In [ ]:
```
from google.colab import files
files.upload()
```

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math as m
from sklearn.impute import KNNImputer
import warnings

warnings.filterwarnings('ignore')
```

In [3]:
```
df1=pd.read_csv('ola_driver_scaler.csv')
df1.head()
```

Out[3]:

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

In [ ]:
```
df1.shape
```

Out[ ]: (19104, 14)

In [ ]:
```
df1.describe()
```

Out[ ]:

	Unnamed: 0	Driver_ID	Age	Gender	Education_Lev
count	19104.000000	19104.000000	19043.000000	19052.000000	19104.00000
mean	9551.500000	1415.591133	34.668435	0.418749	1.02167
std	5514.994107	810.705321	6.257912	0.493367	0.80016
min	0.000000	1.000000	21.000000	0.000000	0.00000
25%	4775.750000	710.000000	30.000000	0.000000	0.00000
50%	9551.500000	1417.000000	34.000000	0.000000	1.00000
75%	14327.250000	2137.000000	39.000000	1.000000	2.00000
max	19103.000000	2788.000000	58.000000	1.000000	2.00000

In [ ]:
```
df1.info()
```

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

In [ ]:
```
df1['Unnamed: 0'].describe()
```

Out[ ]:

	Unnamed: 0
count	19104.000000
mean	9551.500000
std	5514.994107
min	0.000000
25%	4775.750000
50%	9551.500000
75%	14327.250000
max	19103.000000

dtype: float64

In [4]:
```
#we can drop the column Unnamed: 0,since there is no correlation with driver
df1.drop('Unnamed: 0',axis=1,inplace=True)
```

Data Processing and featuring Engineering

In [5]:
```
#convert date columns
df1['LastWorkingDate']=pd.to_datetime(df1['LastWorkingDate'])
```

```
df1['Dateofjoining']=pd.to_datetime(df1['Dateofjoining'])
```

In [6]:
```
df1.isna().sum()
```

Out[6]:

	0
MMM-YY	0
Driver_ID	0
Age	61
Gender	52
City	0
Education_Level	0
Income	0
Dateofjoining	0
LastWorkingDate	17488
Joining Designation	0
Grade	0
Total Business Value	0
Quarterly Rating	0

dtype: int64

In [ ]:
```
#we have null values in Age,Gender
#and LastWorkingDate means the driver has not left the company
```

Impute missing values using KNN imputer

In [7]:
```
num_vars = df1.select_dtypes(np.number)

num_vars.columns
```

Out[7]:
```
Index(['Driver_ID', 'Age', 'Gender', 'Education_Level', 'Income',
       'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating'],
      dtype='object')
```

In [8]:
```
num_vars.drop('Driver_ID',axis=1,inplace=True)
```

In [9]:
```
imputer = KNNImputer(n_neighbors=5, weights='uniform', metric='nan_euclidear
imputer.fit(num_vars)
data_new = imputer.transform(num_vars)
```

In [10]:
```
data_new = pd.DataFrame(data_new)
data_new.columns = num_vars.columns
data_new.isna().sum()
```

Out[10]:

	0
Age	0
Gender	0
Education_Level	0
Income	0
Joining Designation	0
Grade	0
Total Business Value	0
Quarterly Rating	0

dtype: int64

In [11]:
```
#merge 2 KNNdataframes and actual dataframe
resultant_columns = list(set(df1.columns).difference(set(num_vars)))

resultant_columns
```

Out[11]:
```
['LastWorkingDate', 'Driver_ID', 'Dateofjoining', 'City', 'MMM-YY']
```

In [12]:
```
df = pd.concat([data_new, df1[resultant_columns]], axis=1)

df.shape
```

Out[12]:
```
(19104, 13)
```

In [13]:
```
ola=df.copy()
```

In [14]:
```
df.isna().sum()
```

Out[14]:

	0
Age	0
Gender	0
Education_Level	0
Income	0
Joining Designation	0
Grade	0
Total Business Value	0
Quarterly Rating	0
LastWorkingDate	17488
Driver_ID	0
Dateofjoining	0
City	0
MMM-YY	0

dtype: int64

Create Target feature

In [15]:

```
Target=df.groupby('Driver_ID').aggregate({'LastWorkingDate':'last'})['LastWo
Target['LastWorkingDate'].replace({True:1,False:0},inplace=True)
Target.rename(columns={'LastWorkingDate':'Target'},inplace=True)
Target.head()
```

Out[15]:

	Driver_ID	Target
0	1	0
1	2	1
2	4	0
3	5	0
4	6	1

Create Quarterly increase feature

In [16]:

```
QR1=df.groupby('Driver_ID').aggregate({'Quarterly Rating':'first'}).reset_ir
QR2=df.groupby('Driver_ID').aggregate({'Quarterly Rating':'last'}).reset_inc
QR=QR1.merge(QR2,on='Driver_ID')
#create Quaterly increase flag
QR['Got Quaterly Rating']=np.where(QR['Quarterly Rating_x']==QR['Quarterly F
QR.head()
```

Out[16]:

	Driver_ID	Quarterly Rating_x	Quarterly Rating_y	Got Quaterly Rating
0	1	2.0	2.0	0
1	2	1.0	1.0	0
2	4	1.0	1.0	0
3	5	1.0	1.0	0
4	6	1.0	2.0	1

Create Income increase feature

In [17]:

```
in1=df.groupby('Driver_ID').aggregate({'Income':'first'}).reset_index()
in2=df.groupby('Driver_ID').aggregate({'Income':'last'}).reset_index()
IN=in1.merge(in2,on='Driver_ID')
#create monthly income increase flag
IN['Got Income hike']=np.where(IN['Income_x']==IN['Income_y'],0,1)
IN.head()
```

Out[17]:

	Driver_ID	Income_x	Income_y	Got Income hike
0	1	57387.0	57387.0	0
1	2	67016.0	67016.0	0
2	4	65603.0	65603.0	0
3	5	46368.0	46368.0	0
4	6	78728.0	78728.0	0

In [ ]:

```
df.columns
```

Out[ ]:

```
Index(['MMM-YY', 'Age', 'Gender', 'City', 'Education_Level', 'Dateofjoinin
g',
       'LastWorkingDate', 'Grade', 'Total Business Value', 'Income',
       'Joining Designation', 'Quarterly Rating'],
      dtype='object')
```

In [18]:
```
#aggregate the dataset based on driver id
functions = {'MMM-YY':'count',
             'Driver_ID':'first',
             'Age':'max',
             'Gender':'last',
            'City':'last',
             'Education_Level':'last',
             'Dateofjoining':'first',
            'LastWorkingDate':'last',
             'Grade':'last',
             'Total Business Value':'sum',
            'Income':'sum',
            'Joining Designation':'last',
             'Quarterly Rating':'first'}
df = df.groupby([df['Driver_ID']]).aggregate(functions)
```

In [ ]:
```
df.drop('Driver_ID',axis=1,inplace=True)
df.reset_index(inplace=True)
#df.head()
```

In [23]:
```
#merge all df and new columns
df=df.merge(Target,on='Driver_ID')
df=df.merge(QR,on='Driver_ID')
df=df.merge(IN,on='Driver_ID')
df.head()
```

Out[23]:

	Driver_ID	MMM-YY	Age	Gender	City	Education_Level	Dateofjoining	LastW
0	1	3	28.0	0.0	C23	2.0	2018-12-24	
1	2	2	31.0	0.0	C7	2.0	2020-11-06	
2	4	5	43.0	0.0	C13	2.0	2019-12-07	
3	5	3	29.0	0.0	C9	0.0	2019-01-09	
4	6	5	31.0	1.0	C11	1.0	2020-07-31	

In [24]:
```
df['Join month']=df['Dateofjoining'].dt.month
df['Join year']=df['Dateofjoining'].dt.year
```

In [25]:
```
#drop unneccesary columns
df.drop(['Quarterly Rating_x','Quarterly Rating_y','Income_x','Income_y','Da
df.rename(columns={'MMM-YY':'No of Reprotings'},inplace=True)
df.head()
```

Out[25]:

	Driver_ID	No of Reprotings	Age	Gender	City	Education_Level	Grade	Total Business Value
0	1	3	28.0	0.0	C23	2.0	1.0	1715580
1	2	2	31.0	0.0	C7	2.0	2.0	0
2	4	5	43.0	0.0	C13	2.0	2.0	350000
3	5	3	29.0	0.0	C9	0.0	1.0	120360
4	6	5	31.0	1.0	C11	1.0	3.0	1265000

In [26]:

```
#Converting categorical values to numeric values
df['Age']=df['Age'].astype(int)
df['Gender']=df['Gender'].astype(int)
df['City']=df['City'].str.replace('C','')
df['City']=df['City'].astype(int)
df['Education_Level']=df['Education_Level'].astype(int)
df['Grade']=df['Grade'].astype(int)
df['Joining Designation']=df['Joining Designation'].astype(int)
df['Quarterly Rating']=df['Quarterly Rating'].astype(int)
df.head()
```

Out[26]:

	Driver_ID	No of Reprotings	Age	Gender	City	Education_Level	Grade	Total Business Value
0	1	3	28	0	23	2	1	1715580
1	2	2	31	0	7	2	2	0
2	4	5	43	0	13	2	2	350000
3	5	3	29	0	9	0	1	120360
4	6	5	31	1	11	1	3	1265000

In [ ]:

```
df.shape
```

Out[ ]:

```
(2381, 16)
```

In [27]:

```
df.isna().sum()
```

Out[27]:

	0
Driver_ID	0
No of Reprotings	0
Age	0
Gender	0
City	0
Education_Level	0
Grade	0
Total Business Value	0
Income	0
Joining Designation	0
Quarterly Rating	0
Target	0
Got Quaterly Rating	0
Got Income hike	0
Join month	0
Join year	0

dtype: int64

EXploratory Data Analysis

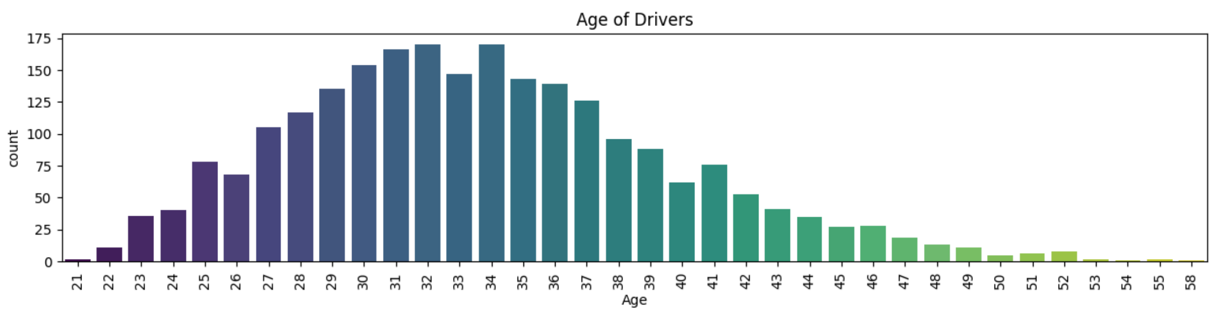
Univariate Analysis

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

Out[ ]:

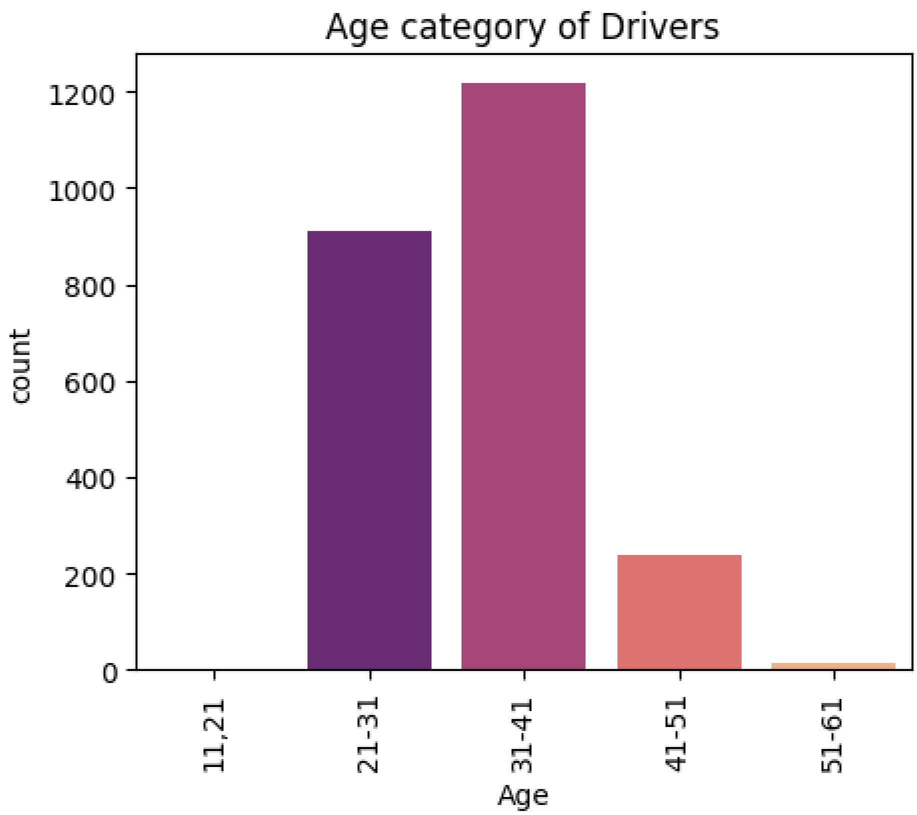
	Driver_ID	No of Reprotings	Age	Gender	City	Educa
count	2381.000000	2381.00000	2381.000000	2381.000000	2381.000000	2
mean	1397.559009	8.02352	33.762285	0.409492	15.335573	
std	806.161628	6.78359	5.933364	0.491843	8.371843	
min	1.000000	1.00000	21.000000	0.000000	1.000000	
25%	695.000000	3.00000	30.000000	0.000000	8.000000	
50%	1400.000000	5.00000	33.000000	0.000000	15.000000	
75%	2100.000000	10.00000	37.000000	1.000000	22.000000	
max	2788.000000	24.00000	58.000000	1.000000	29.000000	

In [ ]:
```
#Age distribution of drivers

plt.figure(figsize=(15,3))
sns.countplot(x=df.Age,palette='viridis')
plt.title('Age of Drivers')
plt.xticks(rotation=90)
plt.show()
```



In [ ]:
```
#create age bins
age_bins=[11,21,31,41,51,61]
age_lables = ['11,21','21-31','31-41','41-51','51-61']
Age_Cat= pd.cut(df['Age'], bins=age_bins, labels = age_lables)
Age_Cat
```

In [ ]:
```
plt.figure(figsize=(5,4))
sns.countplot(x=Age_Cat,palette='magma')
plt.title('Age category of Drivers')
plt.xticks(rotation=90)
plt.show()
```



From the above graph,we can see there is right skew distribution of age.

And we can see most of the drivers are between the age **31 to 41**

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

Out[ ]:

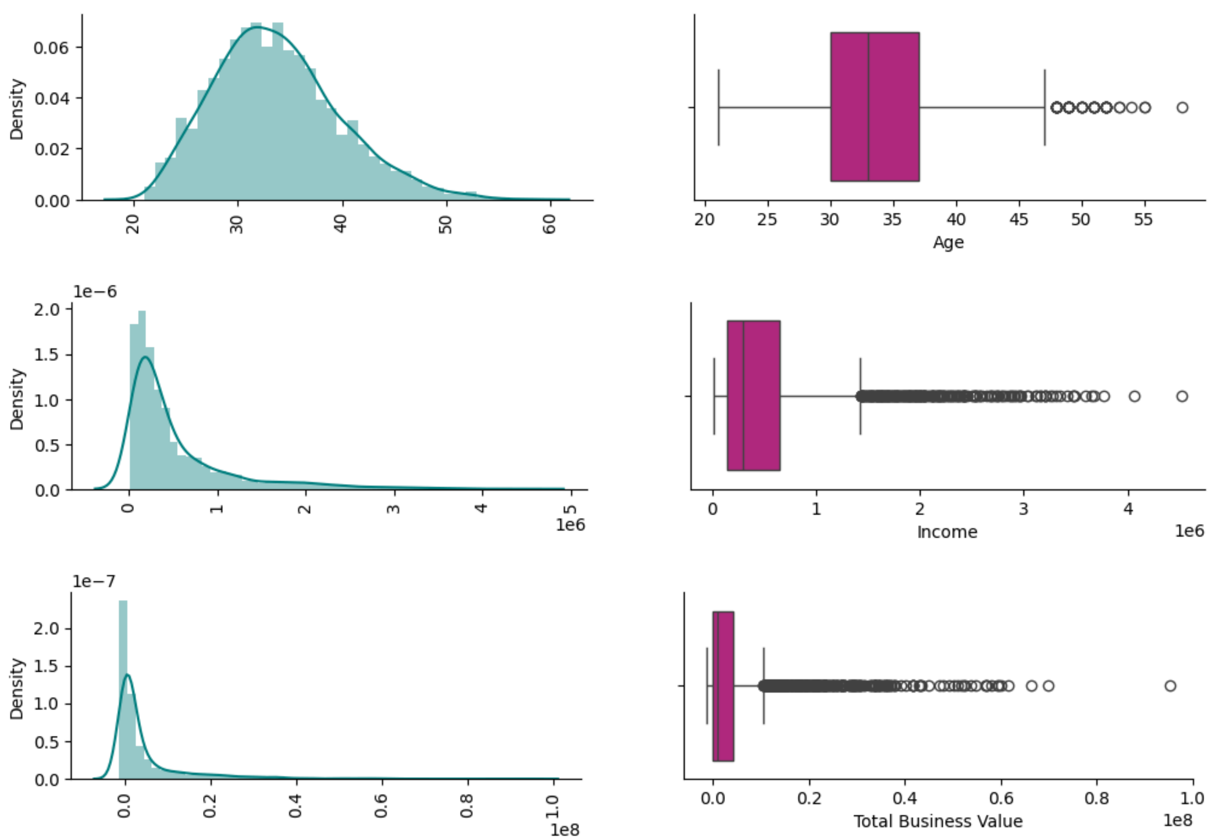
	Driver_ID	No of Reprotings	Age	Gender	City	Education_Level	Grade	Total Business Value
0	1	3	28	0	23	2	1	1715580
1	2	2	31	0	7	2	2	0
2	4	5	43	0	13	2	2	350000
3	5	3	29	0	9	0	1	120360
4	6	5	31	1	11	1	3	1265000

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['Driver_ID', 'No of Reprotings', 'Age', 'Gender', 'City',
       'Education_Level', 'Grade', 'Total Business Value', 'Income',
       'Joining Designation', 'Quarterly Rating', 'Target',
       'Got Quaterly Rating', 'Got Income hike', 'Join month', 'Join yea
r'],
      dtype='object')
```

In [ ]:

```python
b =df[['Age','Income','Total Business Value']]
for i in b:
    plt.figure(figsize=(12,2))
    plt.subplot(121)
    sns.distplot(x=df[i],color='teal')
    plt.title('')
    plt.xticks(rotation=90)

    plt.subplot(122)
    sns.boxplot(x=df[i],color='mediumvioletred')
    plt.title('')
    sns.despine()
    plt.show()
```



From the above graph,we can see few outlier are present in the Age and Total business value,So we need to outlier treatment

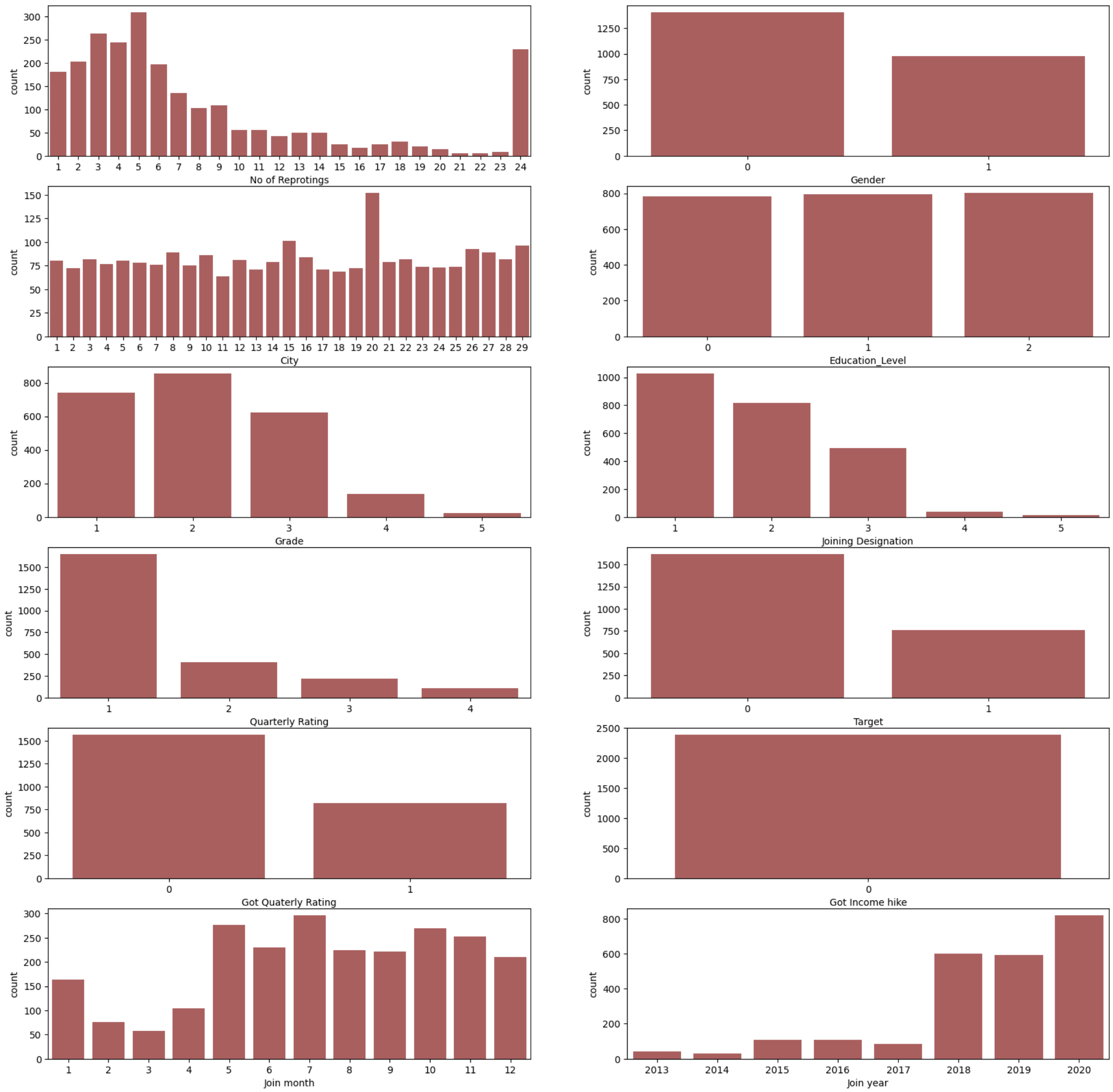
In [28]:

```python
cat_cols=['No of Reprotings','Gender', 'City',
       'Education_Level', 'Grade',
       'Joining Designation', 'Quarterly Rating', 'Target',
       'Got Quaterly Rating', 'Got Income hike', 'Join month', 'Join year']
```

In [ ]:

```python
# countplots for categorical variables
cols, rows = 2, 6
fig, axs = plt.subplots(rows, cols, figsize=(20, 20))

index = 0
for row in range(rows):
    for col in range(cols):
```

```
        plt.title('')
        sns.countplot(x=cat_cols[index], data=df, ax=axs[row, col], alpha=0.
        index += 1

plt.show()
```



In [58]:
```
df['Target'].value_counts()
```

Out[58]:

Target	count
0	1616
1	765

dtype: int64

Out of 2381 drivers,**1616** drivers were left the company

We can see the maximum **no of reprotings are 5 and 24**.

We can **male** drivers are **more** than female drivers.

Most of the drivers are from the **citycode 20**.

We can there is **same** no of drivers are in distributed in all 3 **educational** levels.

We can see the quaterly hike are given to only **1/3** of drivers

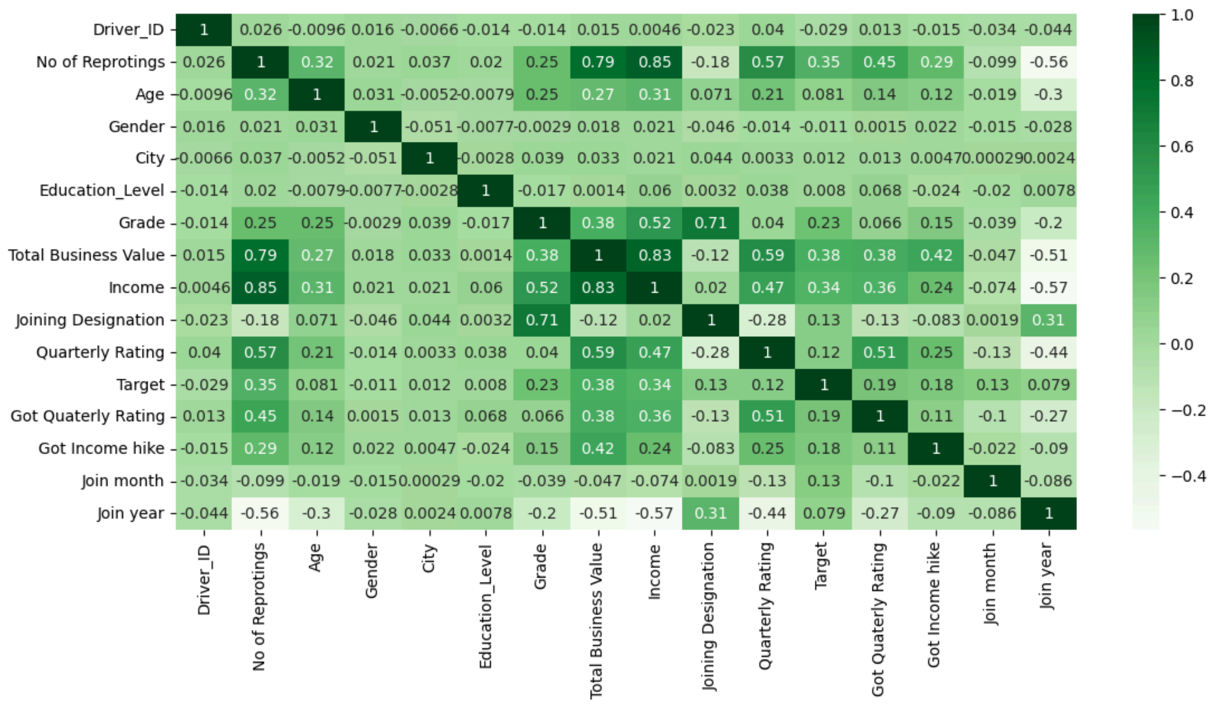
And Income hike are **not even provided to single** driver,this might be the important feature for chrun.We will further analysis for more accurate prediction.

And Most of the drivers are joined at every financial months(**july** and **December**).

There is the drastic increase in the count od drivers joined from **2017 to 2018**.

Bivariate Analysis

In [32]:
```
corr = df.corr()
plt.figure(figsize=(13,6))
sns.heatmap(corr,annot=True,cmap='Greens')
plt.show()
```



From the corelation matrix,we can see driver chrun(target) is highly correlated with Total business value.

Age and Quarterly Rating are positivly correlated.

Total business value and No of reportings also positvly higly correlated.

Joining degination is negaitvley correlated with Quarterly Rating and monthly income hike

In [37]:
```
grouped_rating = df.groupby('Quarterly Rating')['No of Reprotings'].sum().re
grouped_rating['No of Reprotings'] =(grouped_rating['No of Reprotings']/sum(
grouped_rating
```

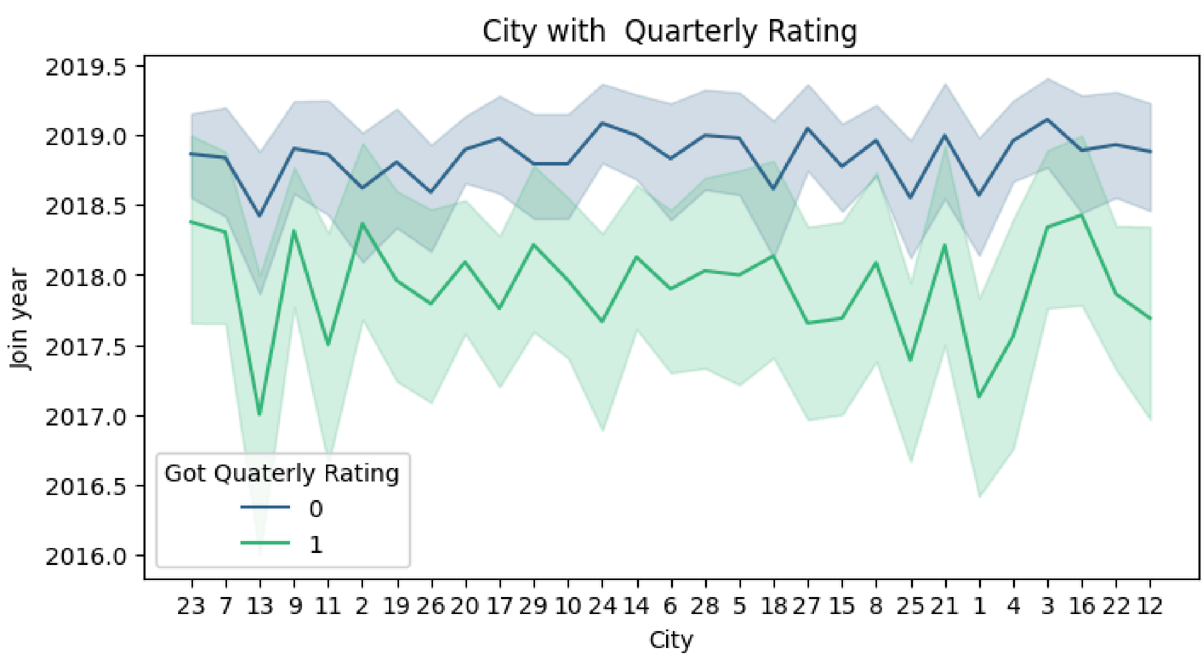
Out[37]:

	Quarterly Rating	No of Reprotings
0	1	47.35
1	2	25.52
2	3	17.79
3	4	9.34

We can see **9% of the drivers got 4 **rating in their Quarterly Rating

In [54]:
```
df['City']=df['City'].astype(str)
```

In [57]:
```
fig = plt.figure(figsize=(8,4))
sns.lineplot(x=df.City,y=df['Join year'],hue=df['Got Quaterly Rating'],palet
plt.title('City with  Quarterly Rating')
plt.show()
```



city -16 got the maximum quaterly rating

In [70]:
```
df['City']=df['City'].astype(int)
```

In [61]:
```python
n = ['Gender','Education_Level','Joining Designation','Grade','Got Income hi

for i in n:
    print("------------------------------------------------------------")
    print(df[i].value_counts(normalize=True) * 100)
```

```
------------------------------------------------------------
Gender
0    59.050819
1    40.949181
Name: proportion, dtype: float64
------------------------------------------------------------
Education_Level
2    33.683326
1    33.389332
0    32.927341
Name: proportion, dtype: float64
------------------------------------------------------------
Joining Designation
1    43.091138
2    34.229315
3    20.705586
4     1.511970
5     0.461991
Name: proportion, dtype: float64
------------------------------------------------------------
Grade
2    35.909282
1    31.121378
3    26.165477
4     5.795884
5     1.007980
Name: proportion, dtype: float64
------------------------------------------------------------
Got Income hike
0    98.194036
1     1.805964
Name: proportion, dtype: float64
------------------------------------------------------------
Got Quaterly Rating
0    65.728685
1    34.271315
Name: proportion, dtype: float64
```

58% of drivers are male while female constitutes around 40%

33% of drivers have completed graduation and 12+ education

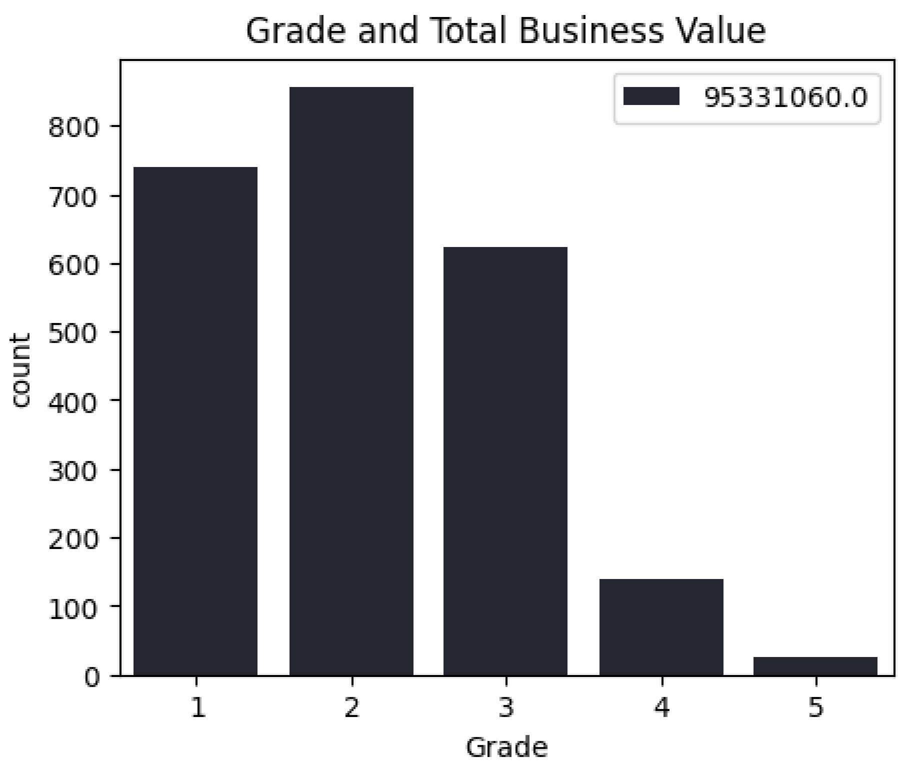
43% of drivers have 1 as joining_designation

Around 36% of drivers graded as 2

Around 73% of drivers rated as 1 on last quarter

Only 15% of drivers rating has been increased on quarterly

In [69]:
```
#check the grade and Total Business Value
plt.figure(figsize=(5,4))
sns.countplot(x=df['Grade'],hue=df['Total Business Value'].max(),color='blue
plt.title('Grade and Total Business Value')
plt.show()
```



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

Ans: False Grade with 2 have the higher Total Business Value

outlier Treatment on Total Business Value

In [71]:
```
len(df[df['Total Business Value'] < 1])
```

Out[71]: 729

As we can notice Total Business Value column has some values in negative.

We consider them as outlier which will affect the results of the our machine learning model.

Considering the parts of datasets that has Total Business Value > 1.

There are exactly 729 Driver having Total Business Value that less than 1.

In [72]:
```
df= df[df['Total Business Value'] > 1]
```

In [73]:
```
df.shape
```

Out[73]:
```
(1652, 16)
```

Model Buliding

In [95]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE
```

Train test Split

Before going to train and test out data,we need to find what is our actual objective and which metric help us to achive in churn analysis.

If our prioritize is **precision**,then we have to reduce **False positive**,which is drivers who are actually retain the company,but predicted as left.

If we prioritize **recall**, we are going to reduce our **false negatives**.means the driver who is actually left but predicted as reatined.

This is useful since usually the cost of hiring a new person is higher than retaining n experienced person. So, by **reducing false negatives**, we would be able to better identify those who are actually going to leave and try to retain them by appropriate measures

In [74]:
```
df.head()
```

Out[74]:

	Driver_ID	No of Reprotings	Age	Gender	City	Education_Level	Grade	Tot Busines Valu
0	1	3	28	0	23	2	1	1715580
2	4	5	43	0	13	2	2	350000
3	5	3	29	0	9	0	1	120360
4	6	5	31	1	11	1	3	1265000
7	12	6	35	0	23	2	1	2607180

In [79]:
```
X = df.drop('Target',axis=1)
y = df['Target']
```

In [81]:
```
# Split the data into training and test data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, r

print(f'Shape of x_train: {X_train.shape}')
print(f'Shape of x_test: {X_test.shape}')
print(f'Shape of y_train: {y_train.shape}')
print(f'Shape of y_test: {y_test.shape}')
```

```
Shape of x_train: (1321, 15)
Shape of x_test: (331, 15)
Shape of y_train: (1321,)
Shape of y_test: (331,)
```

Perform Standardisation

In [82]:
```
scaler = MinMaxScaler()
x_train = scaler.fit_transform(X_train)
x_test = scaler.transform(X_test)
```

Random Forests with imbalance data

In ensemble methods such as Random Forests that utilize bootstrapping, Out-Of-Bag (OOB) data plays a critical role in estimating the model's performance without needing a separate validation set.

In [83]:
```
#Keeping max_depth small to avoid overfitting
params = {
    "max_depth": [2, 3, 4],
    "n_estimators": [50, 100, 150, 200],
}
```

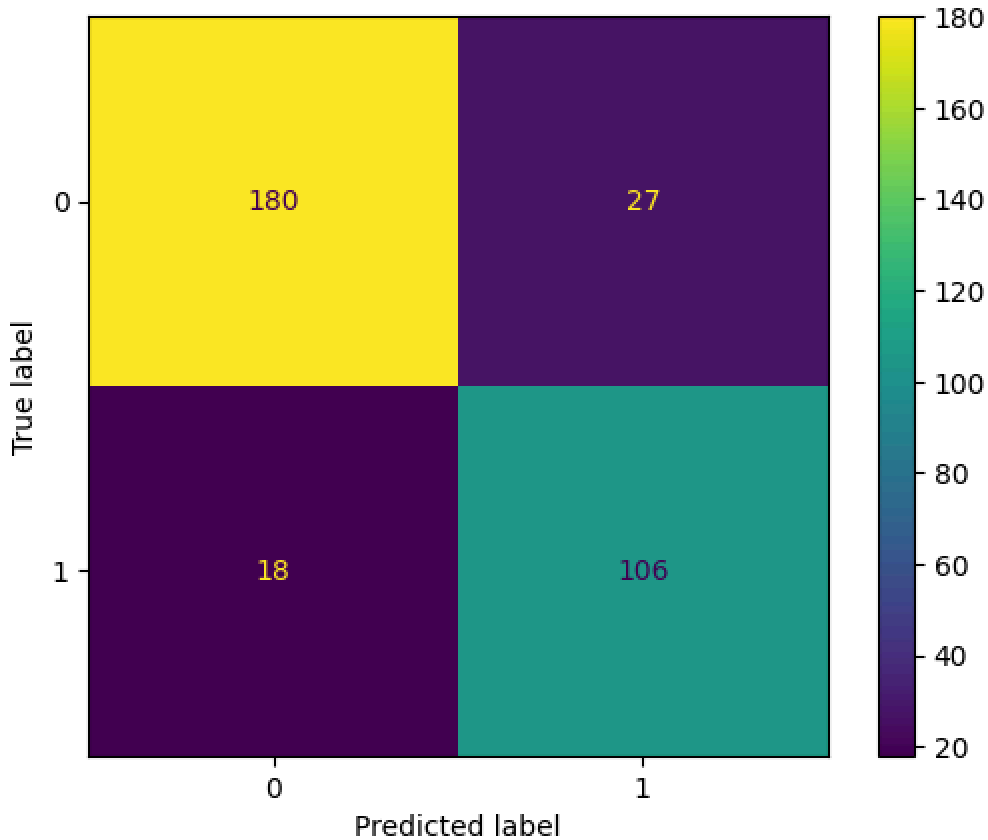
In [85]:
```python
import time
start_time = time.time()
random_forest = RandomForestClassifier(class_weight="balanced")
c = GridSearchCV(estimator=random_forest, param_grid=params, n_jobs=-1, cv=3

c.fit(X_train, y_train)

print("Best Params: ", c.best_params_)
print("Best Score: ", c.best_score_)
elapsed_time = time.time() - start_time

print("\nElapsed Time: ", elapsed_time)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best Params:  {'max_depth': 4, 'n_estimators': 100}
Best Score:  0.8342420152946469

Elapsed Time:  13.163042068481445
```

In [88]:
```python
y_pred = c.predict(X_test)

print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=c.classes_).plot(
```

```
              precision    recall  f1-score   support

           0       0.91      0.87      0.89       207
           1       0.80      0.85      0.82       124

    accuracy                           0.86       331
   macro avg       0.85      0.86      0.86       331
weighted avg       0.87      0.86      0.86       331
```

Out[88]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x79949ed
ffbe0>
```



Random Forest Classifier with imbalanced class weight

Out of all prediction, the measure for correctly predicted 0 is 91% and for 1 is 80% (Precision) Out of all actual 0, the measure for correctly predicted is 87% and for 1 is 85% (Recall) For imbalanced dataset. We consider F1-Score metrics

F1 Score of 0 is 89% F1 Score of 1 is 82%

Balancing Dataset using SMOTE

In [92]:
```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train =

sm = SMOTE(random_state = 7)
X_train, y_train = sm.fit_resample(X_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train.shape

print("After OverSampling, counts of label '1': {}".format(sum(y_train == 1)
print("After OverSampling, counts of label '0': {}".format(sum(y_train == 0)
```

```
Before OverSampling, counts of label '1': 474
Before OverSampling, counts of label '0': 847

After OverSampling, the shape of train_X: (1694, 15)
After OverSampling, the shape of train_y: (1694,)

After OverSampling, counts of label '1': 847
After OverSampling, counts of label '0': 847
```

Random Forests with balance data

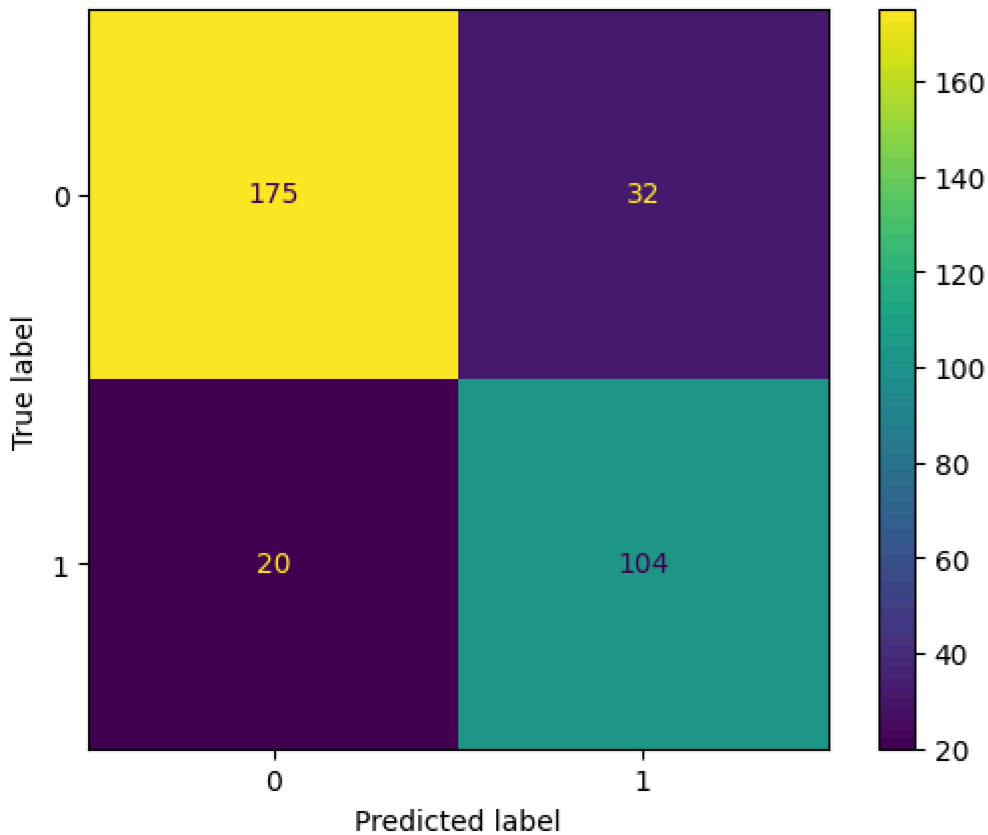
In [93]:

```
params = {
    "max_depth": [2, 3, 4],
    "n_estimators": [50, 100, 150, 200],
}

start_time = time.time()
random_forest = RandomForestClassifier(class_weight="balanced_subsample")
c = GridSearchCV(estimator=random_forest, param_grid=params, n_jobs=-1, cv=3

c.fit(X_train, y_train)

print("Best Params: ", c.best_params_)
print("Best Score: ", c.best_score_)
elapsed_time = time.time() - start_time

print("\nElapsed Time: ", elapsed_time)

y_pred = c.predict(X_test)

print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=c.classes_).plot(
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best Params:  {'max_depth': 4, 'n_estimators': 200}
Best Score:  0.8500159715482636

Elapsed Time:  15.796841621398926
              precision    recall  f1-score   support

           0       0.90      0.85      0.87       207
           1       0.76      0.84      0.80       124

    accuracy                           0.84       331
   macro avg       0.83      0.84      0.84       331
weighted avg       0.85      0.84      0.84       331
```

Out[93]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x79949f0
06230>
```



Random Forest Classifier with balanced class weight

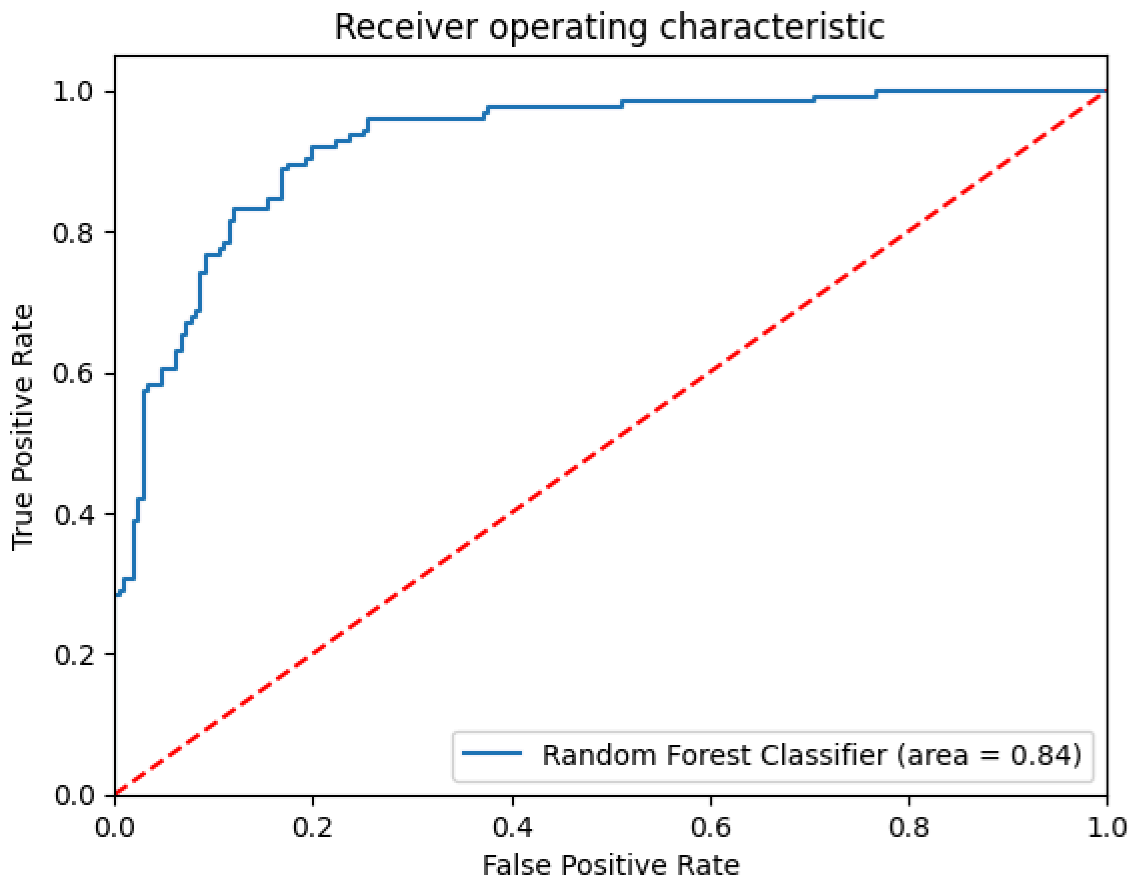
Out of all prediction, the measure for correctly predicted 0 is 90% and for 1 is 76% (Precision) Out of all actual 0.

The measure for correctly predicted is 85% and for 1 is 84% (Recall) For

F1 Score of 0 is 87% F1 Score of 1 is 80%

ROC-AUC Curve

In [96]:
```
logit_roc_auc=roc_auc_score(y_test,y_pred)
fpr,tpr,thresholds=roc_curve(y_test,c.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr,tpr,label='Random Forest Classifier (area = %0.2f)' % logit_roc
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



We can see the ROC is aspiring towards 1,hence it is Generalized model.

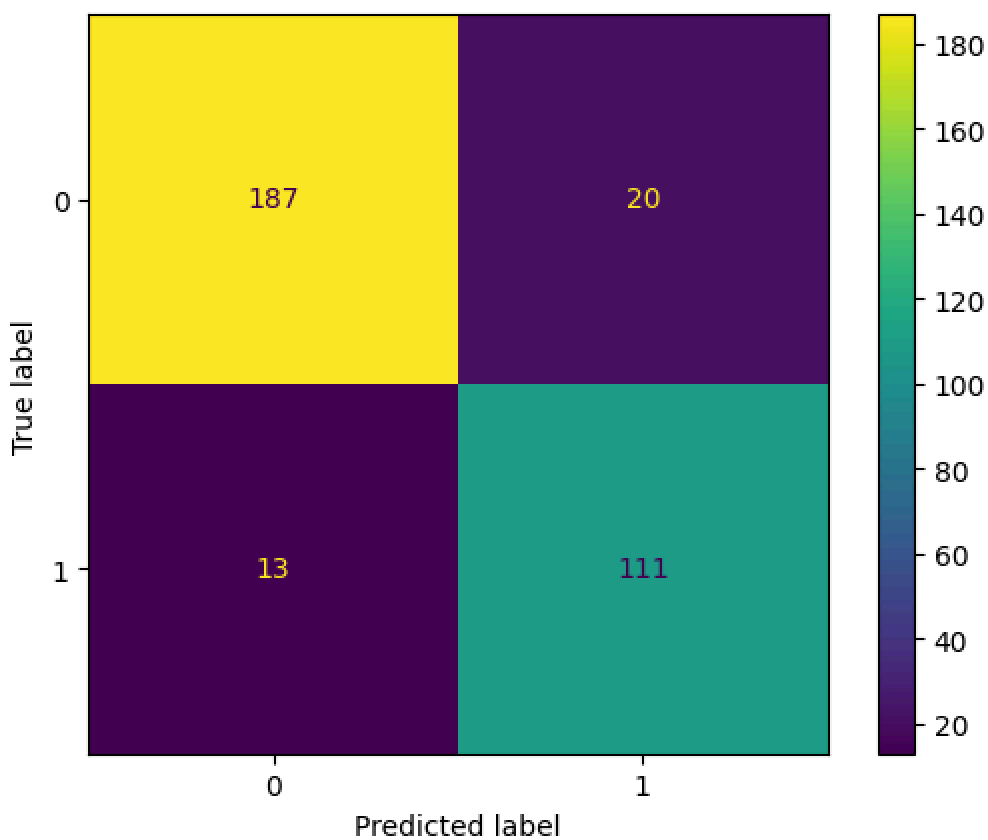
Gradient Boosting Classifier

In [97]:
```
params = {
    "max_depth": [2, 3, 4],
    "loss": ["log_loss", "exponential"],
    "subsample": [0.1, 0.2, 0.5, 0.8, 1],
    "learning_rate": [0.1, 0.2, 0.3],
    "n_estimators": [50,100,150,200]
}

gbdt = GradientBoostingClassifier()
start_time = time.time()
c = GridSearchCV(estimator=gbdt, cv=3, n_jobs=-1, verbose=True, param_grid=p

c.fit(X_train, y_train)
print("Best Params: ", c.best_params_)
print("Best Score: ", c.best_score_)

elapsed_time = time.time() - start_time
print("\n Elapsed Time: ", elapsed_time)

y_pred = c.predict(X_test)

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

```
cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=c.classes_).plot(
```

```
Fitting 3 folds for each of 360 candidates, totalling 1080 fits
Best Params:  {'learning_rate': 0.3, 'loss': 'exponential', 'max_depth': 4,
'n_estimators': 100, 'subsample': 0.8}
Best Score:  0.9138099123412623

 Elapsed Time:  349.3228373527527
              precision    recall  f1-score   support

           0       0.94      0.90      0.92       207
           1       0.85      0.90      0.87       124

    accuracy                           0.90       331
   macro avg       0.89      0.90      0.89       331
weighted avg       0.90      0.90      0.90       331
```

Out[97]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7994905279a0>

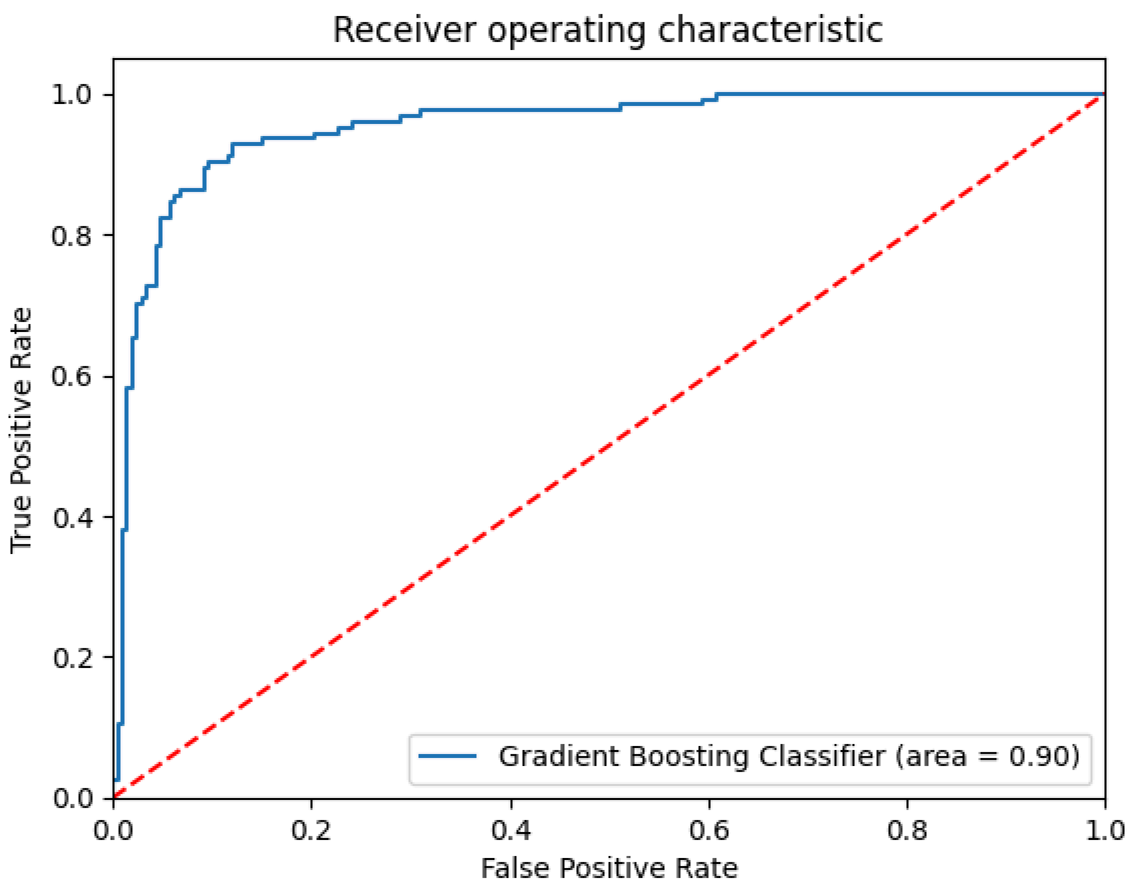


Gradient Boosting Classifier with balanced class weight

Out of all prediction, the measure for correctly predicted 0 is 94% and for 1 is 85% (Precision) Out of all actual 0.

The measure for correctly predicted is 90% and for 1 is 90% (Recall) For imbalanced dataset.

In [98]:
```
logit_roc_auc=roc_auc_score(y_test,y_pred)
fpr,tpr,thresholds=roc_curve(y_test,c.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr,tpr,label='Gradient Boosting Classifier (area = %0.2f)' % logit
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

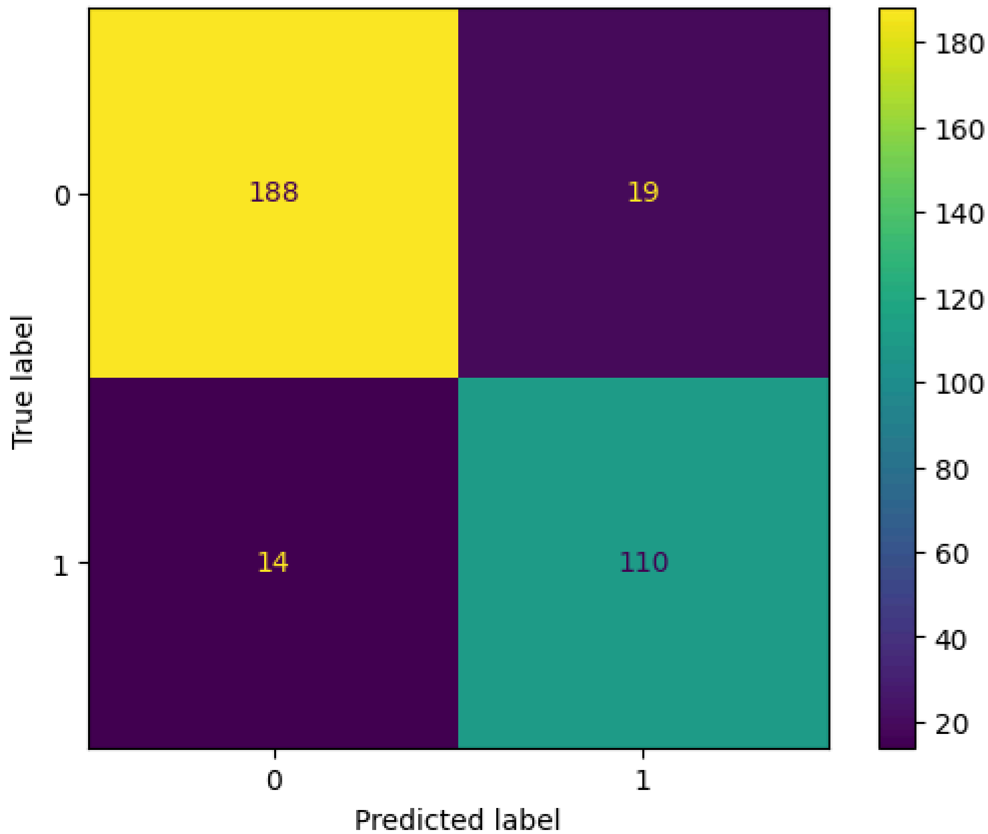


XGBoost Classifier

In [100…
```
import xgboost as xgb
model = xgb.XGBClassifier(class_weight = "balanced")

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print("XGBoost Classifier Score: ", model.score(X_test, y_test))
print("\n", classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.classes_).p
```

```
XGBoost Classifier Score:  0.9003021148036254

              precision    recall  f1-score   support

           0       0.93      0.91      0.92       207
           1       0.85      0.89      0.87       124

    accuracy                           0.90       331
   macro avg       0.89      0.90      0.89       331
weighted avg       0.90      0.90      0.90       331
```

Out[100… <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x79949f03c0d0>



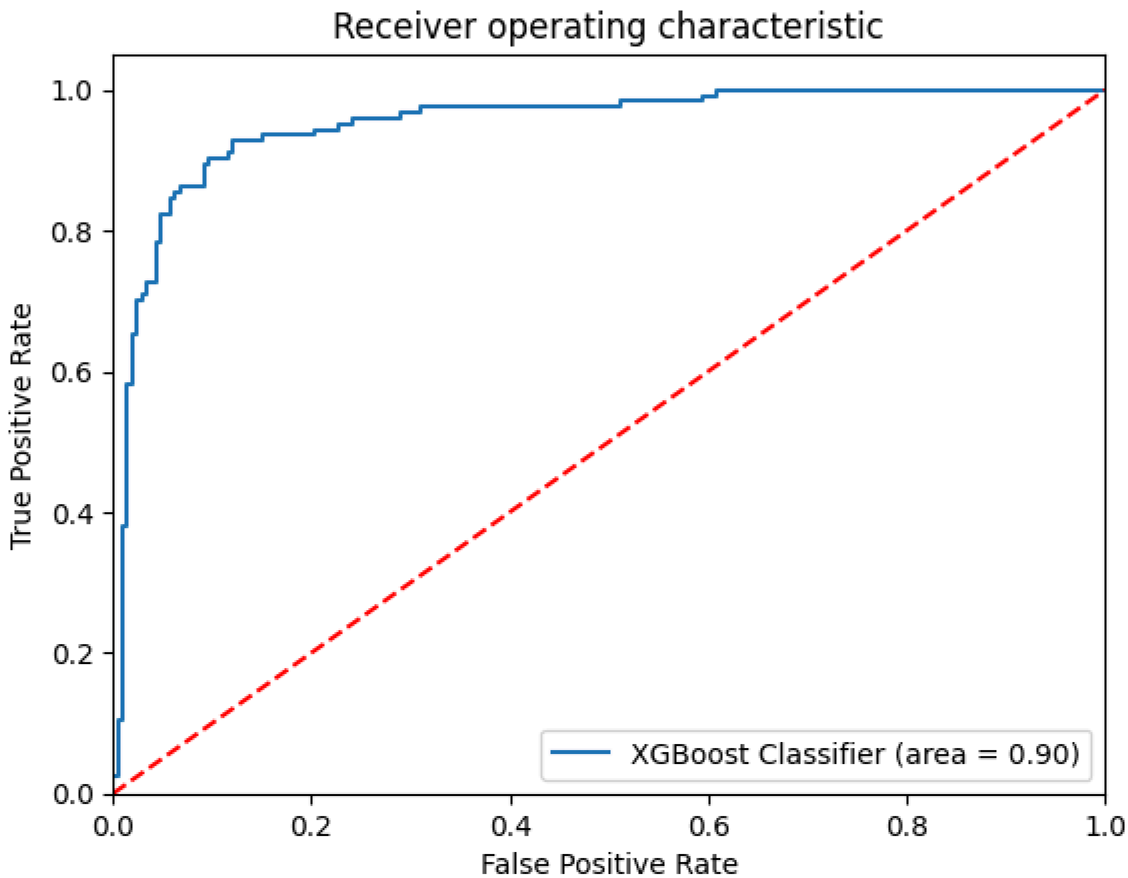
XGBoost Classifier with balanced class weight

Out of all prediction, the measure for correctly predicted 0 is 93% and for 1 is 85% (Precision) Out of all actual 0.

The measure for correctly predicted is 91% and for 1 is 89% (Recall)

In [101…
```
logit_roc_auc=roc_auc_score(y_test,y_pred)
fpr,tpr,thresholds=roc_curve(y_test,c.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr,tpr,label='XGBoost Classifier (area = %0.2f)' % logit_roc_auc)
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



Feature Importance of the best model so far.

(max_depth = 4, n_estimators= 100)

In [107…

```
gbt = GradientBoostingClassifier(max_depth = 4, n_estimators= 100)

gbt.fit(X_train, y_train)
print("Score of GradientBoostingClassifier: ", gbt.score(X_test, y_test))
```

```
Score of GradientBoostingClassifier:  0.9093655589123867
```

In [108…

```
importances = gbt.feature_importances_
importances
```

Out[108…

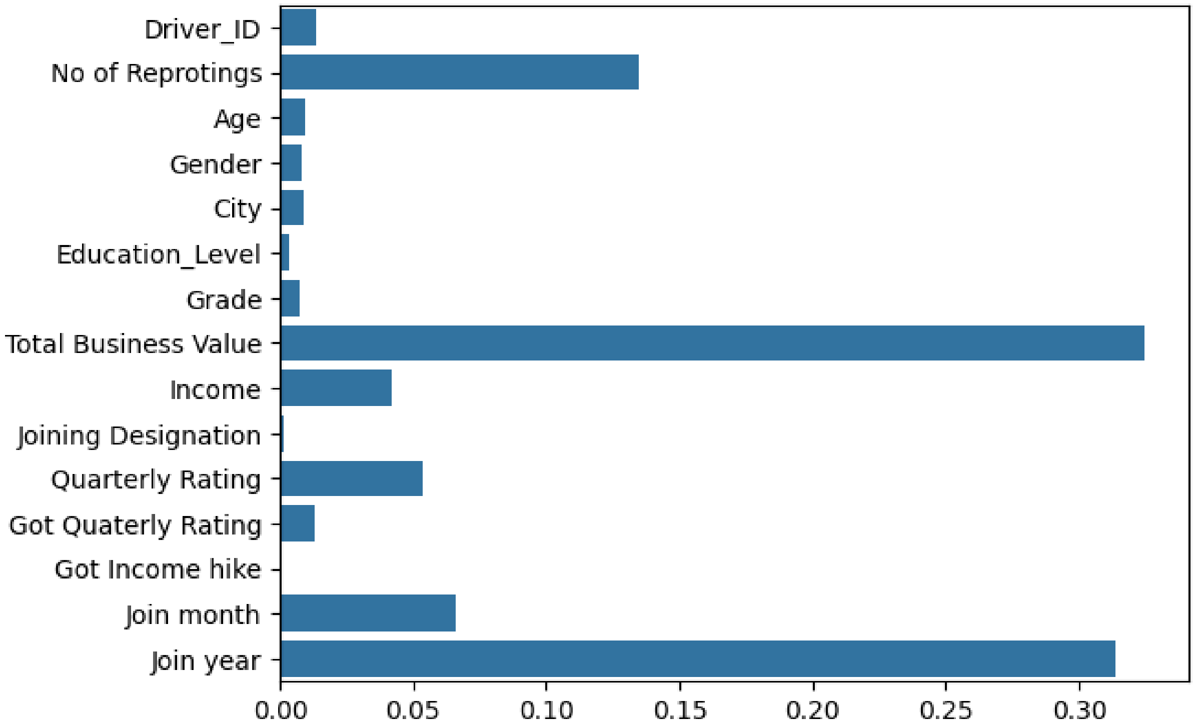
```
array([1.36555102e-02, 1.34693087e-01, 9.24996141e-03, 7.84748333e-03,
       8.62528211e-03, 3.27391822e-03, 7.57226503e-03, 3.25030433e-01,
       4.17269842e-02, 1.51658106e-03, 5.38206268e-02, 1.31380300e-02,
       6.85907280e-05, 6.63006405e-02, 3.13480607e-01])
```

In [116…

```
# Feature names
features = np.array(X_train.columns)

# TODO: Identifying the most important feature
```

```
most_important_feature = features[np.argmax(importances)]

print(f"The most important feature is: {most_important_feature}")

sns.barplot(y= features, x=importances)
plt.show()
```

```
The most important feature is: Total Business Value
```



We can see Total Business value,Quaterly Rating,No of reporting are the important features

Insights:

We can see the maximum no of reprotings are 5 and 24.

Most of the drivers are from the citycode 20.

We can there is same no of drivers are in distributed in all 3 educational levels.

Quaterly hike are given to only 1/3 of drivers

And Income hike are not even provided to single driver,this might be the important feature for chrun.We will further analysis for more accurate prediction.

And Most of the drivers are joined at every financial months(july and December).

There is the drastic increase in the count od drivers joined from 2017 to 2018.

From the corelation matrix,we can see driver chrun(target) is highly correlated with Total business value.

Age and Quarterly Rating are positivly correlated.

Total business value and No of reportings also positvly higly correlated.

Joining degination is negaitvley correlated with Quarterly Rating and monthly income hike.

We can see 9% of the drivers got 4 rating in their Quarterly Rating.

58% of drivers are male while female constitutes around 40%

43% of drivers have 1 as joining_designation

Around 73% of drivers rated as 1 on last quarter

Only 15% of drivers rating has been increased on quarterly.

Drivers with a Grade of ‘A’ are more likely to have a higher Total Business Value. (T/F) Ans: False Grade with 2 have the higher Total Business Value

Recall increased after treatment of data imbalance and is performing better in Gradient Boosting.

Precision dropped after treatment of data imbalance and is performing better in Gradient Boosting.

F1_score incresed after the treatment of imabalanced data and in Gradient Boosting.

Recommendations:

Out of 2381 drivers 1616 have left the company.

We need to incentivise the drivers overtime or other perks to overcome churning The employees whose quarterly rating has increased are less likely to leave the organization.

Company needs to implement the reward system for the customer who provide the feedback and rate drivers

The employees whose monthly salary has not increased are more likely to leave the organization.

Company needs to get in touch with those drivers whose monthly salary has not increased and help them out to earn more by provider bonus and perks.

Out of 2381 employees, 1744 employees had their last quarterly rating as 1.

Out of 2381 employees, the quarterly rating has not increased for 2076 employees. This is red flag for the company which needs to regulate.

Company needs to look why customers are not rating drivers. Last_Quarterly_Rating, Total_Business_Value & Quarterly_Rating_Increased are the most important features. Company needs to tracks these features as predicators

We observe that we are not getting very high recall on target 0 which may be due to small unbalanced dataset. More data will overcome this issue.

The Gradient Boosting Classifier attains the Recall score of 91% for the driver who left the company. Which indicates that model is performing the decent job.