Project Description:

A ride sharing business is trying to retain its drivers as the churn rate among drivers is high and new driver acquisition is more expensive than retaining drivers. I am 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)

I will be looking at the potential reasons for driver attrition among the variables provided in the data mentioned below:

- MMMM-YY : Reporting Date (Monthly)
- Driver_ID : Unique id for drivers
- Age : Age of the driver
- Gender : Gender of the driver – Male : 0, Female: 1
- City : City Code of the driver
- Education_Level : Education level – 0 for 10+ ,1 for 12+ ,2 for graduate
- Income : Monthly average Income of the driver
- Date Of Joining : Joining date for the driver
- LastWorkingDate : Last date of working for the driver
- Joining Designation : Designation of the driver at the time of joining
- Grade : Grade of the driver at the time of reporting
- Total Business Value : The total business value acquired by the driver in a month (negative business indicates

cancellation/refund or car EMI adjustments)

- Quarterly Rating : Quarterly rating of the driver: 1,2,3,4,5 (higher is better)

There is no churn data column present above, so I will looking at the LastWorkingDate column to find whether a driver left or not.

In [196…

```
# useful imports
import numpy as np, seaborn as sns, pandas as pd, matplotlib.pyplot as plt, scipy
df = pd.read_csv('ola_driver_scaler.csv')
# df.head()
```

I am told to keep the data snippet confidential so it would not be visible here.

In [197...
```
df.shape
```

Out[197]:
```
(19104, 14)
```

Since we can see that there are multiple rows for each Driver_ID, we would need to aggregate those rows later on in order to analyze each driver's data at once.

In [198...
```
df.drop(['Unnamed: 0'],axis=1,inplace=True)
```

In [199...
```
df.info() # some missing values are seen
```

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

In [200...
```
df.columns[(df.dtypes == "float64") | (df.dtypes == "int64")]
```

Out[200]:
```
Index(['Driver_ID', 'Age', 'Gender', 'Education_Level', 'Income',
       'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating'],
      dtype='object')
```

The categorical columns are:

In [201...
```
df.columns[df.dtypes =="object"]
```

Out[201]:
```
Index(['MMM-YY', 'City', 'Dateofjoining', 'LastWorkingDate'], dtype='object')
```

checking for null values

There are many missing values in Age, Gender, LastWorkingDate. Since they are less than 10% of the total entries in Age and Gender, we can impute them rather than removing the columns. But LastWorkingDate column has more than 90% missing entries. We would be checking it again after we aggregate the data for each driver.

In [202...
```
df.isna().sum()*100/len(df)
```

Out[202]:
```
MMM-YY                   0.000000
Driver_ID                0.000000
Age                      0.319305
Gender                   0.272194
City                     0.000000
Education_Level          0.000000
Income                   0.000000
Dateofjoining            0.000000
LastWorkingDate         91.541039
Joining Designation      0.000000
Grade                    0.000000
Total Business Value     0.000000
Quarterly Rating         0.000000
dtype: float64
```

checking for duplicated values

There are no duplicate entries.

In [203…
```
df.duplicated().sum()
```

Out[203]:
```
0
```

changing date columns to DateTime format

In [204…
```
df['MMM-YY'] = pd.to_datetime(df['MMM-YY'])
df['Dateofjoining'] = pd.to_datetime(df['Dateofjoining'])
df['LastWorkingDate'] = pd.to_datetime(df['LastWorkingDate'])
```

In [205…
```
cont = df[['Age', 'Gender']].values

from sklearn.impute import KNNImputer

imputer = KNNImputer(missing_values=np.nan, n_neighbors=3)
imputer = imputer.fit(cont)
cont = imputer.transform(cont).astype('int')
cont[:5]
```

Out[205]:
```
array([[28,  0],
       [28,  0],
       [28,  0],
       [31,  0],
       [31,  0]])
```

In [206…
```
df[['Age', 'Gender']] = cont
```

Data aggregation

In [207…
```
# grouping based on Driver ID
agg_df = df.groupby(['Driver_ID']).agg({'MMM-YY':'last','Age':'first','Gender':'first','City':'f
        'Education_Level':'first','Income':['first','last','mean'],'Dateofjoining':'first','Last
         ,'Joining Designation':'first','Grade':'first','Total Business Value':'sum',
                                        'Quarterly Rating':['first','last','mean']})
df2 = agg_df.reset_index()
```

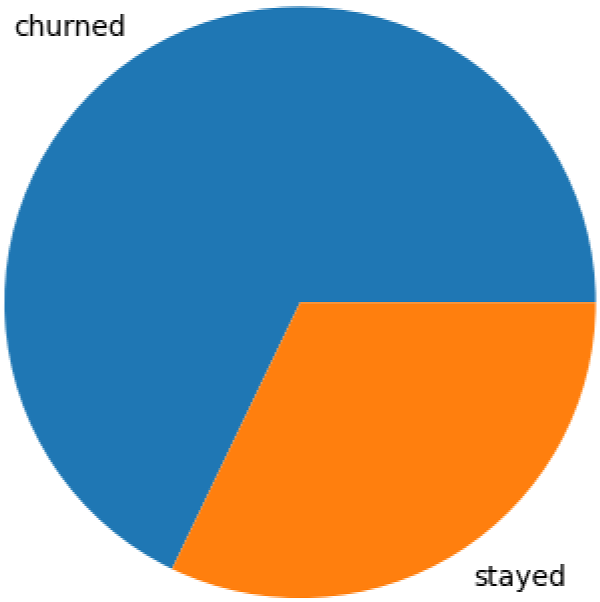
In [208…
```
df2.columns = ['_'.join(col) for col in df2.columns.values]
```

In [209…
```
df2.columns
```
Out[209]:
```
Index(['Driver_ID_', 'MMM-YY_last', 'Age_first', 'Gender_first', 'City_first',
       'Education_Level_first', 'Income_first', 'Income_last', 'Income_mean',
       'Dateofjoining_first', 'LastWorkingDate_last', 'LastWorkingDate_any',
       'Joining Designation_first', 'Grade_first', 'Total Business Value_sum',
       'Quarterly Rating_first', 'Quarterly Rating_last',
       'Quarterly Rating_mean'],
      dtype='object')
```
In [210…
```
df2.shape
```
Out[210]:
```
(2381, 18)
```
Aggregation reduced the number of rows from 19104 to 2381.

So, there are 2381 unique drivers.

In [211…
```
df2['LastWorkingDate_any']
```
Out[211]:
```
0        True
1       False
2        True
3        True
4       False
        ...
2376    False
2377     True
2378     True
2379     True
2380    False
Name: LastWorkingDate_any, Length: 2381, dtype: bool
```
In [212…
```
df2['Churn'] = df2['LastWorkingDate_any']
df2['Churn'] = df2['Churn'].astype('int')
```
In [213…
```
df2['Churn'].value_counts()/len(df2)*100
```
Out[213]:
```
1    67.870643
0    32.129357
Name: Churn, dtype: float64
```
In [214…
```
df2['Churn'].value_counts()
```
Out[214]:
```
1    1616
0     765
Name: Churn, dtype: int64
```
67.87% (n=1616) drivers churned, whereas 32.1% (n=765) drivers stayed.

In [215…
```
fig = plt.plot(figsize=(3,3))
plt.pie(df2['Churn'].value_counts(),labels=['churned','stayed'])
plt.show()
```



There are more drivers that churned than those who stayed.

In [216…
```
df2['RateIncrease'] = np.where((df2['Quarterly Rating_last'] > df2['Quarterly Rating_first']),1,(
```

In [217…
```
df2['Income_Increase'] = np.where((df2['Income_last'] > df2['Income_first']),1,0)
```

In [218…
```
df2['LastDate'] = np.where(df2['LastWorkingDate_last'].notnull(),
                        df2['LastWorkingDate_last'].dt.strftime('%Y-%m-%d'),
                        df2['MMM-YY_last'].dt.strftime('%Y-%m-%d'))
# replacing NaT values in last Working date column with Last reporting date 'MMM-YY'
```

In [219…
```
df2[['MMM-YY_last','LastWorkingDate_last','LastDate']].head()
```

Out[219]:

	MMM-YY_last	LastWorkingDate_last	LastDate
0	2019-03-01	2019-03-11	2019-03-11
1	2020-12-01	NaT	2020-12-01
2	2020-04-01	2020-04-27	2020-04-27
3	2019-03-01	2019-03-07	2019-03-07
4	2020-12-01	NaT	2020-12-01

In [220…
```
df2['tenure'] = pd.to_datetime(df2['LastDate']) - df2['Dateofjoining_first']
```

In [221…
```
df2 = df2.drop(['Income_first','Income_last','Quarterly Rating_first','Quarterly Rating_last','La
          ,'LastWorkingDate_any','MMM-YY_last','Dateofjoining_first'],axis=1)
```

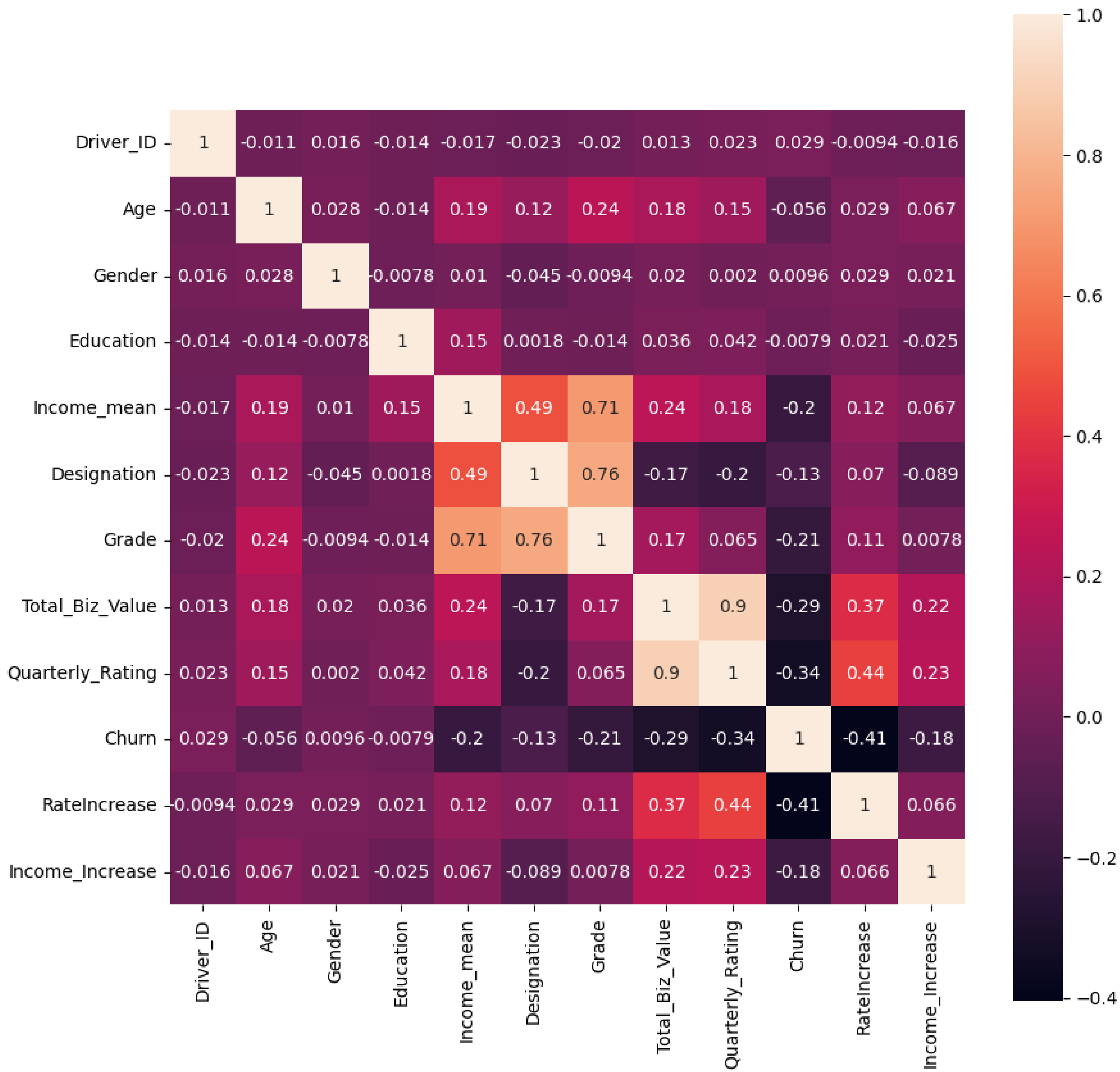
In [222…
```
df2.rename(columns = {'Driver_ID_':'Driver_ID','Age_first':'Age','Gender_first':'Gender','City_fi
                      'Education_Level_first':'Education','Joining Designation_first':'Designatio
                      'Total Business Value_sum':'Total_Biz_Value','Quarterly Rating_mean':'Quart
                     }, inplace=True)
```

In [223…

```
# Spearman's Rank Correlation Coefficient
plt.figure(figsize=(10,10))
sns.heatmap(df2.corr(method='spearman'), square=True,annot=True)
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_10980\2516434410.py:3: FutureWarning: The default va
lue of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fal
se. Select only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(df2.corr(method='spearman'), square=True,annot=True)
```

Out[223]: <AxesSubplot: >



The variables 'Quarterly Rating' and 'Total Business Value' are highly correlated (0.9), we will ignore one of them - Quarterly rating as we are also considering it in RateIncrease variable.

Grade and Designation are also correlated (0.76), and we would ignore Grade variable.

Churn is negatively correlated with all variables, except for gender (0-males, 1-females) so females had more churn rate.

In [224…

```
df2.corr(method='spearman')['Churn']
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_10980\118079.py:1: FutureWarning: The default value
of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False.
Select only valid columns or specify the value of numeric_only to silence this warning.
  df2.corr(method='spearman')['Churn']
```

Out[224]:

```
Driver_ID            0.029218
Age                 -0.056165
Gender               0.009552
Education           -0.007874
Income_mean         -0.200731
Designation         -0.129816
Grade               -0.208645
Total_Biz_Value     -0.292862
Quarterly_Rating    -0.339183
Churn                1.000000
RateIncrease        -0.405072
Income_Increase     -0.176845
Name: Churn, dtype: float64
```

In [225…

```
df2.drop(['Grade','Quarterly_Rating','Driver_ID'],axis=1,inplace=True) # driver ID is not relate
```

In [226…

```
df2['LastDate'] = pd.to_datetime(df2['LastDate']).dt.year
```

In [227…

```
df2.describe() # numerical features
```

Out[227]:

	Age	Gender	Education	Income_mean	Designation	Total_Biz_Value	Churn	RateIncreas
count	2381.000000	2381.000000	2381.00000	2381.000000	2381.000000	2.381000e+03	2381.000000	2381.0000C
mean	33.089038	0.410752	1.00756	59232.460484	1.820244	4.586742e+06	0.678706	0.15035
std	5.839201	0.492074	0.81629	28298.214012	0.841433	9.127115e+06	0.467071	0.35749
min	21.000000	0.000000	0.00000	10747.000000	1.000000	-1.385530e+06	0.000000	0.0000C
25%	29.000000	0.000000	0.00000	39104.000000	1.000000	0.000000e+00	0.000000	0.0000C
50%	33.000000	0.000000	1.00000	55285.000000	2.000000	8.176800e+05	1.000000	0.0000C
75%	37.000000	1.000000	2.00000	75835.000000	2.000000	4.173650e+06	1.000000	0.0000C
max	58.000000	1.000000	2.00000	188418.000000	5.000000	9.533106e+07	1.000000	1.0000C

In [228…

```
df2['tenure'] = df2['tenure'].dt.days
df2['tenure'].describe()
```

Out[228]:

```
count    2381.000000
mean      424.540109
std       564.404943
min       -27.000000
25%        91.000000
50%       180.000000
75%       467.000000
max      2801.000000
Name: tenure, dtype: float64
```

In [229…

```
df2['tenure'] = df2['tenure'].clip(lower=0)
df2['tenure'].describe()
```

Out[229]:
```
count    2381.000000
mean      424.852163
std       564.165833
min         0.000000
25%        91.000000
50%       180.000000
75%       467.000000
max      2801.000000
Name: tenure, dtype: float64
```

In [230…
```
df2['City'].describe()
```

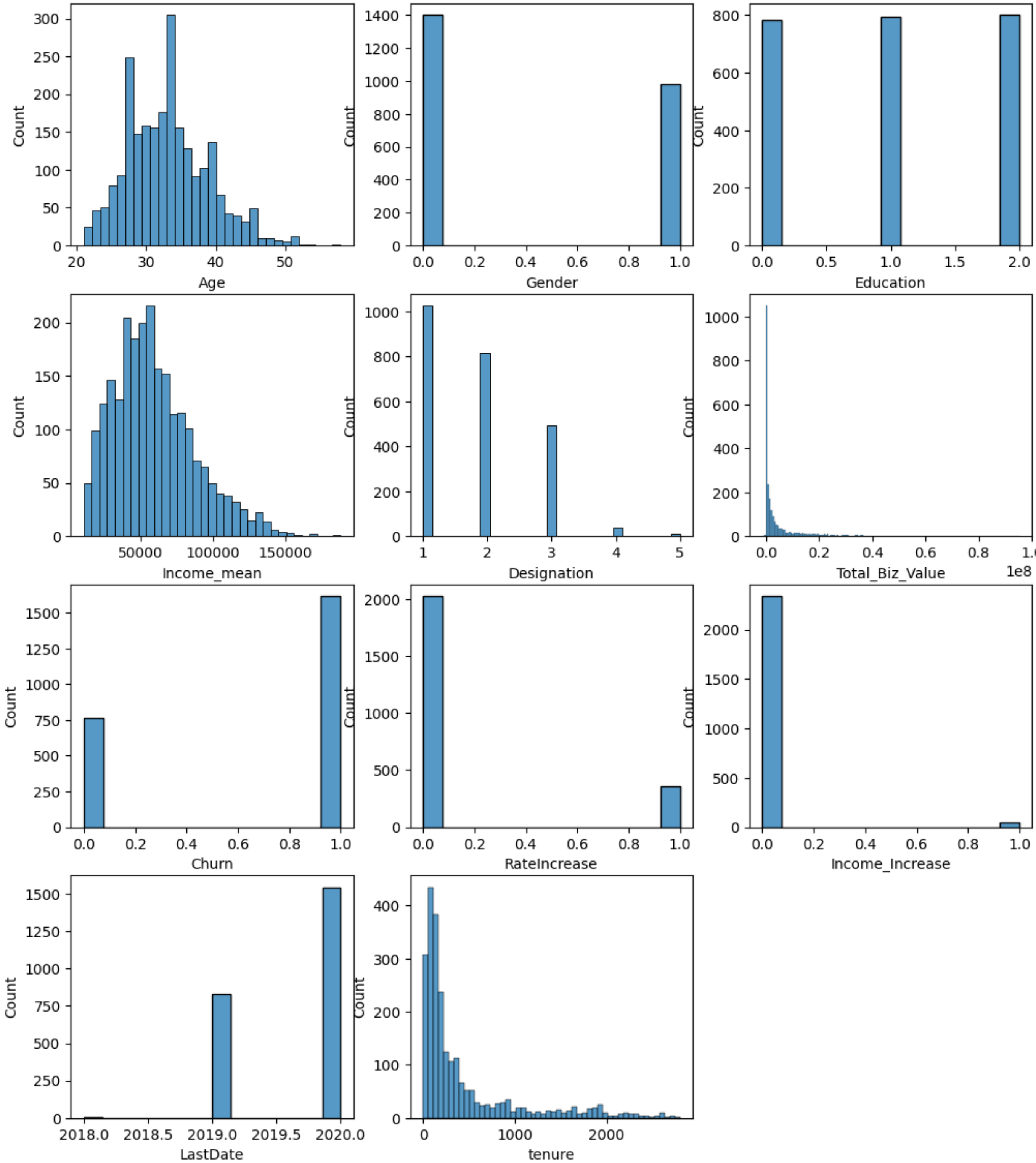
Out[230]:
```
count     2381
unique      29
top        C20
freq       152
Name: City, dtype: object
```

City has 29 unique city codes, more common being C20.

In [231…
```
continuous_cols = df2.columns[(df2.dtypes == 'int64')|(df2.dtypes == 'float64')|(df2.dtypes == ':
continuous_cols
```

Out[231]:
```
Index(['Age', 'Gender', 'Education', 'Income_mean', 'Designation',
       'Total_Biz_Value', 'Churn', 'RateIncrease', 'Income_Increase',
       'LastDate', 'tenure'],
      dtype='object')
```

In [232…
```
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(14)
n = len(continuous_cols)

for i in range(n):
    plt.subplot(4,(n//4)+1,i+1)
    sns.histplot(data=df2, x=continuous_cols[i])
plt.show()
```



The continuous variables look right skewed because of presence of outliers. If we remove the outliers, then we can get bell shaped curves. For the categorical variables:

There are more male drivers (0) than females drivers (1).

Education variable has uniform distribution across all 3 levels.

Most drivers join at the designation levels - 1,2 and 3. Very few join as 4 or 5.

Most drivers churn that means most drivers leave their jobs.

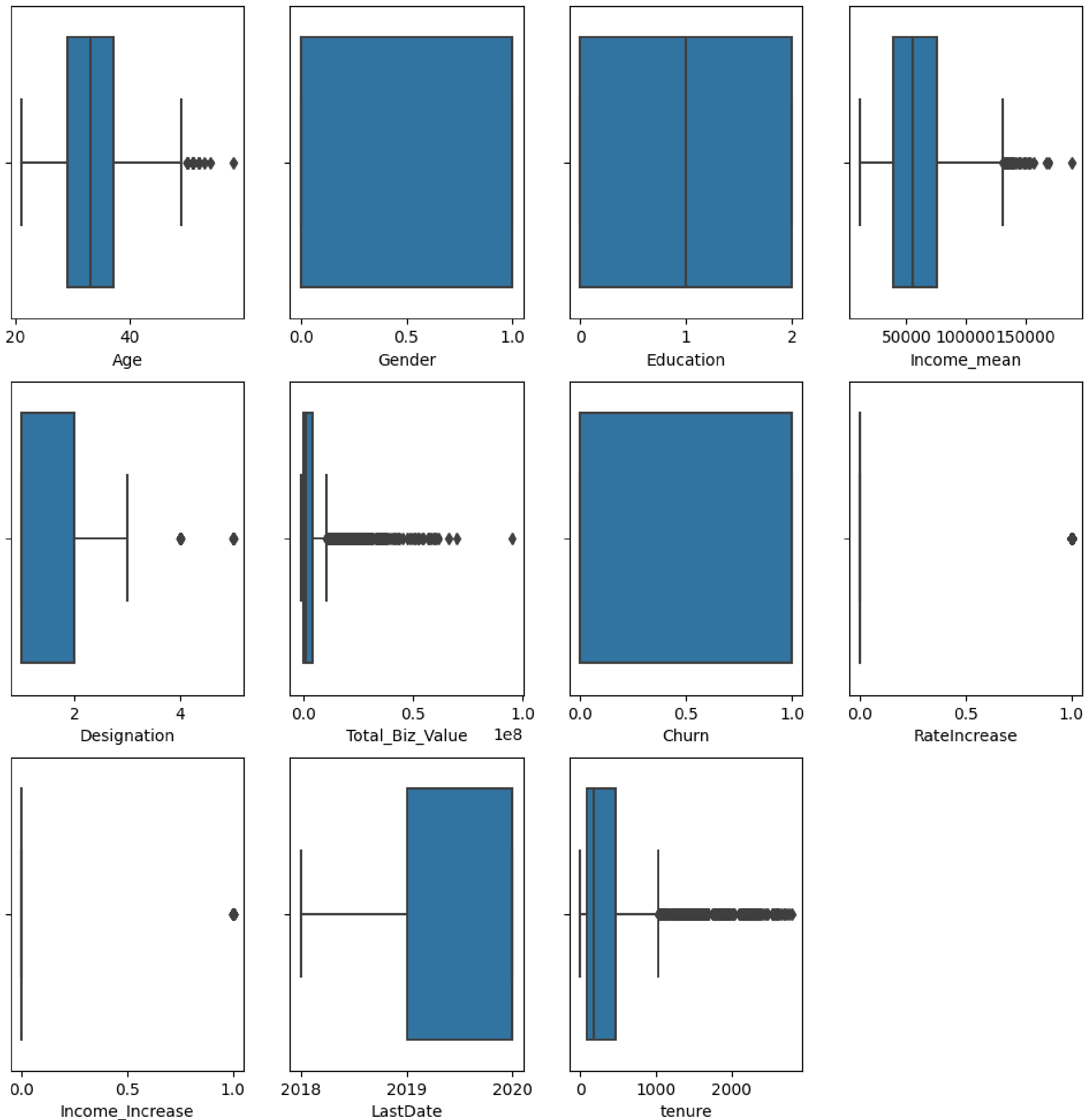
A few drivers had an increase in their quarterly ratings.

Very small number of drivers had an increase in their monthly income compared to when they started.

Most drivers left their jobs in the year 2020 maybe due to the pandemic. Some left in 2019, and very few in 2018.

In [233…

```
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(12)
n = len(continuous_cols)

for i in range(n):
    plt.subplot(3,4,i+1)
    sns.boxplot(data=df2, x=continuous_cols[i])
plt.show()
```



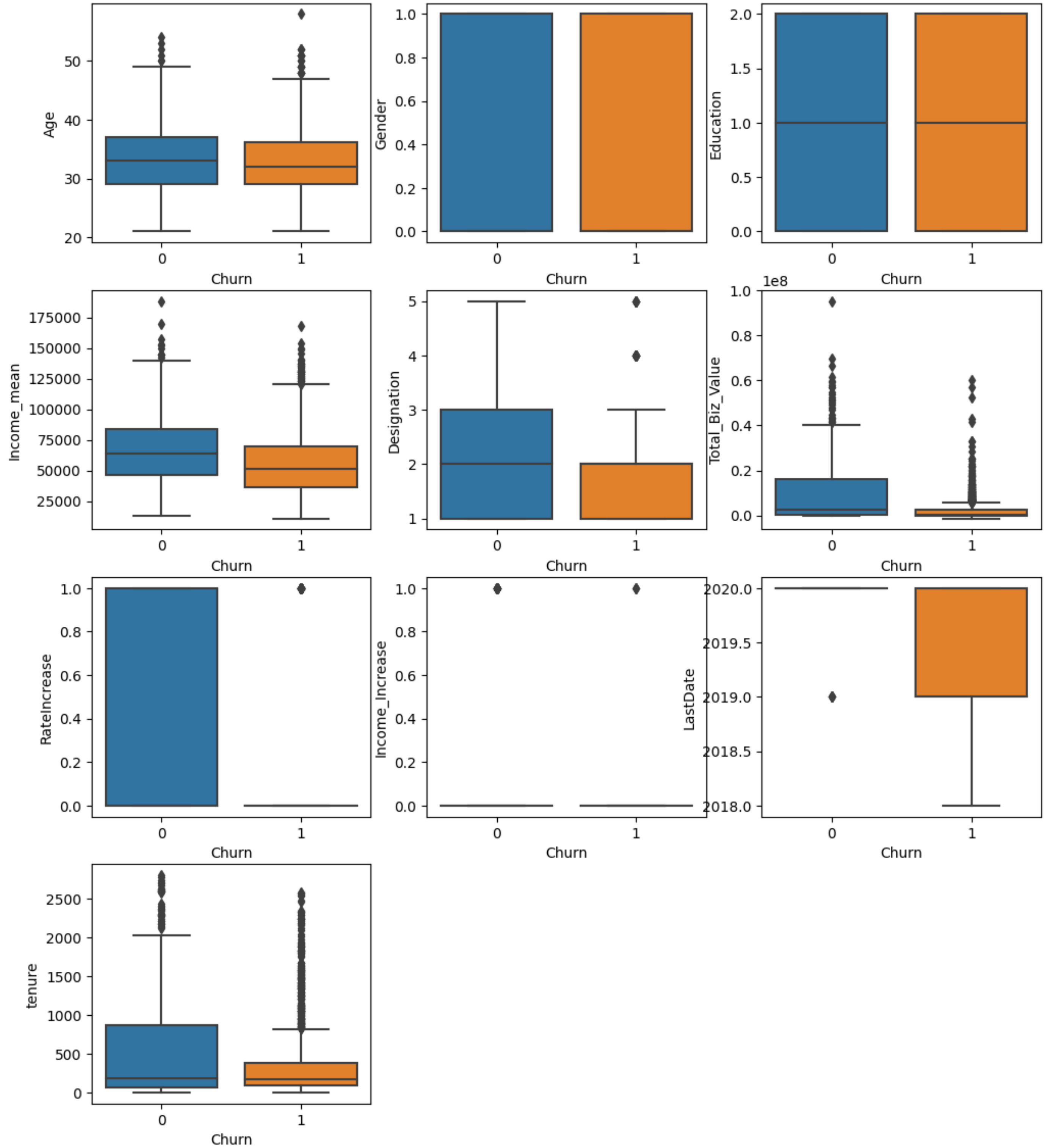
There are many outliers but bagging and boosting algorithms do not assume normality.

Now that we have checked the individual features, let's look at their relationship with each other.

In [234…
```
continuous_cols = continuous_cols.drop(labels = ['Churn'])
continuous_cols
```

Out[234]:
```
Index(['Age', 'Gender', 'Education', 'Income_mean', 'Designation',
       'Total_Biz_Value', 'RateIncrease', 'Income_Increase', 'LastDate',
       'tenure'],
      dtype='object')
```

In [235…
```
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(14)
n = len(continuous_cols)

for i in range(n):
    plt.subplot(4,3,i+1)
    sns.boxplot(data=df2, y=continuous_cols[i],x='Churn')
plt.show()
```



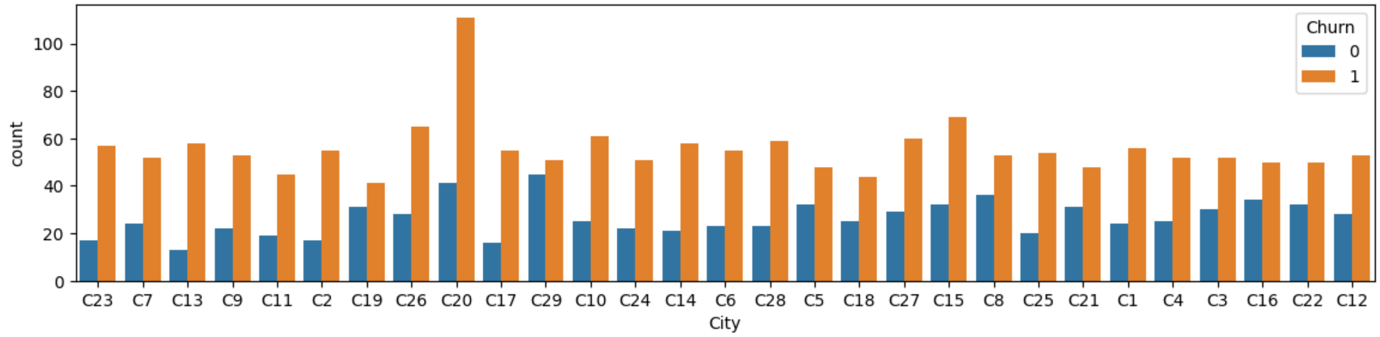
Mean income, designation while joining, and total business value are lower for drivers who churned than those who stayed.

There was no increase in quarterly ratings for those drivers who left, so they might have left due to lower customer evaluation and rating on their skills.

Tenure for those who churned also is lesser than those who stayed.

In [236…

```
f = plt.figure()
f.set_figwidth(14)
f.set_figheight(3)
sns.countplot(data=df2, x='City',hue='Churn')
plt.show()
```



The most number of churns were from city C20. Maybe other cities can be targeted for marketing and calling more drivers to work for their business with enticing ads on job perks. In every city, there were more drivers who churned than those who stayed.

Encoding categorical variables

In [237…

```
X = df2.drop(['Churn'],axis=1)
Y = np.array(df2['Churn']).reshape(-1,1)
print(X.shape, Y.shape)
```

```
(2381, 11) (2381, 1)
```

In [238…

```
# from sklearn.preprocessing import LabelEncoder
# X['City'] = X['City'].apply(LabelEncoder().fit_transform)
# .join(df.select_dtypes(include=['number']))
X['City'] = X['City'].apply(lambda x:x[1:])
X.head()
```

Out[238]:

	Age	Gender	City	Education	Income_mean	Designation	Total_Biz_Value	RateIncrease	Income_Increase	Last
0	28	0	23	2	57387.0	1	1715580	0	0	
1	31	0	7	2	67016.0	2	0	0	0	
2	43	0	13	2	65603.0	2	350000	0	0	
3	29	0	9	0	46368.0	1	120360	0	0	
4	31	1	11	1	78728.0	3	1265000	1	0	

In [239…

```
X.shape
```

Out[239]: (2381, 11)

Splitting data into training and testing dataset

In [240…

```
from sklearn.model_selection import train_test_split
# Create training and test split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_state=4) #10% te
```

In [241…

```
X_train.shape, X_test.shape
```

Out[241]: ((2142, 11), (239, 11))

Class imbalance treatment

In [242…
```
from imblearn.over_sampling import SMOTE
from collections import Counter

smt = SMOTE()
X_sm, y_sm = smt.fit_resample(X_train, y_train)

print('Resampled dataset shape {}'.format(Counter(y_sm)))
```

```
Resampled dataset shape Counter({1: 1448, 0: 1448})
```

In [243…
```
X_sm.shape
```

Out[243]:
```
(2896, 11)
```

In [244…
```
np.unique(y_sm, return_counts=True)
```

Out[244]:
```
(array([0, 1]), array([1448, 1448], dtype=int64))
```

Now, classes are balanced.

Column Standarization

In [245…
```
# Mean centering and Variance scaling (Standard Scaling)
from sklearn.preprocessing import StandardScaler
X_columns = X_sm.columns
scaler = StandardScaler()
X_sm = scaler.fit_transform(X_sm)
X_test = scaler.transform(X_test)
X_sm = pd.DataFrame(X_sm, columns=X_columns)
X_sm.head()
```

Out[245]:

	Age	Gender	City	Education	Income_mean	Designation	Total_Biz_Value	RateIncrease	Income_In
0	-1.086187	-0.740718	0.077179	0.099321	-0.282217	-0.954042	-0.340704	-0.440712	-0.
1	-0.729584	1.350042	-0.048111	-1.151255	-0.848871	0.271193	-0.229959	2.269057	-0.
2	0.161925	1.350042	-1.802171	1.349896	0.253630	1.496429	-0.467264	-0.440712	-0.
3	-1.086187	-0.740718	-0.423981	0.099321	-0.846891	0.271193	-0.558525	-0.440712	-0.
4	0.875131	-0.740718	0.954209	1.349896	0.602511	0.271193	-0.558525	-0.440712	-0.

In [246…
```
X_sm.shape
```

Out[246]:
```
(2896, 11)
```

There are 11 features and 2896 samples, out of which equal number samples are present in each class, so it is a balanced dataset.

Decision tree

Before trying out bagging and boosting, if I only use 1 Decision tree for simplicity, the results are shown below.

In [247…

```
from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(random_state=7)
# Train on training data
print(tree_clf.fit(X_sm,y_sm))

# predict on test data
print(tree_clf.score(X_test,y_test))

from sklearn.model_selection import KFold, cross_validate

kfold = KFold(n_splits=10)
cv_acc_results = cross_validate(tree_clf, X_sm, y_sm, cv = kfold, scoring = 'accuracy', return_tr

print(f"K-Fold Accuracy Mean: Train: {cv_acc_results['train_score'].mean()*100} Validation: {cv_a
print(f"K-Fold Accuracy Std: Train: {cv_acc_results['train_score'].std()*100} Validation: {cv_acc
```

```
DecisionTreeClassifier(random_state=7)
0.8158995815899581
K-Fold Accuracy Mean: Train: 100.0 Validation: 83.08328361770671
K-Fold Accuracy Std: Train: 0.0 Validation: 2.9675605781825722
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: Us
erWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with featu
re names
  warnings.warn(
```

Since the training accuracy was perfect 100% and validation was lower, the model overfitted the training data.

In [248…

```
depths = [5,10,15,20,25,30]

for depth in depths:
    tree_clf = DecisionTreeClassifier(random_state=7, max_depth = depth, min_samples_leaf=10)

    cv_acc_results = cross_validate(tree_clf, X_sm, y_sm, cv = kfold, scoring = 'accuracy', retu

    print(f"K-Fold for depth:{depth} Accuracy Mean: Train: {cv_acc_results['train_score'].mean()
    print(f"K-Fold for depth: {depth} Accuracy Std: Train: {cv_acc_results['train_score'].std()*
    print('***************')
```

```
K-Fold for depth:5 Accuracy Mean: Train: 81.54149007292192 Validation: 75.96014795370482
K-Fold for depth: 5 Accuracy Std: Train: 0.8696823884369445 Validation: 5.90796009545866
***************
K-Fold for depth:10 Accuracy Mean: Train: 87.88752814681295 Validation: 82.18553871853001
K-Fold for depth: 10 Accuracy Std: Train: 0.5852748946584958 Validation: 3.965726101645297
***************
K-Fold for depth:15 Accuracy Mean: Train: 89.43757891337478 Validation: 83.04748836654336
K-Fold for depth: 15 Accuracy Std: Train: 0.32560917664820627 Validation: 3.2126962102824788
***************
K-Fold for depth:20 Accuracy Mean: Train: 89.52966377493028 Validation: 83.04748836654336
K-Fold for depth: 20 Accuracy Std: Train: 0.3075208861801741 Validation: 3.2126962102824788
***************
K-Fold for depth:25 Accuracy Mean: Train: 89.52966377493028 Validation: 83.04748836654336
K-Fold for depth: 25 Accuracy Std: Train: 0.3075208861801741 Validation: 3.2126962102824788
***************
K-Fold for depth:30 Accuracy Mean: Train: 89.52966377493028 Validation: 83.04748836654336
K-Fold for depth: 30 Accuracy Std: Train: 0.3075208861801741 Validation: 3.2126962102824788
***************
```

Since the train and validation results are closer at max depth 10, let's see smaller depths.

In [249…

```python
depths = [5, 6, 7, 8, 9, 10]

for depth in depths:
    tree_clf = DecisionTreeClassifier(random_state=7, max_depth = depth, min_samples_leaf=10)

    cv_acc_results = cross_validate(tree_clf, X_sm, y_sm, cv = kfold, scoring = 'accuracy', retu

    print(f"K-Fold for depth:{depth} Accuracy Mean: Train: {cv_acc_results['train_score'].mean()
    print(f"K-Fold for depth: {depth} Accuracy Std: Train: {cv_acc_results['train_score'].std()*
    print('***************')
```

```
K-Fold for depth:5 Accuracy Mean: Train: 81.54149007292192 Validation: 75.96014795370482
K-Fold for depth: 5 Accuracy Std: Train: 0.8696823884369445 Validation: 5.90796009545866
***************
K-Fold for depth:6 Accuracy Mean: Train: 83.27949634389495 Validation: 80.11144254862188
K-Fold for depth: 6 Accuracy Std: Train: 1.0992257618270365 Validation: 4.828598418085984
***************
K-Fold for depth:7 Accuracy Mean: Train: 84.59173174766207 Validation: 79.76267748478702
K-Fold for depth: 7 Accuracy Std: Train: 0.6537862995420123 Validation: 3.108494384289223
***************
K-Fold for depth:8 Accuracy Mean: Train: 85.87322607738008 Validation: 79.69621763512707
K-Fold for depth: 8 Accuracy Std: Train: 0.8004141381513422 Validation: 4.5586522170021615
***************
K-Fold for depth:9 Accuracy Mean: Train: 86.99356417178969 Validation: 81.45794057988306
K-Fold for depth: 9 Accuracy Std: Train: 0.46716634486531317 Validation: 2.9167272279877285
***************
K-Fold for depth:10 Accuracy Mean: Train: 87.88752814681295 Validation: 82.18553871853001
K-Fold for depth: 10 Accuracy Std: Train: 0.5852748946584958 Validation: 3.965726101645297
***************
```

Max depth 6 is best one.

Ensemble technique - Random Forest Bagging algorithm

We need to use a non-parametric model like Decision Tree to fit a non-normal dataset. Bagging algorithms like Random Forest use an aggregation of decision trees with low bias and high variance, to reduce the variance and overfitting.

In [250…

```
# Defining Parameters
params = {
          'n_estimators' : [20,50,100,200,300],
          'max_depth' : [6],
          'criterion' : ['gini'],
          'bootstrap' : [True],
          'min_samples_leaf' : [5,10]
         }
```

In [251…

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

tuning_function = GridSearchCV(estimator = RandomForestClassifier(),
                               param_grid = params,
                               scoring = 'accuracy',
                               cv = 3,
                               n_jobs=-1
                               )
# Now we will fit all combinations, this will take some time to run. (5-6 mins)
tuning_function.fit(X_sm, y_sm)

parameters = tuning_function.best_params_
score = tuning_function.best_score_
print(parameters)
print(score)
```

```
{'bootstrap': True, 'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 10, 'n_estimators':
50}
0.8418559163546773
```

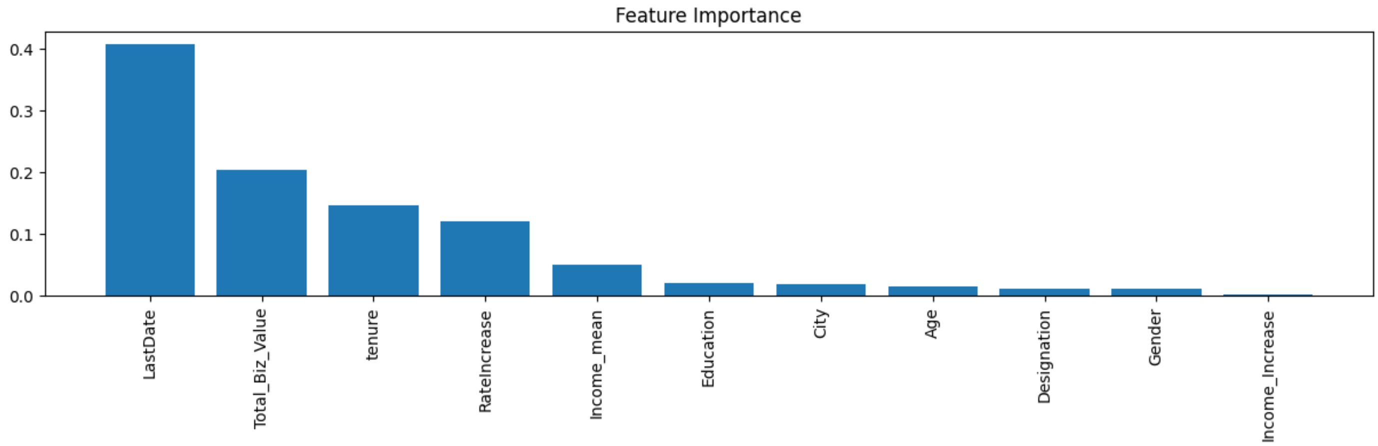
In [252…

```
from sklearn.model_selection import KFold, cross_validate

tree_clf = RandomForestClassifier(random_state=7, max_depth=6, n_estimators=50, min_samples_leaf
kfold = KFold(n_splits=3)
cv_acc_results = cross_validate(tree_clf, X_sm, y_sm, cv = kfold, scoring = 'accuracy', return_t

print(f"K-Fold Accuracy Mean: Train: {cv_acc_results['train_score'].mean()*100} Validation: {cv_
print(f"K-Fold Accuracy Std: Train: {cv_acc_results['train_score'].std()*100} Validation: {cv_ac
```

```
K-Fold Accuracy Mean: Train: 85.99809131800127 Validation: 71.44162313119286
K-Fold Accuracy Std: Train: 0.6843485430972308 Validation: 5.670700072268422
```

In [254…

```
# Feature importance
tree_clf = RandomForestClassifier(random_state=7, max_depth=6, n_estimators=50, min_samples_leaf
tree_clf.fit(X_sm, y_sm)
importances = tree_clf.feature_importances_
indices = np.argsort(importances)[::-1] # Sort feature importances in descending order
names = [X_sm.columns[i] for i in indices] # Rearrange feature names so they match the sorted fe
plt.figure(figsize=(15, 3)) # Create plot
plt.title("Feature Importance") # Create plot title
plt.bar(range(X_sm.shape[1]), importances[indices]) # Add bars
plt.xticks(range(X_sm.shape[1]), names, rotation=90) # Add feature names as x-axis labels
plt.show() # Show plot
```



According to the RF bagging algorithm, the churn outcome was most affected by the Last working date, and then the other important features were Total Business value, tenure duration, mean income, city, age, increase in quarterly ratings, education, starting designation and gender.

As per Spearman's rank correlation coefficients looked at earlier, the churn is negatively correlated with all variables, except for gender (0-males, 1-females) so females had more churn rate. Otherwise churn increased when rest all variables decreased in value.

- Age -0.056165
- Gender 0.009552
- Education -0.007874
- Income_mean -0.200731
- Designation -0.129816
- Grade -0.208645
- Total_Biz_Value -0.292862
- Quarterly_Rating -0.339183
- Churn 1.000000
- RateIncrease -0.405072
- Income_Increase -0.176845

In [255…

```
y_pred = tree_clf.predict(X_test)
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

In [256…

```
len(X_test)
```

Out[256]:

```
239
```

In [257…

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, plot_confusion_matr

testscore = accuracy_score(y_test,y_pred)
print('Test accuracy: ',testscore)

cm = confusion_matrix(y_test, y_pred)

cm_df = pd.DataFrame(cm,index = np.unique(y_test), columns = np.unique(y_test) )
```

```
cm_df.head()
```

```
Test accuracy:  0.8702928870292888
```

Out[257]:

	0	1
0	64	7
1	24	144

In [258…

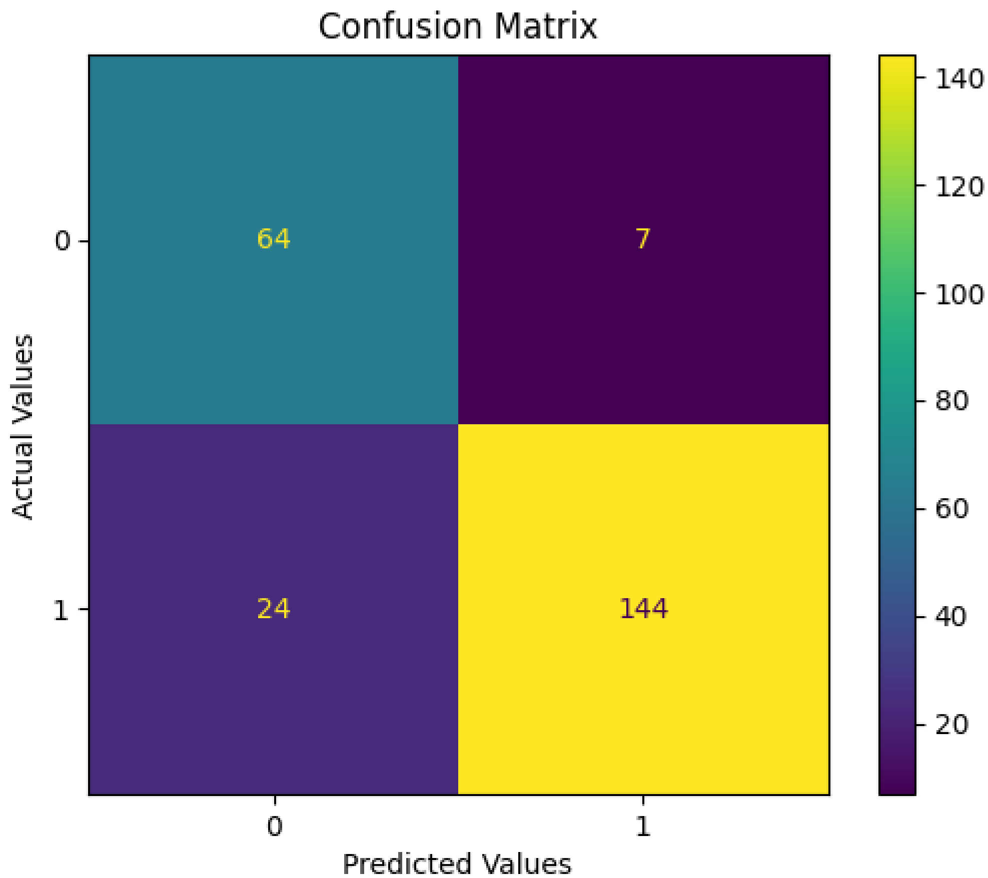
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.73      0.90      0.81        71
           1       0.95      0.86      0.90       168

    accuracy                           0.87       239
   macro avg       0.84      0.88      0.85       239
weighted avg       0.89      0.87      0.87       239
```

In [259…

```
#Plotting the confusion matrix
plt.figure(figsize=(1,1))
plot_confusion_matrix(tree_clf,X_test,y_test)
#sns.heatmap(cm_df, annot=True,cmap='coolwarm')
plt.title('Confusion Matrix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\deprecati
on.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_
matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: Confusion
MatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: Us
erWarning: X does not have valid feature names, but RandomForestClassifier was fitted with featu
re names
  warnings.warn(
<Figure size 100x100 with 0 Axes>
```



In [260…

```
from sklearn.metrics import f1_score
print("Precision score is :",precision_score(y_test,y_pred))
print("Recall score is :",recall_score(y_test,y_pred))
print("F1 score is :",f1_score(y_test,y_pred))
```

```
Precision score is : 0.9536423841059603
Recall score is : 0.8571428571428571
F1 score is : 0.90282131661442
```

Precision score was 0.95 and there were very few false positives, drivers who stayed but were predicted to churn. They don't need much attention and can cause waste of resources but thankfully there were very few in number.

Recall score- 0.86 which means that there were few false negatives that caused financial losses as they churned but were predicted to stay, and this can be improved to decrease further losses.

ROC (Receiver Operating Characteristic curve) and AUC (Area Under Curve)

In [261…

```
y_proba = tree_clf.predict_proba(X_test)
y_proba.shape, y_test.shape
```

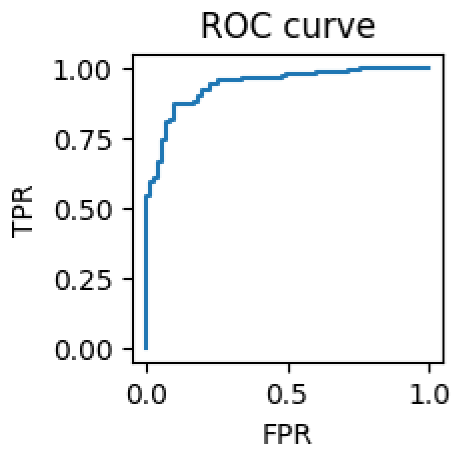
```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

Out[261]: ((239, 2), (239, 1))

In [262…

```
from sklearn.metrics import roc_curve, roc_auc_score
fpr, tpr, thr = roc_curve(y_test, y_proba[:,1])
plt.figure(figsize=(2,2))
plt.plot(fpr,tpr)
plt.title('ROC curve')
```

```
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



There are more true positives than False positives, hence the ROC curve has more area under curve than 0.5.

In [263…

```
roc_auc_score(y_test,y_proba[:,1])
```

Out[263]: 0.938044936284373

0.93 is a good area under curve score.

Ensemble Boosting algorithm - XGBoost

Boosting uses a series of decision stumps that have high bias and low variance (underfitted models), to add their contribution in a way that each reduces the error residual of the previous model and reduces bias and underfitting.

In [68]:

```
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import StratifiedKFold

params = {
        'learning_rate': [0.1, 0.5, 0.8],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5]
        }
xgb = XGBClassifier(n_estimators=100, objective='multi:softmax', num_class=20, silent=True)

folds = 3

skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 1001)

random_search = RandomizedSearchCV(xgb, param_distributions=params, n_iter=10, scoring='accuracy
                                   cv=skf.split(X_sm,y_sm), verbose=3, random_state=1001 )


# start = dt.datetime.now()
random_search.fit(X_sm, y_sm)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[18:54:01] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-0fc7796c
793e6356f-1/xgboost/xgboost-ci-windows/src/learner.cc:767:
Parameters: { "silent" } are not used.
```

Out[68]:

```
RandomizedSearchCV
estimator: XGBClassifier
XGBClassifier
```

In [69]:

```
print('\n Best hyperparameters:')
print(random_search.best_params_)
```

```
 Best hyperparameters:
{'subsample': 0.8, 'max_depth': 5, 'learning_rate': 0.5, 'colsample_bytree': 1.0}
```

In [70]:

```
best_xgb = XGBClassifier(n_estimators=100, objective='multi:softmax', num_class=20,
                         subsample=1.0, max_depth=4, learning_rate=0.8, colsample_bytree=0.6)
best_xgb.fit(X_sm, y_sm)
```

Out[70]:

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.6, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.8, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [71]:

```
y_pred = best_xgb.predict(X_test)
y_pred_train = best_xgb.predict(X_sm)

testscore = accuracy_score(y_test,y_pred)
trscore = accuracy_score(y_sm,y_pred_train)
print('Train accuracy: ',trscore)
print('Test accuracy: ',testscore)
cm = confusion_matrix(y_test, y_pred)

cm_df = pd.DataFrame(cm,index = np.unique(y_test), columns = np.unique(y_test) )

cm_df.head()
```

```
Train accuracy:  0.9996546961325967
Test accuracy:  0.8786610878661087
```

Out[71]:

	0	1
0	61	10
1	19	149

True negatives - 31 (drivers who stayed as predicted)

False positives - 6 (drivers who stayed but were predicted to churn)

False negatives - 8 (drivers who churned but were predicted to stay) - need the most attention as can cause huge financial losses

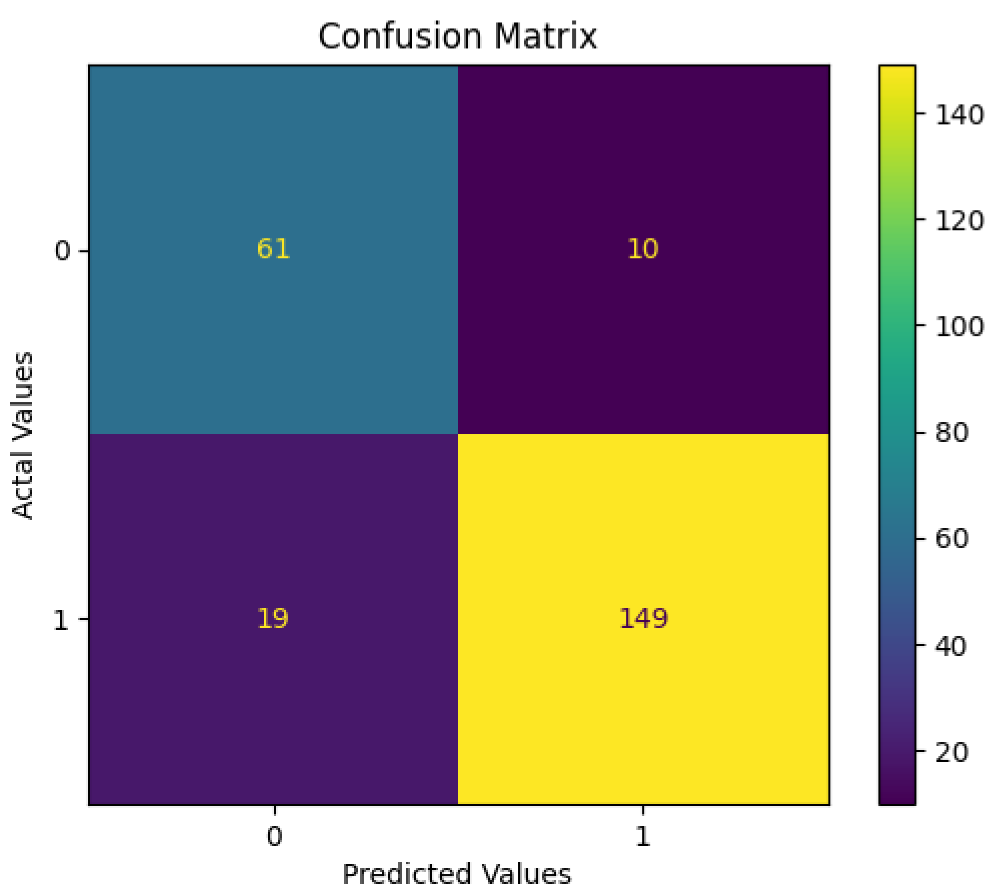
True positives - 75 (drivers who churned as predicted)

In [72]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.86      0.81        71
           1       0.94      0.89      0.91       168

    accuracy                           0.88       239
   macro avg       0.85      0.87      0.86       239
weighted avg       0.89      0.88      0.88       239
```

In [73]:
```
#Plotting the confusion matrix
plt.figure(figsize=(1,1))
plot_confusion_matrix(best_xgb,X_test,y_test)
#sns.heatmap(cm_df, annot=True,cmap='coolwarm')
plt.title('Confusion Matrix')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\utils\deprecati
on.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_
matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: Confusion
MatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
<Figure size 100x100 with 0 Axes>
```



In [74]:
```
from sklearn.metrics import f1_score
print("Precision score is :",precision_score(y_test,y_pred))
print("Recall score is :",recall_score(y_test,y_pred))
print("F1 score is :",f1_score(y_test,y_pred))
```

```
Precision score is : 0.9371069182389937
Recall score is : 0.8869047619047619
F1 score is : 0.9113149847094801
```

Precision is 0.94 which is good as the number of false positives (7) are low, the drivers who stayed but were predicted to churn.

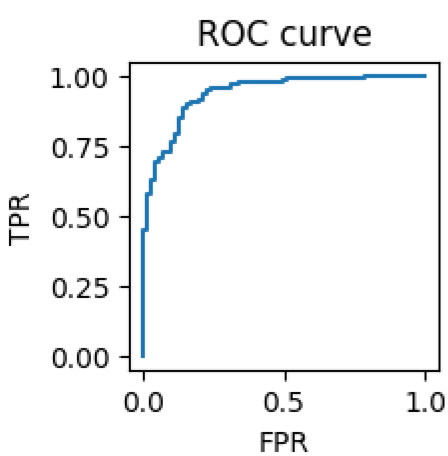
Recall-score: 0.89. Recall is high because of low number of False negatives. This is good as we are not losing too many drivers (9) to churn who were predicted to stay. But we can still improve recall score by encouraging drivers to stay and provide enticing perks and reduce financial losses.

ROC (Receiver Operating Characteristic curve) and AUC (Area Under Curve)

In [75]:
```
# from sklearn.linear_model import
y_proba = best_xgb.predict_proba(X_test)
y_proba.shape, y_test.shape
```

Out[75]:
```
((239, 20), (239, 1))
```

In [76]:
```
from sklearn.metrics import roc_curve, roc_auc_score
fpr, tpr, thr = roc_curve(y_test, y_proba[:,1])
plt.figure(figsize=(2,2))
plt.plot(fpr,tpr)
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



There are more true positives than False positives, hence the ROC curve has more area under curve than 0.5.

In [77]:
```
roc_auc_score(y_test,y_proba[:,1])
```

Out[77]:
```
0.937709590878605
```

0.94 is a good area under curve.

Business insights

There are more male drivers (0) than females drivers (1).

Education variable has uniform distribution across all 3 levels.

Most drivers join at the designation levels - 1,2 and 3. Very few join as 4 or 5.

Most drivers churn that means most drivers leave their jobs.

A few drivers had an increase in their quarterly ratings.

Very small number of drivers had an increase in their monthly income compared to when they started.

Most drivers left their jobs in the year 2020 maybe due to the pandemic. Some left in 2019, and very few in 2018.

Mean income, designation while joining, and total business value are lower for drivers who churned than those who stayed.

There was no increase in quarterly ratings for those drivers who left, so they might have left due to lower customer evaluation and rating on their skills.

Tenure for those who churned also is lesser than those who stayed.

Most common city for drivers to live or work at was C20. In every city, there were more drivers who churned than those who stayed.

According to XGBoost algorithm: test accuracy 87.9%. Precision is 0.94 and Recall score 0.89.

High Precision is good as the number of false positives (7) are low, the drivers who stayed but were predicted to churn. Recall-score: 0.9. Recall is high because of low number of False negatives. This is good as we are not losing too many drivers (9) to churn who were predicted to stay. But we can still improve recall score by encouraging drivers to stay and provide enticing perks and reduce financial losses.

According to the RF bagging algorithm, test accuracy: 87.03%. Precision score was 0.95 and Recall score 0.86.

Both did not do very well because of small dataset (2896 rows).

According to the RF bagging algorithm, the churn outcome was most affected by the Last working date, and then the other important features were Total Business value, tenure duration, mean income, city, age, increase in quarterly ratings, education, starting designation and gender.

As per Spearman's rank correlation coefficients looked at earlier, the churn is negatively correlated with all variables, except for gender (0-males, 1-females) so females had more churn rate. Otherwise churn increased when rest all variables decreased in value.

Recommendations

Since we only have data from 2018, 2019 and 2020, out of which 2020 was the time of pandemic and cannot be used for general analysis, we need to collect more data in order to improve our analysis and reduce errors.

Since there are more males than females, there is an opportunity for the business to increase the number of drivers by encouraging more female drivers to take up jobs.

Drivers who left were those who did not have any increase in their ratings so maybe they can be encouraged to improve their driving skills and ratings by being given tips on customer satisfaction, communication with customers and safe driving tips to increase their ratings and therefore increase their sense of job satisfaction.

Most common city for drivers to live or work at was C20. Maybe other cities can be targeted for marketing and calling more drivers to work for their business with enticing ads on job perks.

It is important to retain drivers by offering them more job perks- such as designated resting areas especially for female drivers, reducing total business value loss by discouraging cancellation of rides and improve quarterly ratings for drivers. A survey can be conducted among drivers to understand their needs.

In [ ]:
```
''
```