```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.impute import KNNImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split,GridSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
import xgboost as xgb
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix,ConfusionMatrixDisplay
from sklearn.metrics import roc_auc_score,roc_curve

import time
```

```
data=pd.read_csv('/content/ola_driver_scaler.csv')
data.head()
```

```
/usr/local/lib/python3.11/dist-packages/google/colab/_dataframe_summarizer.py:88: UserWarning: Could not infer format, so each eleme
  cast_date_col = pd.to_datetime(column, errors="coerce")
```

| | Unnamed: 0 | MMM-YY | Driver_ID | Age | Gender | City | Education_Level | Income | Dateofjoining | LastWorkingDate | Joining Designation | Grade | Bus |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 01/01/19 | 1 | 28.0 | 0.0 | C23 | 2 | 57387 | 24/12/18 | NaN | 1 | 1 | 23 |
| **1** | 1 | 02/01/19 | 1 | 28.0 | 0.0 | C23 | 2 | 57387 | 24/12/18 | NaN | 1 | 1 | -6 |
| **2** | 2 | 03/01/19 | 1 | 28.0 | 0.0 | C23 | 2 | 57387 | 24/12/18 | 03/11/19 | 1 | 1 | |
| **3** | 3 | 11/01/20 | 2 | 31.0 | 0.0 | C7 | 2 | 67016 | 11/06/20 | NaN | 2 | 2 | |

Next steps: Generate code with data View recommended plots New interactive sheet

```
data.drop("Unnamed: 0", axis = 1, inplace = True)
```

```
data.head()
```

```
/usr/local/lib/python3.11/dist-packages/google/colab/_dataframe_summarizer.py:88: UserWarning: Could not infer format, so each eleme
  cast_date_col = pd.to_datetime(column, errors="coerce")
```

| | MMM-YY | Driver_ID | Age | Gender | City | Education_Level | Income | Dateofjoining | LastWorkingDate | Joining Designation | Grade | Total Business Value | Qua |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 01/01/19 | 1 | 28.0 | 0.0 | C23 | 2 | 57387 | 24/12/18 | NaN | 1 | 1 | 2381060 | |
| **1** | 02/01/19 | 1 | 28.0 | 0.0 | C23 | 2 | 57387 | 24/12/18 | NaN | 1 | 1 | -665480 | |
| **2** | 03/01/19 | 1 | 28.0 | 0.0 | C23 | 2 | 57387 | 24/12/18 | 03/11/19 | 1 | 1 | 0 | |
| **3** | 11/01/20 | 2 | 31.0 | 0.0 | C7 | 2 | 67016 | 11/06/20 | NaN | 2 | 2 | 0 | |

Next steps: Generate code with data View recommended plots New interactive sheet

```
data.shape
```

```
(19104, 13)
```

```
data.nunique()
```

| | 0 |
|---|---|
| **MMM-YY** | 24 |
| **Driver_ID** | 2381 |
| **Age** | 36 |
| **Gender** | 2 |
| **City** | 29 |
| **Education_Level** | 3 |
| **Income** | 2383 |
| **Dateofjoining** | 869 |
| **LastWorkingDate** | 493 |
| **Joining Designation** | 5 |
| **Grade** | 5 |
| **Total Business Value** | 10181 |
| **Quarterly Rating** | 4 |

```
data.info()
```

```
<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
```

Converting Features to respective datatypes:

```
data['MMM-YY']=pd.to_datetime(data['MMM-YY'])
data["Dateofjoining"] = pd.to_datetime(data["Dateofjoining"])
data["LastWorkingDate"] = pd.to_datetime(data["LastWorkingDate"])
```

```
/tmp/ipython-input-8-105792943.py:1: UserWarning: Could not infer format, so each element will be parsed individually, falling back
  data['MMM-YY']=pd.to_datetime(data['MMM-YY'])
/tmp/ipython-input-8-105792943.py:2: UserWarning: Could not infer format, so each element will be parsed individually, falling back
  data["Dateofjoining"] = pd.to_datetime(data["Dateofjoining"])
/tmp/ipython-input-8-105792943.py:3: UserWarning: Could not infer format, so each element will be parsed individually, falling back
  data["LastWorkingDate"] = pd.to_datetime(data["LastWorkingDate"])
```

```
data.info()
```

```
<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  datetime64[ns]
 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  datetime64[ns]
 8   LastWorkingDate       1616 non-null   datetime64[ns]
 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: datetime64[ns](3), float64(2), int64(7), object(1)
memory usage: 1.9+ MB
```

Checking for missing values and prepare data for KNN Imputation:

```
data.isnull().sum()/len(data)*100
```

| | 0 |
|---|---|
| **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 |

1.There are missing values found in AGE, Gender.

2.LastWorkingDate feature contains missing values which indicates the driver has not left the company yet.

```
num_vars=data.select_dtypes(np.number)
num_vars.columns
```

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

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

## KNN Imputation

```
imputer=KNNImputer(n_neighbors=5,weights='uniform',metric='nan_euclidean')
imputer.fit(num_vars)
data_new=imputer.transform(num_vars)
```

```
data_new=pd.DataFrame(data_new)
```

```
data_new.columns=num_vars.columns
```

```
data_new.isnull().sum()
```

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

1.We have successfully imputed the missing values using KNNImputer

```
data_new.nunique()
```

| | 0 |
|---|---|
| **Age** | 70 |
| **Gender** | 6 |
| **Education_Level** | 3 |
| **Income** | 2383 |
| **Joining Designation** | 5 |
| **Grade** | 5 |
| **Total Business Value** | 10181 |
| **Quarterly Rating** | 4 |

## Concatenating DataFrames

```
resultant_columns = list(set(data.columns).difference(set(num_vars)))
```

```
resultant_columns
```

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

```
new_df=pd.concat([data_new,data[resultant_columns]],axis=1)
new_df.shape
```

```
(19104, 13)
```

```
new_df.head()
```

| | Age | Gender | Education_Level | Income | Joining Designation | Grade | Total Business Value | Quarterly Rating | Driver_ID | LastWorkingDate | MMM-YY | Dateofjoining |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 28.0 | 0.0 | 2.0 | 57387.0 | 1.0 | 1.0 | 2381060.0 | 2.0 | 1 | NaT | 2019-01-01 | 2018-12-24 |
| **1** | 28.0 | 0.0 | 2.0 | 57387.0 | 1.0 | 1.0 | -665480.0 | 2.0 | 1 | NaT | 2019-02-01 | 2018-12-24 |

Next steps: Generate code with new_df View recommended plots New interactive sheet

## Data Preprocessing

Feature Engineering

```
agg_functions = {
    "Age": "max",
```

```
    "Gender": "first",
    "Education_Level": "last",
    "Income": "last",
    "Joining Designation": "last",
    "Grade": "last",
    "Total Business Value": "sum",
    "Quarterly Rating": "last",
    "LastWorkingDate": "last",
    "City": "first",
    "Dateofjoining": "last"
}
```

```
processed_df=new_df.groupby(["Driver_ID", "MMM-YY"]).aggregate(agg_functions).sort_index(ascending=[True,True])
```

```
processed_df.head()
```

| Driver_ID | MMM-YY | Age | Gender | Education_Level | Income | Joining Designation | Grade | Total Business Value | Quarterly Rating | LastWorkingDate | City | Dateofjoini |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2019-01-01 | 28.0 | 0.0 | 2.0 | 57387.0 | 1.0 | 1.0 | 2381060.0 | 2.0 | NaT | C23 | 2018-12-: |
| | 2019-02-01 | 28.0 | 0.0 | 2.0 | 57387.0 | 1.0 | 1.0 | -665480.0 | 2.0 | NaT | C23 | 2018-12-: |

Next steps: Generate code with processed_df View recommended plots New interactive sheet

```
final_data=pd.DataFrame()
```

```
final_data["Driver_ID"] = new_df["Driver_ID"].unique()
```

```
final_data['Age'] = list(processed_df.groupby('Driver_ID',axis=0).max('MMM-YY')['Age'])
final_data['Gender'] = list(processed_df.groupby('Driver_ID').agg({'Gender':'last'})['Gender'])
final_data['City'] = list(processed_df.groupby('Driver_ID').agg({'City':'last'})['City'])
final_data['Education'] = list(processed_df.groupby('Driver_ID').agg({'Education_Level':'last'})['Education_Level'])
final_data['Income'] = list(processed_df.groupby('Driver_ID').agg({'Income':'last'})['Income'])
final_data['Joining_Designation'] = list(processed_df.groupby('Driver_ID').agg({'Joining Designation':'last'})['Joining Designation'])
final_data['Grade'] = list(processed_df.groupby('Driver_ID').agg({'Grade':'last'})['Grade'])
final_data['Total_Business_Value'] = list(processed_df.groupby('Driver_ID',axis=0).sum('Total Business Value')['Total Business Value'])
final_data['Last_Quarterly_Rating'] = list(processed_df.groupby('Driver_ID').agg({'Quarterly Rating':'last'})['Quarterly Rating'])
```

```
/tmp/ipython-input-24-2206923327.py:1: FutureWarning: The 'axis' keyword in DataFrame.groupby is deprecated and will be removed in a
  final_data['Age'] = list(processed_df.groupby('Driver_ID',axis=0).max('MMM-YY')['Age'])
/tmp/ipython-input-24-2206923327.py:8: FutureWarning: The 'axis' keyword in DataFrame.groupby is deprecated and will be removed in a
  final_data['Total_Business_Value'] = list(processed_df.groupby('Driver_ID',axis=0).sum('Total Business Value')['Total Business Va]
```

```
final_data.head()
```

| | Driver_ID | Age | Gender | City | Education | Income | Joining_Designation | Grade | Total_Business_Value | Last_Quarterly_Rating |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 28.0 | 0.0 | C23 | 2.0 | 57387.0 | 1.0 | 1.0 | 1715580.0 | 2.0 |
| 1 | 2 | 31.0 | 0.0 | C7 | 2.0 | 67016.0 | 2.0 | 2.0 | 0.0 | 1.0 |
| 2 | 4 | 43.0 | 0.0 | C13 | 2.0 | 65603.0 | 2.0 | 2.0 | 350000.0 | 1.0 |
| 3 | 5 | 29.0 | 0.0 | C9 | 0.0 | 46368.0 | 1.0 | 1.0 | 120360.0 | 1.0 |
| 4 | 6 | 31.0 | 1.0 | C11 | 1.0 | 78728.0 | 3.0 | 3.0 | 1265000.0 | 2.0 |

Next steps: Generate code with final_data View recommended plots New interactive sheet

```
final_data.shape
```

```
(2381, 10)
```

Create a column which tells whether the quarterly rating has increased for that driver - for those whose quarterly rating has increased we assign the value 1

```
first_quarter = processed_df.groupby(["Driver_ID"]).agg({"Quarterly Rating": "first"})
```

```
last_quarter = processed_df.groupby(["Driver_ID"]).agg({"Quarterly Rating": "last"})
```

```
qr = (last_quarter["Quarterly Rating"] > first_quarter["Quarterly Rating"]).reset_index()
```

```
empid = qr[qr["Quarterly Rating"] == True]["Driver_ID"]

qrl = []
for i in final_data["Driver_ID"]:
    if i in empid.values:
        qrl.append(1)
    else:
        qrl.append(0)
```

```
final_data.head()
```

| | Driver_ID | Age | Gender | City | Education | Income | Joining_Designation | Grade | Total_Business_Value | Last_Quarterly_Rating |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 28.0 | 0.0 | C23 | 2.0 | 57387.0 | 1.0 | 1.0 | 1715580.0 | 2.0 |
| 1 | 2 | 31.0 | 0.0 | C7 | 2.0 | 67016.0 | 2.0 | 2.0 | 0.0 | 1.0 |
| 2 | 4 | 43.0 | 0.0 | C13 | 2.0 | 65603.0 | 2.0 | 2.0 | 350000.0 | 1.0 |
| 3 | 5 | 29.0 | 0.0 | C9 | 0.0 | 46368.0 | 1.0 | 1.0 | 120360.0 | 1.0 |
| 4 | 6 | 31.0 | 1.0 | C11 | 1.0 | 78728.0 | 3.0 | 3.0 | 1265000.0 | 2.0 |

Next steps: Generate code with final_data View recommended plots New interactive sheet

Target variable creation: Create a column called target which tells whether the driver has left the company- driver whose last working day is present will have the value 1

```
lwd = (processed_df.groupby(["Driver_ID"]).agg({"LastWorkingDate": "last"})["LastWorkingDate"].isna()).reset_index()

lwrid = lwd[lwd["LastWorkingDate"] == True]["Driver_ID"]
target = []

for i in final_data["Driver_ID"]:
    if i in lwrid.values:
        target.append(0)
    else:
        target.append(1)

final_data["target"] = target
```

```
final_data.head()
```

| | Driver_ID | Age | Gender | City | Education | Income | Joining_Designation | Grade | Total_Business_Value | Last_Quarterly_Rating | target |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 28.0 | 0.0 | C23 | 2.0 | 57387.0 | 1.0 | 1.0 | 1715580.0 | 2.0 | 1 |
| 1 | 2 | 31.0 | 0.0 | C7 | 2.0 | 67016.0 | 2.0 | 2.0 | 0.0 | 1.0 | 0 |
| 2 | 4 | 43.0 | 0.0 | C13 | 2.0 | 65603.0 | 2.0 | 2.0 | 350000.0 | 1.0 | 1 |
| 3 | 5 | 29.0 | 0.0 | C9 | 0.0 | 46368.0 | 1.0 | 1.0 | 120360.0 | 1.0 | 1 |
| 4 | 6 | 31.0 | 1.0 | C11 | 1.0 | 78728.0 | 3.0 | 3.0 | 1265000.0 | 2.0 | 0 |

Next steps: Generate code with final_data View recommended plots New interactive sheet

Create a column which tells whether the monthly income has increased for that driver - for those whose monthly income has increased we assign the value 1

```
final_data = pd.DataFrame()
final_data["Driver_ID"] = new_df["Driver_ID"].unique()
final_data['Age'] = list(processed_df.groupby('Driver_ID').max('MMM-YY')['Age'])
final_data['Gender'] = list(processed_df.groupby('Driver_ID').agg({'Gender':'last'})['Gender'])
final_data['City'] = list(processed_df.groupby('Driver_ID').agg({'City':'last'})['City'])
final_data['Education'] = list(processed_df.groupby('Driver_ID').agg({'Education_Level':'last'})['Education_Level'])
final_data['Income'] = list(processed_df.groupby('Driver_ID').agg({'Income':'last'})['Income'])
final_data['Joining_Designation'] = list(processed_df.groupby('Driver_ID').agg({'Joining Designation':'last'})['Joining Designation'])
final_data['Grade'] = list(processed_df.groupby('Driver_ID').agg({'Grade':'last'})['Grade'])
final_data['Total_Business_Value'] = list(processed_df.groupby('Driver_ID').sum('Total Business Value')['Total Business Value'])
final_data['Last_Quarterly_Rating'] = list(processed_df.groupby('Driver_ID').agg({'Quarterly Rating':'last'})['Quarterly Rating'])

# Create the target column here
lwd = (processed_df.groupby(["Driver_ID"]).agg({"LastWorkingDate": "last"})["LastWorkingDate"].isna()).reset_index()
lwrid = lwd[lwd["LastWorkingDate"] == True]["Driver_ID"]
target = []
```

```
for i in final_data["Driver_ID"]:
    if i in lwrid.values:
        target.append(0)
    else:
        target.append(1)

final_data["target"] = target


mrf = processed_df.groupby(["Driver_ID"]).agg({"Income": "first"})

mrl = processed_df.groupby(["Driver_ID"]).agg({"Income": "last"})

mr = (mrl["Income"] > mrf["Income"]).reset_index()

empid = mr[mr["Income"] == True]["Driver_ID"]
income = []
for i in final_data["Driver_ID"]:
    if i in empid.values:
        income.append(1)
    else:
        income.append(0)

final_data["Salary_Increased"] = income
```

```
final_data.head()
```

| | Driver_ID | Age | Gender | City | Education | Income | Joining_Designation | Grade | Total_Business_Value | Last_Quarterly_Rating | Salary_ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 28.0 | 0.0 | C23 | 2.0 | 57387.0 | 1.0 | 1.0 | 1715580.0 | 2.0 | |
| 1 | 2 | 31.0 | 0.0 | C7 | 2.0 | 67016.0 | 2.0 | 2.0 | 0.0 | 1.0 | |
| 2 | 4 | 43.0 | 0.0 | C13 | 2.0 | 65603.0 | 2.0 | 2.0 | 350000.0 | 1.0 | |
| 3 | 5 | 29.0 | 0.0 | C9 | 0.0 | 46368.0 | 1.0 | 1.0 | 120360.0 | 1.0 | |
| 4 | 6 | 31.0 | 1.0 | C11 | 1.0 | 78728.0 | 3.0 | 3.0 | 1265000.0 | 2.0 | |

Next steps: Generate code with final_data View recommended plots New interactive sheet

```
final_data['Salary_Increased'].value_counts(normalize=True)
```

| | proportion |
|---|---|
| **Salary_Increased** | |
| 0 | 0.98194 |
| 1 | 0.01806 |

Around 1.8% drivers income have been increased.

## Statistical Summary

```
final_data.describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Driver_ID** | 2381.0 | 1.397559e+03 | 8.061616e+02 | 1.0 | 695.0 | 1400.0 | 2100.0 | 2788.0 |
| **Age** | 2381.0 | 3.377018e+01 | 5.933265e+00 | 21.0 | 30.0 | 33.0 | 37.0 | 58.0 |
| **Gender** | 2381.0 | 4.105838e-01 | 4.914963e-01 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |
| **Education** | 2381.0 | 1.007560e+00 | 8.162900e-01 | 0.0 | 0.0 | 1.0 | 2.0 | 2.0 |
| **Income** | 2381.0 | 5.933416e+04 | 2.838367e+04 | 10747.0 | 39104.0 | 55315.0 | 75986.0 | 188418.0 |
| **Joining_Designation** | 2381.0 | 1.820244e+00 | 8.414334e-01 | 1.0 | 1.0 | 2.0 | 2.0 | 5.0 |
| **Grade** | 2381.0 | 2.096598e+00 | 9.415218e-01 | 1.0 | 1.0 | 2.0 | 3.0 | 5.0 |
| **Total_Business_Value** | 2381.0 | 4.586742e+06 | 9.127115e+06 | -1385530.0 | 0.0 | 817680.0 | 4173650.0 | 95331060.0 |
| **Last_Quarterly_Rating** | 2381.0 | 1.427971e+00 | 8.098389e-01 | 1.0 | 1.0 | 1.0 | 2.0 | 4.0 |
| **Salary_Increased** | 2381.0 | 1.805964e-02 | 1.331951e-01 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |

1.There are total of 2831 different drivers data.

2.Age of drivers range from 21years to 58years.

3.75% drivers monthly income is <= 75986.

4.75% drivers acquired 4173650 as total business values.

```
final_data.describe(include='object')
```

| | City |
|---|---|
| **count** | 2381 |
| **unique** | 29 |
| **top** | C20 |
| **freq** | 152 |

Majority of drivers are coming from C20 city

```
final_data['Gender'].value_counts()
```

| Gender | count |
|---|---|
| **0.0** | 1400 |
| **1.0** | 975 |
| **0.6** | 3 |
| **0.2** | 2 |
| **0.4** | 1 |

Majority of drivers are male

```
final_data["Education"].value_counts()
```

| Education | count |
|---|---|
| **2.0** | 802 |
| **1.0** | 795 |
| **0.0** | 784 |

Majority of drivers have completed their graduation.

```
final_data["target"].value_counts()
```

| target | count |
|---|---|
| **1** | 1616 |
| **0** | 765 |

Out of 2381 drivers 1616 have left the company.

```
n = ['Gender','Education','Joining_Designation','Grade','Last_Quarterly_Rating','Quarterly_Rating_Increased']

# Create the 'Quarterly_Rating_Increased' column
first_quarter = processed_df.groupby(["Driver_ID"]).agg({"Quarterly Rating": "first"})
last_quarter = processed_df.groupby(["Driver_ID"]).agg({"Quarterly Rating": "last"})
qr = (last_quarter["Quarterly Rating"] > first_quarter["Quarterly Rating"]).reset_index()
empid_qr = qr[qr["Quarterly Rating"] == True]["Driver_ID"]
```

```
qrl = []
for i in final_data["Driver_ID"]:
    if i in empid_qr.values:
        qrl.append(1)
    else:
        qrl.append(0)

final_data["Quarterly_Rating_Increased"] = qrl

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

```
------------------------------------------------------------
Gender
0.0    58.798824
1.0    40.949181
0.6     0.125997
0.2     0.083998
0.4     0.041999
Name: proportion, dtype: float64
------------------------------------------------------------
Education
2.0    33.683326
1.0    33.389332
0.0    32.927341
Name: proportion, dtype: float64
------------------------------------------------------------
Joining_Designation
1.0    43.091138
2.0    34.229315
3.0    20.705586
4.0     1.511970
5.0     0.461991
Name: proportion, dtype: float64
------------------------------------------------------------
Grade
2.0    35.909282
1.0    31.121378
3.0    26.165477
4.0     5.795884
5.0     1.007980
Name: proportion, dtype: float64
------------------------------------------------------------
Last_Quarterly_Rating
1.0    73.246535
2.0    15.203696
3.0     7.055859
4.0     4.493910
Name: proportion, dtype: float64
------------------------------------------------------------
Quarterly_Rating_Increased
0    84.964301
1    15.035699
Name: proportion, dtype: float64
```

1.58% of drivers are male while female constitutes around 40%

2.33% of drivers have completed graduation and 12+ education

3.43% of drivers have 1 as joining_designation

4.Around 36% of drivers graded as 2

5.Around 73% of drivers rated as 1 on last quarter

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

## Univariate Analysis

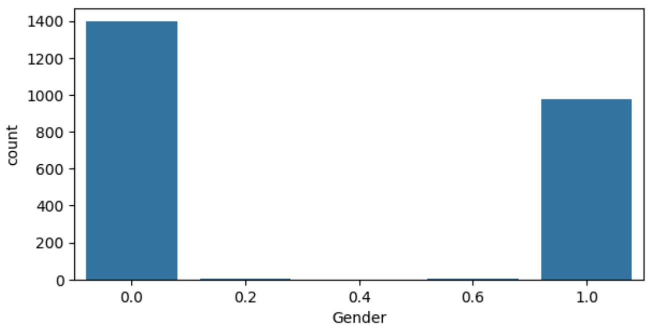
```
plt.figure(figsize=(15, 15))
plt.subplot(421)
sns.countplot(data=final_data, x="Gender")
# final_data["Gender"].value_counts(normalize=True).plot.bar('Gender')

plt.subplot(422)
sns.countplot(data=final_data, x="City")
plt.xticks(rotation="45")

plt.subplot(423)
sns.countplot(data=final_data, x="Joining_Designation")

plt.subplot(424)
```

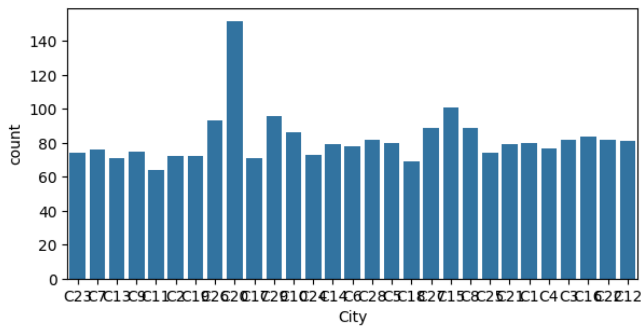
```
sns.countplot(data=final_data, x="Education")

plt.subplot(425)
sns.countplot(data=final_data, x="Grade")

plt.subplot(426)
sns.countplot(data=final_data, x="Last_Quarterly_Rating")

plt.subplot(427)
sns.countplot(data=final_data, x="Quarterly_Rating_Increased")

plt.subplot(428)
sns.countplot(data=final_data, x="Salary_Increased")
plt.tight_layout()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-51-739048282.py in <cell line: 0>()
      6 plt.subplot(422)
      7 sns.countplot(data=final_data, x="City")
----> 8 plt.xticks(rotation="45")
      9 
     10 plt.subplot(423)
```

3 frames

```
/usr/local/lib/python3.11/dist-packages/matplotlib/text.py in set_rotation(self, s)
   1242             self._rotation = 90.
   1243         else:
-> 1244             raise ValueError("rotation must be 'vertical', 'horizontal' or "
   1245                              f"a number, not {s}")
   1246         self.stale = True

ValueError: rotation must be 'vertical', 'horizontal' or a number, not 45
```




Next steps: Explain error

## Insights-

1.Out of 2381 employees, 1404 employees are of the Male gender and 977 are females.

2.Out of 2381 employees, 152 employees are from city C20 and 101 from city C15.

3.Out of 2381 employees, 802 employees have their education as Graduate and 795 have completed their 12.

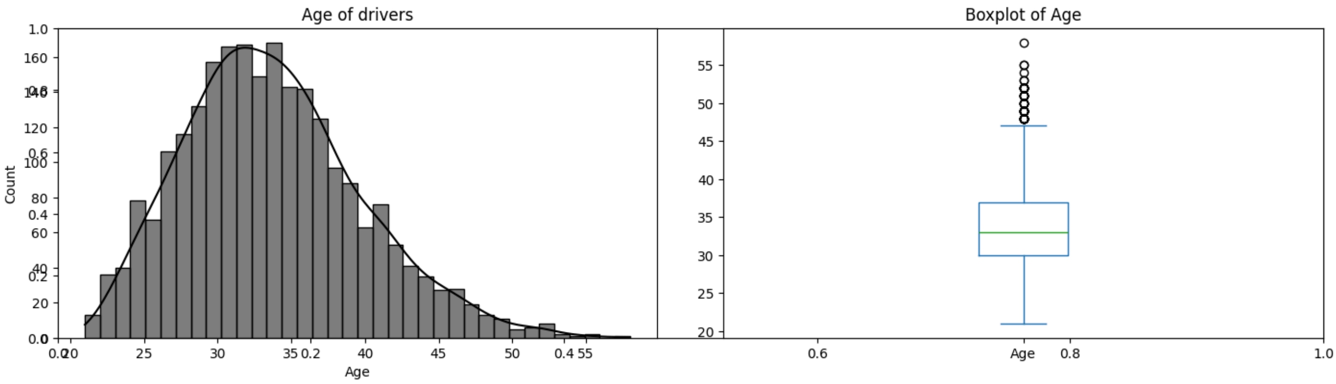
4.Out of 2381 employees, 1026 joined with the grade as 1, 815 employees joined with the grade 2.

5.Out of 2381 employees, 855 employees had their designation as 2 at the time of reporting.

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

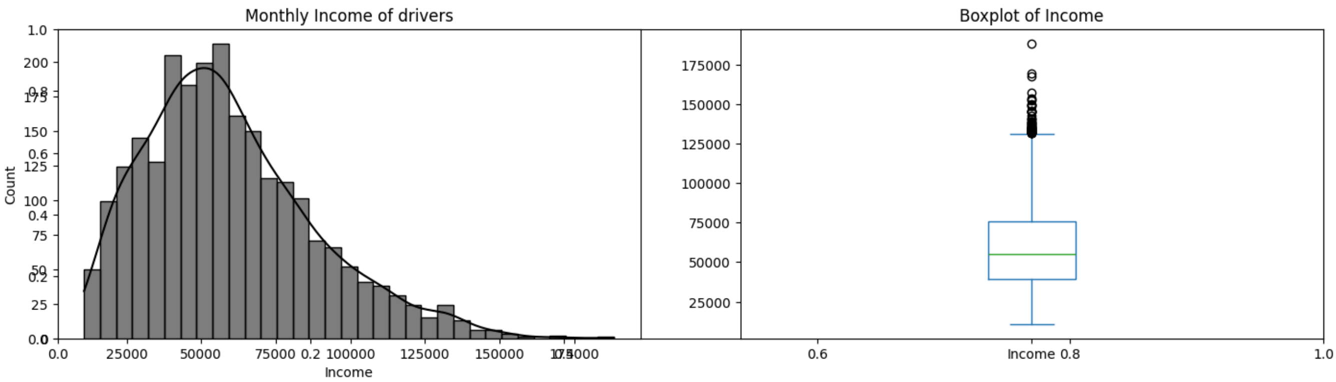
7.Out of 2381 employees, the quarterly rating has not increased for 2076 employees.

```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.histplot(final_data['Age'],color='black', kde=True)
plt.title("Age of drivers")
plt.subplot(122)
final_data['Age'].plot.box(title='Boxplot of Age')
plt.tight_layout(pad=3)
```



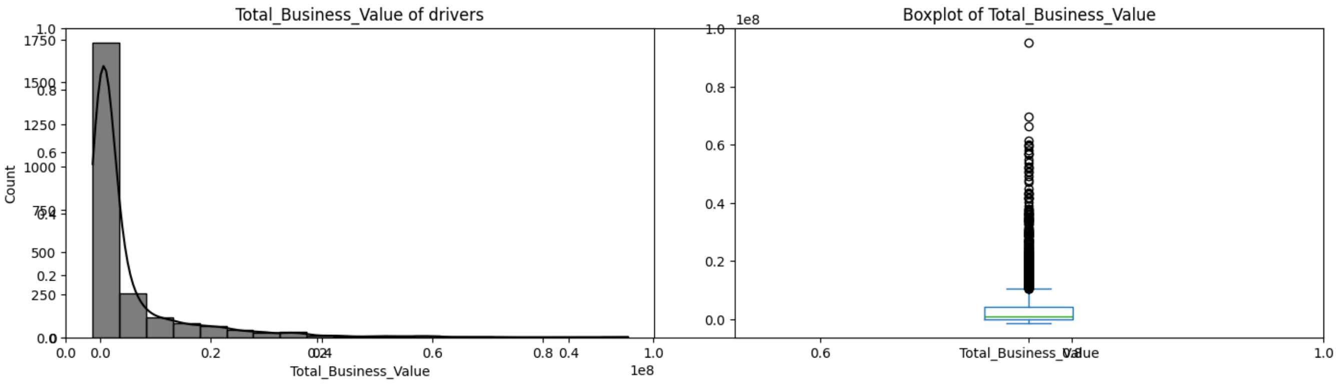

The distribution of age slightly skewed on right which might indicate the outliers in the data

```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.histplot(final_data['Income'],color='black', kde=True)
plt.title("Monthly Income of drivers")
plt.subplot(122)
final_data['Income'].plot.box(title='Boxplot of Income')
plt.tight_layout(pad=3)
```




The distribution of monthly income skewed on right which might indicate the outliers in the data

```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.histplot(final_data['Total_Business_Value'],color='black', kde=True, bins=20)
plt.title("Total_Business_Value of drivers")
plt.subplot(122)
final_data['Total_Business_Value'].plot.box(title='Boxplot of Total_Business_Value')
plt.tight_layout(pad=3)
```

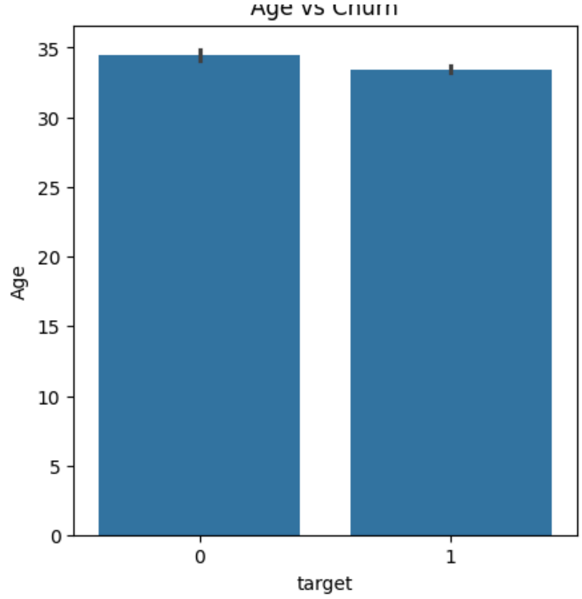



The distribution of total business value highly skewed on right which might indicate the outliers in the data

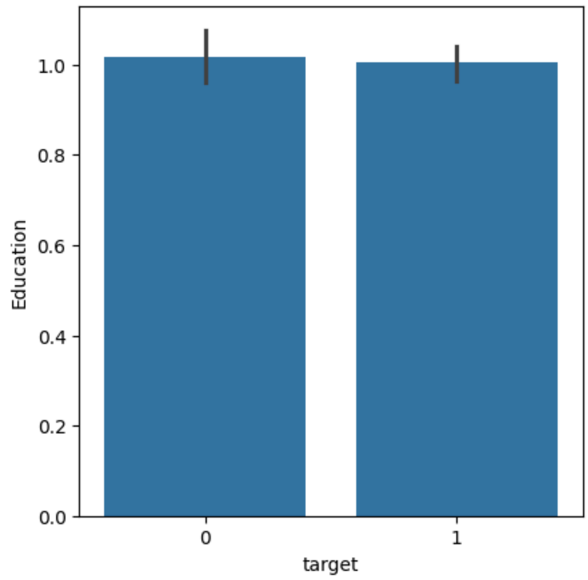
## Bivariate Analysis

```
plt.figure(figsize=(10,20))

plt.subplot(421)
sns.barplot(data=final_data, x="target", y="Age")
plt.title("Age vs Churn")

plt.subplot(422)
sns.barplot(data=final_data, x="target", y="Education")
plt.title("Education vs Churn")

plt.subplot(423)
sns.barplot(data=final_data, x="target", y="Gender")
plt.title("Gender vs Churn")

plt.subplot(425)
sns.barplot(data=final_data, x="target", y="Grade")
plt.title("Grade vs Churn")

plt.subplot(426)
sns.barplot(data=final_data, x="target", y="Joining_Designation")
plt.title("Joining_Designation vs Churn")

plt.subplot(427)
sns.barplot(data=final_data, x="target", y="Salary_Increased")
plt.title("Salary_Increased vs Churn")

plt.subplot(428)
sns.barplot(data=final_data, x="target", y="Quarterly_Rating_Increased")
plt.title("Quarterly_Rating_Increased vs Churn")

plt.tight_layout(pad=3)
```

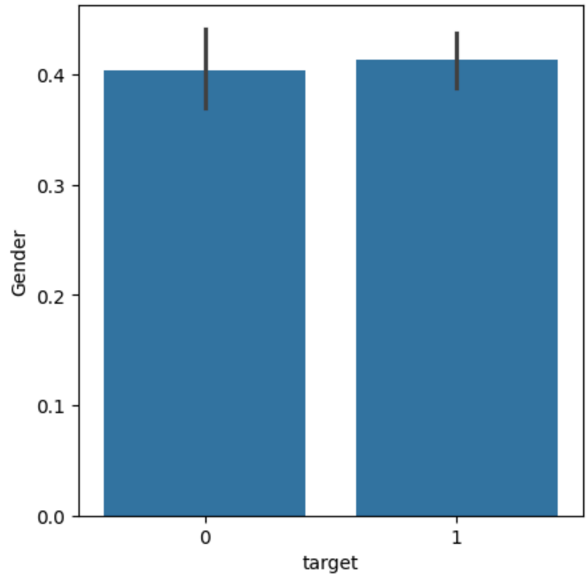
Age vs Churn



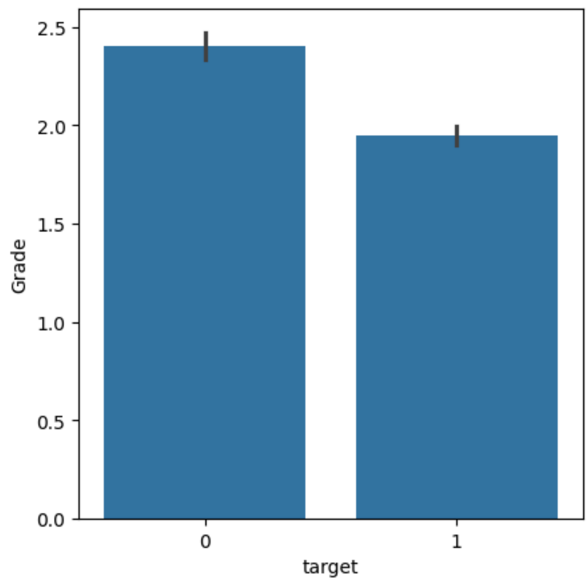

Education vs Churn



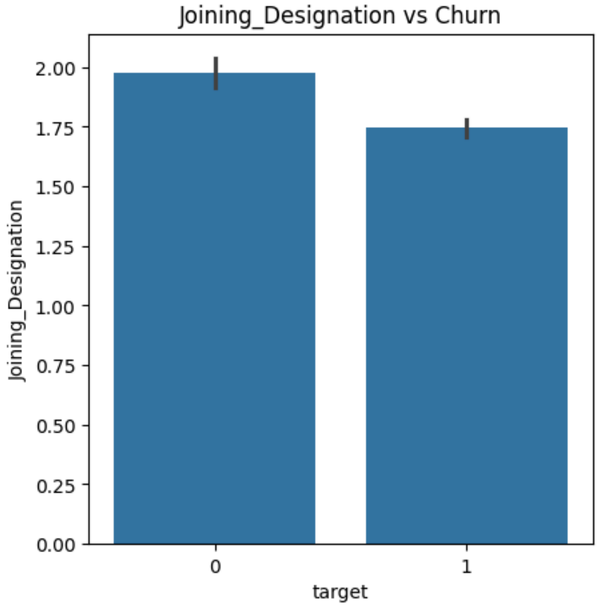

Gender vs Churn



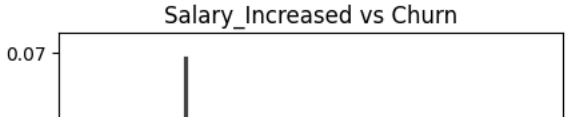

Grade vs Churn



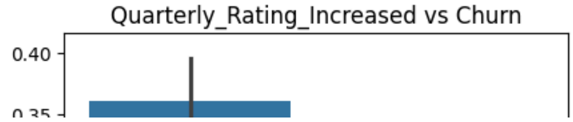

Joining_Designation vs Churn

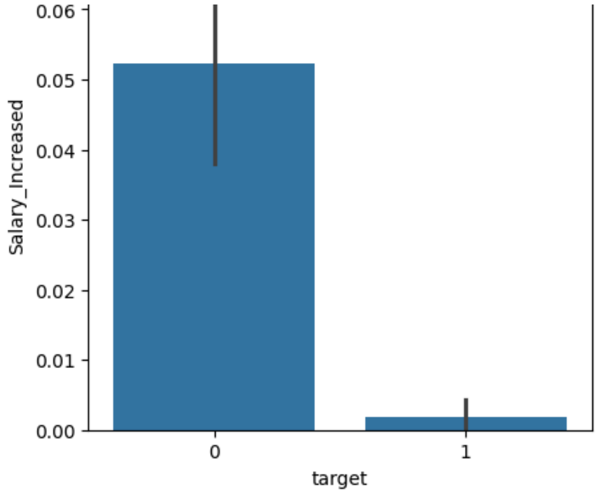


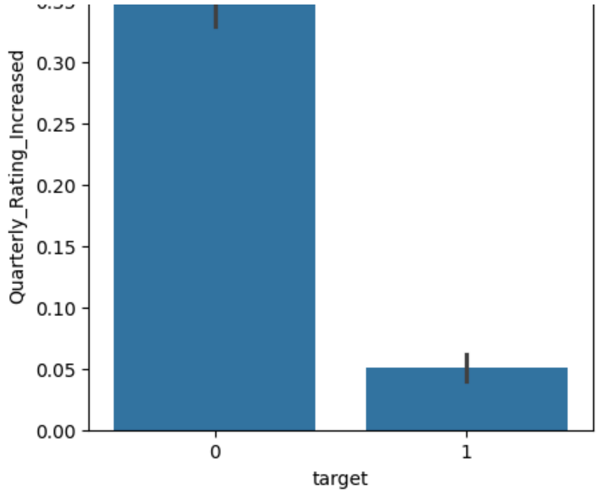

Salary_Increased vs Churn




Quarterly_Rating_Increased vs Churn






Insights-

1.The proportion of Age, gender and education is more or less the same for both the employees who left the organization and those who did not leave.

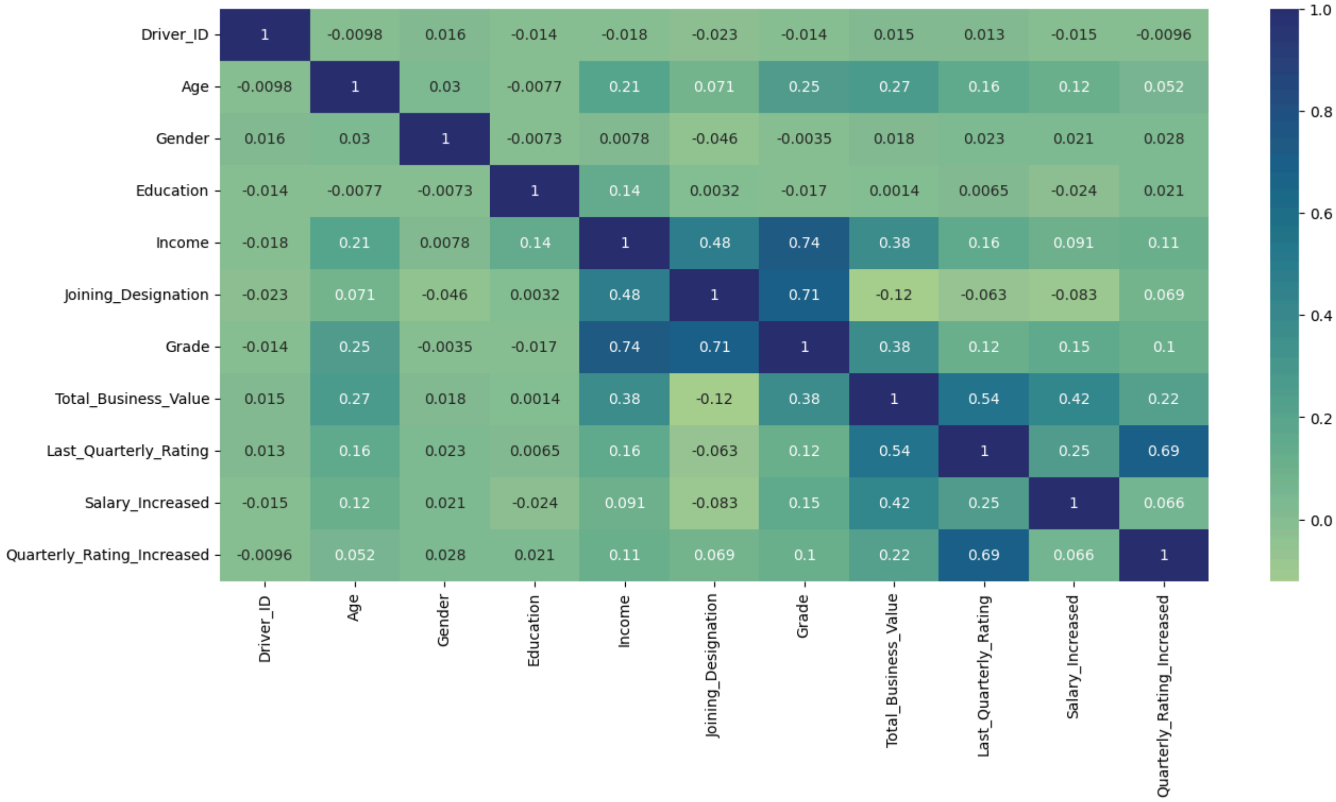
2.The employees who have their grade as 3 or 4 at the time of joining are less likely to leave the organization.

3.The employees whose quarterly rating has increased are less likely to leave the organization.

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

## Correlation Analysis

```
plt.figure(figsize=(15, 7))

# Drop the 'City' column before calculating correlation
sns.heatmap(final_data.drop('City', axis=1).corr(method="pearson"), annot=True, cmap="crest")
plt.show()
```




## Insights-

1.Income and Grade is highly correlated

2.Joining Designation and Grade is highly correlated

3.Total Business value and salary increament is correlated

## One Hot Encoding

As there is only one categorical values in our dataset. We will opt one hot encoder to convert it to numerical.

```
final_data=pd.concat([final_data,final_data['City']],axis=1)
```

```
# Create the target column
lwd = (processed_df.groupby(["Driver_ID"]).agg({"LastWorkingDate": "last"})["LastWorkingDate"].isna()).reset_index()
lwrid = lwd[lwd["LastWorkingDate"] == True]["Driver_ID"]
target = []

for i in final_data["Driver_ID"]:
    if i in lwrid.values:
        target.append(0)
    else:
        target.append(1)

final_data["target"] = target
```

```
final_data.shape
```

## Standardisation for training data

```
X=final_data.drop(['Driver_ID','target','City'],axis=1)
X_cols=X.columns
scaler=MinMaxScaler()
scaler.fit_transform(X)
```

```
array([[0.18918919, 0.        , 1.        , ..., 0.33333333, 0.        ,
        0.        ],
       [0.27027027, 0.        , 1.        , ..., 0.        , 0.        ,
        0.        ],
       [0.59459459, 0.        , 1.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.64864865, 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.18918919, 1.        , 1.        , ..., 0.        , 0.        ,
        0.        ],
       [0.24324324, 0.        , 1.        , ..., 0.33333333, 0.        ,
        1.        ]])
```

```
X=pd.DataFrame(X)
X.columns=X_cols
X
```

| | Age | Gender | Education | Income | Joining_Designation | Grade | Total_Business_Value | Last_Quarterly_Rating | Salary_Increased | Qu |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 28.0 | 0.0 | 2.0 | 57387.0 | 1.0 | 1.0 | 1715580.0 | 2.0 | 0 | |
| **1** | 31.0 | 0.0 | 2.0 | 67016.0 | 2.0 | 2.0 | 0.0 | 1.0 | 0 | |
| **2** | 43.0 | 0.0 | 2.0 | 65603.0 | 2.0 | 2.0 | 350000.0 | 1.0 | 0 | |
| **3** | 29.0 | 0.0 | 0.0 | 46368.0 | 1.0 | 1.0 | 120360.0 | 1.0 | 0 | |
| **4** | 31.0 | 1.0 | 1.0 | 78728.0 | 3.0 | 3.0 | 1265000.0 | 2.0 | 0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2376** | 34.0 | 0.0 | 0.0 | 82815.0 | 2.0 | 3.0 | 21748820.0 | 4.0 | 0 | |
| **2377** | 34.0 | 1.0 | 0.0 | 12105.0 | 1.0 | 1.0 | 0.0 | 1.0 | 0 | |
| **2378** | 45.0 | 0.0 | 0.0 | 35370.0 | 2.0 | 2.0 | 2815090.0 | 1.0 | 0 | |
| **2379** | 28.0 | 1.0 | 2.0 | 69498.0 | 1.0 | 1.0 | 977830.0 | 1.0 | 0 | |
| **2380** | 30.0 | 0.0 | 2.0 | 70254.0 | 2.0 | 2.0 | 2298240.0 | 2.0 | 0 | |

2381 rows × 10 columns

Next steps: Generate code with X | View recommended plots | New interactive sheet

Train Test Split

```
y = final_data["target"]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=7,shuffle=True)
```

```
print("X_train Shape: ", X_train.shape)
print("X_test Shape: ", X_test.shape)
print("y_train Shape: ", y_train.shape)
print("y_test Shape: ", y_test.shape)
```

```
X_train Shape:  (1904, 10)
X_test Shape:  (477, 10)
y_train Shape:  (1904,)
y_test Shape:  (477,)
```

## Random Forest Classifier before Balancing

keeping max_depth small to avoid overfitting

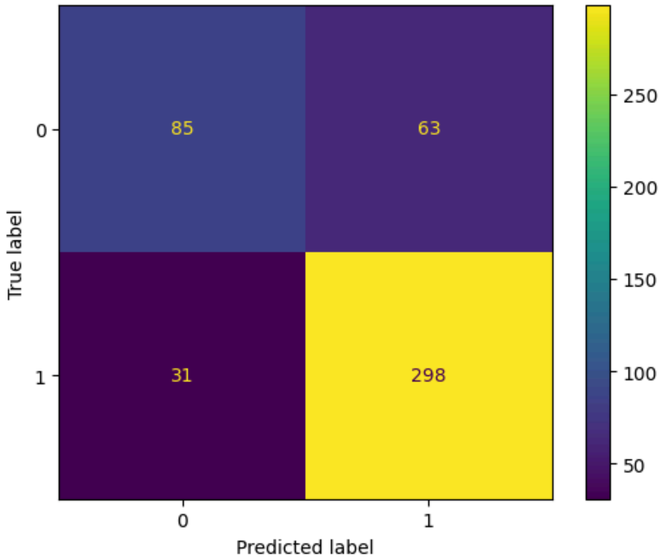
```
params= {
  "max_depth":[2,3,4],
 "n_estimators":[50,100,150,200],

}
start_time=time.time()
random_forest=RandomForestClassifier(class_weight="balanced")
c=GridSearchCV(estimator=random_forest,param_grid=params,n_jobs=-1,cv=3,verbose=True,scoring='f1')

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': 50}
Best Score:  0.8626575080063451

Elapsed Time:  12.047347068786621
```

```
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       0.73      0.57      0.64       148
           1       0.83      0.91      0.86       329

    accuracy                           0.80       477
   macro avg       0.78      0.74      0.75       477
weighted avg       0.80      0.80      0.80       477

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x79188121e310>
```




Random Forest Classifier with balanced class weight

Out of all prediction, the measure for correctly predicted 0 is 73% and for 1 is 82% (Precision)

Out of all actual 0, the measure for correctly predicted is 57% and for 1 is 90% (Recall)

As this is imbalanced dataset. We give importance to F1-Score metrics

F1 Score of 0 is 64%

F! Score of 1 is 86%

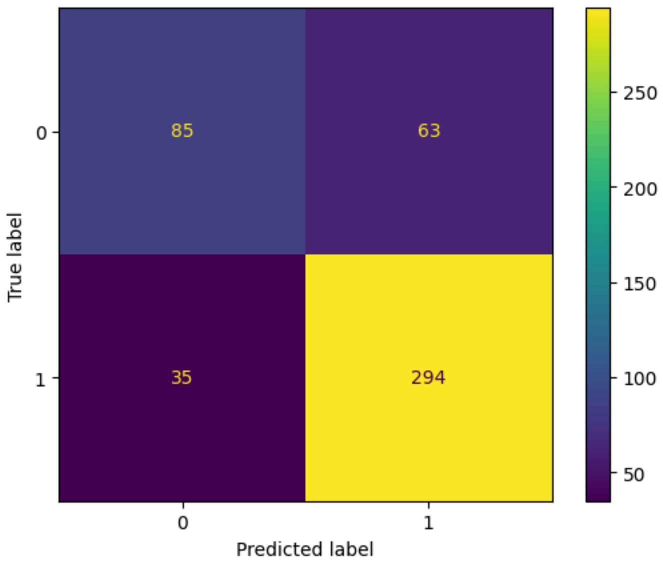
Lets try out bootstrapped random forest using subsample

```
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, verbose=True, scoring='f1')

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': 150}
Best Score:  0.861917343462629

Elapsed Time:  12.531663179397583
```

```
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       0.71      0.57      0.63       148
           1       0.82      0.89      0.86       329

    accuracy                           0.79       477
   macro avg       0.77      0.73      0.75       477
weighted avg       0.79      0.79      0.79       477

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7918812df190>
```




Random Forest Classifier with balanced class weight

Out of all prediction, the measure for correctly predicted 0 is 75% and for 1 is 83% (Precision)

Out of all actual 0, the measure for correctly predicted is 57% and for 1 is 91% (Recall)

As this is imbalanced dataset. We give importance to F1-Score metrics

F1 Score of 0 is 65%

F! Score of 1 is 87%

There is not much significant difference in the matrices observed for bootstrapped Random Forest and Weighted Random Forest

Lets try balancing

## Balancing Dataset using SMOTE

As the target variable is imbalanced towards 1. We will use SMOTE to balance the dataset

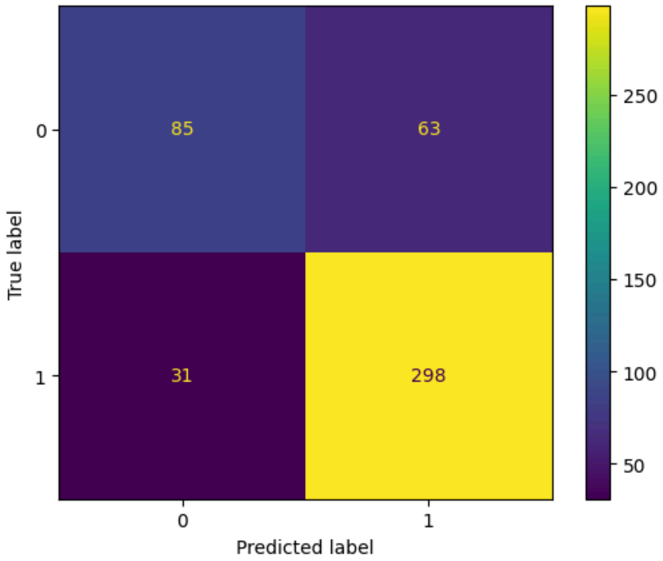
```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train == 0)))

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': 1287
Before OverSampling, counts of label '0': 617

After OverSampling, the shape of train_X: (2574, 10)
After OverSampling, the shape of train_y: (2574,)

After OverSampling, counts of label '1': 1287
After OverSampling, counts of label '0': 1287
/tmp/ipython-input-63-4167766004.py:5: FutureWarning: Series.ravel is deprecated. The underlying array is already 1D, so ravel is nc
  X_train,y_train=sm.fit_resample(X_train,y_train.ravel())
```

## Ensemble Learning: Bagging

```
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, verbose=True, scoring='f1')

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': 150}
Best Score:  0.8190827560355075

Elapsed Time:  19.900463342666626
              precision    recall  f1-score   support

           0       0.73      0.57      0.64       148
           1       0.83      0.91      0.86       329

    accuracy                           0.80       477
   macro avg       0.78      0.74      0.75       477
weighted avg       0.80      0.80      0.80       477

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x791880f04450>
```




Random Forest Classifier with balanced class weight

Out of all prediction, the measure for correctly predicted 0 is 74% and for 1 is 83% (Precision)

Out of all actual 0, the measure for correctly predicted is 57% and for 1 is 91% (Recall)

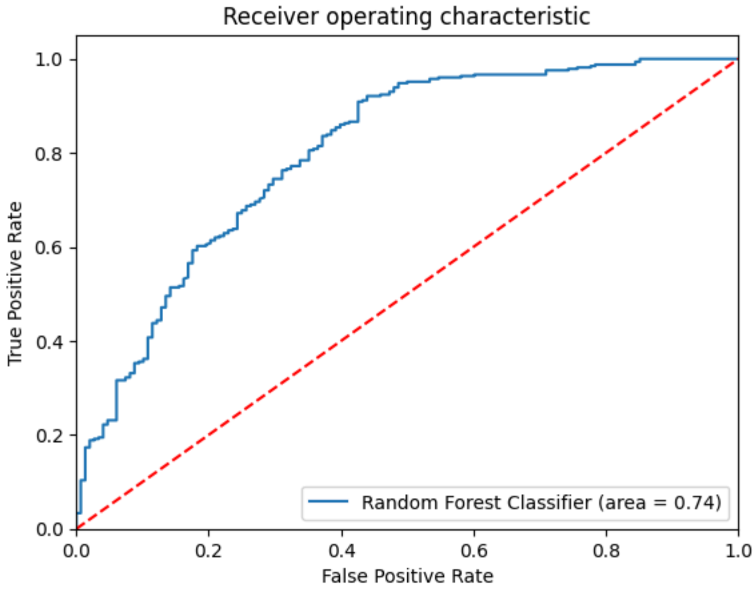
As this is imbalanced dataset. We give importance to F1-Score metrics-

F1 Score of 0 is 65%

F! Score of 1 is 87%

## ROC-AUC Curve

```
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_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()
```

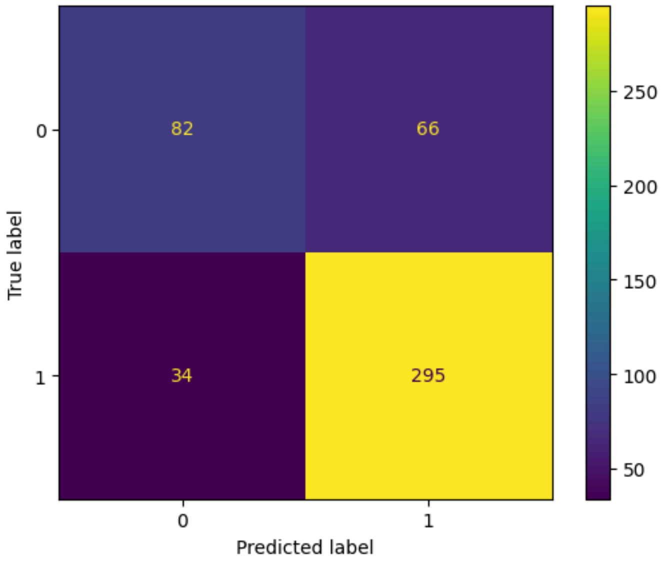



## Ensemble Learning: Boosting

Gradient Boosting Classifier

```
params ={
    "max_depth" : [2,3,4],
    "n_estimators" : [20,100,150,200],
    "loss" : ['log_loss','exponential'],
    "learning_rate" : [0.1,0.2,0.3],
    "subsample" : [0.1, 0.2, 0.5, 0.8, 1]

}

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

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': 0.1, 'loss': 'log_loss', 'max_depth': 3, 'n_estimators': 100, 'subsample': 1}
Best Score:  0.8391608391608392

 Elapsed Time:  361.077513217926
              precision    recall  f1-score   support

           0       0.71      0.55      0.62       148
           1       0.82      0.90      0.86       329

    accuracy                           0.79       477
   macro avg       0.76      0.73      0.74       477
weighted avg       0.78      0.79      0.78       477

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7918813f0b50>
```




Gradient Boosting Classifier Metrics

Out of all prediction, the measure for correctly predicted 0 is 62% and for 1 is 82% (Precision)

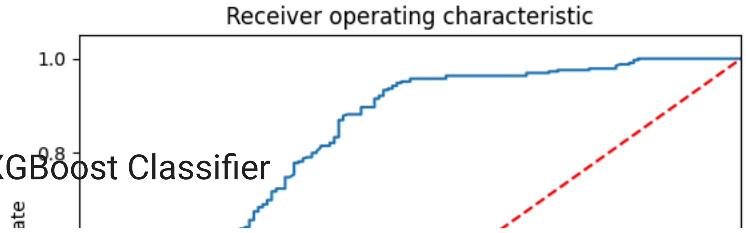
Out of all actual 0, the measure for correctly predicted is 60% and for 1 is 83% (Recall)

As this is imbalanced dataset. We give importance to F1-Score metrics

F1 Score of 0 is 61%

F1 Score of 1 is 83%

```
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_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()
```

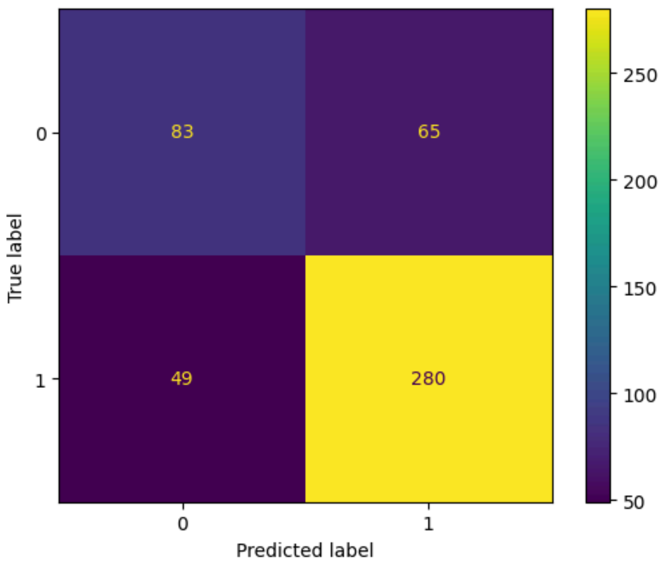



## XGBoost Classifier

```
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_).plot()
```

Gradient Boosting Classifier (area = 0.73)
0.0 0.2 0.4 0.6 0.8 1.0
False Positive Rate

```
XGBoost Classifier Score:  0.7610062893081762

              precision    recall  f1-score   support

           0       0.63      0.56      0.59       148
           1       0.81      0.85      0.83       329

    accuracy                           0.76       477
   macro avg       0.72      0.71      0.71       477
weighted avg       0.75      0.76      0.76       477

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7918726e87d0>
```




XGBoost Classifier with balanced class weight-

Out of all prediction, the measure for correctly predicted 0 is 62% and for 1 is 81% (Precision)

Out of all actual 0, the measure for correctly predicted is 57% and for 1 is 84% (Recall)

As this is imbalanced dataset. We give importance to F1-Score metrics-

F1 Score of 0 is 60%

F1 Score of 1 is 83%

```
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])
```