To build a model that could predict Churn among ola drivers

Problem Statement - To build a model that could predict Churn among ola drivers

Insights :

1. Total Business value and Quarterly rating mean has a corr of 0.9
2. No of days served and Quarterly rating mean has a corr 0.66
3. Grade and first income has a corr of 0.71
4. Total Business Value has more than 14% outliers
5. There are more drivers from C20 City
6. Median age of drivers who have quit is less than those who did not quit
7. Quarterly rating means of drivers who quit has a median less than those who did not quit
8. Those who have quarterly rating values less than 0 have quit the most
9. Education level has no influence on the drivers quitting
10. Grades 1 and 2 are more in count than other grades
11. Drivers who quit, quit within a year most
12. Data is an imbalance as the number of people who quit is more
13. Data is available on the drivers who quit in 2019 and 2020 years
14. Drivers who have joint recently in years 2018,2019 and 2020 have quit the most
15. Drivers join with designation 1,2 or 3 the most
16. Drivers join with grade 1,2 or 3 the most
17. 67.7% of the driver's Quarterly_Rating remained the same. For others it increased and decreased

F2 SCORE IS MY PRIME FOCUS. F2 SCORE GIVES MORE WEIGHTAGE TO RECALL THAN PRECISION. OUR FOCUS IS TO REDUCE FALSE NEGATIVE

Random Forest Classifier:

1. Best parameters are {'bootstrap': True, 'ccp_alpha': 0, 'class_weight': 'balanced_subsample', 'criterion': 'log_loss', 'max_features': 30, 'n_estimators': 100, 'n_jobs': -1, 'random_state': 9}
2. Cross Validation : test_precision : 84.83463864970851 train_precision : 96.61671601049319 test_recall : 89.23022896474222 train_recall : 98.2373620663061 test_f1 : 86.58440240891856 train_f1 : 97.41972142258018
3. F2 score for train and test data is below(0.998818432453722, 0.8840486867392696)
4. Recall on test data is 0.91 # Precision on test data is 0.84
5. AOC-ROC curve for train data is 0.88
6. Date_Diff,Quarterly_Rating_Value,Total Business Value,Quarterly_Rating_Mean has the highest importance

GradientBoostingClassifier :

1. Best parameters are {'ccp_alpha': 0, 'criterion': 'friedman_mse', 'learning_rate': 0.15, 'loss': 'log_loss', 'max_features': 45, 'n_estimators': 200, 'random_state': 9, 'subsample': 0.9}
2. Cross Validation : test_precision : 86.52222752960618 train_precision : 94.60994175516304 test_recall : 91.3850259867959 train_recall : 98.19504209928247 test_f1 : 88.8753141396598 train_f1 : 96.36906098988803
3. F2 score for train and test data is below(0.9675174013921114, 0.902423469387755)
4. Recall on test data is 0.92 # Precision on test data is 0.84
5. AOC-ROC curve for train data is 0.90
6. Date_Diff,Quarterly_Rating_Value,Total Business Value,Quarterly_Rating_Mean has the highest importance

VIF: Features 9,10,11,12 has the highest score. This needs to be dropped Remaining all columns have VIF score of less than 5

Recommendations :

1. Use GradientBoostingClassifier for the given data set
2. As there only 1800 unique driver ID's we will need more data for more accuracy
3. For the drivers who are predicted they would leave have lesser profit for the company
4. Assign more rides to these drivers who are predicted they would quit
5. FP is high because we are considering F2 score as our metric in CV. In order to reduce FN we are increasing FP in our model because we cant afford to lose a driver.
6. We can give promotional offers to particular drivers who tend to leave
7. Try to match the profit ratio with our competitors (Uber, city taxi, Rapido etc)
8. Ola can issue 5 year bond with a driver and them with electric cars, CNG where the operation cost is the least
9. Based on the driver performance Ola can start giving small loans to them with low interest rates to retain them.
10. Ola should increase its marketing and add additional drivers to balance out the drivers who could leave in the next 3-4 months
11. Slightly increase the profit margin for those who do not tend to quit ola
12. Build surge pricing model for ola drivers
13. Car drivers to park these vehicles in regions in cities there the demand is more in their respective cities

In [2]:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge,Lasso,ElasticNet
import sklearn.metrics as metrics
from sklearn.preprocessing import PolynomialFeatures
from sklearn import decomposition
from  scipy import stats
from sklearn import decomposition
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
from imblearn.over_sampling import SMOTE
from sklearn.impute import KNNImputer
from category_encoders import TargetEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve,roc_auc_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import precision_score,recall_score,f1_score,fbeta_score
from imblearn.metrics import geometric_mean_score
from scipy.stats import pearsonr,spearmanr,kendalltau
import datetime as dt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import classification_report
from sklearn.metrics import fbeta_score, make_scorer
```

In [3]:

```
df=pd.read_csv('ola_driver_scaler.csv')
df.drop('Unnamed: 0',inplace=True,axis=1)
# Droppinig 'Unnamed: 0'
```

In [4]:

```
df
```

Out[4]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterly Rating
0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	2381060	2
1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	-665480	2
2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	03/11/19	1	1	0	2
3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	0	1
4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
19099	08/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2	740280	3
19100	09/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2	448370	3
19101	10/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2	0	2
19102	11/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2	200420	2
19103	12/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	NaN	2	2	411480	2

19104 rows × 13 columns

In [5]:

```
df.info()
# There are null values in 'Last Working date', 'Age' and 'Gender'
```

```
<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 [6]:

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

Out[6]:

```
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
```

In [7]:

```
df1=df[['Driver_ID','MMM-YY','Quarterly Rating']]
df1['MM']=df1['MMM-YY'].apply(lambda x : x[:2])
df1['YY']=df1['MMM-YY'].apply(lambda x : x[-2:])
df1.drop('MMM-YY',axis=1,inplace=True)
df2=df1.copy('deep')
df2.drop(['Quarterly Rating','MM','YY'],axis=1,inplace=True)
df2.drop_duplicates(keep='first',ignore_index=True,inplace=True)
df2['ele']=''
```

In [8]:

```
prev_rate=df1.iloc[0][1]
prev_id=df1.iloc[0][0]
for i in range(1,len(df1)):
    if df1.iloc[i][0]==prev_id:
        if df1.iloc[i][1]>prev_rate:

            new=df2[df2['Driver_ID']==df1.iloc[i][0]].iloc[0][1]
            ind=df2[df2['Driver_ID']==df1.iloc[i][0]].index
            df2.iloc[ind, df2.columns.get_loc('ele')] += ' increased'

        elif df1.iloc[i][1]<prev_rate :
            new=df2[df2['Driver_ID']==df1.iloc[i][0]].iloc[0][1]
            ind=df2[df2['Driver_ID']==df1.iloc[i][0]].index
            df2.iloc[ind, df2.columns.get_loc('ele')] += ' decreased'

        elif df1.iloc[i][1]==prev_rate :
            new=df2[df2['Driver_ID']==df1.iloc[i][0]].iloc[0][1]
            ind=df2[df2['Driver_ID']==df1.iloc[i][0]].index
            df2.iloc[ind, df2.columns.get_loc('ele')] += ' const'
        prev_id=df1.iloc[i][0]
        prev_rate=df1.iloc[i][1]
    else:
        new=df2[df2['Driver_ID']==df1.iloc[i][0]].iloc[0][1]
        ind=df2[df2['Driver_ID']==df1.iloc[i][0]].index
        df2.iloc[ind, df2.columns.get_loc('ele')] += ' const'
        prev_id=df1.iloc[i][0]
        prev_rate=df1.iloc[i][1]
```

In [9]:

```
df2['ele1']=df2['ele'].apply(lambda x : x.split(' '))
df2['ele1']=df2['ele1'].apply(lambda x : x[1:])
```

In [10]:

```
list_val=[]
for i in range(len(df2)):
    c=0
    for j in df2['ele1'][i]:
        if j=='const':
            c=c+0
        if j=='decreased':
            c=c-1
        if j=='increased':
            c=c+1
    list_val.append(c)
```

In [11]:

```
df2['ele2']=list_val
df2.drop(['ele','ele1'],axis=1,inplace=True)
df2.columns=['Driver_ID','Quarterly_Rating_Value']
# Quaterly_Rating_Value is updated with-3,-2,-1,0,1,2,3 based on the value of rating it has increased or decreased
```

In [12]:

```
df2
```

Out[12]:

	Driver_ID	Quarterly_Rating_Value
0	1	0
1	2	0
2	4	0
3	5	0
4	6	1
...	...	...
2376	2784	1
2377	2785	0
2378	2786	-1
2379	2787	-1
2380	2788	0

2381 rows × 2 columns

In [13]:

```
df['MM-YY']=pd.to_datetime(df['MMM-YY'])
df['Date_of_joining']=pd.to_datetime(df['Dateofjoining'])
df['Last_Working_Date']=pd.to_datetime(df['LastWorkingDate'])
df.drop(['MMM-YY','Dateofjoining','LastWorkingDate'],inplace=True,axis=1)
```

In [14]:

```
df3=df.groupby('Driver_ID')[['Quarterly Rating']].agg({'mean'}).reset_index()
```

In [15]:

```
df3.columns=['Driver_ID','Quarterly_Rating_Mean']
# Finding the mean of Quarterly_Rating
```

In [16]:

```
df4=df.groupby('Driver_ID')[['Age','Gender','City','Education_Level','Date_of_joining','Joining Designation','Grade','Total Business Valu
                                                                                            'Age':'last',
                                                                                            'Gender':'last',
                                                                                            'City':'last',
                                                                                            'Education_Level':'last',
                                                                                             'Total Business Value':'sum',
                                                                                            'Date_of_joining':'first',
                                                                                            'Joining Designation':'first',
                                                                                            'Grade':'first',
                                                                                            'Last_Working_Date':'last'}).reset_index()
```

In [17]:

```
# Aggregating for each driver ID: age based on last value,gender on last value, city on last value, education level on last value,
# Total business as sum, date of joining as first value, joining designation as first, grade as first and last working date as last.
df4
```

Out[17]:

	Driver_ID	Age	Gender	City	Education_Level	Total Business Value	Date_of_joining	Joining Designation	Grade	Last_Working_Date
0	1	28.0	0.0	C23	2	1715580	2018-12-24	1	1	2019-03-11
1	2	31.0	0.0	C7	2	0	2020-11-06	2	2	NaT
2	4	43.0	0.0	C13	2	350000	2019-12-07	2	2	2020-04-27
3	5	29.0	0.0	C9	0	120360	2019-01-09	1	1	2019-03-07
4	6	31.0	1.0	C11	1	1265000	2020-07-31	3	3	NaT
...	...	...	...	...	...	...	...	...	...	...
2376	2784	34.0	0.0	C24	0	21748820	2015-10-15	2	3	NaT
2377	2785	34.0	1.0	C9	0	0	2020-08-28	1	1	2020-10-28
2378	2786	45.0	0.0	C19	0	2815090	2018-07-31	2	2	2019-09-22
2379	2787	28.0	1.0	C20	2	977830	2018-07-21	1	1	2019-06-20
2380	2788	30.0	0.0	C27	2	2298240	2020-06-08	2	2	NaT

2381 rows × 10 columns

In [18]:

```
df5=df.groupby('Driver_ID')[['MM-YY']].agg({'max','min'}).reset_index()
```

In [19]:

```
df5.columns = ['Driver_ID', 'Max_Date', 'Min_Date']
```

In [20]:

```
df5['Date_Diff']=abs(df5['Min_Date']-df5['Max_Date'])
# Caluculating difference in dates for which data is avaialbles
```

In [21]:

```
df5
```

Out[21]:

	Driver_ID	Max_Date	Min_Date	Date_Diff
0	1	2019-03-01	2019-01-01	59 days
1	2	2020-12-01	2020-11-01	30 days
2	4	2020-04-01	2019-12-01	122 days
3	5	2019-03-01	2019-01-01	59 days
4	6	2020-12-01	2020-08-01	122 days
...	...	...	...	...
2376	2784	2020-12-01	2019-01-01	700 days
2377	2785	2020-10-01	2020-08-01	61 days
2378	2786	2019-09-01	2019-01-01	243 days
2379	2787	2019-06-01	2019-01-01	151 days
2380	2788	2020-12-01	2020-06-01	183 days

2381 rows × 4 columns

In [22]:

```
df6=df.groupby('Driver_ID')[['Income']].agg({'first','last'}).reset_index()
```

In [23]:

```
df6.columns = ['Driver_ID', 'First_Income', 'Last_Income']
```

In [24]:

```
df6['Income_diff']=df6['Last_Income']-df6['First_Income']
```

In [25]:

```
# Difference, first and last income
df6
```

Out[25]:

	Driver_ID	First_Income	Last_Income	Income_diff
0	1	57387	57387	0
1	2	67016	67016	0
2	4	65603	65603	0
3	5	46368	46368	0
4	6	78728	78728	0
...	...	...	...	...
2376	2784	82815	82815	0
2377	2785	12105	12105	0
2378	2786	35370	35370	0
2379	2787	69498	69498	0
2380	2788	70254	70254	0

2381 rows × 4 columns

In [26]:

```
df7=df2.merge(df3, on='Driver_ID')
df8=df7.merge(df4, on='Driver_ID')
df9=df8.merge(df5, on='Driver_ID')
df10=df9.merge(df6, on='Driver_ID')
# Merging all data frames to a single frame
```

In [27]:

```
df10['No_of_days_served']=df10['Last_Working_Date']-df10['Date_of_joining']
df10['Quit']=np.isnan(df10['No_of_days_served'])
temp_dict = {False: 1, True: 0 }
df10['Quit'] = df10['Quit'].map(temp_dict)
# Encoding 'Quit' columns as 1 and 0
```

In [28]:

```
df10
```

Out[28]:

	Driver_ID	Quarterly_Rating_Value	Quarterly_Rating_Mean	Age	Gender	City	Education_Level	Total Business Value	Date_of_joining	Joining Designation	Grade	Last
0	1	0	2.000000	28.0	0.0	C23	2	1715580	2018-12-24	1	1	
1	2	0	1.000000	31.0	0.0	C7	2	0	2020-11-06	2	2	
2	4	0	1.000000	43.0	0.0	C13	2	350000	2019-12-07	2	2	
3	5	0	1.000000	29.0	0.0	C9	0	120360	2019-01-09	1	1	
4	6	1	1.600000	31.0	1.0	C11	1	1265000	2020-07-31	3	3	
...	...	...	...	...	...	...	...	...	...	...	...	
2376	2784	1	2.625000	34.0	0.0	C24	0	21748820	2015-10-15	2	3	
2377	2785	0	1.000000	34.0	1.0	C9	0	0	2020-08-28	1	1	
2378	2786	-1	1.666667	45.0	0.0	C19	0	2815090	2018-07-31	2	2	
2379	2787	-1	1.500000	28.0	1.0	C20	2	977830	2018-07-21	1	1	
2380	2788	0	2.285714	30.0	0.0	C27	2	2298240	2020-06-08	2	2	

2381 rows × 20 columns

In [29]:

```
df10.info()
```

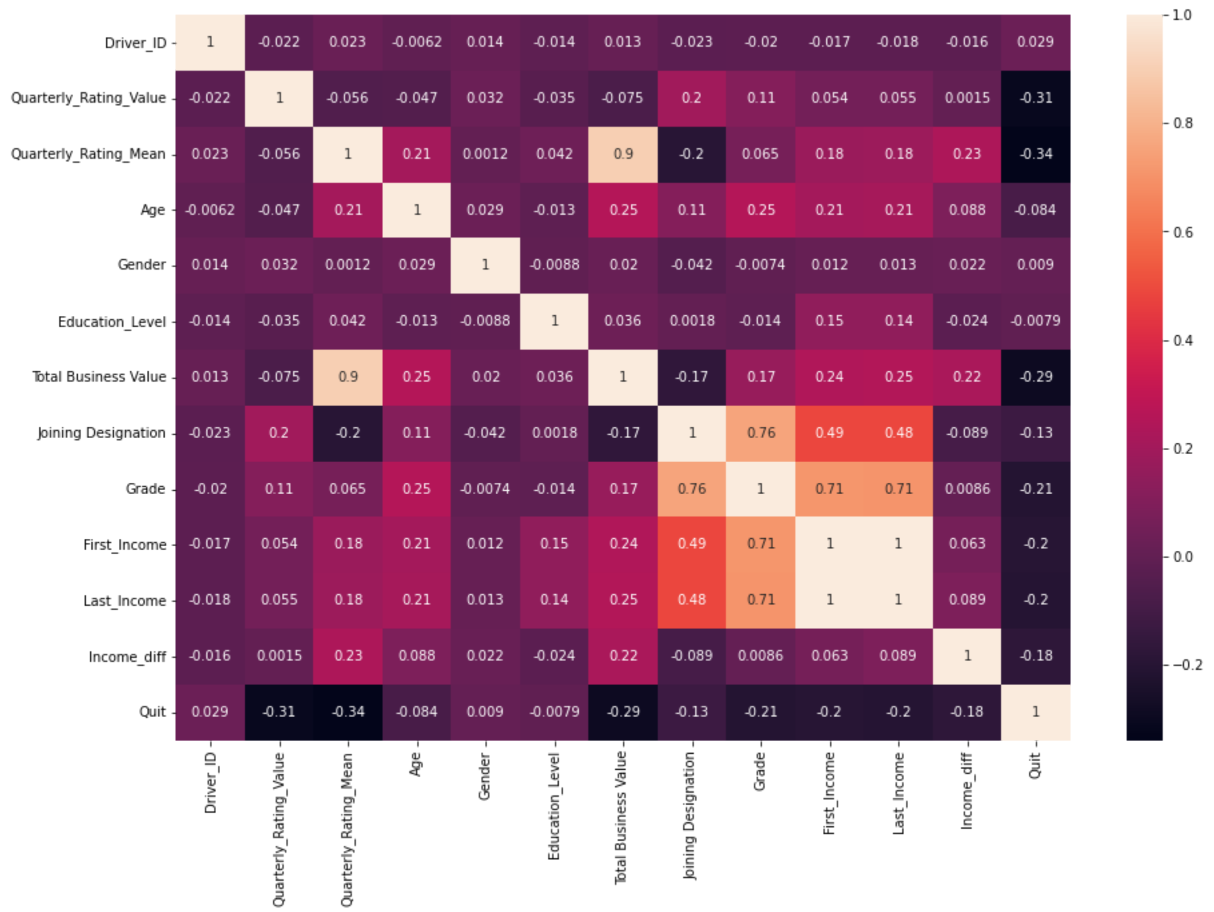
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2381 entries, 0 to 2380
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Driver_ID               2381 non-null   int64
 1   Quarterly_Rating_Value  2381 non-null   int64
 2   Quarterly_Rating_Mean   2381 non-null   float64
 3   Age                     2381 non-null   float64
 4   Gender                  2381 non-null   float64
 5   City                    2381 non-null   object
 6   Education_Level         2381 non-null   int64
 7   Total Business Value    2381 non-null   int64
 8   Date_of_joining         2381 non-null   datetime64[ns]
 9   Joining Designation     2381 non-null   int64
 10  Grade                   2381 non-null   int64
 11  Last_Working_Date       1616 non-null   datetime64[ns]
 12  Max_Date                2381 non-null   datetime64[ns]
 13  Min_Date                2381 non-null   datetime64[ns]
 14  Date_Diff               2381 non-null   timedelta64[ns]
 15  First_Income            2381 non-null   int64
 16  Last_Income             2381 non-null   int64
 17  Income_diff             2381 non-null   int64
 18  No_of_days_served       1616 non-null   timedelta64[ns]
 19  Quit                    2381 non-null   int64
dtypes: datetime64[ns](4), float64(3), int64(10), object(1), timedelta64[ns](2)
memory usage: 390.6+ KB
```

In [30]:

```
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(df10.corr(method ='spearman'),annot=True,ax=ax)
# Total Business value and Quaterly rating mean has a corr of 0.9
# No of days served and Quaterly rating mean has a corr 0.66
# Grade and first income has has a corr of 0.71
```

Out[30]:

```
<AxesSubplot:>
```

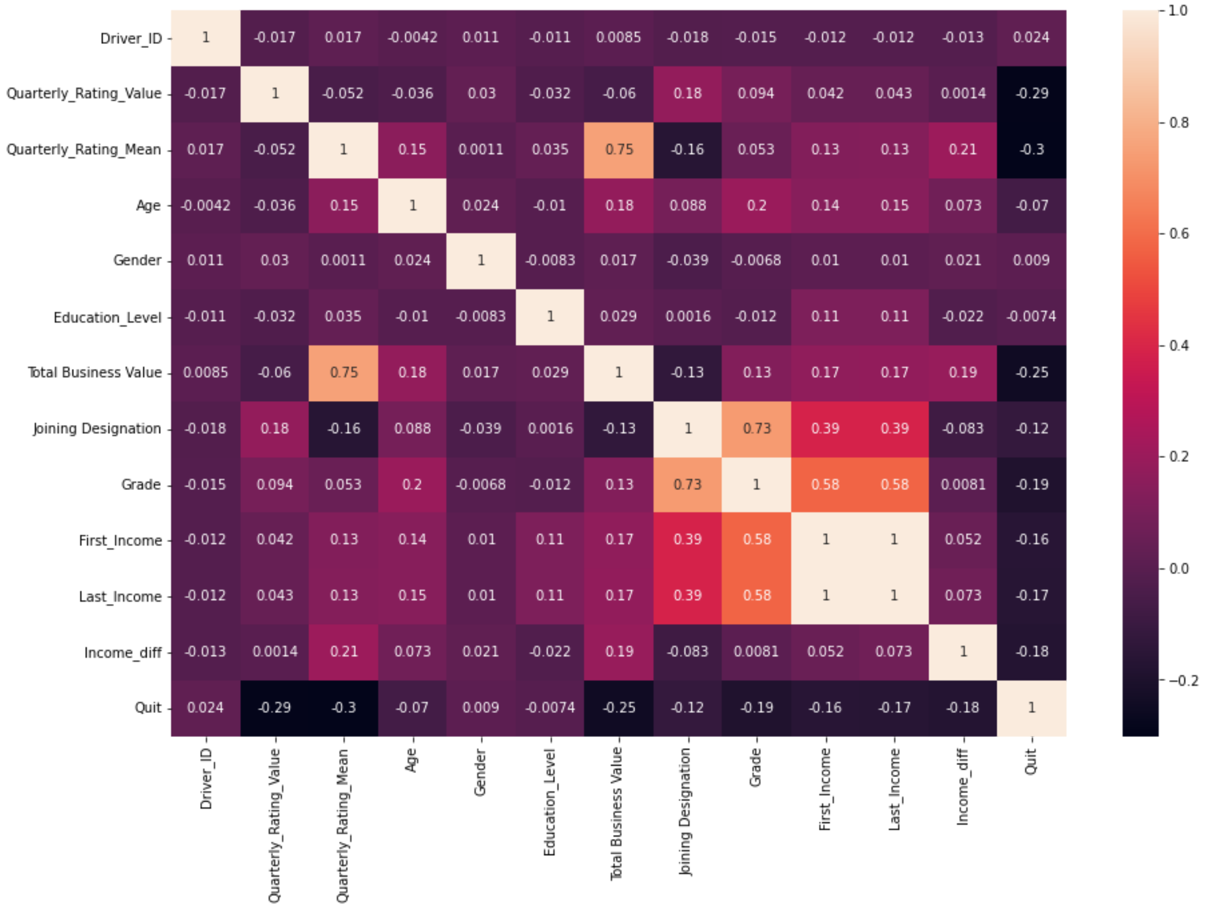


In [31]:

```
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(df10.corr(method ='kendall'),annot=True,ax=ax)
```

Out[31]:

```
<AxesSubplot:>
```



In [32]:

```
df10['No_of_days_served']=df10['No_of_days_served'].dt.days
df10['Date_Diff']=df10['Date_Diff'].dt.days
```

In [33]:

```
df10
```

Out[33]:

	Driver_ID	Quarterly_Rating_Value	Quarterly_Rating_Mean	Age	Gender	City	Education_Level	Total Business Value	Date_of_joining	Joining Designation	Grade	Last
0	1	0	2.000000	28.0	0.0	C23	2	1715580	2018-12-24	1	1	
1	2	0	1.000000	31.0	0.0	C7	2	0	2020-11-06	2	2	
2	4	0	1.000000	43.0	0.0	C13	2	350000	2019-12-07	2	2	
3	5	0	1.000000	29.0	0.0	C9	0	120360	2019-01-09	1	1	
4	6	1	1.600000	31.0	1.0	C11	1	1265000	2020-07-31	3	3	
...	...	...	...	...	...	...	...	...	...	...	...	
2376	2784	1	2.625000	34.0	0.0	C24	0	21748820	2015-10-15	2	3	
2377	2785	0	1.000000	34.0	1.0	C9	0	0	2020-08-28	1	1	
2378	2786	-1	1.666667	45.0	0.0	C19	0	2815090	2018-07-31	2	2	
2379	2787	-1	1.500000	28.0	1.0	C20	2	977830	2018-07-21	1	1	
2380	2788	0	2.285714	30.0	0.0	C27	2	2298240	2020-06-08	2	2	

2381 rows × 20 columns

In [34]:

```
df10.describe()
# Total Business Value has lot of outliners
```

Out[34]:

	Driver_ID	Quarterly_Rating_Value	Quarterly_Rating_Mean	Age	Gender	Education_Level	Total Business Value	Joining Designation	Grade
count	2381.000000	2381.000000	2381.000000	2381.000000	2381.000000	2381.00000	2.381000e+03	2381.000000	2381.000000
mean	1397.559009	-0.042419	1.566304	33.663167	0.410332	1.00756	4.586742e+06	1.820244	2.078538
std	806.161628	0.718517	0.719652	5.983375	0.491997	0.81629	9.127115e+06	0.841433	0.931321
min	1.000000	-3.000000	1.000000	21.000000	0.000000	0.00000	-1.385530e+06	1.000000	1.000000
25%	695.000000	0.000000	1.000000	29.000000	0.000000	0.00000	0.000000e+00	1.000000	1.000000
50%	1400.000000	0.000000	1.000000	33.000000	0.000000	1.00000	8.176800e+05	2.000000	2.000000
75%	2100.000000	0.000000	2.000000	37.000000	1.000000	2.00000	4.173650e+06	2.000000	3.000000
max	2788.000000	3.000000	4.000000	58.000000	1.000000	2.00000	9.533106e+07	5.000000	5.000000

In [35]:

```
df10.describe(include='object')
# There are more drivers from C20 City
```

Out[35]:

	City
count	2381
unique	29
top	C20
freq	152

In [36]:

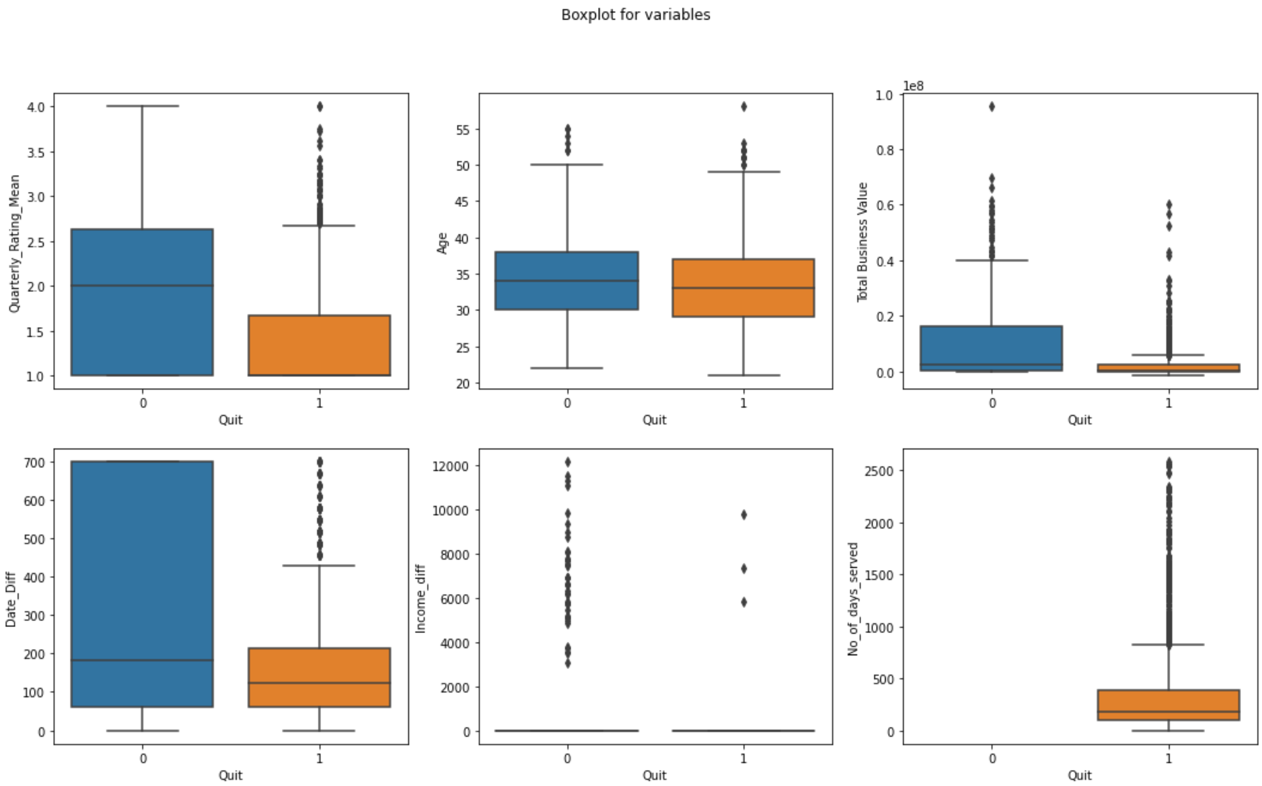
```
df10.shape
# Shape of the Data
```

Out[36]:

```
(2381, 20)
```

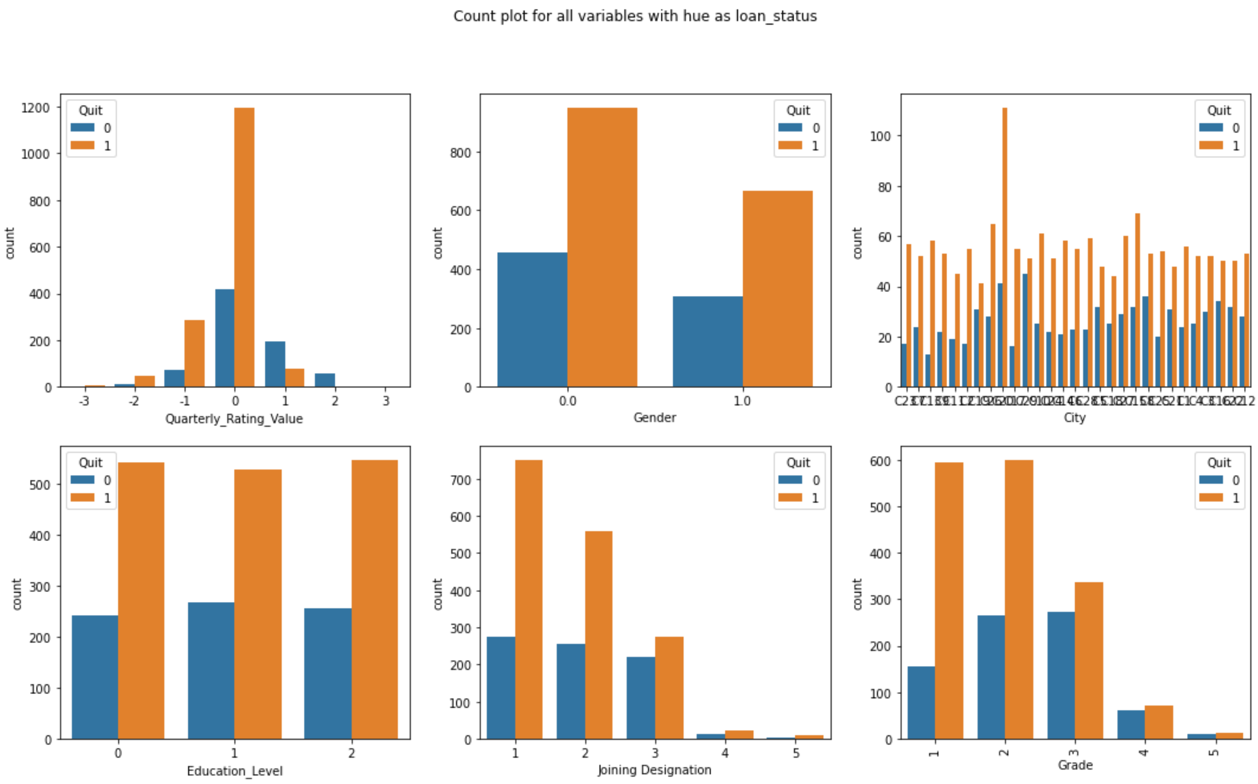
In [37]:

```
fig, axes = plt.subplots(2, 3, figsize=(18, 10))

fig.suptitle('Boxplot for variables')
sns.boxplot(ax=axes[0, 0], data=df10, y='Quarterly_Rating_Mean',x='Quit')
sns.boxplot(ax=axes[0, 1], data=df10, y='Age',x='Quit')
sns.boxplot(ax=axes[0, 2], data=df10, y='Total Business Value',x='Quit')
sns.boxplot(ax=axes[1, 0], data=df10, y='Date_Diff',x='Quit')
sns.boxplot(ax=axes[1, 1], data=df10, y='Income_diff',x='Quit')
sns.boxplot(ax=axes[1, 2], data=df10, y='No_of_days_served',x='Quit')

plt.show()
# Median age of drivers who have quit is less than those who did not quit
# Quaterly rating mean of drivers who quit have median less than those who did not quit
```



In [38]:

```
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
plt.xticks(rotation = 90)
fig.suptitle('Count plot for all variables with hue as loan_status')
sns.countplot(ax=axes[0, 0], data=df10, x='Quarterly_Rating_Value',hue='Quit')
sns.countplot(ax=axes[0, 1], data=df10, x='Gender',hue='Quit')
sns.countplot(ax=axes[0, 2], data=df10, x='City',hue='Quit' )
sns.countplot(ax=axes[1, 0], data=df10, x='Education_Level',hue='Quit' )
sns.countplot(ax=axes[1, 1], data=df10, x='Joining Designation',hue='Quit')
sns.countplot(ax=axes[1, 2], data=df10, x='Grade',hue='Quit')
plt.show()
# Those who have quaterly rating value less than 0 have quit most
# Education level has no influence on the drivers quiting
# Grade 1 and 2 are more in count than other grades
```

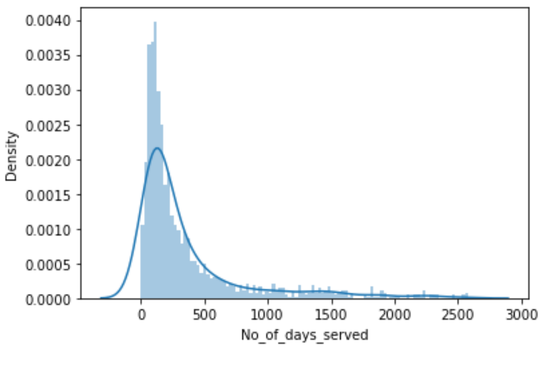


In [39]:

```
sns.distplot(df10['No_of_days_served'],bins=100)
# Drivers who quit, quit within a year most
```

Out[39]:

```
<AxesSubplot:xlabel='No_of_days_served', ylabel='Density'>
```



In [40]:

```
df10['Quit'].value_counts()
# Data is imbalance as the number of people who quit is more
```

Out[40]:

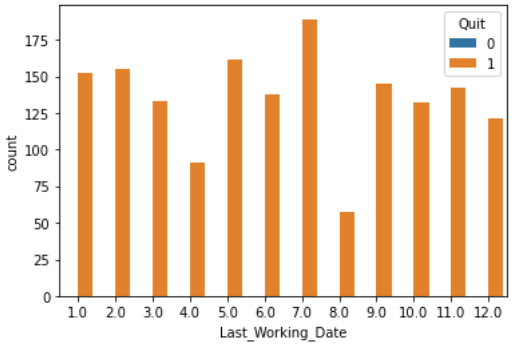
```
1    1616
0     765
Name: Quit, dtype: int64
```

In [41]:

```
sns.countplot(x=df10['Last_Working_Date'].dt.month,hue=df10['Quit'])
# Drivers quit the most in the middle of the year
```

Out[41]:

<AxesSubplot:xlabel='Last_Working_Date', ylabel='count'>

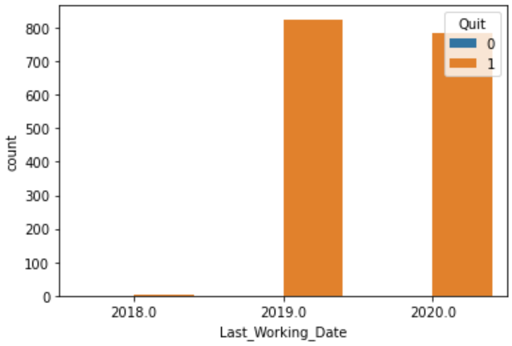


In [42]:

```
sns.countplot(x=df10['Last_Working_Date'].dt.year,hue=df10['Quit'])
# Data is avaialble of the drivers who quit in 2019 and 2020 years
```

Out[42]:

<AxesSubplot:xlabel='Last_Working_Date', ylabel='count'>

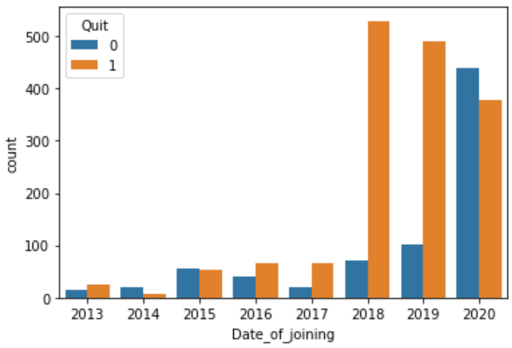


In [43]:

```
sns.countplot(x=df10['Date_of_joining'].dt.year,hue=df10['Quit'])
# Drivers who have joint recently in years 2018,2019 and 2020 have quit the most
```

Out[43]:

<AxesSubplot:xlabel='Date_of_joining', ylabel='count'>

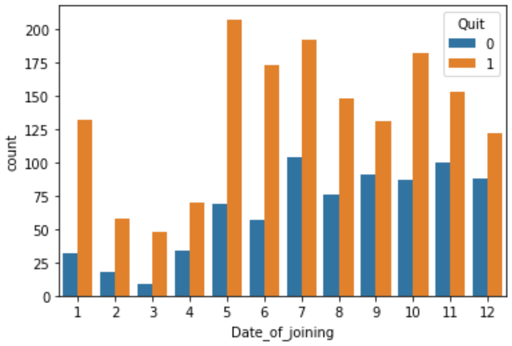


In [44]:

```
sns.countplot(x=df10['Date_of_joining'].dt.month,hue=df10['Quit'])
```

Out[44]:

```
<AxesSubplot:xlabel='Date_of_joining', ylabel='count'>
```



In [45]:

```
df10['Joining Designation'].value_counts(normalize=True)
# Drivers join with designation 1,2 or 3 the most
```

Out[45]:

```
1    0.430911
2    0.342293
3    0.207056
4    0.015120
5    0.004620
Name: Joining Designation, dtype: float64
```

In [46]:

```
df10['Grade'].value_counts(normalize=True)
# Drivers join with grade 1,2 or 3 the most
```

Out[46]:

```
2    0.363713
1    0.315414
3    0.256615
4    0.055439
5    0.008820
Name: Grade, dtype: float64
```

In [47]:

```
df10['Quarterly_Rating_Value'].value_counts(normalize=True)
# 67.7% of the drivers Quarterly_Rating remained the same.
# For others it increased and decreased
```

Out[47]:

```
 0    0.677866
-1    0.151617
 1    0.115078
 2    0.025199
-2    0.024360
-3    0.004200
 3    0.001680
Name: Quarterly_Rating_Value, dtype: float64
```

In [48]:

```
dff=df10.copy('deep')
```

In [49]:

```
dr=['Driver_ID','Date_of_joining','Last_Working_Date','Max_Date','Min_Date','No_of_days_served']
```

In [50]:

```
dff.drop(dr,axis=1,inplace=True)
# Dropping columns from dr as they are not needed
```

In [51]:

```
dff.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2381 entries, 0 to 2380
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Quarterly_Rating_Value  2381 non-null   int64  
 1   Quarterly_Rating_Mean   2381 non-null   float64
 2   Age                    2381 non-null   float64
 3   Gender                 2381 non-null   float64
 4   City                   2381 non-null   object 
 5   Education_Level        2381 non-null   int64  
 6   Total Business Value   2381 non-null   int64  
 7   Joining Designation    2381 non-null   int64  
 8   Grade                  2381 non-null   int64  
 9   Date_Diff              2381 non-null   int64  
 10  First_Income           2381 non-null   int64  
 11  Last_Income            2381 non-null   int64  
 12  Income_diff            2381 non-null   int64  
 13  Quit                   2381 non-null   int64  
dtypes: float64(3), int64(10), object(1)
memory usage: 343.6+ KB
```

Removing outliers

In [52]:

```
def outliners(x,col):
    Q1 = np.percentile(x[col], 25)
    Q3 = np.percentile(x[col], 75)
    IQR = Q3 - Q1
    upper = Q3 +1.5*IQR
    lower = Q1 - 1.5*IQR
    #print(upper,lower)
    ls=list(x.iloc[((x[col]<lower) | (x[col]>upper)).values].index)
    return ls
```

In [53]:

```
len(outliners(dff,'Age'))/len(dff)
```

Out[53]:

```
0.010499790004199917
```

In [54]:

```
dff.drop(outliners(dff,'Age'),axis=0,inplace=True)
dff.reset_index(inplace=True)
dff.drop('index',axis=1,inplace=True)
```

In [55]:

```
len(outliners(dff,'First_Income'))/len(dff)
```

Out[55]:

```
0.02037351443123939
```

In [56]:

```
dff.drop(outliners(dff,'Last_Income'),axis=0,inplace=True)
dff.reset_index(inplace=True)
dff.drop('index',axis=1,inplace=True)
```

In [57]:

```
dff['Last_Income_bin']=dff['Last_Income'].apply(lambda x: 1 if x>1 else 0)
```

In [58]:

```
X= dff.drop('Quit',axis=1)
y=dff['Quit']
X = pd.get_dummies(X, columns = ['City'],drop_first=True)
# One hot encoding
```

In [59]:

```
scaler = StandardScaler()
X=scaler.fit_transform(X)
# Scaling data
```

In [60]:

```
sm=SMOTE(k_neighbors=7)
X,y=sm.fit_resample(pd.DataFrame(X),pd.DataFrame(y))
# Balancing the data
```

In [61]:

```
y.value_counts()
# Data is balanced
```

Out[61]:

```
Quit
0       1581
1       1581
dtype: int64
```

In [62]:

```
X.isnull().sum()
# there are no null values in any columns so no need for applying KNN
```

Out[62]:

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
11    0
12    0
13    0
14    0
15    0
16    0
17    0
18    0
19    0
20    0
21    0
22    0
23    0
24    0
25    0
26    0
27    0
28    0
29    0
30    0
31    0
32    0
33    0
34    0
35    0
36    0
37    0
38    0
39    0
40    0
dtype: int64
```

RandomForestClassifier

Hyperparameter tuning

In [63]:

```
para={ 'n_estimators':[80,90,100,120,130,150],
      'criterion':['log_loss'],
      'random_state':[9],
      'bootstrap':[True],
      'n_jobs':[-1],
      'class_weight':['balanced_subsample'],
      'max_features':[30,35,40,45],
      'ccp_alpha':[0,0.05,0.1],
}

ftwo_scorer = make_scorer(fbeta_score, beta=2)
```

In [64]:

```
rfc = RandomForestClassifier()
model=GridSearchCV(rfc,para,scoring=ftwo_scorer,cv=7,return_train_score=True, refit=True)
```

In [173]:

```
model.fit(X,y)
```

Out[173]:

```
GridSearchCV(cv=7, estimator=RandomForestClassifier(),
             param_grid={'bootstrap': [True], 'ccp_alpha': [0, 0.05, 0.1],
                         'class_weight': ['balanced_subsample'],
                         'criterion': ['log_loss'],
                         'max_features': [30, 35, 40, 45],
                         'n_estimators': [80, 90, 100, 120, 130, 150],
                         'n_jobs': [-1], 'random_state': [9]},
             return_train_score=True, scoring=make_scorer(fbeta_score, beta=2))
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [174]:

```
model.best_params_
# Best parameters for RandomForestClassifier are the following
```

Out[174]:

```
{'bootstrap': True,
 'ccp_alpha': 0,
 'class_weight': 'balanced_subsample',
 'criterion': 'log_loss',
 'max_features': 30,
 'n_estimators': 100,
 'n_jobs': -1,
 'random_state': 9}
```

In [175]:

```
model.best_score_
# Best model score is 0.88
```

Out[175]:

```
0.880337225187451
```

Cross Validation

In [65]:

```
scoring = ['precision', 'recall', 'f1']
rfc = RandomForestClassifier(n_estimators=100,max_features=30,
                             criterion='log_loss',random_state=9,
                             bootstrap=True, n_jobs=-1,
                             class_weight='balanced_subsample',ccp_alpha=0.0017)
cv_acc_results = cross_validate(rfc, X, y, cv = 7, scoring = scoring, return_train_score = True)

print('test_precision :', cv_acc_results['test_precision'].mean()*100)
print('train_precision :', cv_acc_results['train_precision'].mean()*100)
print('test_recall :', cv_acc_results['test_recall'].mean()*100)
print('train_recall :', cv_acc_results['train_recall'].mean()*100)
print('test_f1 :', cv_acc_results['test_f1'].mean()*100)
print('train_f1 :', cv_acc_results['train_f1'].mean()*100)
```

```
test_precision : 84.76867710561068
train_precision : 96.43363319358748
test_recall : 88.5517628880461
train_recall : 98.29228886473442
test_f1 : 86.20134989000819
train_f1 : 97.3533468951765
```

In [66]:

```
X= dff.drop('Quit',axis=1)
y=dff['Quit']
X = pd.get_dummies(X, columns = ['City'],drop_first=True)
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=7,test_size=0.2,shuffle=True)
scaler = StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)
```

In [67]:

```
rfc = RandomForestClassifier(n_estimators=100,max_features=30,criterion='log_loss',random_state=9,bootstrap=True, n_jobs=-1,class_weight=
```

In [68]:

```
rfc.fit(X_train,y_train)
# Applying the best parameter on the model
```

Out[68]:

```
RandomForestClassifier
RandomForestClassifier(ccp_alpha=0.0017, class_weight='balanced_subsample',
                       criterion='log_loss', max_features=30, n_jobs=-1,
                       oob_score=True, random_state=9)
```

In [69]:

```
fbeta_score(y_train, rfc.predict(X_train),beta=1),fbeta_score(y_test, rfc.predict(X_test),beta=2)
# F2 score for train and test data is below
```

Out[69]:

```
(0.9968454258675079, 0.8943348185868873)
```

In [70]:

```
print(classification_report(y_test, rfc.predict(X_test),labels=[0, 1]))
# Classification report for test data
# Recall on test is 0.91
# Precision on test is 0.84
```

```
              precision    recall  f1-score   support

           0       0.78      0.67      0.72       152
           1       0.85      0.91      0.88       310

    accuracy                           0.83       462
   macro avg       0.81      0.79      0.80       462
weighted avg       0.83      0.83      0.83       462
```

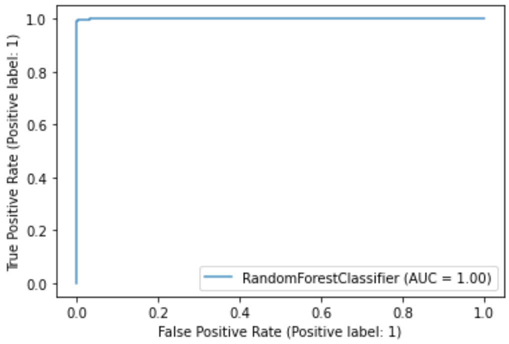
In [71]:

```
print(classification_report(y_train, rfc.predict(X_train),labels=[0, 1]))
# Classification report for train data
# Recall on test is 1
# Precision on test is 1
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       575
           1       1.00      0.99      1.00      1271

    accuracy                           1.00      1846
   macro avg       0.99      1.00      0.99      1846
weighted avg       1.00      1.00      1.00      1846
```

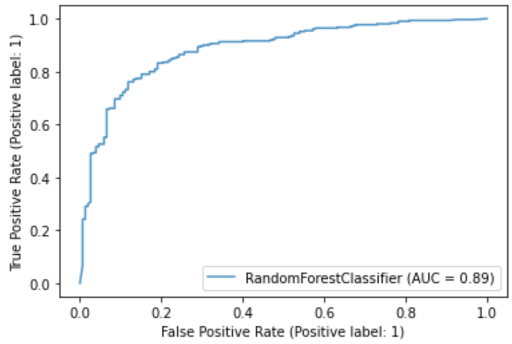
In [72]:

```
ax = plt.gca()
rfc_disp = RocCurveDisplay.from_estimator(rfc, X_train, y_train, ax=ax, alpha=0.8)
plt.show()
# ROC curve for train data
```



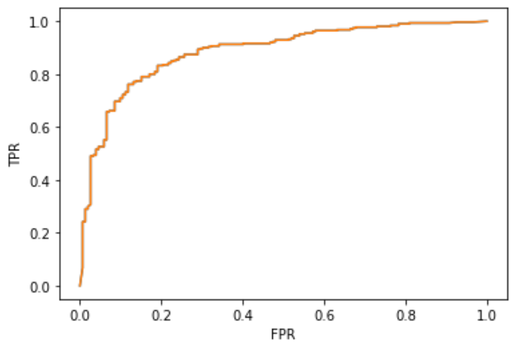
In [73]:

```
ax = plt.gca()
tree_clf_disp = RocCurveDisplay.from_estimator(rfc, X_test, y_test, ax=ax, alpha=0.8)
plt.show()
# ROC curve for train data. AUC=0.88
```



In [74]:

```
fpr,tpr,thres=roc_curve(y_test,rfc.predict_proba(X_test)[:,1])
plt.plot(fpr,tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.plot(fpr,tpr)
plt.show()
# TPR vs FPR
```



In [75]:

```
cm = confusion_matrix(y_train, rfc.predict(X_train))
print ("Confusion Matrix : \n", cm)
```

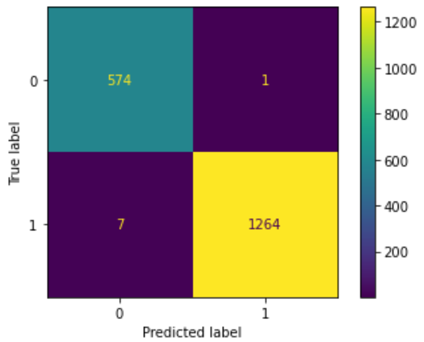
```
Confusion Matrix :
 [[ 574    1]
 [   7 1264]]
```

In [76]:

```
ConfusionMatrixDisplay(cm).plot()
# Confusion matrix on train data
```

Out[76]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c250cee160>
```

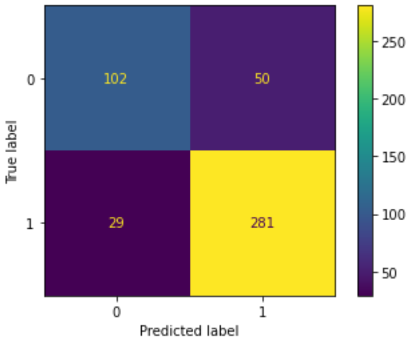


In [77]:

```
cm = confusion_matrix(y_test, rfc.predict(X_test))
ConfusionMatrixDisplay(cm).plot()
# Confusion matrix on test data
```

Out[77]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c250d243d0>
```



In [78]:

```
rfc.feature_importances_
```

Out[78]:

```
array([1.18955107e-01, 6.43656432e-02, 7.23689155e-02, 1.24235152e-02,
       2.59161034e-02, 9.29525481e-02, 5.52420858e-02, 1.07106570e-02,
       2.73455607e-01, 6.95264173e-02, 6.74618228e-02, 2.27033881e-04,
       0.00000000e+00, 4.17693491e-03, 2.43933601e-03, 6.06884489e-03,
       3.73328054e-03, 4.48711331e-03, 5.04547034e-03, 5.52417668e-03,
       3.80217851e-03, 3.40895030e-03, 6.35411684e-03, 6.39608043e-03,
       6.46305052e-03, 5.22959812e-03, 6.21720955e-03, 4.56556576e-03,
       4.99135077e-03, 7.34452747e-03, 3.28308132e-03, 4.77748238e-03,
       2.42961551e-03, 6.54890957e-03, 7.09634533e-03, 4.41908788e-03,
       5.49286128e-03, 5.09099535e-03, 2.99086634e-03, 5.52735901e-03,
       2.49015509e-03])
```

In [79]:

```
arr=rfc.feature_importances_.reshape(-1)
col=X.columns
dic={}
for i in range(len(col)-1):
    dic[col[i]]=arr[i]
dict(sorted(dic.items(), key=lambda item: abs(item[1])))
# Date_Diff,Quarterly_Rating_Value,Total Business Value,Quarterly_Rating_Mean has the highest importances
```

Out[79]:

```
{'Last_Income_bin': 0.0,
 'Income_diff': 0.00022703388085778208,
 'City_C28': 0.0024296155051501756,
 'City_C11': 0.0024393360100922914,
 'City_C7': 0.0029908663353631413,
 'City_C26': 0.003283081321633581,
 'City_C18': 0.0034089503035573294,
 'City_C13': 0.0037332805435759445,
 'City_C17': 0.0038021785094058376,
 'City_C10': 0.004176934911548122,
 'City_C4': 0.004419087883660952,
 'City_C14': 0.004487113308538201,
 'City_C23': 0.004565565757409881,
 'City_C27': 0.00477748238021075,
 'City_C24': 0.004991350768462313,
 'City_C15': 0.005045470343429957,
 'City_C6': 0.0050909953508729055,
 'City_C21': 0.005229598118545529,
 'City_C5': 0.005492861280981417,
 'City_C16': 0.005524176682662261,
 'City_C8': 0.005527359104421079,
 'City_C12': 0.006068844893276041,
 'City_C22': 0.006217209551254903,
 'City_C19': 0.006354116840445133,
 'City_C2': 0.006396080428722562,
 'City_C20': 0.006463050518787476,
 'City_C29': 0.006548909566757556,
 'City_C3': 0.007096345334232528,
 'City_C25': 0.0073445274704890436,
 'Grade': 0.010710656961271174,
 'Gender': 0.012423515245092328,
 'Education_Level': 0.025916103438292357,
 'Joining Designation': 0.05524208577097585,
 'Quarterly_Rating_Mean': 0.06436564319843929,
 'Last_Income': 0.06746182284825794,
 'First_Income': 0.06952641726556921,
 'Age': 0.07236891551083584,
 'Total Business Value': 0.09295254814036055,
 'Quarterly_Rating_Value': 0.1189551069429439,
 'Date_Diff': 0.27345560677547226}
```

GradientBoostingClassifier

Hyperparameter tuning

In [80]:

```
para={'loss':['log_loss'],
'learning_rate':[0.05,0.1,0.15],
'n_estimators':[200,260,265,270,280,300],
'subsample':[0.7,0.8,0.9,1],
'criterion':['friedman_mse'],
'random_state':[9],
'max_features':[10,20,30,40,45],
'ccp_alpha':[0,0.1,0.15]
}
```

In [81]:

```
ftwo_scorer = make_scorer(fbeta_score, beta=2)
```

In [82]:

```
tree_clf = GradientBoostingClassifier()
model=GridSearchCV(tree_clf,para,scoring=ftwo_scorer,cv=7,return_train_score=True)
```

In [132]:

```
model.fit(X,y)
```

Out[132]:

```
GridSearchCV(cv=7, estimator=GradientBoostingClassifier(),
             param_grid={'ccp_alpha': [0, 0.1, 0.15],
                         'criterion': ['friedman_mse'],
                         'learning_rate': [0.05, 0.1, 0.15],
                         'loss': ['log_loss'],
                         'max_features': [10, 20, 30, 40, 45],
                         'n_estimators': [200, 260, 265, 270, 280, 300],
                         'random_state': [9], 'subsample': [0.7, 0.8, 0.9, 1]},
             return_train_score=True, scoring=make_scorer(fbeta_score, beta=2))
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [133]:

```
model.best_params_
# Best parameters for RandomForestClassifier are the following
```

Out[133]:

```
{'ccp_alpha': 0,
 'criterion': 'friedman_mse',
 'learning_rate': 0.15,
 'loss': 'log_loss',
 'max_features': 45,
 'n_estimators': 200,
 'random_state': 9,
 'subsample': 0.9}
```

In [134]:

```
model.best_score_
# Best Score
```

Out[134]:

```
0.897292421271724
```

Cross Validation

In [83]:

```
tree_clf = GradientBoostingClassifier(ccp_alpha=0,random_state=9, n_estimators= 200, learning_rate = 0.15,criterion='friedman_mse',loss='l
cv_acc_results = cross_validate(tree_clf, X, y, cv = 7, scoring = scoring, return_train_score = True)
print('test_precision :', cv_acc_results['test_precision'].mean()*100)
print('train_precision :', cv_acc_results['train_precision'].mean()*100)
print('test_recall :', cv_acc_results['test_recall'].mean()*100)
print('train_recall :', cv_acc_results['train_recall'].mean()*100)
print('test_f1 :', cv_acc_results['test_f1'].mean()*100)
print('train_f1 :', cv_acc_results['train_f1'].mean()*100)
```

```
test_precision : 86.34422691813619
train_precision : 94.64469616922545
test_recall : 91.39907290349768
train_recall : 98.17629295129868
test_f1 : 88.78871192563766
train_f1 : 96.37780042935609
```

In [84]:

```
X= dff.drop('Quit',axis=1)
y=dff['Quit']
X = pd.get_dummies(X, columns = ['City'],drop_first=True)
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=7,test_size=0.2,shuffle=True)
scaler = StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)
```

In [85]:

```
tree_clf = GradientBoostingClassifier(ccp_alpha=0,random_state=9, n_estimators= 200, learning_rate = 0.15,criterion='friedman_mse',loss='l
```

In [86]:

```
tree_clf.fit(X_train,y_train)
# Applying the best parameter on the model
```

Out[86]:

```
GradientBoostingClassifier

GradientBoostingClassifier(ccp_alpha=0, learning_rate=0.15, max_features=45,
                           n_estimators=200, random_state=9, subsample=0.9)
```

In [87]:

```
fbeta_score(y_train, tree_clf.predict(X_train),beta=1),fbeta_score(y_test, tree_clf.predict(X_test),beta=2)
# F2 score for train and test data is 0.9624 and 0.882 respectively
```

Out[87]:

```
(0.9653579676674365, 0.9001272264631044)
```

In [88]:

```
print(classification_report(y_test, tree_clf.predict(X_test),labels=[0, 1]))
# Classification report for test data
# Recall on test is 0.91
# Precision on test is 0.85
```

```
              precision    recall  f1-score   support

           0       0.79      0.68      0.73       152
           1       0.85      0.91      0.88       310

    accuracy                           0.84       462
   macro avg       0.82      0.80      0.81       462
weighted avg       0.83      0.84      0.83       462
```

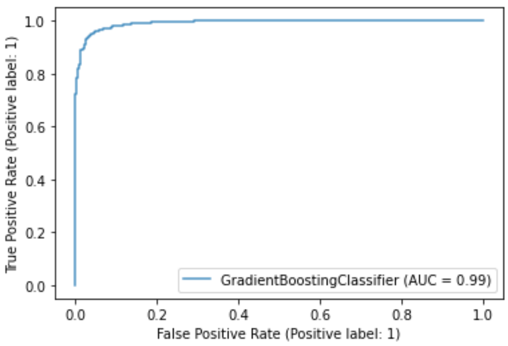
In [89]:

```
print(classification_report(y_train, tree_clf.predict(X_train),labels=[0, 1]))
# Classification report for test data
# Recall on test is 0.99
# Precision on test is 0.94
```

```
              precision    recall  f1-score   support

           0       0.97      0.87      0.92       575
           1       0.94      0.99      0.97      1271

    accuracy                           0.95      1846
   macro avg       0.96      0.93      0.94      1846
weighted avg       0.95      0.95      0.95      1846
```

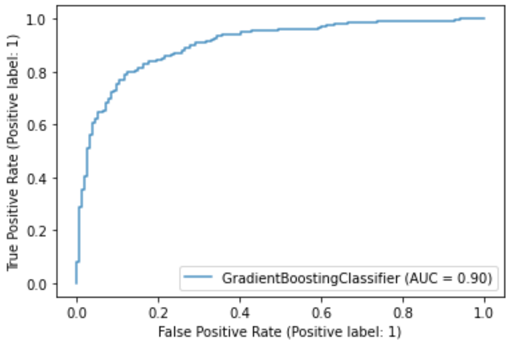
In [90]:

```
ax = plt.gca()
tree_clf_disp = RocCurveDisplay.from_estimator(tree_clf, X_train, y_train, ax=ax, alpha=0.8)
plt.show()
# ROC curve for train data
# AOC = 0.99
```



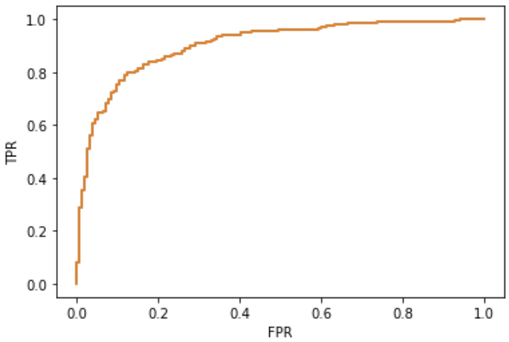
In [91]:

```
ax = plt.gca()
tree_clf_disp = RocCurveDisplay.from_estimator(tree_clf, X_test, y_test, ax=ax, alpha=0.8)
plt.show()
# ROC curve for test data
# AOC = 0.90
```



In [92]:

```
fpr,tpr,thres=roc_curve(y_test,tree_clf.predict_proba(X_test)[:,1])
plt.plot(fpr,tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.plot(fpr,tpr)
plt.show()
# TPR vs FPR
```



In [93]:

```
cm = confusion_matrix(y_train, tree_clf.predict(X_train))
print ("Confusion Matrix : \n", cm)
```

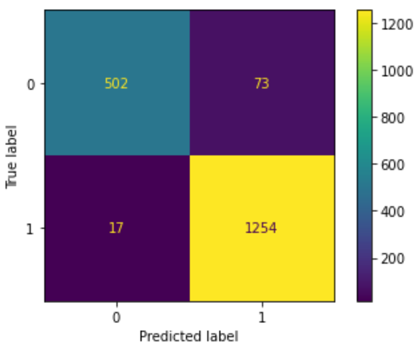
```
Confusion Matrix :
 [[ 502   73]
 [  17 1254]]
```

In [94]:

```
ConfusionMatrixDisplay(cm).plot()
# Confusion matrix on train data
```

Out[94]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c24f4e3a90>
```

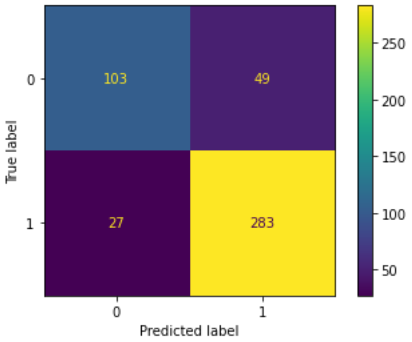


In [95]:

```
cm = confusion_matrix(y_test, tree_clf.predict(X_test))
ConfusionMatrixDisplay(cm).plot()
# Confusion matrix on test data
```

Out[95]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c250ce8670>
```



In [96]:

```
tree_clf.feature_importances_
```

Out[96]:

```
array([1.76998423e-01, 7.64102890e-02, 2.77318057e-02, 3.61024300e-03,
       6.79818561e-03, 7.85996593e-02, 6.05828986e-02, 4.34380791e-03,
       4.06874468e-01, 4.69955736e-02, 4.53284984e-02, 4.65320904e-04,
       0.00000000e+00, 1.26786887e-03, 1.58654157e-03, 2.95123476e-03,
       2.99375475e-03, 7.84991567e-04, 1.71624653e-03, 2.81287718e-03,
       2.31003114e-03, 3.29599080e-04, 2.64830528e-03, 5.79813360e-03,
       1.10894622e-03, 1.15353237e-03, 2.74264738e-03, 3.71008928e-03,
       2.48563010e-03, 4.73491615e-03, 1.78959331e-03, 1.97605792e-03,
       8.94296656e-04, 2.17190662e-03, 4.85939092e-03, 2.05735369e-03,
       3.07134746e-03, 2.97536914e-03, 8.48325361e-04, 3.13680121e-03,
       3.45039525e-04])
```

In [97]:

```
arr=tree_clf.feature_importances_.reshape(-1)
col=X.columns
dic={}
for i in range(len(col)-1):
    dic[col[i]]=arr[i]
dict(sorted(dic.items(), key=lambda item: abs(item[1])))
```

Out[97]:

```
{'Last_Income_bin': 0.0,
 'City_C18': 0.00032959908027993344,
 'Income_diff': 0.0004653209037284973,
 'City_C14': 0.0007849915671416656,
 'City_C7': 0.0008483253611701909,
 'City_C28': 0.0008942966563178738,
 'City_C20': 0.0011089462225296943,
 'City_C21': 0.0011535323656577687,
 'City_C10': 0.0012678688713173545,
 'City_C11': 0.0015865415713938033,
 'City_C15': 0.0017162465278863634,
 'City_C26': 0.001789593313648574,
 'City_C27': 0.001976057920546707,
 'City_C4': 0.00205735368501313,
 'City_C29': 0.002171906615491024,
 'City_C17': 0.0023100311424338175,
 'City_C24': 0.00248563009734803,
 'City_C19': 0.002648305275330649,
 'City_C22': 0.0027426473775899293,
 'City_C16': 0.002812877179366813,
 'City_C12': 0.0029512347639128514,
 'City_C6': 0.0029753691358416073,
 'City_C13': 0.0029937547465057,
 'City_C5': 0.003071347459749465,
 'City_C8': 0.003136801207576888,
 'Gender': 0.003610243002692875,
 'City_C23': 0.003710089277130706,
 'Grade': 0.004343807907174628,
 'City_C25': 0.004734916154211861,
 'City_C3': 0.00485939091681111,
 'City_C2': 0.005798133600028769,
 'Education_Level': 0.006798185614391292,
 'Age': 0.02773180571849187,
 'Last_Income': 0.04532849844686002,
 'First_Income': 0.04699557363165848,
 'Joining Designation': 0.060582898559427875,
 'Quarterly_Rating_Mean': 0.07641028904813203,
 'Total Business Value': 0.07859965926746816,
 'Quarterly_Rating_Value': 0.17699842258418666,
 'Date_Diff': 0.4068744676985437}
```

In [98]:

```
# Date_Diff,Quarterly_Rating_Value,Total Business Value,Quarterly_Rating_Mean has the highest importances
```

In [99]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
X =pd.DataFrame(X_train)
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns[:]
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns[:]))]
print(vif_data)
```

```
    feature       VIF
0         0  1.081996
1         1  2.680243
2         2  1.162398
3         3  1.023740
4         4  1.074669
5         5  4.141454
6         6  3.650826
7         7  5.230941
8         8  3.288081
9         9       inf
10       10       inf
11       11       inf
12       12       NaN
13       13  2.090993
14       14  1.834531
15       15  2.070430
16       16  1.950165
17       17  2.009394
18       18  2.330057
19       19  2.138600
20       20  1.887860
21       21  1.835241
22       22  1.916419
23       23  2.002449
24       24  2.894539
25       25  1.969786
26       26  2.087235
27       27  1.911325
28       28  1.883721
29       29  2.010851
30       30  2.136575
31       31  2.138734
32       32  2.026161
33       33  2.146836
34       34  2.138147
35       35  2.057022
36       36  1.963833
37       37  2.119156
38       38  1.962931
39       39  2.108456
40       40  1.996470
```

In [ ]:

In [ ]: