OLA - Ensemble Learning

In [1]:
```
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 StandardScaler
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

import warnings
warnings.filterwarnings("ignore")
```

Import the dataset

In [2]:
```
df=pd.read_csv("ola_driver_scaler.csv")
df.head()
```

Out[2]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarterl Ratin
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	2381060	
1	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	
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	0	
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	0	

Basic Metric

In [3]:
```
df.shape
```
Out[3]:
```
(19104, 14)
```

In [4]:
```
df.ndim
```
Out[4]:
```
2
```

In [5]:
```
df.size
```
Out[5]:
```
267456
```

In [6]:
```
df.dtypes
```

Out[6]:
```
Unnamed: 0                  int64
MMM-YY                     object
Driver_ID                   int64
Age                       float64
Gender                    float64
City                       object
Education_Level             int64
Income                      int64
Dateofjoining              object
LastWorkingDate            object
Joining Designation         int64
Grade                       int64
Total Business Value        int64
Quarterly Rating            int64
dtype: object
```

In [7]:
```
#Unique Drivers
df["Driver_ID"].nunique()
```

Out[7]:
```
2381
```

In [8]:
```
df.drop("Unnamed: 0", axis = 1, inplace = True)
```

Converting respective data type

In [9]:
```
df['MMM-YY'] = pd.to_datetime(df['MMM-YY'])
df['Dateofjoining'] = pd.to_datetime(df['Dateofjoining'])
df['LastWorkingDate'] = pd.to_datetime(df['LastWorkingDate'])
```

In [10]:
```
df.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
```

Check for Null values

In [11]:
```python
df.isna().sum()
```

Out[11]:
```
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
```

KNN Imputation

In [12]:
```
num_col = df.select_dtypes(np.number)
num_col.columns
```
Out[12]:
```
Index(['Driver_ID', 'Age', 'Gender', 'Education_Level', 'Income',
       'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating'],
      dtype='object')
```
In [13]:
```
num_col.drop(["Driver_ID"], axis = 1, inplace = True)
```
In [14]:
```
imputer = KNNImputer(n_neighbors=5, weights='uniform', metric='nan_euclidean')
imputer.fit(num_col)
df_new = imputer.transform(num_col)
```
In [15]:
```
data_new  = pd.DataFrame(df_new)
```
In [16]:
```
data_new .columns = num_col.columns
```
In [17]:
```
data_new .isnull().sum()
```
Out[17]:
```
Age                     0
Gender                  0
Education_Level         0
Income                  0
Joining Designation     0
Grade                   0
Total Business Value    0
Quarterly Rating        0
dtype: int64
```
Merge data

In [18]:
```
columns= list(set(df.columns).difference(set(num_col)))
columns
```
Out[18]:
```
['LastWorkingDate', 'Dateofjoining', 'City', 'MMM-YY', 'Driver_ID']
```
In [19]:
```
new_df  = pd.concat([data_new, df[columns]], axis=1)
new_df.shape
```

Out[19]: (19104, 13)

In [20]:
```python
new_df.head()
```

Out[20]:

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

Aggregation of data

In [21]:
```python
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()
```

Out[21]:

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

In [22]:

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

In [23]:

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

In [24]:

```
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 Designati
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 Va
final_data['Last_Quarterly_Rating'] = list(processed_df.groupby('Driver_ID').agg({'Quarterly Rating':'last'})['Quarterly Rating']
```

In [25]:

```
final_data.shape
```

Out[25]:

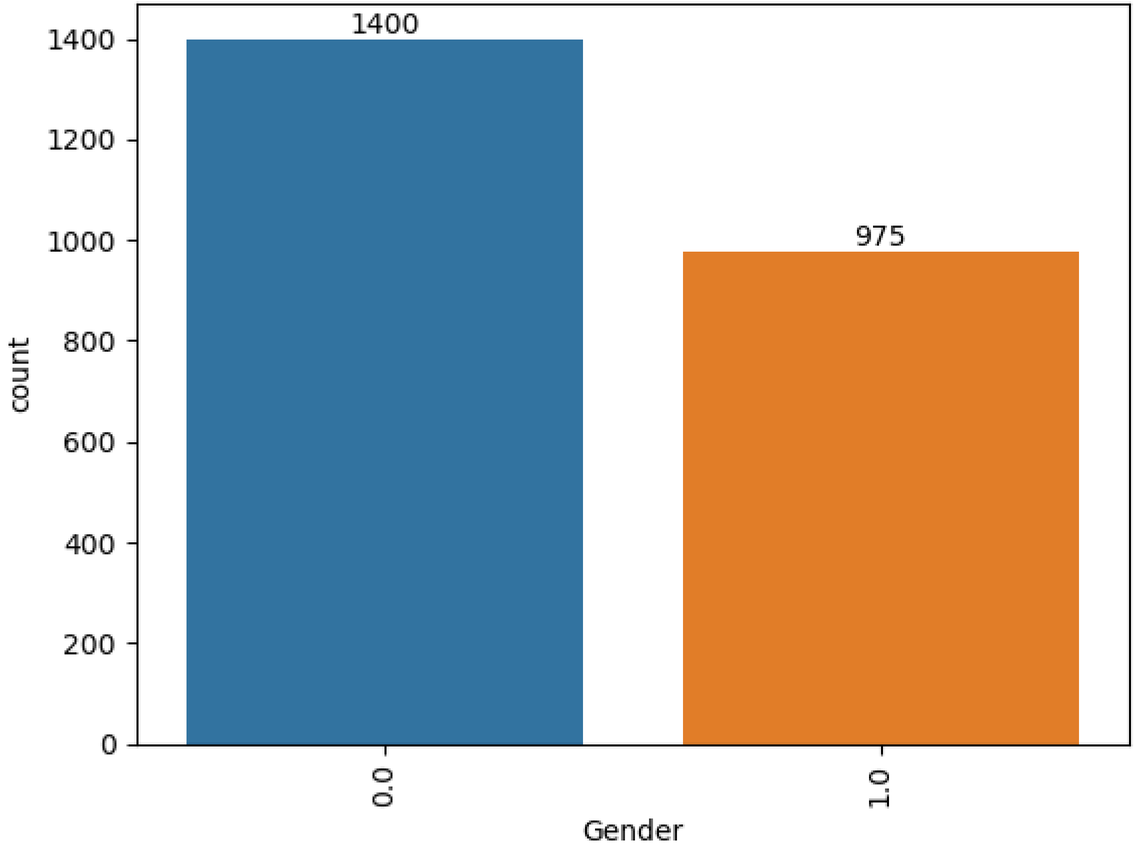
```
(2381, 10)
```

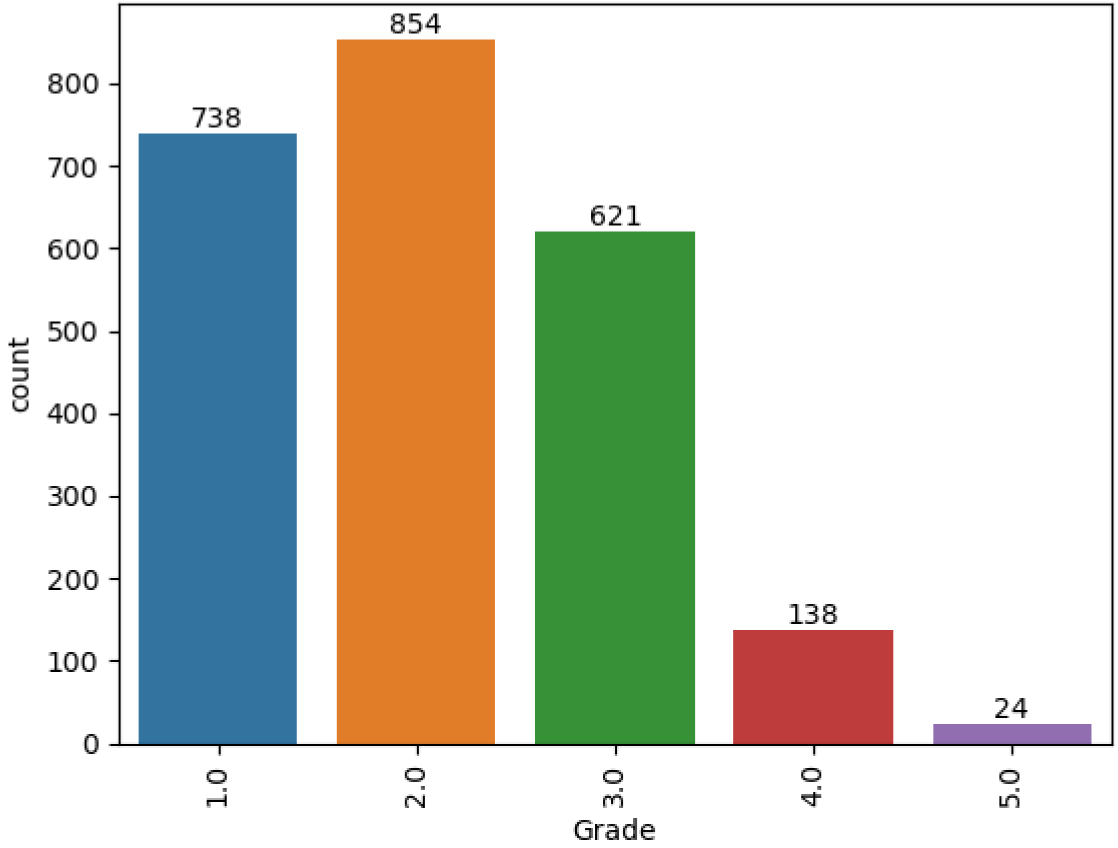
In [26]:

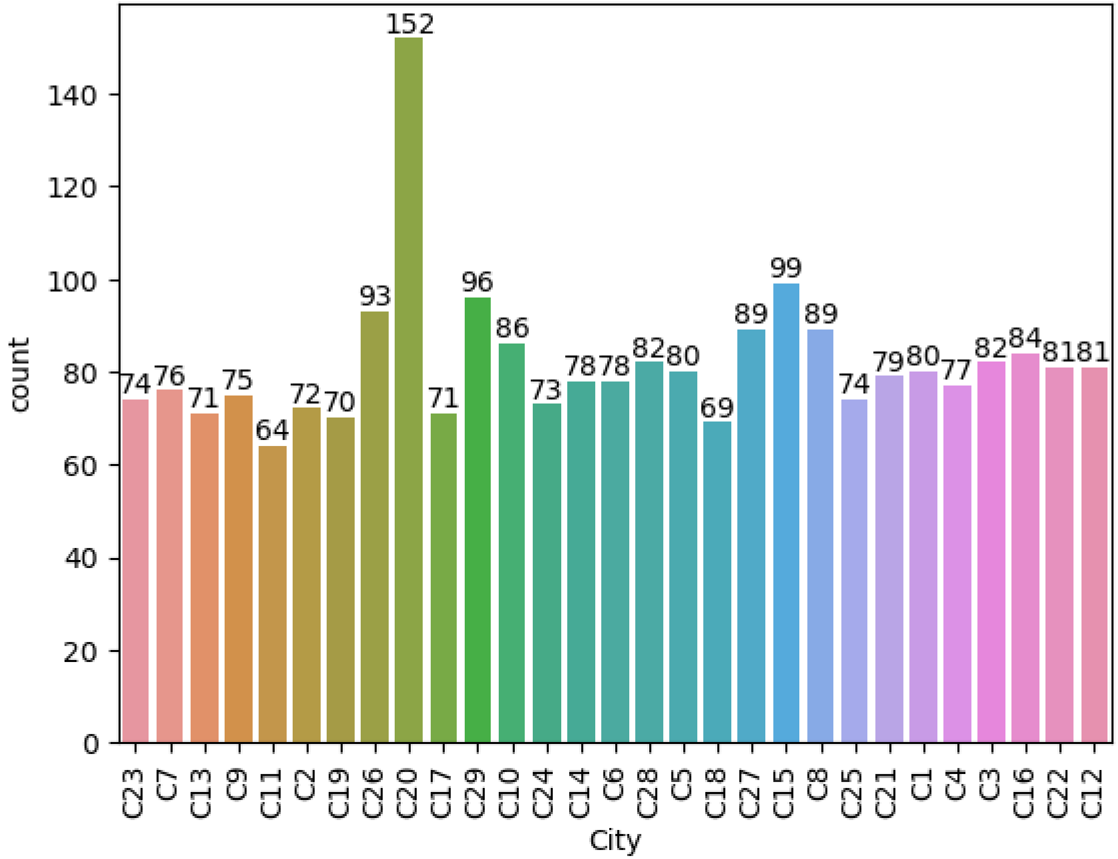
```
final_data = final_data[final_data['Gender'].isin([0.0, 1.0])]
```

Univariate Analysis

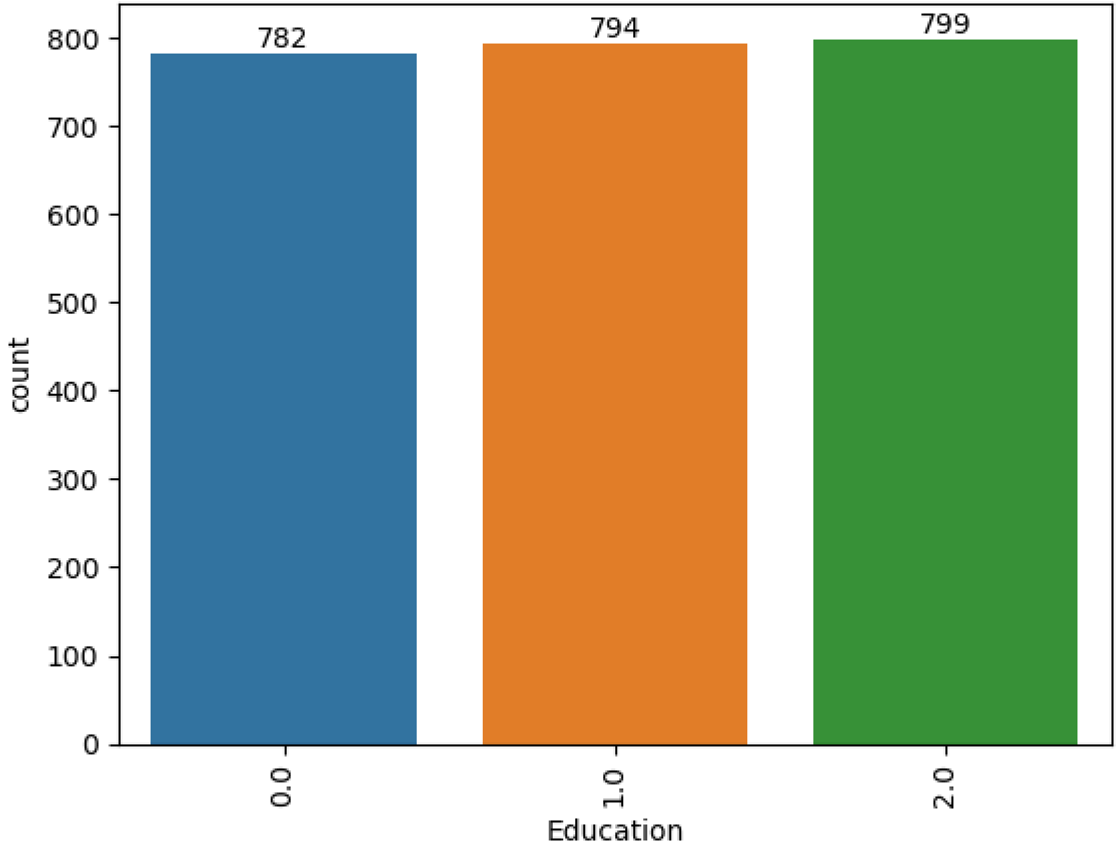
In [27]:
```
col=['Gender', 'Grade','City','Education','Last_Quarterly_Rating']
for i in col:
    sns.countplot(data=final_data, x=i)
    plt.xticks(rotation=90)
    ax=plt.gca()
    for i in ax.containers:
        ax.bar_label(i)
    plt.show()
```



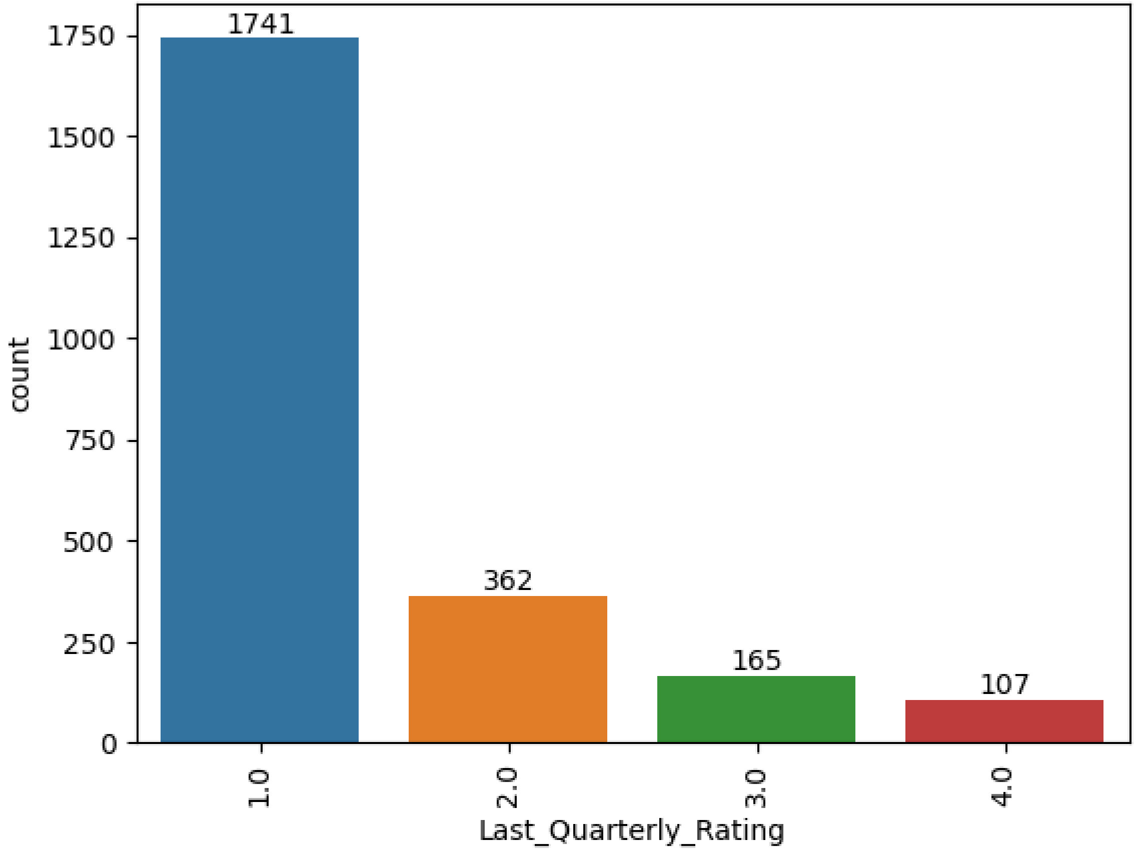

854
738
621
138
24
800
700
600
500
400
300
200
100
0
count
1.0
2.0
3.0
4.0
5.0
Grade



count
City
0
20
40
60
80
100
120
140
74 76 71 75 64 72 70 93 152 71 96 86 73 78 78 82 80 69 89 99 89 74 79 80 77 82 84 81 81
C23 C7 C13 C9 C11 C2 C19 C26 C20 C17 C29 C10 C24 C14 C6 C28 C5 C18 C27 C15 C8 C25 C21 C1 C4 C3 C16 C22 C12



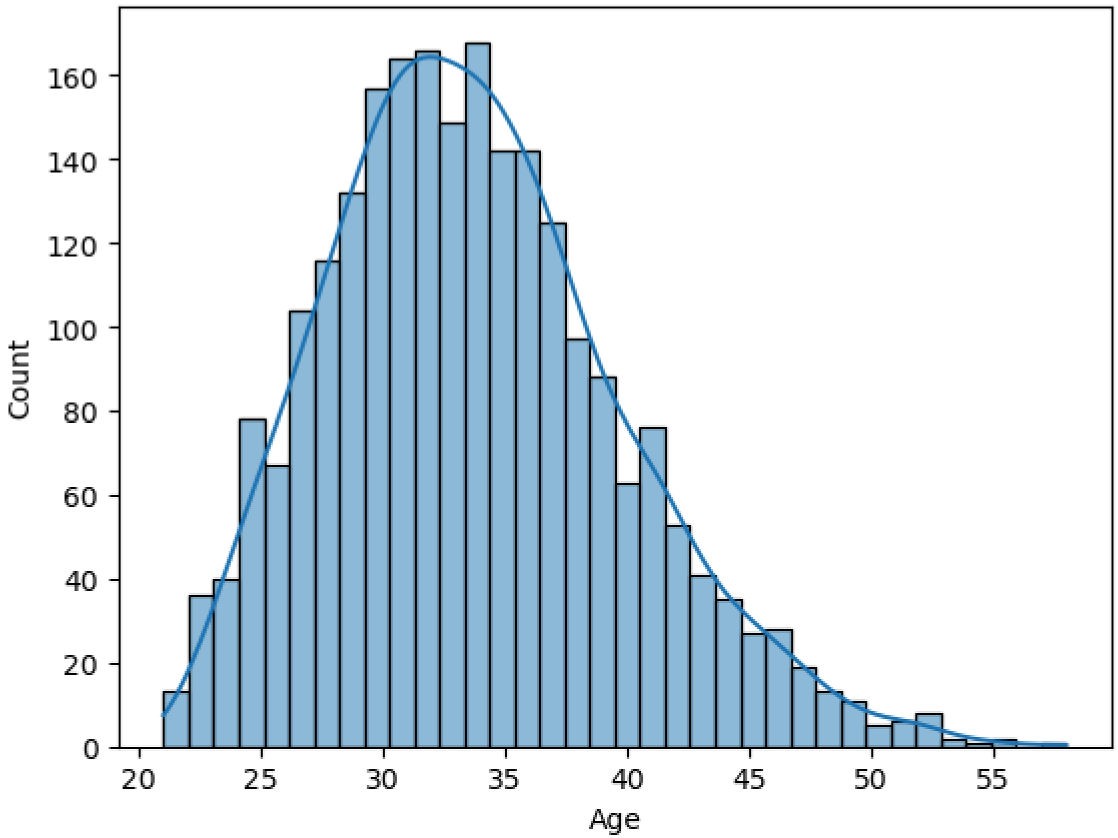
782
794
799
800
700
600
500
400
300
200
100
0
count
0.0
1.0
2.0
Education



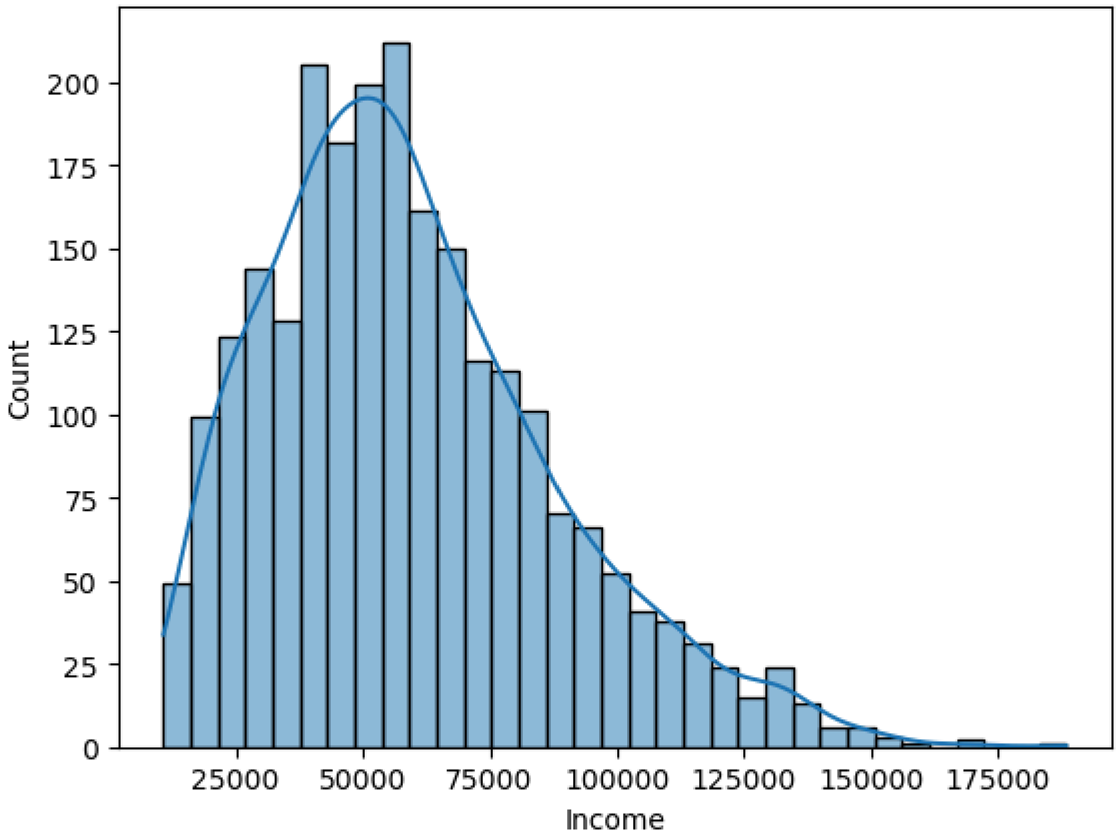
Insights

- Compared to females, males occur more frequently.
- As we move to higher grades, the number of driver occurrences seems to decrease.
- From C20 onwards, the number of drivers appears to be higher.
- The distribution of education levels is almost the same.
- As the rating increases, the occurrence of people decreases.

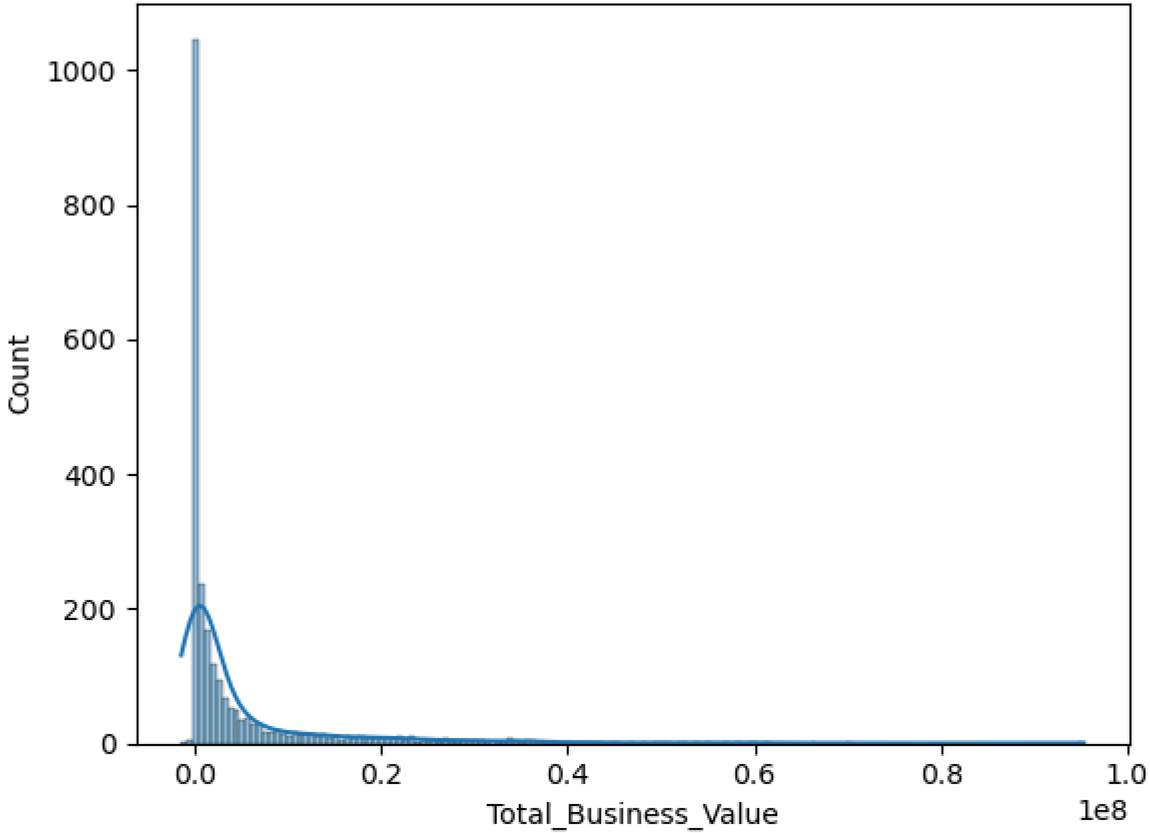
In [28]:
```
col=['Age','Income','Total_Business_Value']
for i in col:
    sns.histplot(data=final_data,x=i,kde=True)
    plt.show()
```



Count
Age
0
20
40
60
80
100
120
140
160
20
25
30
35
40
45
50
55



0
25
50
75
100
125
150
175
200
Count
25000
50000
75000
100000
125000
150000
175000
Income

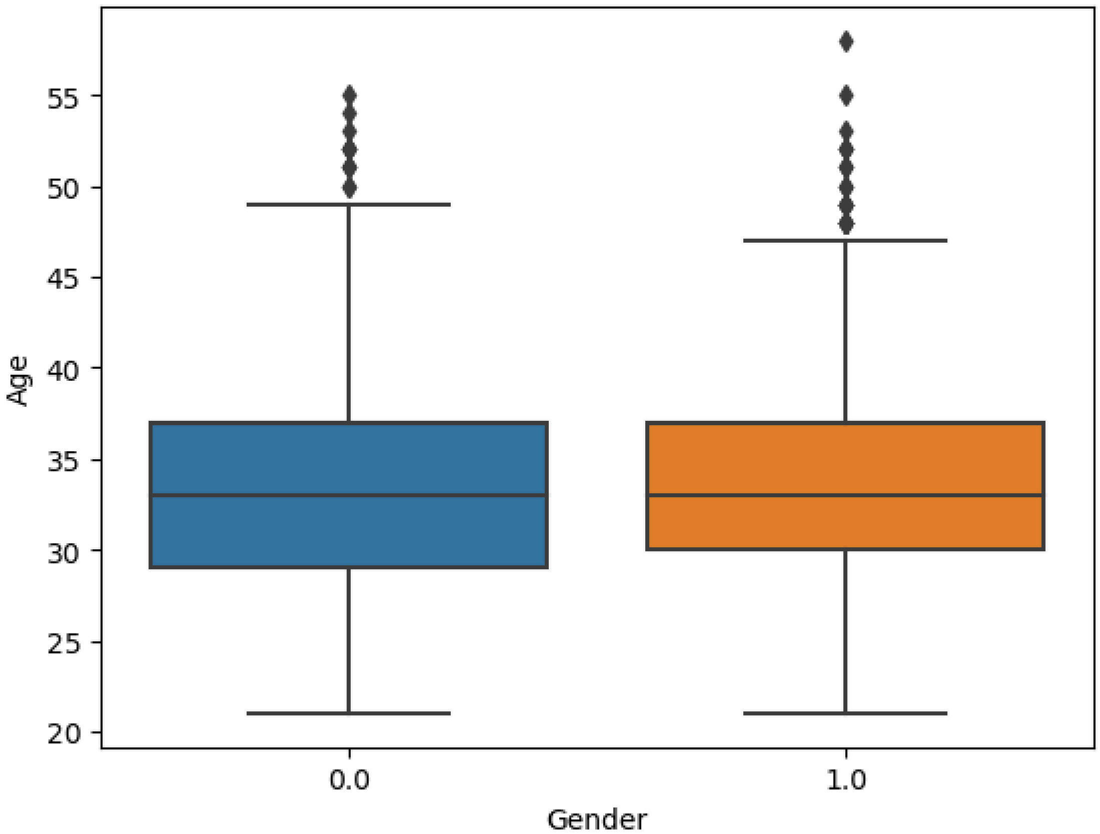


Bivariate Analysis

In [29]:
```
final_data.columns
```

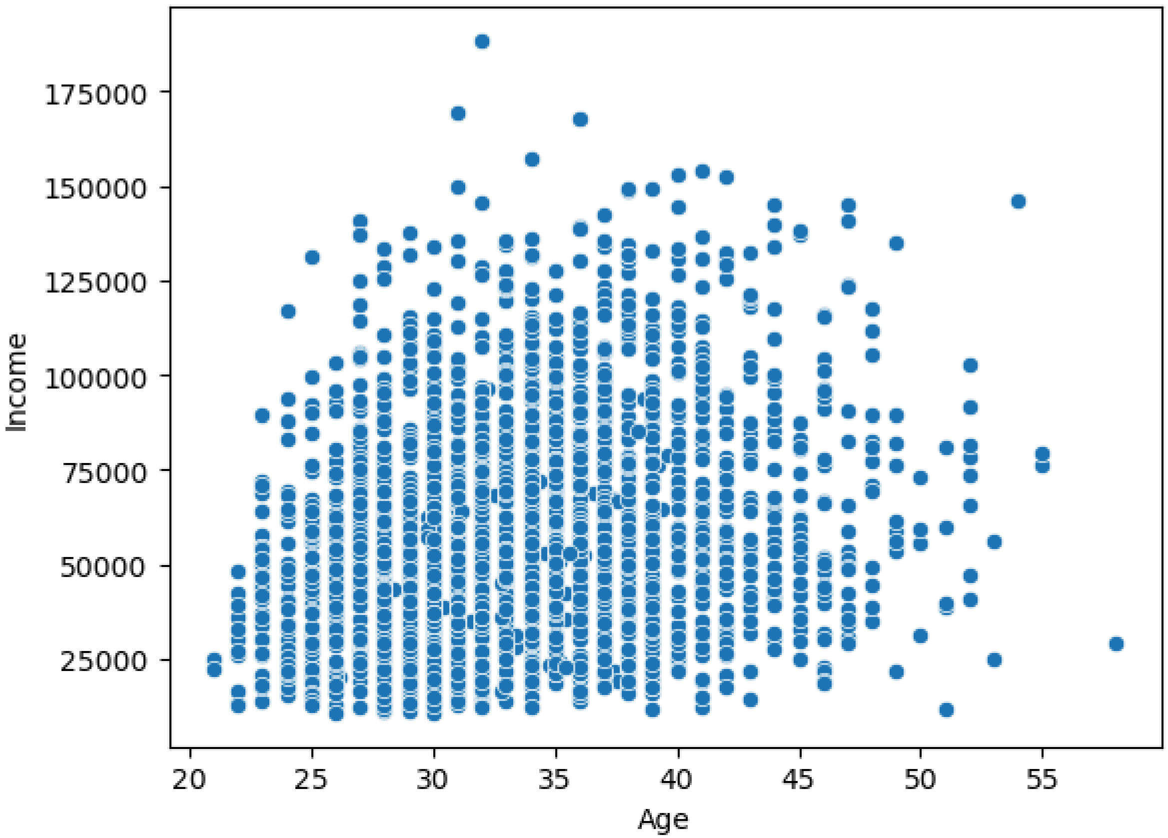
Out[29]:
```
Index(['Driver_ID', 'Age', 'Gender', 'City', 'Education', 'Income',
       'Joining_Designation', 'Grade', 'Total_Business_Value',
       'Last_Quarterly_Rating'],
      dtype='object')
```

In [30]:
```
sns.boxplot(x='Gender', y='Age', data=final_data)
plt.show()
```



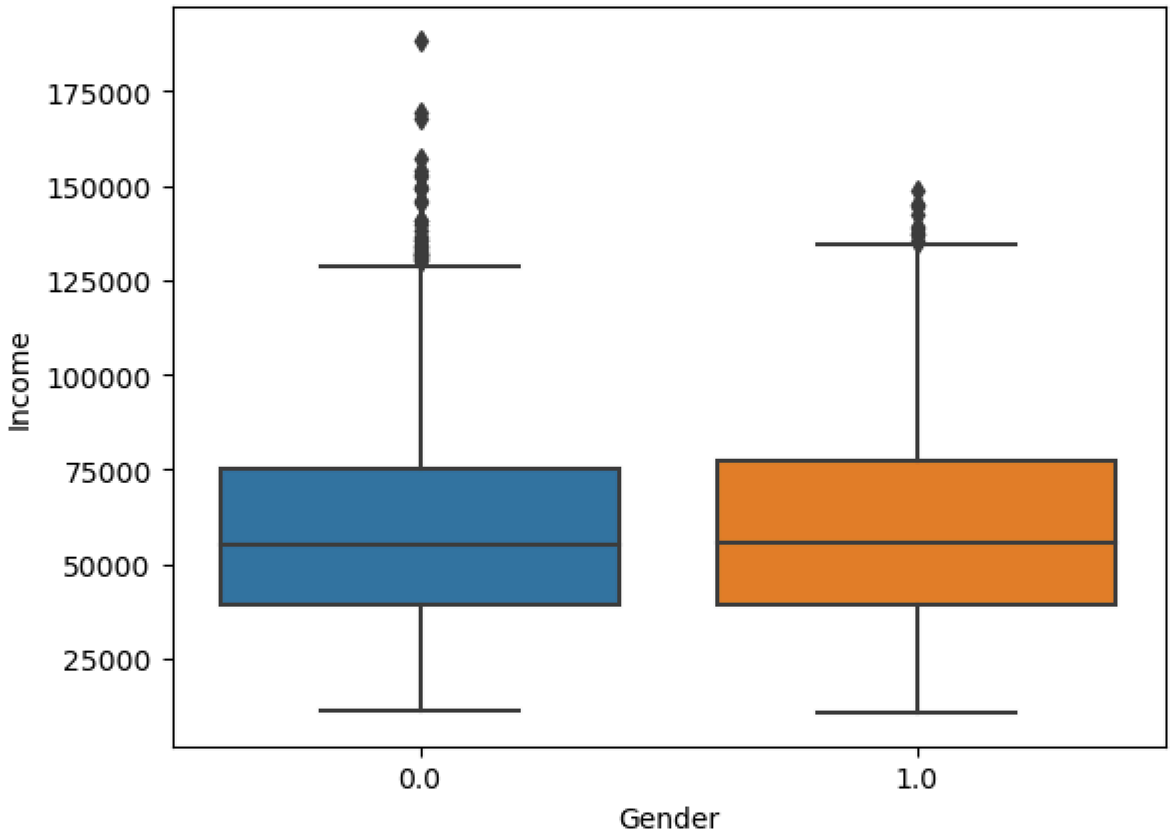
- There is no big differce

In [31]:
```
sns.scatterplot(data=final_data, x='Age', y='Income')
plt.show()
```



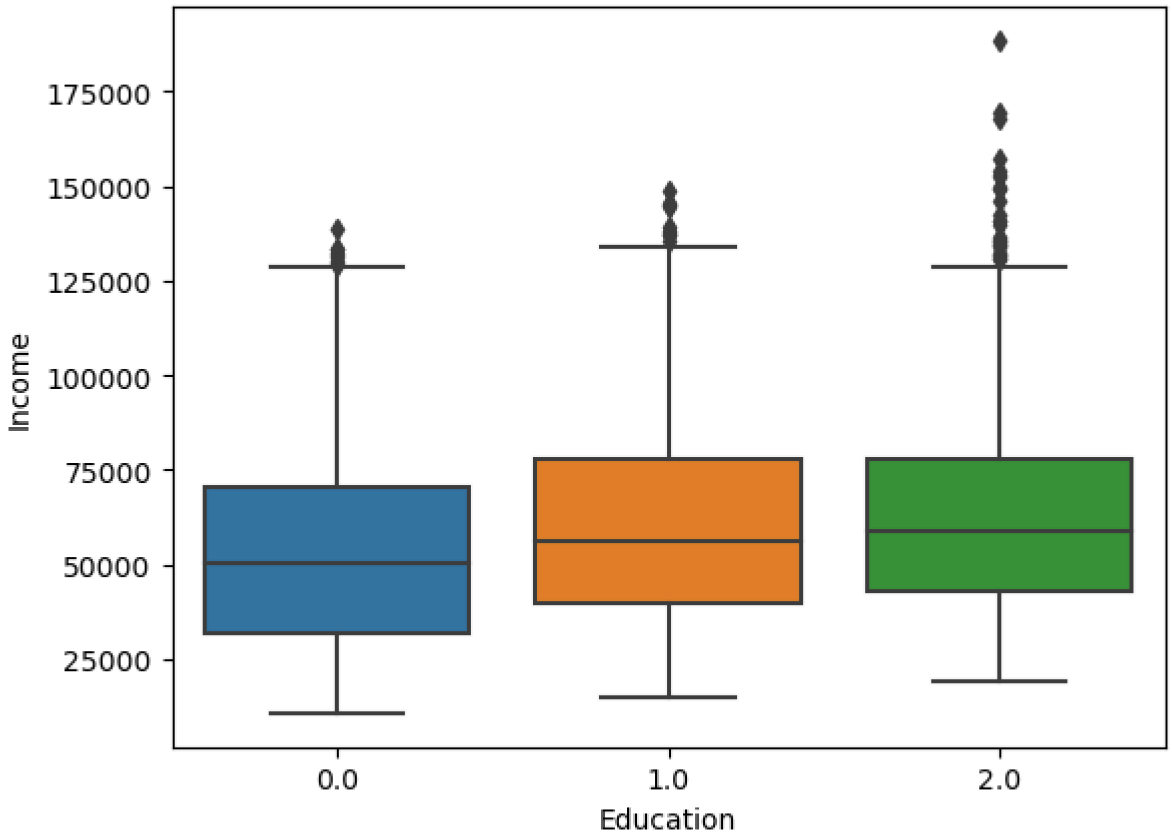
- Income appears to increase initially with age but then levels off or slightly decreases.
- There is no strong linear relationship between Age and Income.

In [32]:
```
sns.boxplot(x='Gender', y='Income', data=final_data)
plt.show()
```



- Avg income for both the gender is similar

In [33]:
```
sns.boxplot(x='Education', y='Income', data=final_data)
plt.show()
```



- As we go in higher education, income differs slightly.

Feature Engineering

In [34]:
```
final_data.head()
```

Out[34]:

	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

Quarterly rating

In [35]:

```
first_qtr = processed_df.groupby(["Driver_ID"]).agg({"Quarterly Rating": "first"})
last_qtr=processed_df.groupby(["Driver_ID"]).agg({"Quarterly Rating": "last"})
qr = (last_qtr["Quarterly Rating"] > first_qtr["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["Quarterly_Rating_Increased"] = qrl
final_data.head()
```

Out[35]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_Business_Value	Last_Quarterly_Rating	Quarterly_Rating_Increased
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	1715580.0	2.0	0
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	0
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	120360.0	1.0	0
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	1265000.0	2.0	1

Target variable creation

In [36]:
```
# Find the last working date for each driver
lwd = processed_df.groupby(["Driver_ID"]).agg({"LastWorkingDate": "last"})["LastWorkingDate"].reset_index()

# Identify drivers who have a LastWorkingDate (i.e., they have left the company)
lwrid = lwd[lwd["LastWorkingDate"].notna()]["Driver_ID"]

# Assign target values: 1 if driver has left (has LastWorkingDate), otherwise 0
final_data["target"] = final_data["Driver_ID"].isin(lwrid).astype(int)
```

Income has increased or not

In [37]:
```
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
```

In [38]:
```
final_data.head()
```

Out[38]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_Business_Value	Last_Quarterly_Rating	Quarterly_Rating_Increased
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	1715580.0	2.0	0
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	0
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	120360.0	1.0	0
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	1265000.0	2.0	1

Statistical summary

In [39]:

```
col=['Gender','Education','Grade','Last_Quarterly_Rating','target','Salary_Increased','target']
for i in col:
    print(final_data[i].value_counts())
    print('*'*100)
```

```
Gender
0.0    1400
1.0     975
Name: count, dtype: int64
****************************************************************************************************
Education
2.0    799
1.0    794
0.0    782
Name: count, dtype: int64
****************************************************************************************************
Grade
2.0    854
1.0    738
3.0    621
4.0    138
5.0     24
Name: count, dtype: int64
****************************************************************************************************
Last_Quarterly_Rating
1.0    1741
2.0     362
3.0     165
4.0     107
Name: count, dtype: int64
****************************************************************************************************
target
1    1614
0     761
Name: count, dtype: int64
****************************************************************************************************
Salary_Increased
0    2332
1      43
Name: count, dtype: int64
****************************************************************************************************
target
1    1614
0     761
Name: count, dtype: int64
****************************************************************************************************
```

In [40]:

```
final_data.describe()
```

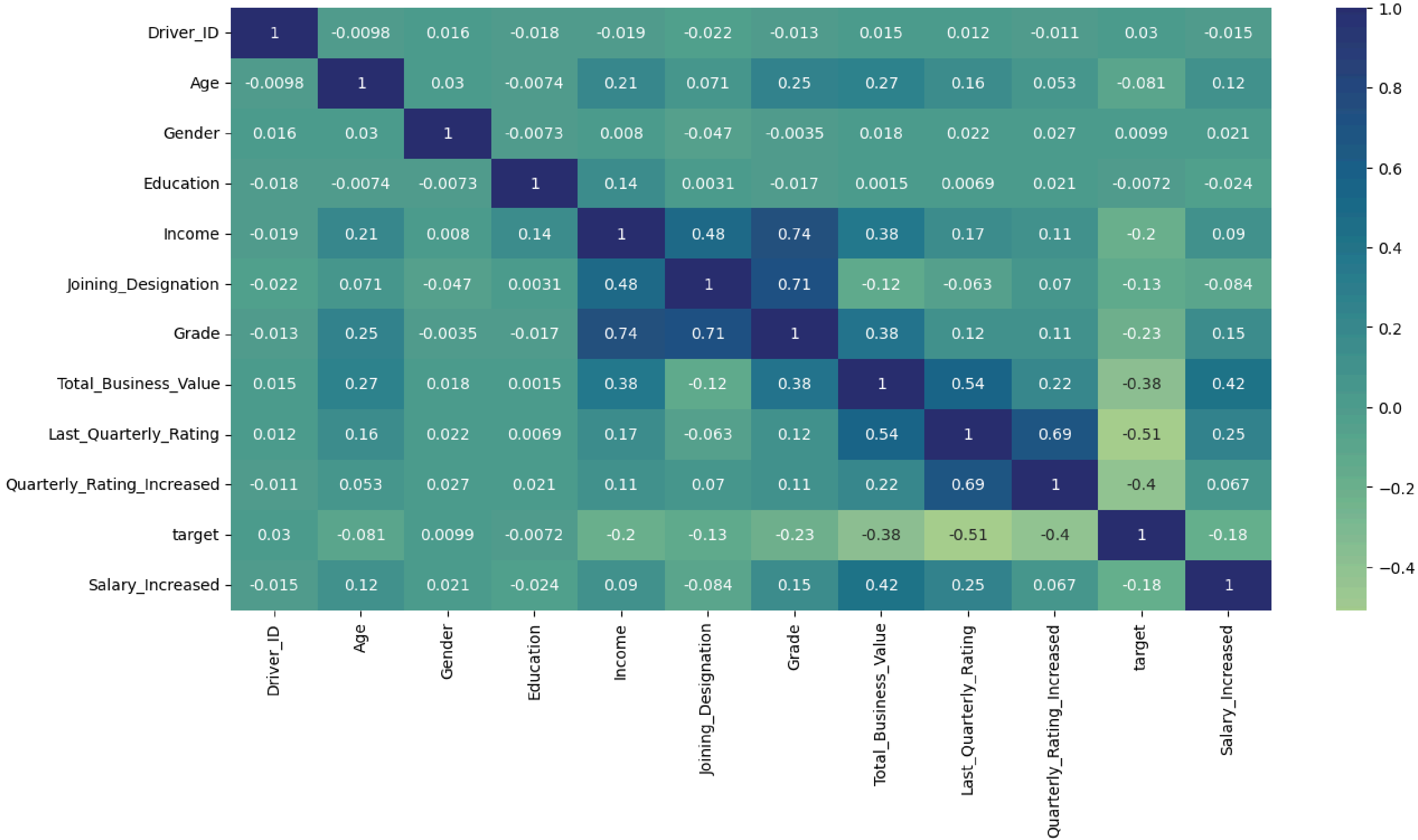
Out[40]:

	Driver_ID	Age	Gender	Education	Income	Joining_Designation	Grade	Total_Business_Value	Last_Quarterly_Rating
count	2375.000000	2375.000000	2375.000000	2375.000000	2375.000000	2375.000000	2375.000000	2.375000e+03	2375.000000
mean	1397.372211	33.778526	0.410526	1.007158	59375.624842	1.820211	2.097263	4.587891e+06	1.426526
std	805.633913	5.936808	0.492033	0.816035	28380.861583	0.841287	0.941537	9.133864e+06	0.808789
min	1.000000	21.000000	0.000000	0.000000	10747.000000	1.000000	1.000000	-1.385530e+06	1.000000
25%	695.500000	30.000000	0.000000	0.000000	39120.000000	1.000000	1.000000	0.000000e+00	1.000000
50%	1399.000000	33.000000	0.000000	1.000000	55344.000000	2.000000	2.000000	8.176800e+05	1.000000
75%	2100.500000	37.000000	1.000000	2.000000	76007.500000	2.000000	3.000000	4.171355e+06	2.000000
max	2788.000000	58.000000	1.000000	2.000000	188418.000000	5.000000	5.000000	9.533106e+07	4.000000

Correlation Analysis

In [41]:

```
plt.figure(figsize=(15, 7))
sns.heatmap(final_data.corr(numeric_only=True),annot=True, cmap="crest")
plt.show()
```



- Income and Grade (0.74) → Higher grades tend to be associated with higher income
- Joining_Designation and Grade (0.71) → Higher grades are linked to the joining designation.
- Education has very low correlation with all variables, suggesting it does not play a significant role in salary, ratings, or business value.

One hot encoding of the categorical variable

In [42]:
```
final_encoded = pd.get_dummies(final_data,'City', drop_first=True)*1
final_encoded.head()
```

Out[42]:

	Driver_ID	Age	Gender	Education	Income	Joining_Designation	Grade	Total_Business_Value	Last_Quarterly_Rating	Quarterly_Rating_Increased	...
0	1	28.0	0.0	2.0	57387.0	1.0	1.0	1715580.0	2.0	0	...
1	2	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	0	...
2	4	43.0	0.0	2.0	65603.0	2.0	2.0	350000.0	1.0	0	...
3	5	29.0	0.0	0.0	46368.0	1.0	1.0	120360.0	1.0	0	...
4	6	31.0	1.0	1.0	78728.0	3.0	3.0	1265000.0	2.0	1	...

5 rows × 40 columns

In [43]:
```
final_encoded.shape
```

Out[43]:
```
(2375, 40)
```

In [44]:
```
final_encoded.drop(['Driver_ID'],axis=1,inplace=True)
final_encoded
```

Out[44]:

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

2375 rows × 39 columns

Train Test Split

In [45]:

```
# Splitting the data
X = final_encoded.drop(["target"], axis=1)
y = final_encoded["target"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7, stratify=y)
```

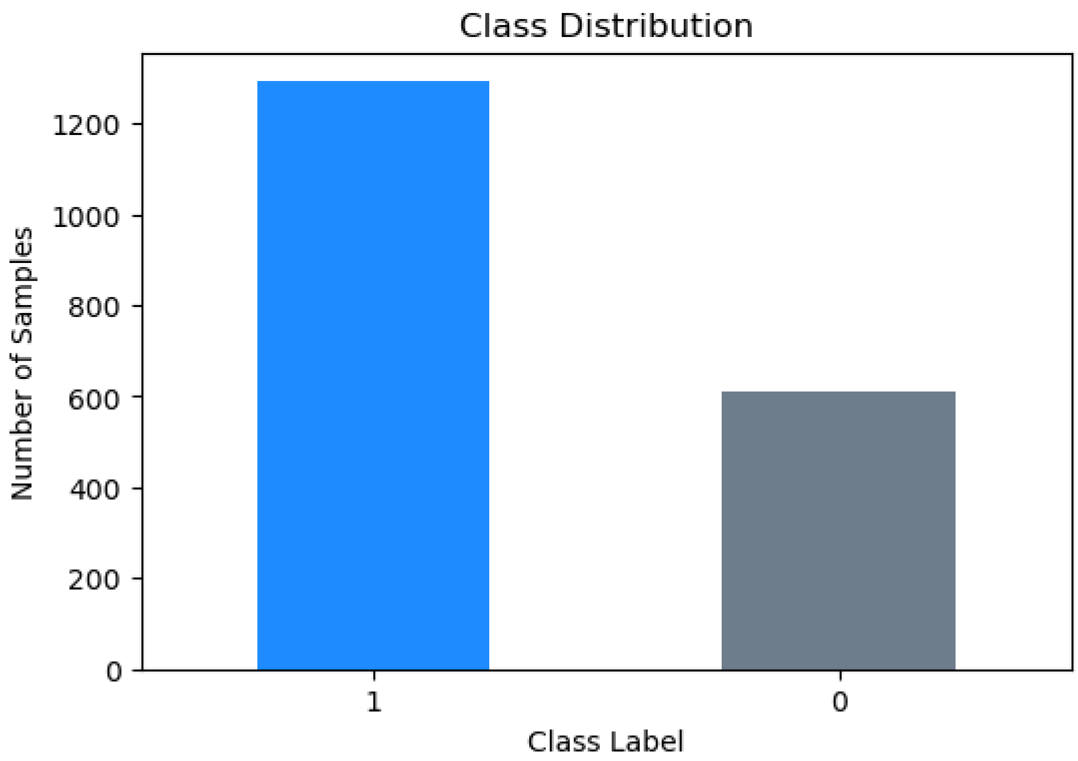
In [46]:

```
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:  (1900, 38)
X_test Shape:  (475, 38)
y_train Shape:  (1900,)
y_test Shape:  (475,)
```

Class Imbalance Treatment

In [47]:
```python
# Count class frequencies
class_counts = y_train.value_counts()
plt.figure(figsize=(6, 4))
class_counts.plot(kind='bar', color=['dodgerblue', 'slategray'])
plt.xlabel('Class Label')
plt.ylabel('Number of Samples')
plt.title('Class Distribution')
plt.xticks(rotation=0)
plt.show()
```



In [48]:
```
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)))

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

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': 1291
Before OverSampling, counts of label '0': 609

After OverSampling, the shape of train_X: (2582, 38)
After OverSampling, the shape of train_y: (2582,)

After OverSampling, counts of label '1': 1291
After OverSampling, counts of label '0': 1291
```

Standardization of training data

In [49]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Using Ensemble learning - Bagging with some hyper-parameter tuning

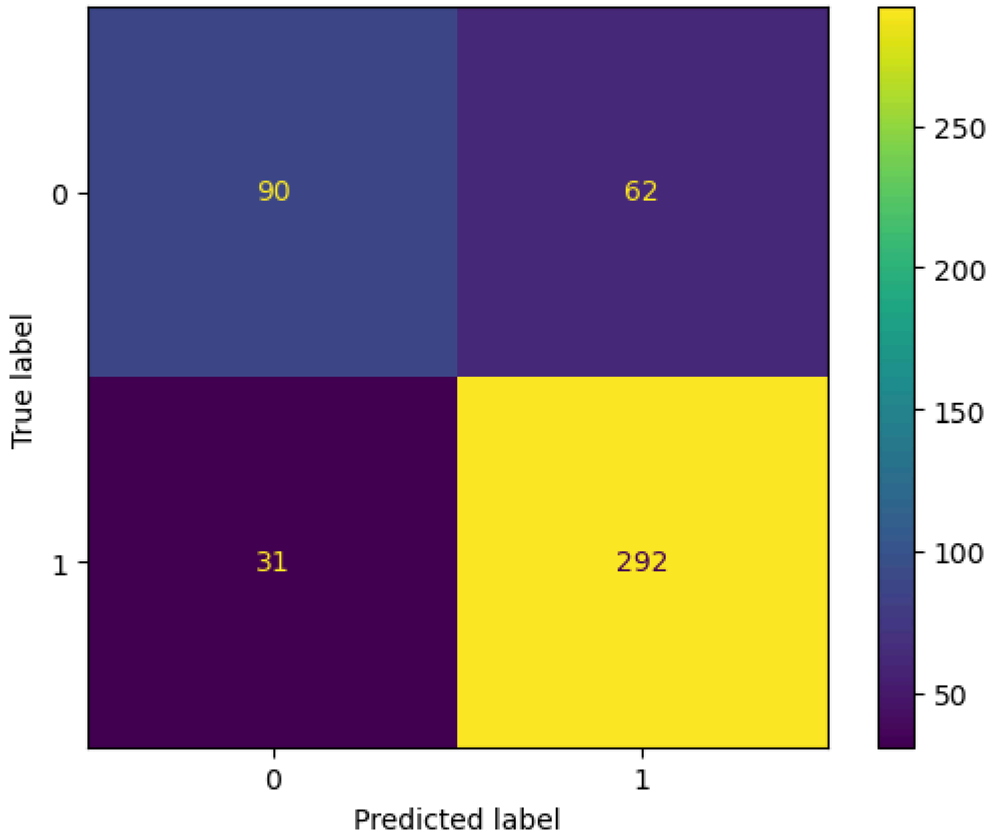
In [50]:
```
params = {
    "max_depth": [2, 3, 4],
    "n_estimators": [50, 100, 150, 200],
}

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)
```

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

print("*"*100)
print("Elapsed Time: ", elapsed_time)
print("*"*100)

y_pred = c.predict(X_test)

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

print("*"*100)
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:  0.8231703653164039
****************************************************************************************************
Elapsed Time:  18.14702320098877
****************************************************************************************************
              precision    recall  f1-score   support

           0       0.74      0.59      0.66       152
           1       0.82      0.90      0.86       323

    accuracy                           0.80       475
   macro avg       0.78      0.75      0.76       475
weighted avg       0.80      0.80      0.80       475

****************************************************************************************************
```

Out[50]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20b5d121750>
```



Using a Random Forest Classifier with balanced class:

- Precision: 74% for class 0 and 82% for class 1
- Recall: 59% for class 0 and 90% for class 1 ### As this is imbalanced dataset. We give importance to F1-Score metrics

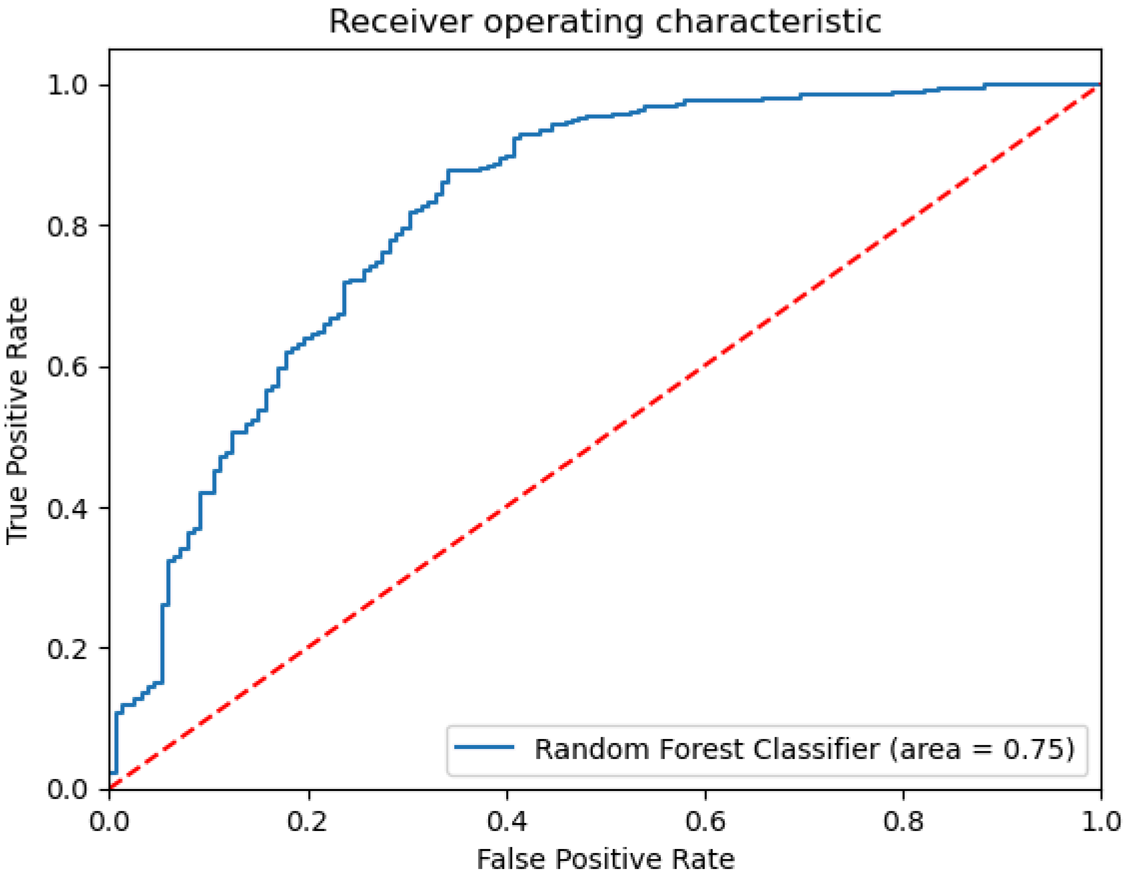
- F1 Score of 0 is 66%

- F1 Score of 1 is 86%

ROC Curve

In [51]:

```
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()
```



Using Ensemble learning - Boosting with some hyper-parameter tuning

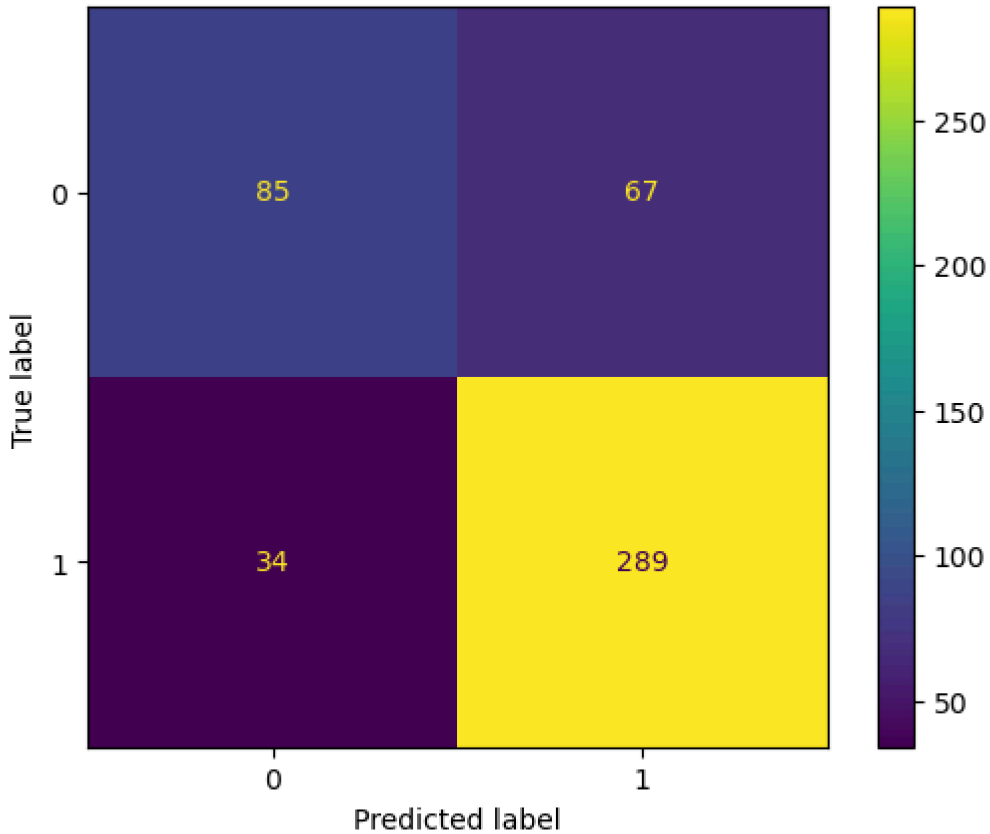
In [52]:

```
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=params)

c.fit(X_train, y_train)
print("Best Params: ", c.best_params_)
print("Best Score: ", c.best_score_)
print("*"*100)
elapsed_time = time.time() - start_time
print("\n Elapsed Time: ", elapsed_time)
print("*"*100)
y_pred = c.predict(X_test)

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

print("*"*100)
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': 'exponential', 'max_depth': 2, 'n_estimators': 100, 'subsample': 1}
Best Score:  0.8334953947546119
****************************************************************************************************

 Elapsed Time:  189.97909951210022
****************************************************************************************************
              precision    recall  f1-score   support

           0       0.71      0.56      0.63       152
           1       0.81      0.89      0.85       323

    accuracy                           0.79       475
   macro avg       0.76      0.73      0.74       475
weighted avg       0.78      0.79      0.78       475

****************************************************************************************************
```

Out[52]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20b5c7d97d0>
```



Using a Gradient Boosting Classifier

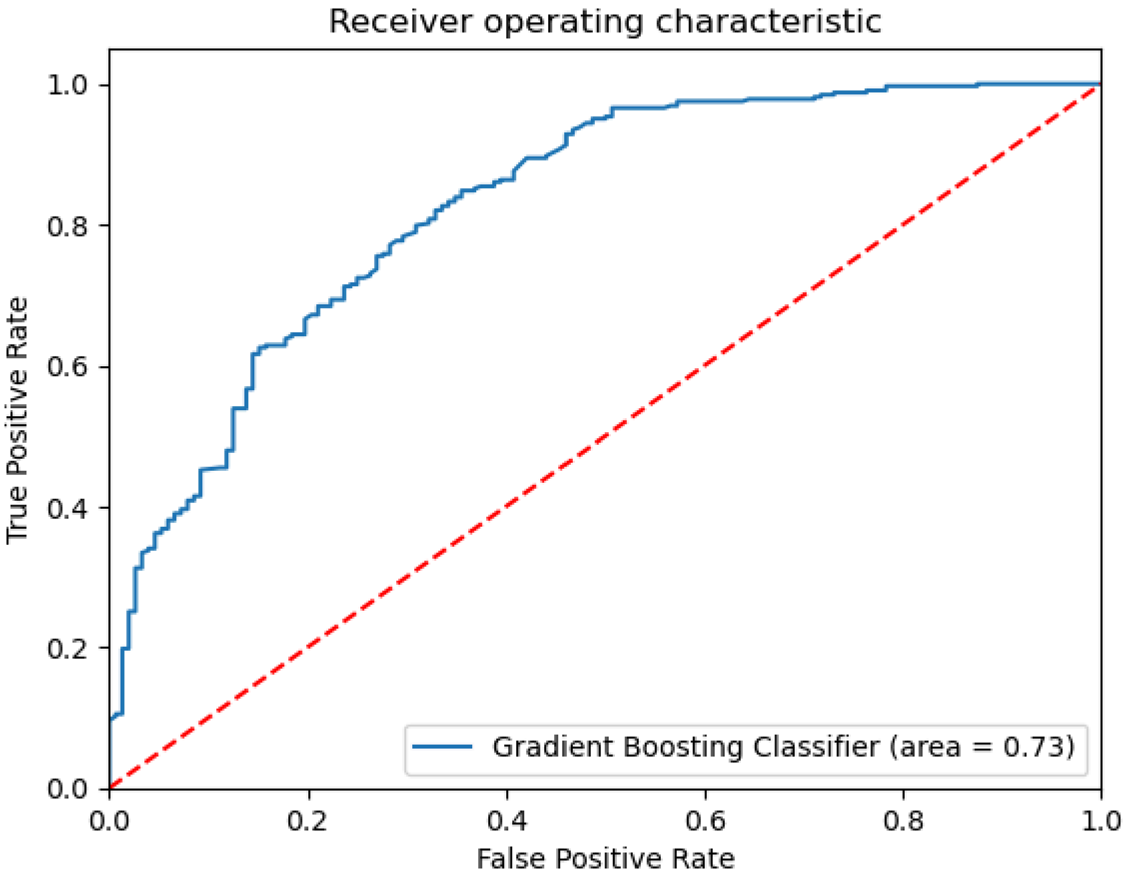
- Precision: 71% for class 0 and 81% for class 1
- Recall: 56% for class 0 and 89% for class 1 ### As this is imbalanced dataset. We give importance to F1-Score metrics

- F1 Score of 0 is 63%

- F1 Score of 1 is 85%

ROC Curve

In [53]:
```
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()
```

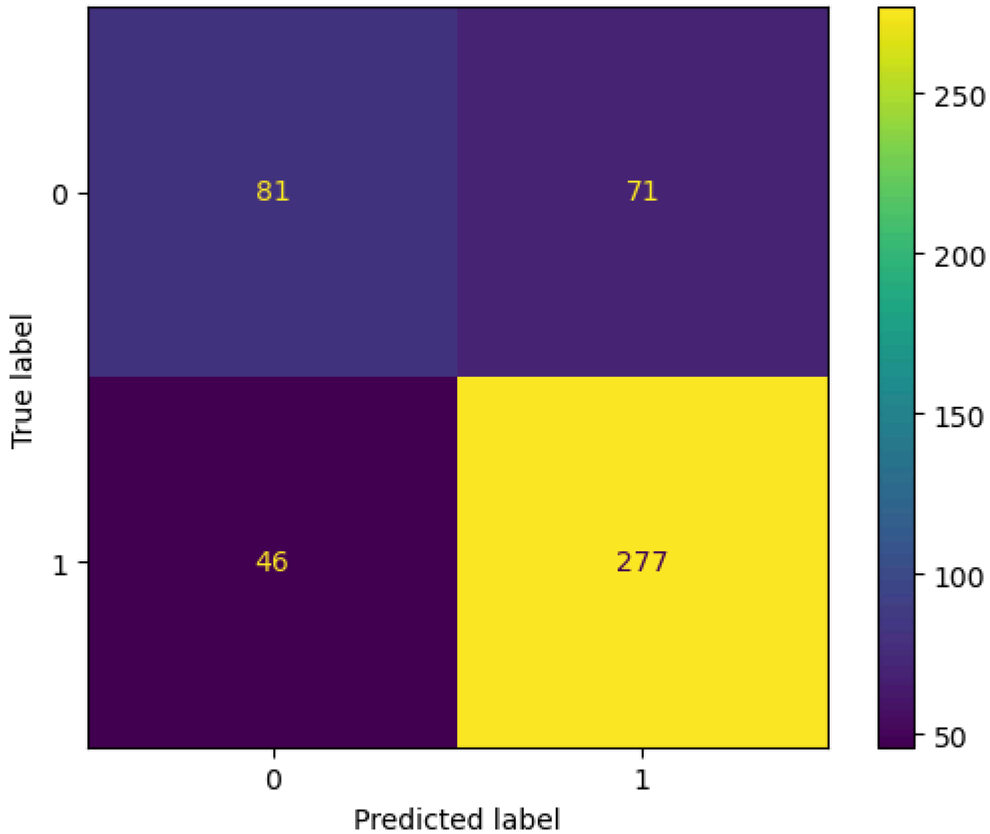


XGBoost Classifier

In [54]:
```
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()
```

```
XGBoost Classifier Score:  0.7536842105263157

               precision    recall  f1-score   support

           0       0.64      0.53      0.58       152
           1       0.80      0.86      0.83       323

    accuracy                           0.75       475
   macro avg       0.72      0.70      0.70       475
weighted avg       0.75      0.75      0.75       475
```

Out[54]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20b5d802f10>
```



Using a XGBoost Classifier

- Precision: 64% for class 0 and 80% for class 1
- Recall: 53% for class 0 and 86% for class 1

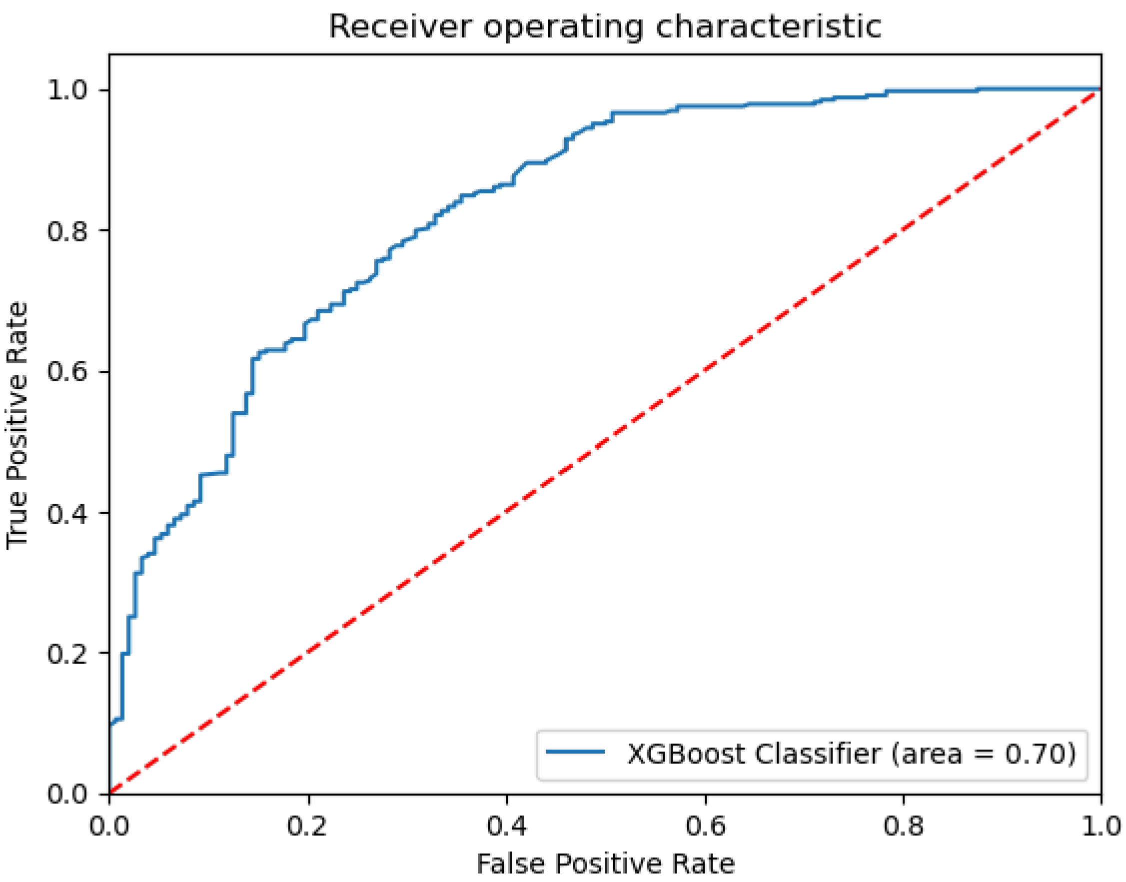
As this is imbalanced dataset. We give importance to F1-Score metrics

- F1 Score of 0 is 58%
- F1 Score of 1 is 83%

ROC Curve

In [55]:

```
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()
```



Insights:

- Gender Distribution: Males are more frequent compared to females.
- Grade Levels: The number of drivers decreases as we move to higher grades. However, from C20 onwards, the number of drivers increases.
- Education Levels: The distribution of education levels is almost the same across employees.
- Ratings and Attrition: As ratings increase, the number of employees decreases. Employees with an increased quarterly rating are less likely to leave.
- Income and Age: Income initially rises with age but then levels off or slightly decreases. There is no strong linear relationship between age and income. Higher education does not significantly impact income.
- Attrition: Out of 2,381 drivers, 1,616 have left the company.
- Salary and Retention: Employees whose monthly salary has not increased are more likely to leave the company.
- Quarterly Ratings:
  - 1,744 employees had their last quarterly rating as 1.
  - 2,076 employees did not see an increase in their quarterly rating.

Recommendations

Address Gender-Based Disparities

- Investigate the reasons for the gender imbalance in the workforce.
- Implement targeted hiring strategies to encourage more female participation.
- Ensure workplace policies support work-life balance, especially for female employees.

Improve Compensation and Performance-Based Incentives

- Employees whose salary has not increased are more likely to leave—implement periodic salary adjustments based on performance.
- Introduce structured performance-based incentives, especially for employees with improved quarterly ratings.
- Ensure salary increments align with inflation and market standards.

Address Attrition Causes

- 1,616 out of 2,381 drivers have left—conduct exit interviews to identify key reasons.
- Offer retention bonuses and competitive benefits to high-performing employees.
- Improve working conditions and ensure job satisfaction to reduce voluntary turnover.

Age and Income Optimization

- Since income levels off or slightly decreases with age, introduce long-term financial incentives.
- Provide financial planning support and retirement benefits to retain experienced employees.