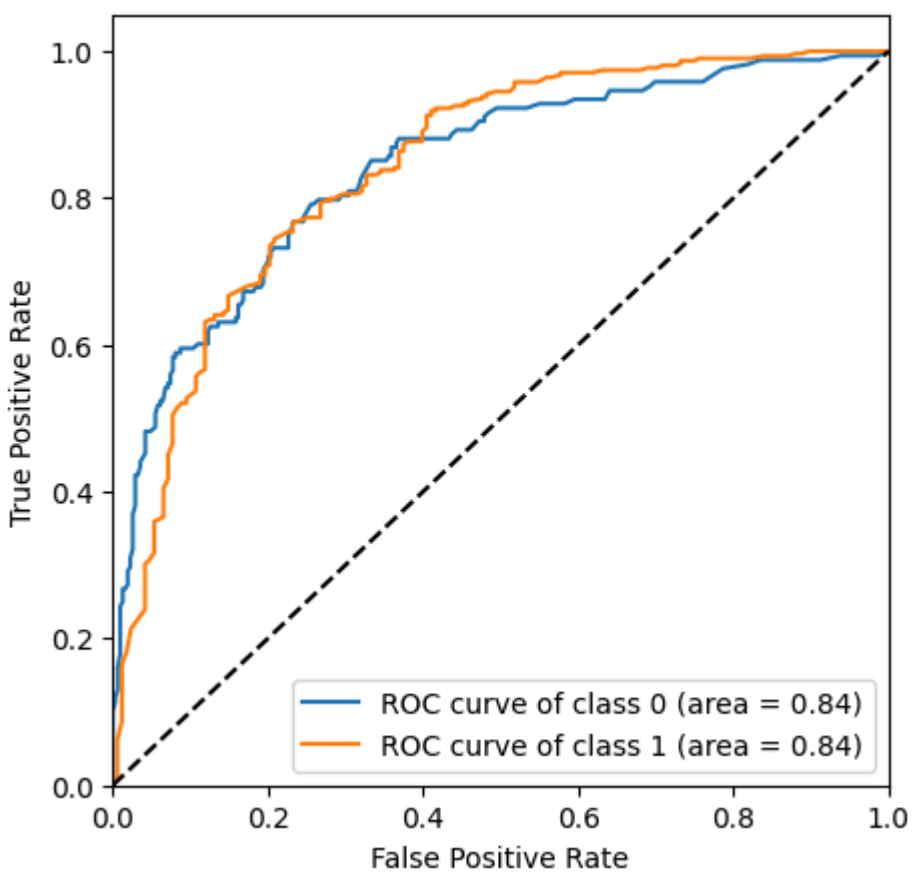


ROC Curve

In [68]:
```
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()
```

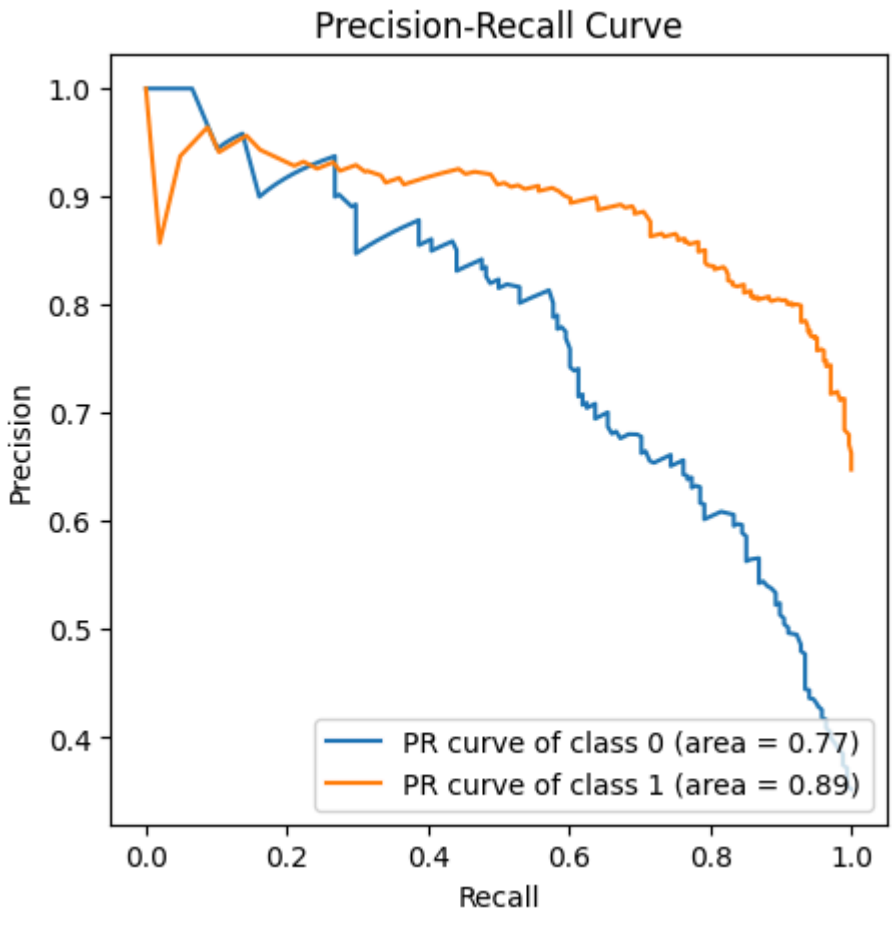
In [69]:
```
plot_roc_curve(rf_random.best_estimator_, X_train, X_test, y_train, y_test)
```



Precision-Recall Curve

In [70]:
```
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()
```

In [71]:
```
plot_pr_curve(rf_random.best_estimator_, X_train, X_test, y_train, y_test)
```



- The top 5 features as per the RandomForestCLassifier are

    1. Months of Service
    2. Income
    3. Total Business Value
    4. Age
    5. Quarterly Rating 1
- Both the classes 0 and 1 have a decent Area Under the ROC curve of 0.84
- The Area Under the PR curve for class 0 is 0.78 and class 1 is 0.89

Ensemble Learning: Boosting - GradientBoostingClassifier

This algorithm builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n_classes_ regression trees are fit on the negative gradient of the loss function, e.g. binary or multiclass log loss. The hyper-parameters of the GradientBoostingClassifier will be selected using random search cross validation

In [72]:
```
# Define parameter grid (FAST)
param_grid = {
    'n_estimators': np.arange(100, 401, 100),
    'learning_rate': np.logspace(-3, 0, 10),
    'max_depth': np.arange(3, 11, 1),
    'min_samples_split': np.arange(2, 11, 2),
    'min_samples_leaf': np.arange(1, 11, 2),
    'subsample': np.linspace(0.5, 1.0, 6)
}

# Randomized search
gb = GradientBoostingClassifier()

gb_random = RandomizedSearchCV(
    estimator=gb,
    param_distributions=param_grid,
    n_iter=30,
    cv=3,
    verbose=1,
    random_state=42,
    n_jobs=-1
)

# Fit model
gb_random.fit(X_train, y_train)

# Best parameters
print("Best parameters found for GradientBoostingClassifier: ", gb_random.best_params_)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
Best parameters found for GradientBoostingClassifier:  {'subsample': np.float64(0.6), 'n_estimators': np.int64(300), 'min_sampl
es_split': np.int64(2), 'min_samples_leaf': np.int64(9), 'max_depth': np.int64(10), 'learning_rate': np.float64(0.0464158883361
27774)}
```

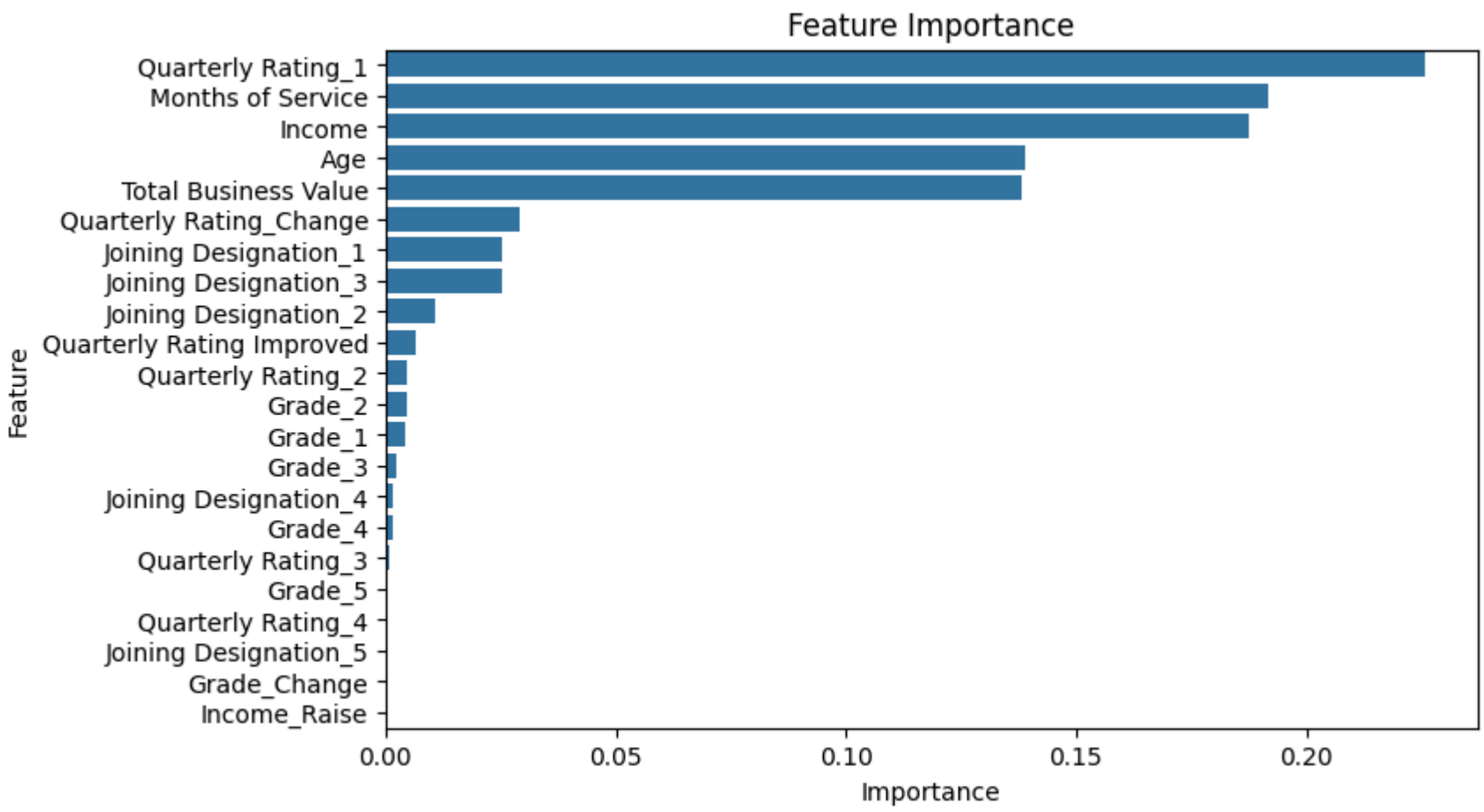
In [73]:
```
color = '\033[91m'
bold = '\033[1m'
end = '\033[0m'
# Predict and evaluate performance
y_true = y_train
y_pred = gb_random.predict(X_train)
print(color + bold + "Train data:" + color + end)
print("Accuracy: ", accuracy_score(y_true, y_pred))
print("Classification Report:\n", classification_report(y_true, y_pred))
y_true = y_test
y_pred = gb_random.predict(X_test)
print(color + bold + "Test data:" + color + end)
print("Accuracy: ", accuracy_score(y_true, y_pred))
print("Classification Report:\n", classification_report(y_true, y_pred))
```

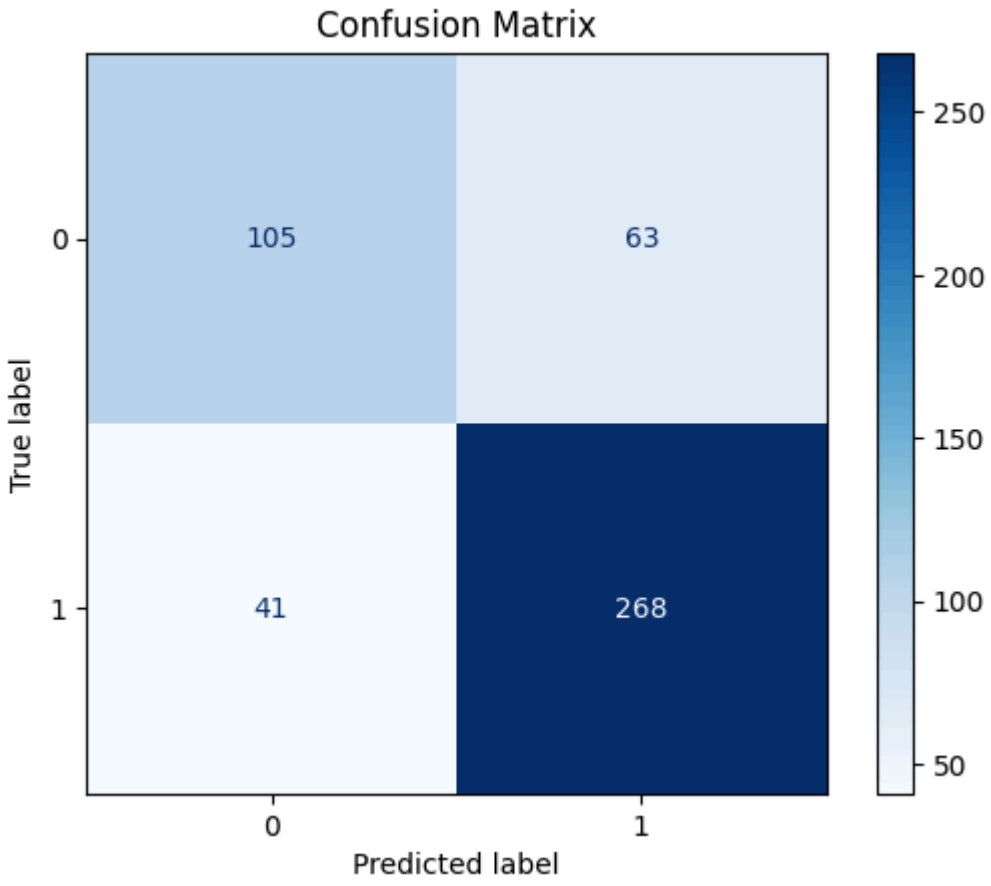
```
Train data:
Accuracy:  0.9992348890589136
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1307
           1       1.00      1.00      1.00      1307

    accuracy                           1.00      2614
   macro avg       1.00      1.00      1.00      2614
weighted avg       1.00      1.00      1.00      2614

Test data:
Accuracy:  0.7819706498951782
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.62      0.67       168
           1       0.81      0.87      0.84       309

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

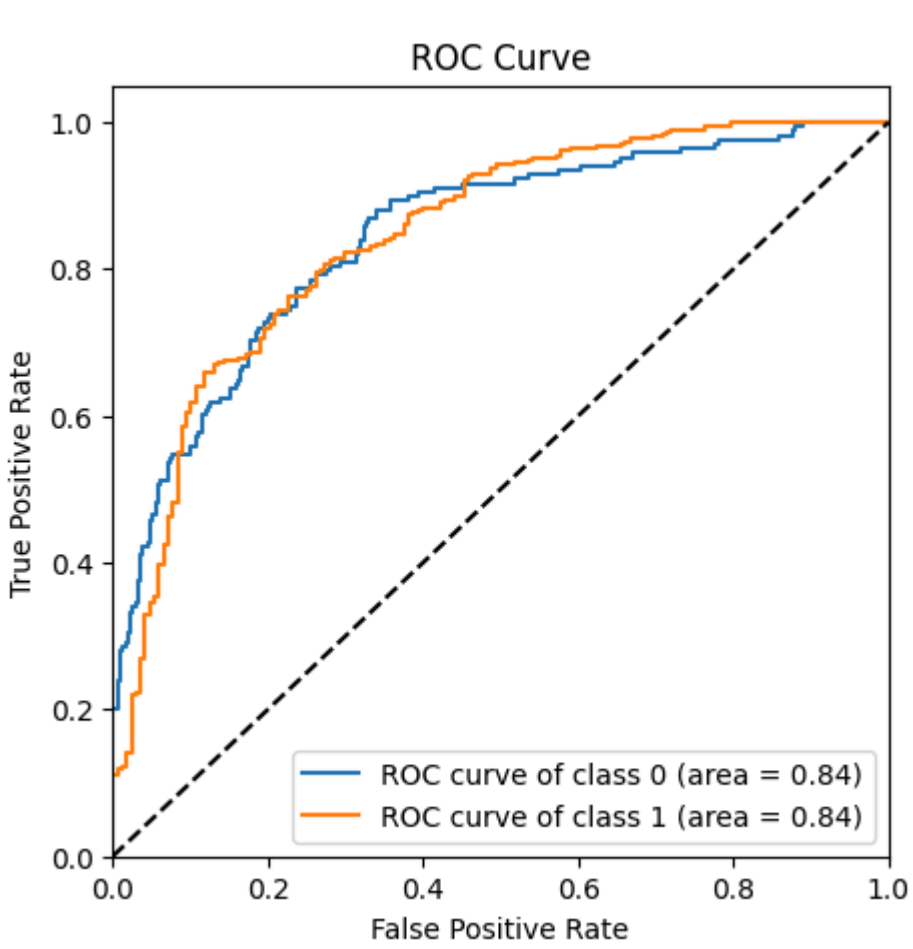
- The training accuracy is 0.999 while the testing accuracy is 0.78, which clearly indicates overfitting. The model performs almost perfectly on the training data but shows noticeably lower performance on unseen data. Although the test accuracy is acceptable, the large gap between training and testing accuracy confirms that the model has over-learned from the training set and is not generalizing effectively.

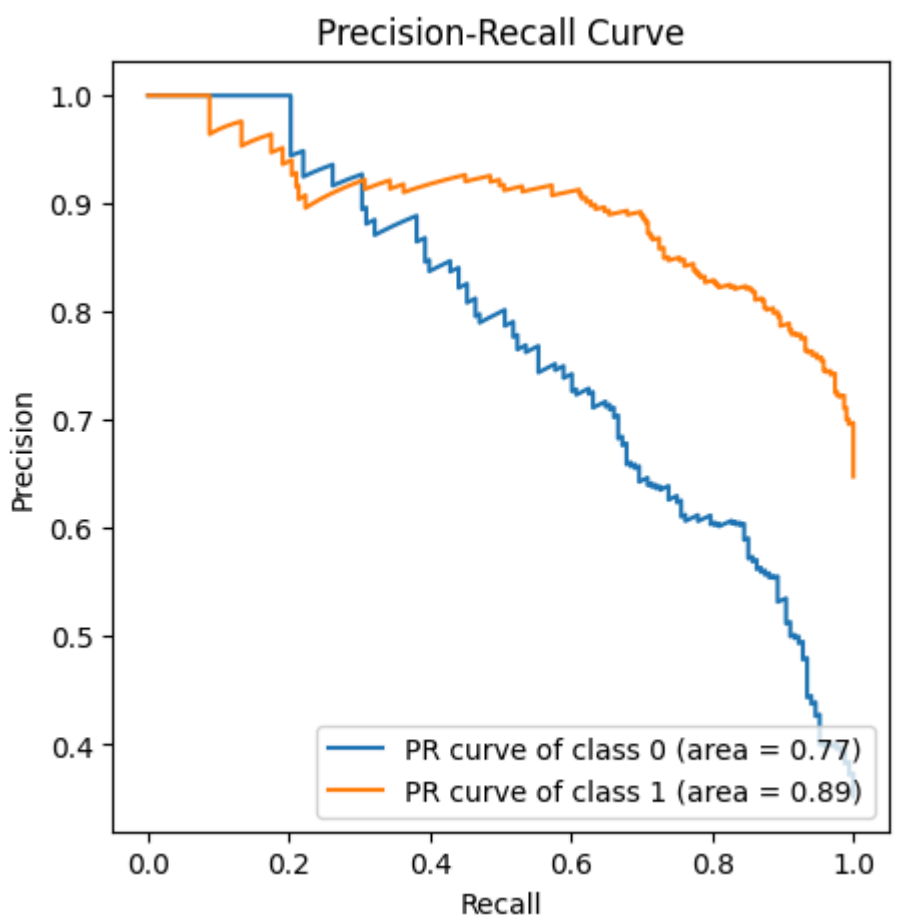
Performance

In [74]:
```
plot_feature_importance(gb_random.best_estimator_, X_train.columns)
display_confusion_matrix(y_test, y_pred)
plot_roc_curve(gb_random.best_estimator_, X_train, X_test, y_train, y_test)
plot_pr_curve(gb_random.best_estimator_, X_train, X_test, y_train, y_test)
```









- The GradientBoostingClassifier identified Quarterly Rating_1, Months of Service,Income,Age and Total Business Value as the top five drivers of driver attrition.

- The ROC-AUC for both classes is approximately 0.84, indicating strong model ability to separate attrition vs. non-attrition cases.
- The Precision-Recall AUC is 0.77 for class 0 and 0.89 for class 1, suggesting the model is especially effective at correctly identifying employees who are likely to leave the company.

Ensemble Learning: Boosting - XGBClassifier

XGBClassifier is a highly optimized version of GBM. It includes regularization to prevent overfitting and various other enhancements. The hyper-parameters of the XGBClassifier will be selected using random search cross validation

In [75]:
```
# Define parameter grid
param_grid = {
    'n_estimators': np.arange(100, 1001, 100),
    'learning_rate': np.logspace(-3, 0, 10),
    'max_depth': np.arange(3, 11, 1),
    'min_child_weight': np.arange(1, 11, 1),
    'gamma': np.logspace(-3, 1, 10),
    'subsample': np.linspace(0.5, 1.0, 6),
    'colsample_bytree': np.linspace(0.5, 1.0, 6)
}

# Initialize classifier and RandomizedSearchCV
xgb = XGBClassifier(eval_metric='mlogloss')

xgb_random = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=param_grid,
    n_iter=30,     # ↓ from 300 to 30 (10x faster)
    cv=3,
    verbose=1,     # ↓ faster log output
    random_state=42,
    n_jobs=-1
)

# Fit the model
xgb_random.fit(X_train, y_train)

# Evaluate best parameters
print("Best parameters found for XGBoost: ", xgb_random.best_params_)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
Best parameters found for XGBoost:  {'subsample': np.float64(0.5), 'n_estimators': np.int64(700), 'min_child_weight': np.int64
(1), 'max_depth': np.int64(4), 'learning_rate': np.float64(0.1), 'gamma': np.float64(0.001), 'colsample_bytree': np.float64(0.
7)}
```

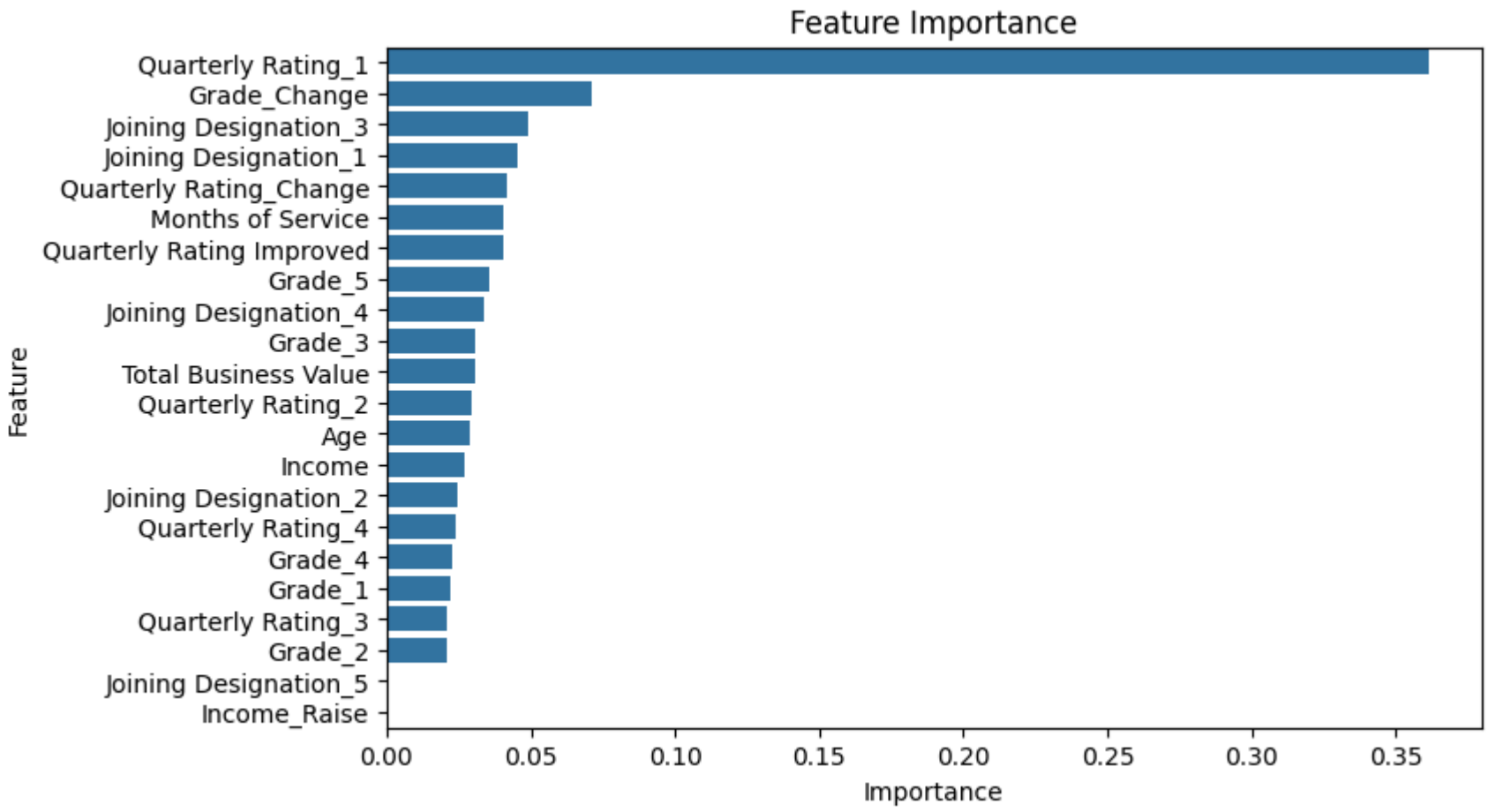
In [76]:
```
color = '\033[91m'
bold = '\033[1m'
end = '\033[0m'
# Predict and evaluate performance
y_true = y_train
y_pred = xgb_random.predict(X_train)
print(color + bold + "Train data:" + color + end)
print("Accuracy: ", accuracy_score(y_true, y_pred))
print("Classification Report:\n", classification_report(y_true, y_pred))
y_true = y_test
y_pred = xgb_random.predict(X_test)
print(color + bold + "Test data:" + color + end)
print("Accuracy: ", accuracy_score(y_true, y_pred))
print("Classification Report:\n", classification_report(y_true, y_pred))
```

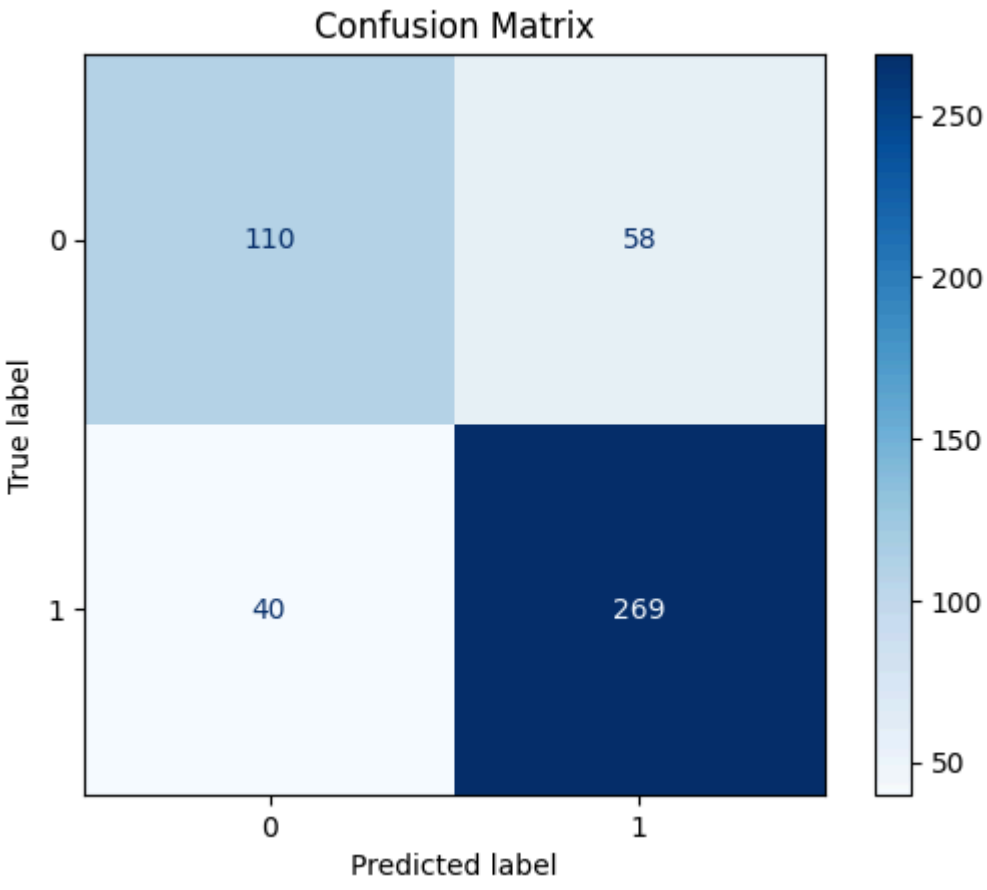
```
Train data:
Accuracy:  0.9713083397092579
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      1307
           1       0.96      0.98      0.97      1307

    accuracy                           0.97      2614
   macro avg       0.97      0.97      0.97      2614
weighted avg       0.97      0.97      0.97      2614

Test data:
Accuracy:  0.7945492662473794
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.65      0.69       168
           1       0.82      0.87      0.85       309

    accuracy                           0.79       477
   macro avg       0.78      0.76      0.77       477
weighted avg       0.79      0.79      0.79       477
```

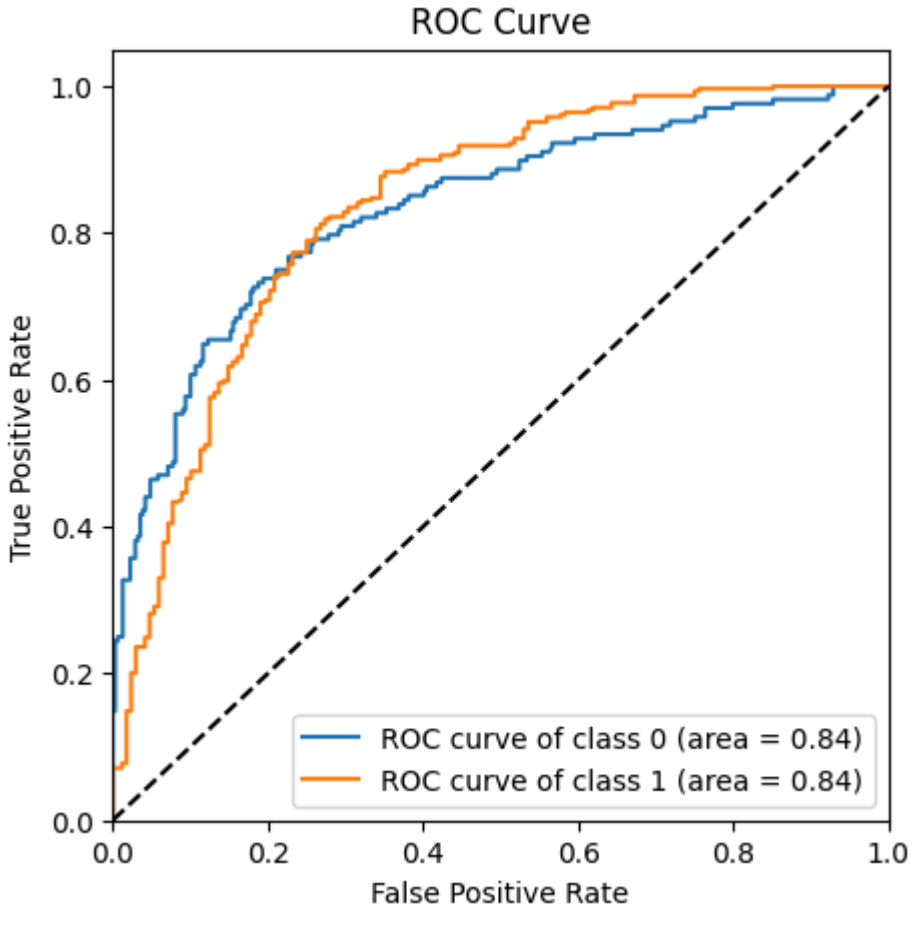
- The training accuracy is 0.97 while the testing accuracy is 0.79, indicating that overfitting is still present. However, the gap between training and testing performance has reduced compared to the previous models, showing improved generalization. This model also runs faster than the earlier ensemble models, making it a more efficient choice.

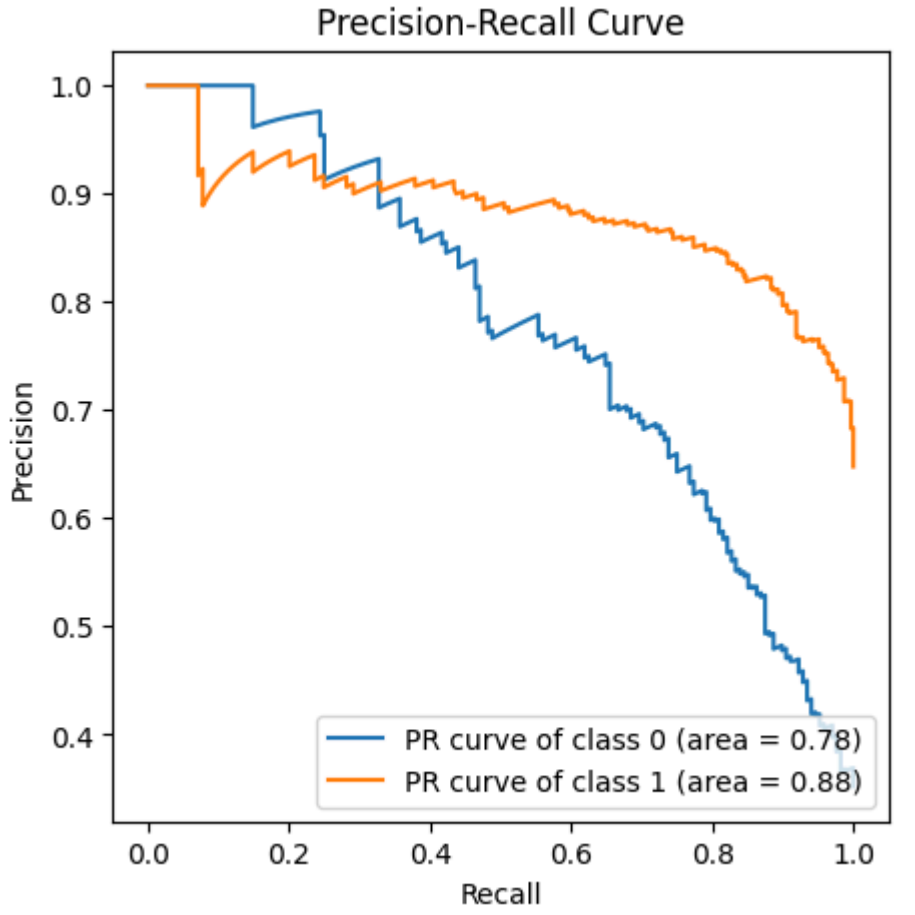
Performance

In [77]:
```
plot_feature_importance(xgb_random.best_estimator_, X_train.columns)
display_confusion_matrix(y_test, y_pred)
plot_roc_curve(xgb_random.best_estimator_, X_train, X_test, y_train, y_test)
plot_pr_curve(xgb_random.best_estimator_, X_train, X_test, y_train, y_test)
```









- The top 5 features as per the XGBClassifier are
  1.Quarterly Rating 1
  2.Grade_Change
  3.Joining Designation_3
  4.Joining Designation_1 5.Quarterly Rating_Change

- Both the classes 0 and 1 have a decent Area Under the ROC curve of 0.84
- The Area Under the PR curve for class 0 is 0.78 and class 1 is 0.88

Insights

- The churn rate among drivers is quite high, with 68% of drivers having left the company at some point.
- The majority of drivers are in the age group of 30 to 35 years, and most are male (59%) with a right-skewed income distribution centered between 40k and 75k.
- Key features influencing driver attrition include Quarterly Rating, Months of Service, Income, Age, and Total Business Value; lower values in these metrics correlate with higher churn.
- Drivers with higher grades, increased income, or improved Quarterly Rating show significantly lower churn rates.

- Cities show distinct patterns: certain cities like C13 have much higher churn rates, while others, such as C29, show marked improvements in driver ratings year-over-year.
- A significant drop in a driver's rating leads to a decrease in Total Business Value, often serving as a precursor to churn as drivers become less motivated to accept rides.
- Model performance shows a notable gap between training and testing accuracy (e.g., 99% train, 78% test in some models), indicating overfitting concerns.

Recommendations

- Focus retention efforts on drivers with low Quarterly Ratings, short tenure, and negative changes in income or grade, as these populations are at elevated risk of churn.
- Implement targeted programs in high-churn cities (like C13), taking successful engagement strategies from lower-churn areas such as C29 for potential adaptation.
- Increase monitoring and support for drivers who experience a significant decline in their Quarterly Ratings or show early signs of disengagement to prevent loss of motivation and revenue.
- Consider creating incentives for drivers achieving higher grades or who demonstrate improvement in key metrics, as this correlates strongly with retention.
- Address data imbalance through continued use of SMOTE or similar techniques, but review model complexity and apply regularization or further feature selection to reduce overfitting and improve model generalization.
- Maintain continuous feedback and regular review cycles for model predictions and operational changes, ensuring that insights remain actionable and up to date.

Questions and Answers for Project

1. What is the main problem addressed by this project?

- The project focuses on understanding and predicting driver attrition (churn) for Ola using demographic, tenure, and performance data, as driver churn is a critical business issue due to high replacement costs and operational challenges.

---

2. Which features are most influential in predicting driver churn?

- Key features affecting churn include Quarterly Rating, Months of Service, Income, Age, and Total Business Value, with lower values of these features associated with a higher likelihood of a driver leaving the company.

---

3. How was data imbalance in the churn target handled?

- The dataset was imbalanced, with a much higher proportion of drivers who had churned. SMOTE (Synthetic Minority Over-sampling Technique) was used to balance the classes during model training.

---

4. What machine learning models did you use, and why?

- Ensemble methods, specifically Random Forest (bagging) and Gradient Boosting/XGBoost (boosting), were used for predictive modeling. These models handle complex relationships well and are robust to overfitting with proper tuning.

---

5. What were the main insights from exploratory analysis?

- A high overall churn rate was found (about 68%), with most churned drivers being younger, receiving lower ratings, and earning less income compared to non-churned drivers. Some cities and groups had especially high or low churn rates.

---

6. How did you engineer additional features for better modeling?

- New features included flags for income increase, quarterly rating improvement, and changes in grade, which helped capture trends in driver satisfaction and performance. A target variable was also created to directly indicate churn.

---

7. What did your model evaluation reveal?

- The models performed well, with ROC AUC scores around 0.84 and PR AUC above 0.77, indicating strong discriminative power. However, some overfitting was observed, as shown by higher training versus test accuracy.

---

8. What actionable business recommendations emerged from your study?

- Recommendations include targeting retention programs toward drivers with declining ratings or incomes, focusing on high-churn cities, and incentivizing improvement in key performance metrics such as ratings and grades.

---

9. Why is driver retention more cost-effective than acquisition for Ola?

- Retaining drivers helps avoid the high costs associated with training and onboarding new hires, minimizes service disruptions, and stabilizes company morale and reputation.

---

10. What future improvements can be made to this predictive system?

- Improvements might include more advanced feature engineering, collecting additional behavioral data, ongoing model retraining, and integrating model predictions into operational workflows for timely intervention.

In [ ]: