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 2018, 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 [1]:
```
# useful imports
import numpy as np, seaborn as sns, pandas as pd, matplotlib.pyplot as plt
df = pd.read_csv('driver.csv')
df.head(10)
```

Out[1]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	03/11/19
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN
5	5	12/01/19	4	43.0	0.0	C13	2	65603	12/07/19	NaN
6	6	01/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN
7	7	02/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN
8	8	03/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN
9	9	04/01/20	4	43.0	0.0	C13	2	65603	12/07/19	27/04/20

In [2]:

```
df.shape
```

Out[2]:

```
(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 [3]:

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

In [4]:

```
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 [5]:

```
##Converting all date features to datetime type
df['MMM-YY'] = pd.to_datetime(df['MMM-YY'])
df['Dateofjoining'] = pd.to_datetime(df['Dateofjoining'])
df['LastWorkingDate'] = pd.to_datetime(df['LastWorkingDate'])
```

In [6]:
```
df.columns[(df.dtypes == "float64") | (df.dtypes == "int64")]
```

Out[6]:
```
Index(['Driver_ID', 'Age', 'Gender', 'Education_Level', 'Income',
       'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating'],
      dtype='object')
```

The categorical columns are:

In [7]:
```
df.columns[df.dtypes =="object"]
```

Out[7]:
```
Index(['City'], 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 [8]:
```
df.isna().sum()*100/len(df)
```

Out[8]:
```
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 [9]:
```
df.duplicated().sum()
```

Out[9]:
```
0
```

Handling null values

In [10]:
```
nums = df.select_dtypes(np.number)
nums.head()
```

Out[10]:

	Driver_ID	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating
0	1	28.0	0.0	2	57387	1	1	2381060	2
1	1	28.0	0.0	2	57387	1	1	-665480	2
2	1	28.0	0.0	2	57387	1	1	0	2
3	2	31.0	0.0	2	67016	2	2	0	1
4	2	31.0	0.0	2	67016	2	2	0	1

In [11]:
```
nums.drop('Driver_ID',axis=1,inplace=True)
columns=nums.columns
```

In [12]:
```
from sklearn.impute import KNNImputer
imputer = KNNImputer(missing_values=np.nan, n_neighbors=5, weights='uniform', metric='nan_euclid
imputer = imputer.fit(nums)
cont = imputer.transform(nums)
cont[:5]
```

Out[12]:
```
array([[ 2.80000e+01,  0.00000e+00,  2.00000e+00,  5.73870e+04,
         1.00000e+00,  1.00000e+00,  2.38106e+06,  2.00000e+00],
       [ 2.80000e+01,  0.00000e+00,  2.00000e+00,  5.73870e+04,
         1.00000e+00,  1.00000e+00, -6.65480e+05,  2.00000e+00],
       [ 2.80000e+01,  0.00000e+00,  2.00000e+00,  5.73870e+04,
         1.00000e+00,  1.00000e+00,  0.00000e+00,  2.00000e+00],
       [ 3.10000e+01,  0.00000e+00,  2.00000e+00,  6.70160e+04,
         2.00000e+00,  2.00000e+00,  0.00000e+00,  1.00000e+00],
       [ 3.10000e+01,  0.00000e+00,  2.00000e+00,  6.70160e+04,
         2.00000e+00,  2.00000e+00,  0.00000e+00,  1.00000e+00]])
```

In [13]:
```
df_new=pd.DataFrame(cont)
df_new.columns=columns
```

In [14]:
```
remaining_columns=list(set(df.columns).difference(set(columns)))
data=pd.concat([df_new, df[remaining_columns]],axis=1)
```

In [15]:
```
data.head()
```

Out[15]:

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	Dateofjoining	City	MMM-YY
0	28.0	0.0	2.0	57387.0	1.0	1.0	2381060.0	2.0	2018-12-24	C23	2019-01-01
1	28.0	0.0	2.0	57387.0	1.0	1.0	-665480.0	2.0	2018-12-24	C23	2019-02-01
2	28.0	0.0	2.0	57387.0	1.0	1.0	0.0	2.0	2018-12-24	C23	2019-03-01
3	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2020-11-06	C7	2020-11-01
4	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2020-11-06	C7	2020-12-01

In [16]:
```
data.isnull().sum()
```

```
Out[16]: Age                         0
         Gender                      0
         Education_Level             0
         Income                      0
         Joining Designation         0
         Grade                       0
         Total Business Value        0
         Quarterly Rating            0
         Dateofjoining               0
         City                        0
         MMM-YY                      0
         LastWorkingDate         17488
         Driver_ID                   0
         dtype: int64
```

Data aggregation

```
In [17]: # grouping based on Driver ID
agg_df = data.groupby(['Driver_ID']).agg({'MMM-YY':'last','Age':'last','Gender':'first','City':'
        'Education_Level':'first','Income':['first','last','mean'],'Dateofjoining':'first','LastW
         ,'Joining Designation':'first','Grade':'first','Total Business Value':'sum',
                                        'Quarterly Rating':['first','last','mean']})
df2 = agg_df.reset_index()
df2.head()
```

Out[17]:

	Driver_ID	MMM-YY	Age	Gender	City	Education_Level	Income			Dateofjoining	LastWorkingD	
		last	last	first	first	first	first	last	mean	first	last	a
0	1	2019-03-01	28.0	0.0	C23	2.0	57387.0	57387.0	57387.0	2018-12-24	2019-03-11	T
1	2	2020-12-01	31.0	0.0	C7	2.0	67016.0	67016.0	67016.0	2020-11-06	NaT	Fa
2	4	2020-04-01	43.0	0.0	C13	2.0	65603.0	65603.0	65603.0	2019-12-07	2020-04-27	T
3	5	2019-03-01	29.0	0.0	C9	0.0	46368.0	46368.0	46368.0	2019-01-09	2019-03-07	T
4	6	2020-12-01	31.0	1.0	C11	1.0	78728.0	78728.0	78728.0	2020-07-31	NaT	Fa

```
In [23]: df2.columns = ['_'.join(col) for col in df2.columns.values]
```

```
In [24]: df2.columns
```

```
Out[24]: 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 [25]: df2.shape
```

Out[25]: (2381, 18)

Aggregation reduced the number of rows from 19104 to 2381.

How many drivers do we have?

2381 drivers.

In [26]:
```
df2['LastWorkingDate_any']
```

Out[26]:
```
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 [29]:
```
df2['Churn'] = df2['LastWorkingDate_any']
df2['Churn'] = df2['Churn'].astype('int')
df2['Age_first'] = df2['Age_first'].astype('int')
df2['Gender_first'] = df2['Gender_first'].astype('int')
```

In [30]:
```
df2['Churn'].value_counts()/len(df2)*100
```

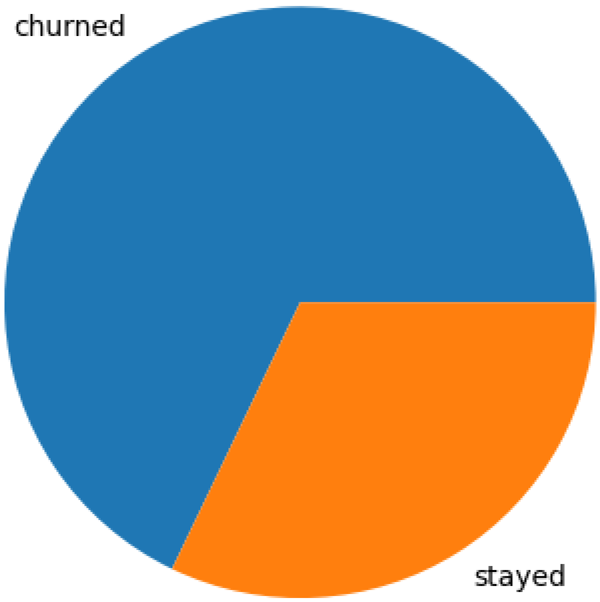
Out[30]:
```
1    67.870643
0    32.129357
Name: Churn, dtype: float64
```

In [31]:
```
df2['Churn'].value_counts()
```

Out[31]:
```
1    1616
0     765
Name: Churn, dtype: int64
```

67.87% (n=1616) drivers churned, whereas 32.1% (n=765) drivers stayed.

In [32]:
```
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 [33]:
```
df2['RateIncrease'] = np.where((df2['Quarterly Rating_last'] > df2['Quarterly Rating_first']),1,(
```

In [34]:
```
df2['Income_Increase'] = np.where((df2['Income_last'] > df2['Income_first']),1,0)
```

In [35]:
```
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 [36]:
```
df2[['MMM-YY_last','LastWorkingDate_last','LastDate']].head()
```

Out[36]:

	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 [37]:
```
df2['tenure'] = pd.to_datetime(df2['LastDate']) - df2['Dateofjoining_first']
```

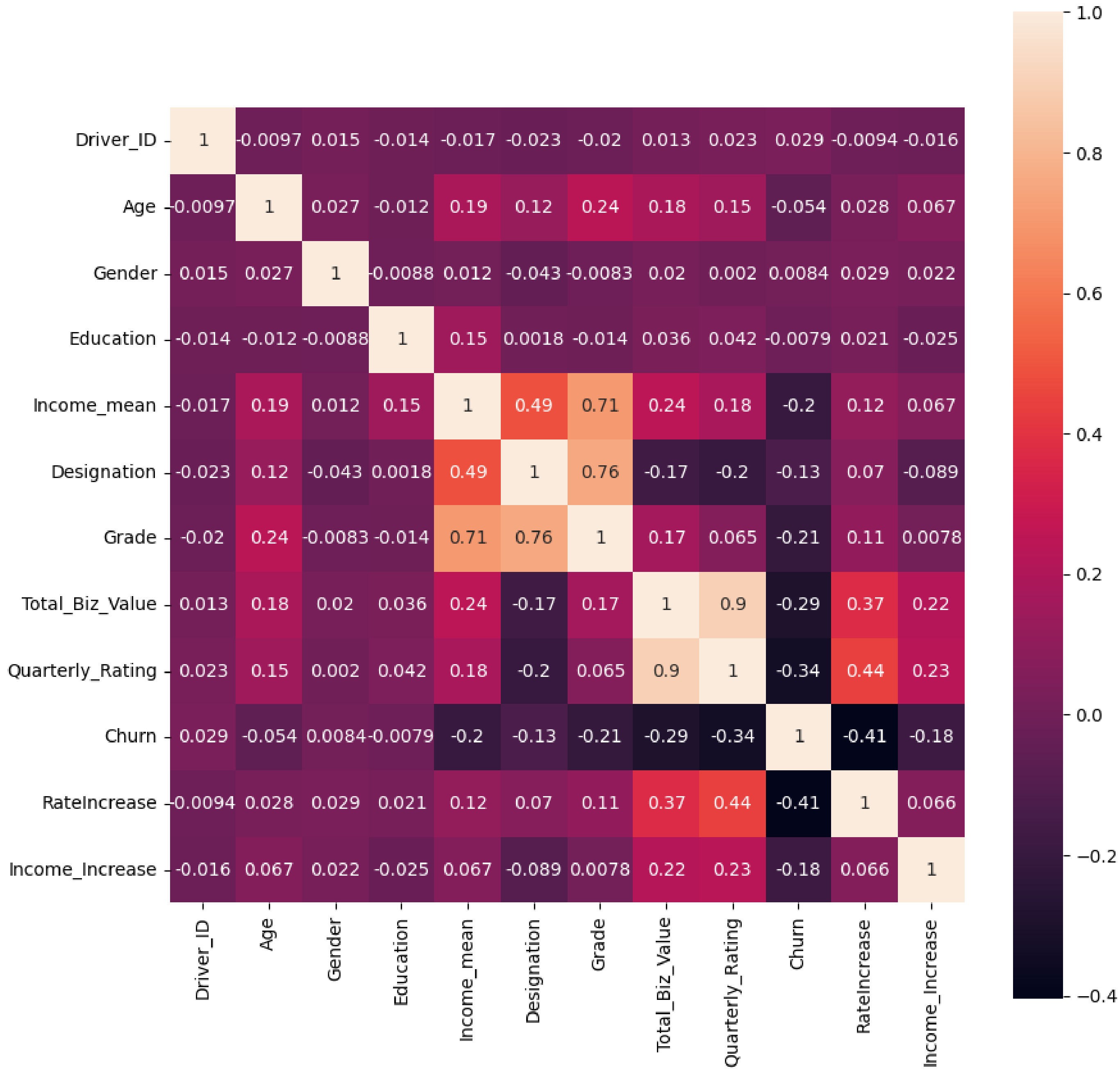
In [38]:
```
df2 = df2.drop(['Income_first','Income_last','Quarterly Rating_first','Quarterly Rating_last','La
         ,'LastWorkingDate_any','MMM-YY_last','Dateofjoining_first'],axis=1)
```

In [39]:
```
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 [40]:
```python
# 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_11516\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[40]: <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.

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

In [41]:
```python
df2.corr(method='spearman')['Churn']
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_11516\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[41]:

```
Driver_ID            0.029218
Age                 -0.053831
Gender               0.008380
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 [42]:

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

In [43]:

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

In [44]:

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

Out[44]:

	Age	Gender	Education	Income_mean	Designation	Grade	Total_Biz_Value	Churn
count	2381.000000	2381.000000	2381.00000	2381.000000	2381.000000	2381.000000	2.381000e+03	2381.00000
mean	33.103738	0.409912	1.00756	59232.460484	1.820244	2.078538	4.586742e+06	0.67870
std	5.835971	0.491920	0.81629	28298.214012	0.841433	0.931321	9.127115e+06	0.46707
min	21.000000	0.000000	0.00000	10747.000000	1.000000	1.000000	-1.385530e+06	0.00000
25%	29.000000	0.000000	0.00000	39104.000000	1.000000	1.000000	0.000000e+00	0.00000
50%	33.000000	0.000000	1.00000	55285.000000	2.000000	2.000000	8.176800e+05	1.00000
75%	37.000000	1.000000	2.00000	75835.000000	2.000000	3.000000	4.173650e+06	1.00000
max	58.000000	1.000000	2.00000	188418.000000	5.000000	5.000000	9.533106e+07	1.00000

In [45]:

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

Out[45]:

```
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 [46]:

```
df2['tenure'] = df2['tenure'].clip(lower=0) # remove all negative values for tenure period.
df2['tenure'].describe()
```

Out[46]:
```
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 [47]:
```
df2['City'].describe()
```

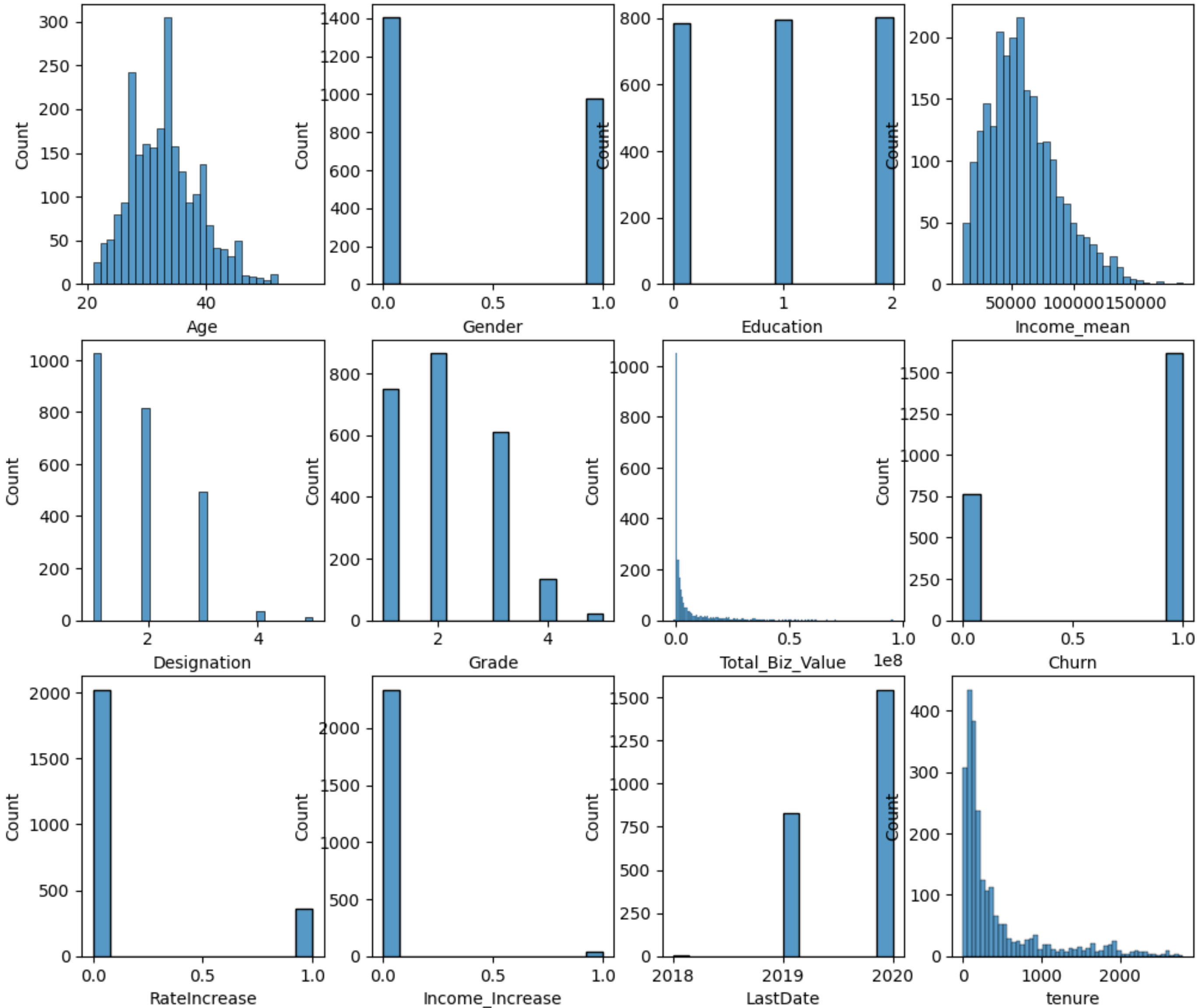
Out[47]:
```
count     2381
unique      29
top        C20
freq       152
Name: City, dtype: object
```

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

In [48]:
```
continuous_cols = df2.columns[(df2.dtypes == 'int64')|(df2.dtypes == 'float64')|(df2.dtypes == ':
continuous_cols
```

Out[48]:
```
Index(['Age', 'Gender', 'Education', 'Income_mean', 'Designation', 'Grade',
       'Total_Biz_Value', 'Churn', 'RateIncrease', 'Income_Increase',
       'LastDate', 'tenure'],
      dtype='object')
```

In [49]:
```
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.

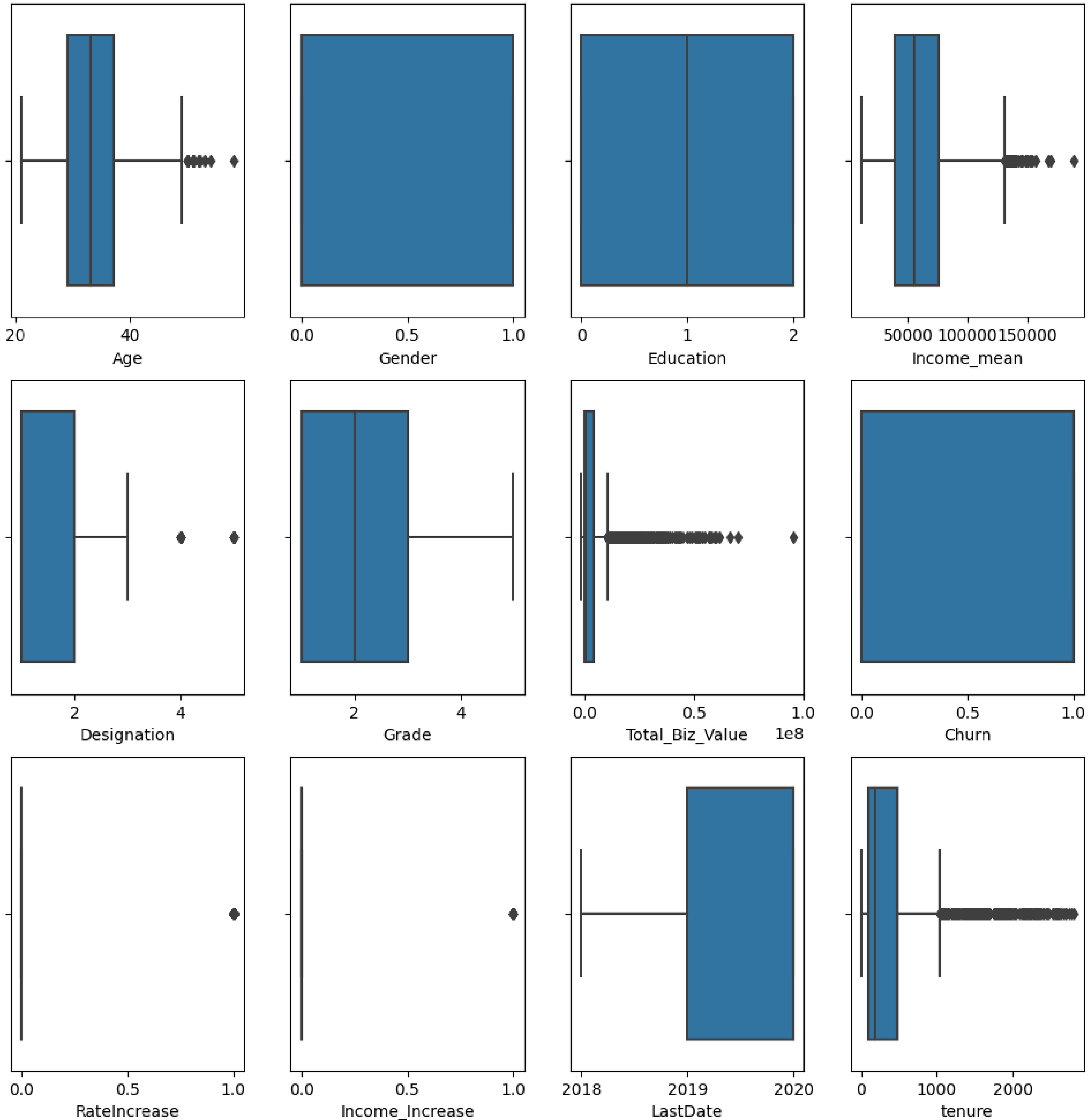
A few drivers had an increase in their quarterly ratings.

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 [50]:
```
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(12)
n = len(continuous_cols)

for i in range(n):
```

```python
    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.

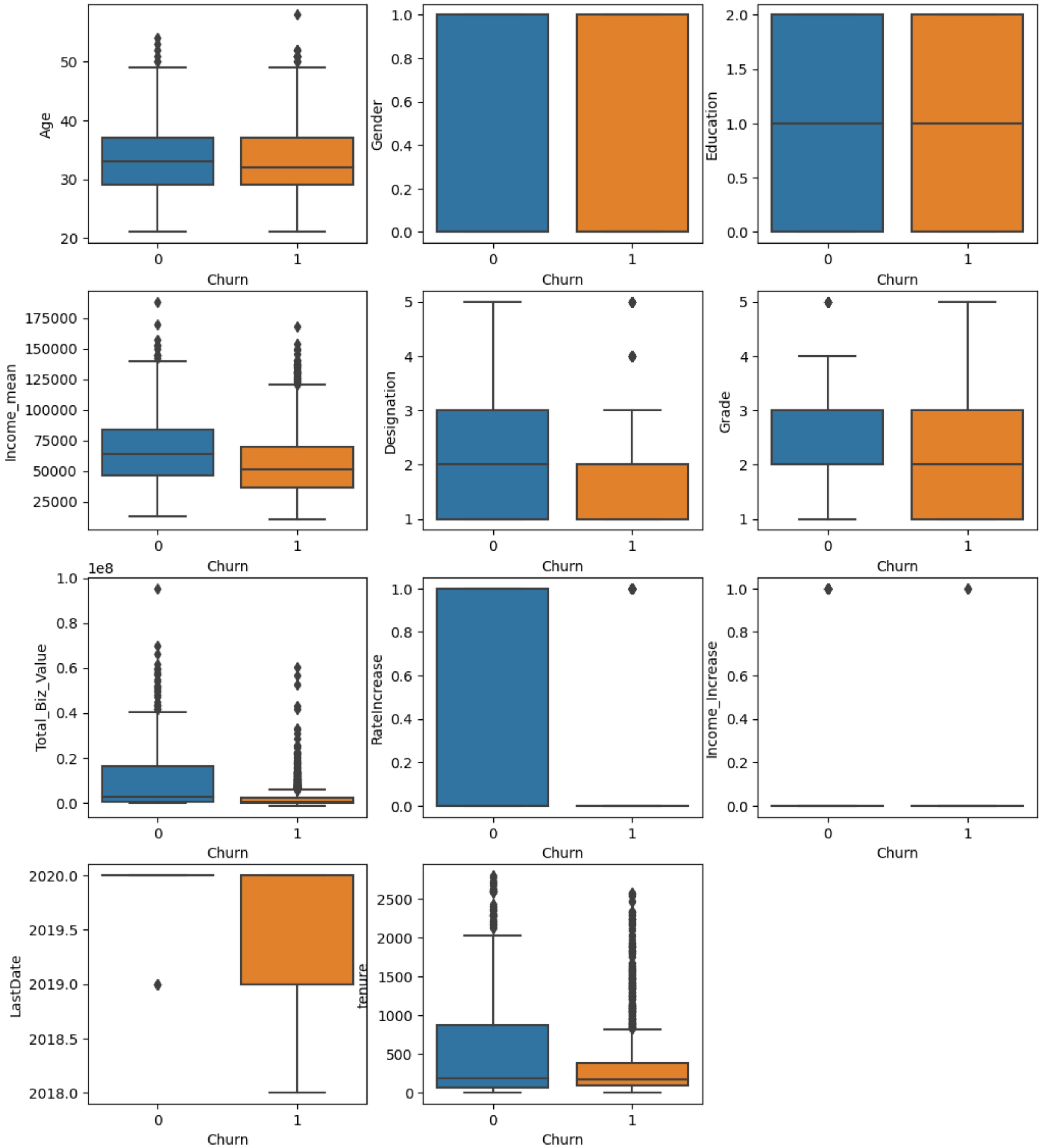
Now that we have checked the individual features, let's look at their relationship with each other.

In [51]:
```python
continuous_cols = continuous_cols.drop(labels = ['Churn'])
continuous_cols
```

Out[51]:
```
Index(['Age', 'Gender', 'Education', 'Income_mean', 'Designation', 'Grade',
       'Total_Biz_Value', 'RateIncrease', 'Income_Increase', 'LastDate',
       'tenure'],
      dtype='object')
```

In [52]:
```python
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()
```

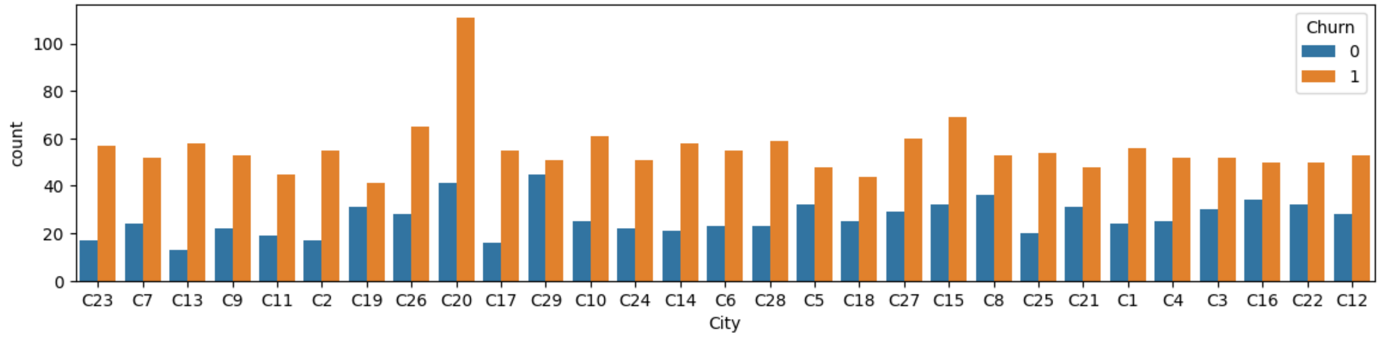


Mean income, designation while joining, and total business value are lower for drivers who churned than those who stayed.

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 [53]:
```
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 [54]:
```
X = df2.drop(['Churn'],axis=1)
Y = np.array(df2['Churn']).reshape(-1,1)
print(X.shape, Y.shape)
```

```
(2381, 12) (2381, 1)
```

In [55]:
```
X['City'] = X['City'].apply(lambda x:x[1:])
X.head()
```

Out[55]:

	Age	Gender	City	Education	Income_mean	Designation	Grade	Total_Biz_Value	RateIncrease	Income_Increas
0	28	0	23	2.0	57387.0	1.0	1.0	1715580.0	0	
1	31	0	7	2.0	67016.0	2.0	2.0	0.0	0	
2	43	0	13	2.0	65603.0	2.0	2.0	350000.0	0	
3	29	0	9	0.0	46368.0	1.0	1.0	120360.0	0	
4	31	1	11	1.0	78728.0	3.0	3.0	1265000.0	1	

In [56]:
```
X.shape
```

Out[56]:
```
(2381, 12)
```

Splitting data into training and testing dataset

In [57]:
```
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=0.2, random_state=4) #20% te
```

In [58]:
```
X_train.shape, X_test.shape
```

Out[58]: ((1904, 12), (477, 12))

Class imbalance treatment

In [59]:
```
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: 1282, 0: 1282})
```

In [60]:
```
X_sm.shape
```

Out[60]: (2564, 12)

In [61]:
```
np.unique(y_sm, return_counts=True)
```

Out[61]: (array([0, 1]), array([1282, 1282], dtype=int64))

Now, classes are balanced.

Column Standarization

In [62]:
```
# Mean centering and Variance scaling (Standard Scaling as typical minimum and maximum values are
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[62]:

	Age	Gender	City	Education	Income_mean	Designation	Grade	Total_Biz_Value	RateIncrease
0	0.542572	1.350477	1.592938	-1.303570	-1.267825	-1.030311	-1.245881	-0.551406	-0.463860
1	1.789286	-0.740479	1.216562	-0.018589	0.097855	1.395505	0.941358	-0.551406	-0.463860
2	-0.704142	1.350477	1.718396	-0.018589	-0.506328	-1.030311	-0.152261	1.500785	-0.463860
3	-1.416550	1.350477	1.592938	1.266392	-0.452047	-1.030311	-1.245881	2.076673	-0.463860
4	-0.347938	1.350477	1.718396	-0.018589	-0.810707	-1.030311	-1.245881	-0.142870	2.155824

In [63]:
```
X_sm.shape
```

Out[63]: (2564, 12)

There are 12 features and 2564 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 [64]:
```
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) # 10-fold CV
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)
0.8427672955974843
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(
K-Fold Accuracy Mean: Train: 100.0 Validation: 83.1924854085603
K-Fold Accuracy Std: Train: 0.0 Validation: 4.702191431682615
```

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

In [65]:
```
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', retur

    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()*1
    print('***************')
```

```
K-Fold for depth:5 Accuracy Mean: Train: 82.5316157816727 Validation: 80.4575024319066
K-Fold for depth: 5 Accuracy Std: Train: 0.7075841788697959 Validation: 3.9653376361758745
***************
K-Fold for depth:10 Accuracy Mean: Train: 89.13589790397548 Validation: 82.95704644941632
K-Fold for depth: 10 Accuracy Std: Train: 0.2848397472272346 Validation: 2.0132256027432054
***************
K-Fold for depth:15 Accuracy Mean: Train: 90.35359192390877 Validation: 83.54419990272373
K-Fold for depth: 15 Accuracy Std: Train: 0.3041546994602327 Validation: 3.396463167508258
***************
K-Fold for depth:20 Accuracy Mean: Train: 90.35359192390877 Validation: 83.54419990272373
K-Fold for depth: 20 Accuracy Std: Train: 0.3041546994602327 Validation: 3.396463167508258
***************
K-Fold for depth:25 Accuracy Mean: Train: 90.35359192390877 Validation: 83.54419990272373
K-Fold for depth: 25 Accuracy Std: Train: 0.3041546994602327 Validation: 3.396463167508258
***************
K-Fold for depth:30 Accuracy Mean: Train: 90.35359192390877 Validation: 83.54419990272373
K-Fold for depth: 30 Accuracy Std: Train: 0.3041546994602327 Validation: 3.396463167508258
***************
```

Since the train and validation results are closer at max depth 15, let's see smaller depths.

In [66]:

```python
depths = [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

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: 82.5316157816727 Validation: 80.4575024319066
K-Fold for depth: 5 Accuracy Std: Train: 0.7075841788697959 Validation: 3.9653376361758745
***************
K-Fold for depth:6 Accuracy Mean: Train: 84.27800552759705 Validation: 80.92412451361868
K-Fold for depth: 6 Accuracy Std: Train: 0.6224891247159411 Validation: 3.961031425584502
***************
K-Fold for depth:7 Accuracy Mean: Train: 85.50009240207072 Validation: 80.61162451361866
K-Fold for depth: 7 Accuracy Std: Train: 0.5133386382571106 Validation: 4.551591849938311
***************
K-Fold for depth:8 Accuracy Mean: Train: 87.31149789766508 Validation: 83.73540856031128
K-Fold for depth: 8 Accuracy Std: Train: 0.4704292108332778 Validation: 2.0801704641735324
***************
K-Fold for depth:9 Accuracy Mean: Train: 88.08290494080634 Validation: 83.85366001945525
K-Fold for depth: 9 Accuracy Std: Train: 0.465854920614268 Validation: 1.9331734861367886
***************
K-Fold for depth:10 Accuracy Mean: Train: 89.13589790397548 Validation: 82.95704644941632
K-Fold for depth: 10 Accuracy Std: Train: 0.2848397472272346 Validation: 2.0132256027432054
***************
K-Fold for depth:11 Accuracy Mean: Train: 89.63427748717454 Validation: 83.4639469844358
K-Fold for depth: 11 Accuracy Std: Train: 0.38775363467559776 Validation: 2.4847857077468065
***************
K-Fold for depth:12 Accuracy Mean: Train: 90.09357775559128 Validation: 83.38886186770426
K-Fold for depth: 12 Accuracy Std: Train: 0.2575032746487283 Validation: 3.6947069893223183
***************
K-Fold for depth:13 Accuracy Mean: Train: 90.33192626765499 Validation: 83.85563594357976
K-Fold for depth: 13 Accuracy Std: Train: 0.30176089628244035 Validation: 3.227574071723548
***************
K-Fold for depth:14 Accuracy Mean: Train: 90.36225743517393 Validation: 83.54389591439688
K-Fold for depth: 14 Accuracy Std: Train: 0.3084925098051531 Validation: 3.41618328498675
***************
K-Fold for depth:15 Accuracy Mean: Train: 90.35359192390877 Validation: 83.54419990272373
K-Fold for depth: 15 Accuracy Std: Train: 0.3041546994602327 Validation: 3.396463167508258
***************
```

Max depth 8 is the 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 [67]:
```
# Defining Parameters
params = {
          'n_estimators' : [20,50,100,200,300],
          'max_depth' : [7,8],
          'criterion' : ['gini'],
          'bootstrap' : [True],
          'min_samples_leaf' : [5,10,15],
          'max_features' : [6,7,8,9,10,11]
         }
```

In [68]:
```
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': 8, 'max_features': 6, 'min_samples_leaf':
5, 'n_estimators': 300}
0.8595993627054886
```

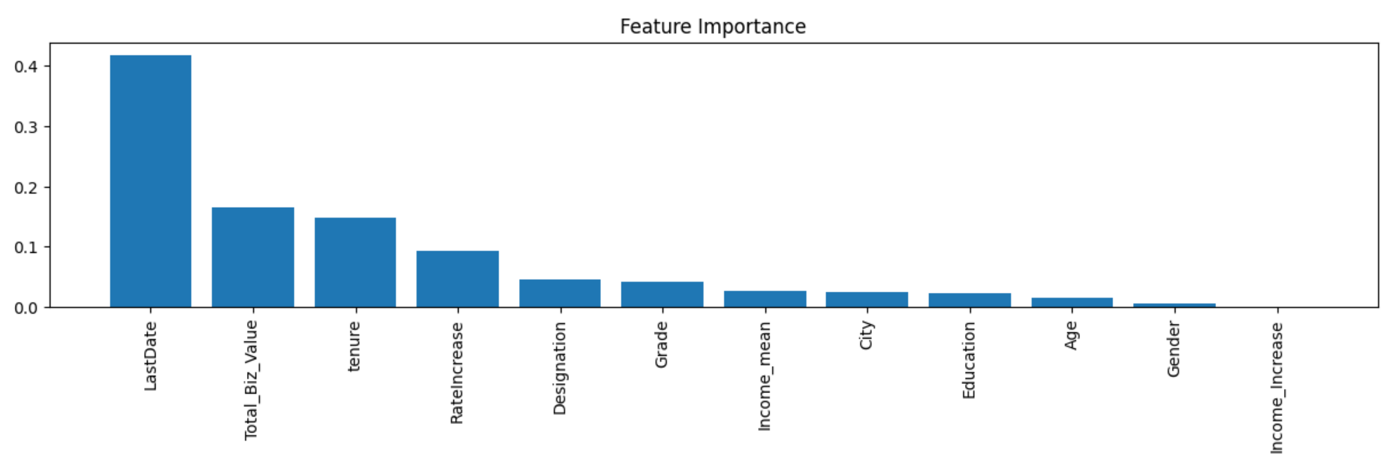
In [69]:

```
from sklearn.model_selection import KFold, cross_validate

tree_clf = RandomForestClassifier(random_state=7, max_depth=8, max_features=6, n_estimators=300,
kfold = KFold(n_splits=3)
cv_acc_results = cross_validate(tree_clf, X_sm, y_sm, cv = kfold, scoring = 'accuracy', return_tı

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_ac
```

```
K-Fold Accuracy Mean: Train: 90.67903325702594 Validation: 79.13248512903387
K-Fold Accuracy Std: Train: 1.477415557807954 Validation: 3.0597491651533786
```

In [70]:

```
# Feature importance
tree_clf = RandomForestClassifier(random_state=7, max_depth=8, max_features=6, n_estimators=300,
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, increase in quarterly ratings, starting designation, grade, mean income, city, age, education, 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 [71]:
```
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 [72]:
```
len(X_test)
```

Out[72]: 477

In [73]:
```
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:  0.8888888888888888
```

Out[73]:

	0	1
0	125	18
1	35	299

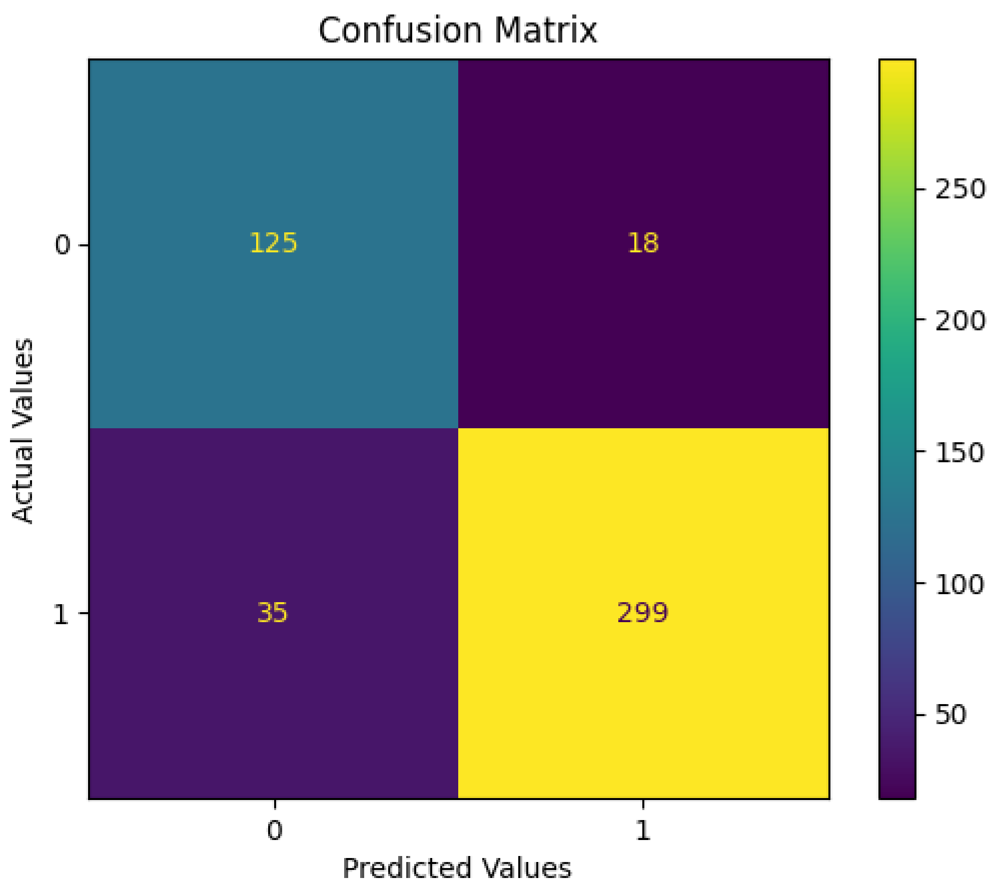
In [74]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.87      0.83       143
           1       0.94      0.90      0.92       334

    accuracy                           0.89       477
   macro avg       0.86      0.88      0.87       477
weighted avg       0.89      0.89      0.89       477
```

In [75]:
```
#Plotting the confusion matrix
plt.figure(figsize=(1,1))
plot_confusion_matrix(tree_clf,X_test,y_test)
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 [76]:
```python
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 : 0.943217665615142
Recall score is : 0.8952095808383234
F1 score is : 0.9185867895545314
```

Precision score was 0.94 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- 0.89 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.

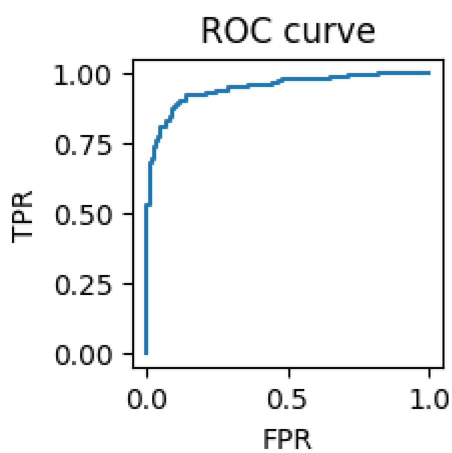
ROC (Receiver Operating Characteristic curve) and AUC (Area Under Curve)

In [77]:
```
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[77]:
```
((477, 2), (477, 1))
```

In [78]:
```
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 [79]:
```
roc_auc_score(y_test,y_proba[:,1])
```

Out[79]:
```
0.9445793727230853
```

0.94 is a good area under curve score.

Model lift

In [126…
```
y_pred = tree_clf.predict(X_test)
y_pred = np.array(y_pred).reshape(-1,1).flatten()
yt = np.array(y_test).flatten()
y_pred.shape, yt.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[126]:
```
((477,), (477,))
```

In [127…
```
y_pred[:5]
```

Out[127]:
```
array([0, 1, 1, 1, 1])
```

In [129…

```python
# Create a DataFrame to store predictions and actual labels
df3 = pd.DataFrame({'Actual': yt, 'Predicted': y_pred})
df3 = df3.sort_values(by='Predicted', ascending=False) # Sort the predicted probabilities in desc
total_positives = df3['Actual'].sum()  # total number of positive cases

lift_percentage = 0.1  # Calculate the number of observations needed for a certain percentage
required = int((lift_percentage * len(df3)) + 1)
top_N = df3.head(required)

positives_in_top_N = top_N['Actual'].sum() # the number of positive cases in the top N

# Calculate the lift
baseline_rate = total_positives / len(df3)
lift = (positives_in_top_N / required) / baseline_rate

print(f"Model Lift for top decile: {lift:.2f}")
```

```
Model Lift for top decile: 1.37
```

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 [80]:

```python
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
[23:42:50] 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[80]:

```
▸ RandomizedSearchCV
▸ estimator: XGBClassifier
    ▸ XGBClassifier
```

In [82]:

```
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 [83]:

```
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[83]:

```
▾ 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 [84]:

```
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:  1.0
Test accuracy:  0.8721174004192872
```

Out[84]:

	0	1
0	116	27
1	34	300

True negatives - 116 (drivers who stayed as predicted)

False positives - 27 (drivers who stayed but were predicted to churn)

False negatives - 34 (drivers who churned but were predicted to stay) - need the most attention as can cause huge financial losses

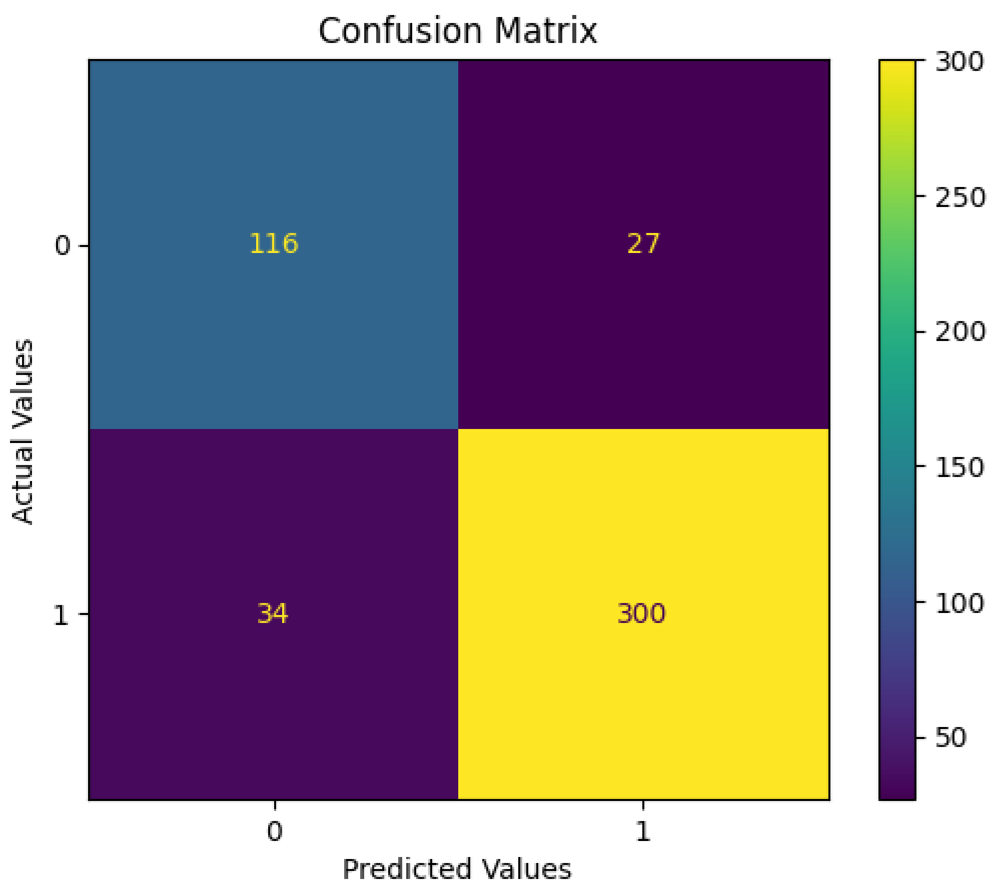
True positives - 300 (drivers who churned as predicted)

In [85]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.81      0.79       143
           1       0.92      0.90      0.91       334

    accuracy                           0.87       477
   macro avg       0.85      0.85      0.85       477
weighted avg       0.87      0.87      0.87       477
```

In [86]:
```
#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 [87]:
```
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 : 0.9174311926605505
Recall score is : 0.8982035928143712
F1 score is : 0.9077155824508321
```

Precision is 0.92 which is good as the number of false positives (7) are low, the drivers who stayed but were predicted to churn.

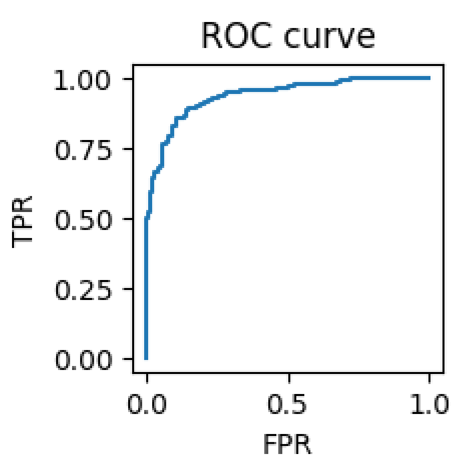
Recall-score: 0.89. Recall is high because of low number of False negatives. This is good as we are not losing too many drivers (9) to churn who were predicted to stay. But we can still improve recall score by encouraging drivers to stay and provide enticing perks and reduce financial losses.

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

In [95]:
```
# from sklearn.linear_model import
y_proba = best_xgb.predict_proba(X_test)
y_proba.shape, y_test.shape
```

Out[95]: ((477, 20), (477, 1))

In [89]:
```
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 [90]:
```
roc_auc_score(y_test,y_proba[:,1])
```

Out[90]: 0.9353251538880282

0.93 is a good area under curve.

Model lift

In [116…
```
y_pred = best_xgb.predict(X_test)
y_pred = np.array(y_pred).reshape(-1,1).flatten()
yt = np.array(y_test).flatten()
y_pred.shape, yt.shape
```

Out[116]: ((477,), (477,))

In [117…
```
y_pred[:5]
```

Out[117]: array([0, 1, 1, 1, 1])

In [130…
```
# Create a DataFrame to store predictions and actual labels
df3 = pd.DataFrame({'Actual': yt, 'Predicted': y_pred})
df3 = df3.sort_values(by='Predicted', ascending=False) # Sort the predicted probabilities in desc
total_positives = df3['Actual'].sum()  # total number of positive cases

lift_percentage = 0.1  # Calculate the number of observations needed for a certain percentage
required = int((lift_percentage * len(df3)) + 1)
top_N = df3.head(required)

positives_in_top_N = top_N['Actual'].sum() # the number of positive cases in the top N

# Calculate the lift
baseline_rate = total_positives / len(df3)
lift = (positives_in_top_N / required) / baseline_rate

print(f"Model Lift for top decile: {lift:.2f}")
```

Model Lift for top decile: 1.37

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 87.2%. Precision is 0.92, Recall score 0.89, F1-score 0.91.

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: 88.8%. Precision score was 0.94, Recall score 0.89, F1-score 0.91.

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 [78]:
```
''
```

Out[78]:
```
''
```