In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import (
    accuracy_score, confusion_matrix, classification_report,
    roc_auc_score, roc_curve, auc,
    ConfusionMatrixDisplay, RocCurveDisplay
)
from statsmodels.stats.outliers_influence import variance_inflation_factor
from imblearn.over_sampling import SMOTE
import time
import pickle
import os
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
from lightgbm import LGBMClassifier
```

In [3]:
```
df_OLA = pd.read_csv('/Users/Ramv/Downloads/ola_driver_scaler.csv')
df_OLA
```

Out[3]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateof
0	0	01/01/19	1	28.0	0.0	C23	2	57387	2
1	1	02/01/19	1	28.0	0.0	C23	2	57387	2
2	2	03/01/19	1	28.0	0.0	C23	2	57387	2
3	3	11/01/20	2	31.0	0.0	C7	2	67016	1
4	4	12/01/20	2	31.0	0.0	C7	2	67016	1
...	...	...	...	...	...	...	...	...	
19099	19099	08/01/20	2788	30.0	0.0	C27	2	70254	06
19100	19100	09/01/20	2788	30.0	0.0	C27	2	70254	06
19101	19101	10/01/20	2788	30.0	0.0	C27	2	70254	06
19102	19102	11/01/20	2788	30.0	0.0	C27	2	70254	06
19103	19103	12/01/20	2788	30.0	0.0	C27	2	70254	06

19104 rows × 14 columns

In [4]:

```
df_OLA.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 [5]:

```
df_OLA.isnull().sum()
```

Out[5]:
```
Unnamed: 0                  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 [6]:
```
df_OLA.drop("Unnamed: 0", axis = 1, inplace = True)
```

In [7]:
```
df_OLA
```

Out[7]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	Las
0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	
1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	
2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	
3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	
4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	
...	...	...	...	...	...	...	...	...	
19099	08/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	
19100	09/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	
19101	10/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	
19102	11/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	
19103	12/01/20	2788	30.0	0.0	C27	2	70254	06/08/20	

19104 rows × 13 columns

In [8]:
```
df_OLA.nunique()
```

Out[8]:
```
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
dtype: int64
```

In [9]:
```
df_OLA["MMM-YY"] = pd.to_datetime(df_OLA["MMM-YY"], errors='coerce')
df_OLA["Dateofjoining"] = pd.to_datetime(df_OLA["Dateofjoining"], errors='cc
df_OLA["LastWorkingDate"] = pd.to_datetime(df_OLA["LastWorkingDate"], errors
```

In [10]:
```
df_OLA.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
```

Missing values in Percentage

In [11]:
```
df_OLA.isnull().sum() / len(df_OLA) * 100
```

Out[11]:
```
MMM-YY                  0.000000
Driver_ID               0.000000
Age                     0.319305
Gender                  0.272194
City                    0.000000
Education_Level         0.000000
Income                  0.000000
Dateofjoining           0.000000
LastWorkingDate        91.541039
Joining Designation     0.000000
Grade                   0.000000
Total Business Value    0.000000
Quarterly Rating        0.000000
dtype: float64
```

There are missing values found in AGE, Gender.

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

In [12]:
```
num_var = df_OLA.select_dtypes(np.number)
num_var.columns
```

Out[12]:
```
Index(['Driver_ID', 'Age', 'Gender', 'Education_Level', 'Income',
       'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating'],
      dtype='object')
```

In [13]:
```
num_var.drop(["Driver_ID"], axis = 1, inplace = True)
```

KNN Imputation

In [14]:
```
# Reimport necessary modules
from sklearn.impute import KNNImputer


# Apply KNN imputation
imputer = KNNImputer(n_neighbors=5, weights='uniform', metric='nan_euclidean
data_imputed = imputer.fit_transform(num_var)

# Convert the imputed array back to a DataFrame
data_imputed_df = pd.DataFrame(data_imputed, columns=num_var.columns)

# Display the first few rows of the imputed dataset
data_imputed_df.head()
```

Out[14]:

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating
0	28.0	0.0	2.0	57387.0	1.0	1.0	2381060.0	2.0
1	28.0	0.0	2.0	57387.0	1.0	1.0	-665480.0	2.0
2	28.0	0.0	2.0	57387.0	1.0	1.0	0.0	2.0
3	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0
4	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0

In [15]:
```
data_imputed_df.isnull().sum()
```

Out[15]:
```
Age                     0
Gender                  0
Education_Level         0
Income                  0
Joining Designation     0
Grade                   0
Total Business Value    0
Quarterly Rating        0
dtype: int64
```

In [16]:
```
data_imputed_df.nunique()
```

Out[16]:
```
Age                        70
Gender                      6
Education_Level             3
Income                   2383
Joining Designation         5
Grade                       5
Total Business Value    10181
Quarterly Rating            4
dtype: int64
```

Feature Engineering

Concatenating dataframes

In [17]:
```
res_columns = list(set(df_OLA.columns).difference(set(num_var)))
res_columns
```

Out[17]:
```
['LastWorkingDate', 'Dateofjoining', 'MMM-YY', 'Driver_ID', 'City']
```

In [18]:
```
df_OLA_new = pd.concat([data_imputed_df, df_OLA[res_columns]], axis=1)
df_OLA_new.shape
```

Out[18]: (19104, 13)

In [19]:
```
df_OLA_new.head()
```

Out[19]:

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	LastW
0	28.0	0.0	2.0	57387.0	1.0	1.0	2381060.0	2.0	
1	28.0	0.0	2.0	57387.0	1.0	1.0	-665480.0	2.0	
2	28.0	0.0	2.0	57387.0	1.0	1.0	0.0	2.0	
3	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	
4	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	

Data Preprocessing

In [20]:
```
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 = df_OLA_new.groupby(["Driver_ID", "MMM-YY"]).aggregate(agg_fun
processed_df.head()
```

Out[20]:

		Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	C
Driver_ID	MMM-YY								
1	2019-01-01	28.0	0.0	2.0	57387.0	1.0	1.0	2381060.0	
	2019-02-01	28.0	0.0	2.0	57387.0	1.0	1.0	-665480.0	
	2019-03-01	28.0	0.0	2.0	57387.0	1.0	1.0	0.0	
2	2020-11-01	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	
	2020-12-01	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	

In [21]:
```
df_OLA_final = pd.DataFrame()
df_OLA_final["Driver_ID"] = df_OLA_new["Driver_ID"].unique()
```

In [22]:
```
df_OLA_final['Age'] = list(processed_df.groupby('Driver_ID',axis=0).max('MMM
df_OLA_final['Gender'] = list(processed_df.groupby('Driver_ID').agg({'Gender
df_OLA_final['City'] = list(processed_df.groupby('Driver_ID').agg({'City':'l
df_OLA_final['Education'] = list(processed_df.groupby('Driver_ID').agg({'Edu
df_OLA_final['Income'] = list(processed_df.groupby('Driver_ID').agg({'Income
df_OLA_final['Joining_Designation'] = list(processed_df.groupby('Driver_ID')
df_OLA_final['Grade'] = list(processed_df.groupby('Driver_ID').agg({'Grade':
df_OLA_final['Total_Business_Value'] = list(processed_df.groupby('Driver_ID'
df_OLA_final['Last_Quarterly_Rating'] = list(processed_df.groupby('Driver_ID
```

In [23]:
```
df_OLA_final.head(10)
```

Out[23]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_Bu
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	
1	2	31.0	0.0	C7	2.0	67016.0	2.0	2.0	
2	4	43.0	0.0	C13	2.0	65603.0	2.0	2.0	
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	
5	8	34.0	0.0	C2	0.0	70656.0	3.0	3.0	
6	11	28.0	1.0	C19	2.0	42172.0	1.0	1.0	
7	12	35.0	0.0	C23	2.0	28116.0	1.0	1.0	
8	13	31.0	0.0	C19	2.0	119227.0	1.0	4.0	
9	14	39.0	1.0	C26	0.0	19734.0	3.0	3.0	

Creating a column for the drivers whose quarterly rating has increased - assigning the value 1

In [24]:

```
first_quarter = processed_df.groupby(["Driver_ID"]).agg({"Quarterly Rating":
last_quarter = processed_df.groupby(["Driver_ID"]).agg({"Quarterly Rating":
quart = (last_quarter["Quarterly Rating"] > first_quarter["Quarterly Rating"]

empid = quart[quart["Quarterly Rating"] == True]["Driver_ID"]

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


df_OLA_final["Quarterly_Rating_Increased"] = qrl
df_OLA_final
```

Out [24]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	
1	2	31.0	0.0	C7	2.0	67016.0	2.0	2.0	
2	4	43.0	0.0	C13	2.0	65603.0	2.0	2.0	
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	
...	...	...	...	...	...	...	...	...	
2376	2784	34.0	0.0	C24	0.0	82815.0	2.0	3.0	
2377	2785	34.0	1.0	C9	0.0	12105.0	1.0	1.0	
2378	2786	45.0	0.0	C19	0.0	35370.0	2.0	2.0	
2379	2787	28.0	1.0	C20	2.0	69498.0	1.0	1.0	
2380	2788	30.0	0.0	C27	2.0	70254.0	2.0	2.0	

2381 rows × 11 columns

Creating Target Variable Column - assigning value 0 for the driver still working and 1 for the driver who has left

In [25]:

```
last_wd = (processed_df.groupby(["Driver_ID"]).agg({"LastWorkingDate": "last

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

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

df_OLA_final["Target"] = target
df_OLA_final
```

Out[25]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	
1	2	31.0	0.0	C7	2.0	67016.0	2.0	2.0	
2	4	43.0	0.0	C13	2.0	65603.0	2.0	2.0	
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	
...	...	...	...	...	...	...	...	...	
2376	2784	34.0	0.0	C24	0.0	82815.0	2.0	3.0	
2377	2785	34.0	1.0	C9	0.0	12105.0	1.0	1.0	
2378	2786	45.0	0.0	C19	0.0	35370.0	2.0	2.0	
2379	2787	28.0	1.0	C20	2.0	69498.0	1.0	1.0	
2380	2788	30.0	0.0	C27	2.0	70254.0	2.0	2.0	

2381 rows × 12 columns

Creating column that denotes whether there was an increase in drivers monthly income - we assign 1 for an increase else assign 0

In [26]:

```
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 df_OLA_final["Driver_ID"]:
    if i in empid.values:
        income.append(1)
    else:
        income.append(0)

df_OLA_final["Salary_Increased"] = income
df_OLA_final
```

Out[26]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation	Grade	Total_
0	1	28.0	0.0	C23	2.0	57387.0	1.0	1.0	
1	2	31.0	0.0	C7	2.0	67016.0	2.0	2.0	
2	4	43.0	0.0	C13	2.0	65603.0	2.0	2.0	
3	5	29.0	0.0	C9	0.0	46368.0	1.0	1.0	
4	6	31.0	1.0	C11	1.0	78728.0	3.0	3.0	
...	...	...	...	...	...	...	...	...	
2376	2784	34.0	0.0	C24	0.0	82815.0	2.0	3.0	
2377	2785	34.0	1.0	C9	0.0	12105.0	1.0	1.0	
2378	2786	45.0	0.0	C19	0.0	35370.0	2.0	2.0	
2379	2787	28.0	1.0	C20	2.0	69498.0	1.0	1.0	
2380	2788	30.0	0.0	C27	2.0	70254.0	2.0	2.0	

2381 rows × 13 columns

In [27]:

```
df_OLA_final["Salary_Increased"].value_counts(normalize=True)
```

Out[27]:

```
0    0.98194
1    0.01806
Name: Salary_Increased, dtype: float64
```

It shows the income increased only for 1.8% of drivers.

Statistical Summary

In [28]:

```
df_OLA_final.describe().T
```

Out[28]:

	count	mean	std	min	25%	5
Driver_ID	2381.0	1.397559e+03	8.061616e+02	1.0	695.0	14
Age	2381.0	3.377018e+01	5.933265e+00	21.0	30.0	
Gender	2381.0	4.105838e-01	4.914963e-01	0.0	0.0	
Education	2381.0	1.007560e+00	8.162900e-01	0.0	0.0	
Income	2381.0	5.933416e+04	2.838367e+04	10747.0	39104.0	553
Joining_Designation	2381.0	1.820244e+00	8.414334e-01	1.0	1.0	
Grade	2381.0	2.096598e+00	9.415218e-01	1.0	1.0	
Total_Business_Value	2381.0	4.586742e+06	9.127115e+06	-1385530.0	0.0	8176
Last_Quarterly_Rating	2381.0	1.427971e+00	8.098389e-01	1.0	1.0	
Quarterly_Rating_Increased	2381.0	1.503570e-01	3.574961e-01	0.0	0.0	
Target	2381.0	6.787064e-01	4.670713e-01	0.0	0.0	
Salary_Increased	2381.0	1.805964e-02	1.331951e-01	0.0	0.0	

Observation:

There are total of 2831 different drivers data.

Age of drivers range from 21yrs to 58yrs.

75% of drivers monthly income is <= 75986.

75% of drivers contributed to 4173650 as total in terms of business value.

In [29]:

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

Out[29]:

	City
count	2381
unique	29
top	C20
freq	152

Observation: C20 is the city most drivers come from.

In [30]:

```
df_OLA_final["Gender"].value_counts()
```

Out[30]:
```
0.0    1400
1.0     975
0.6       3
0.2       2
0.4       1
Name: Gender, dtype: int64
```

Observation: Majority of the drivers are male.

In [31]:
```
df_OLA_final["Education"].value_counts()
```

Out[31]:
```
2.0    802
1.0    795
0.0    784
Name: Education, dtype: int64
```

Observation:More than 1/3 of the drivers have completed graduation

In [32]:
```
df_OLA_final["Target"].value_counts()
```

Out[32]:
```
1    1616
0     765
Name: Target, dtype: int64
```

Observation: 1616 drivers have left the company

In [33]:
```
n = ['Gender','Education','Joining_Designation','Grade',
     'Last_Quarterly_Rating','Quarterly_Rating_Increased']

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

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

Observation:

58% of drivers are male while female constitutes around 41%.

33% of drivers have completed graduation and around 33% have completed 12+ education.

43% of drivers have 1 as joining_designation.

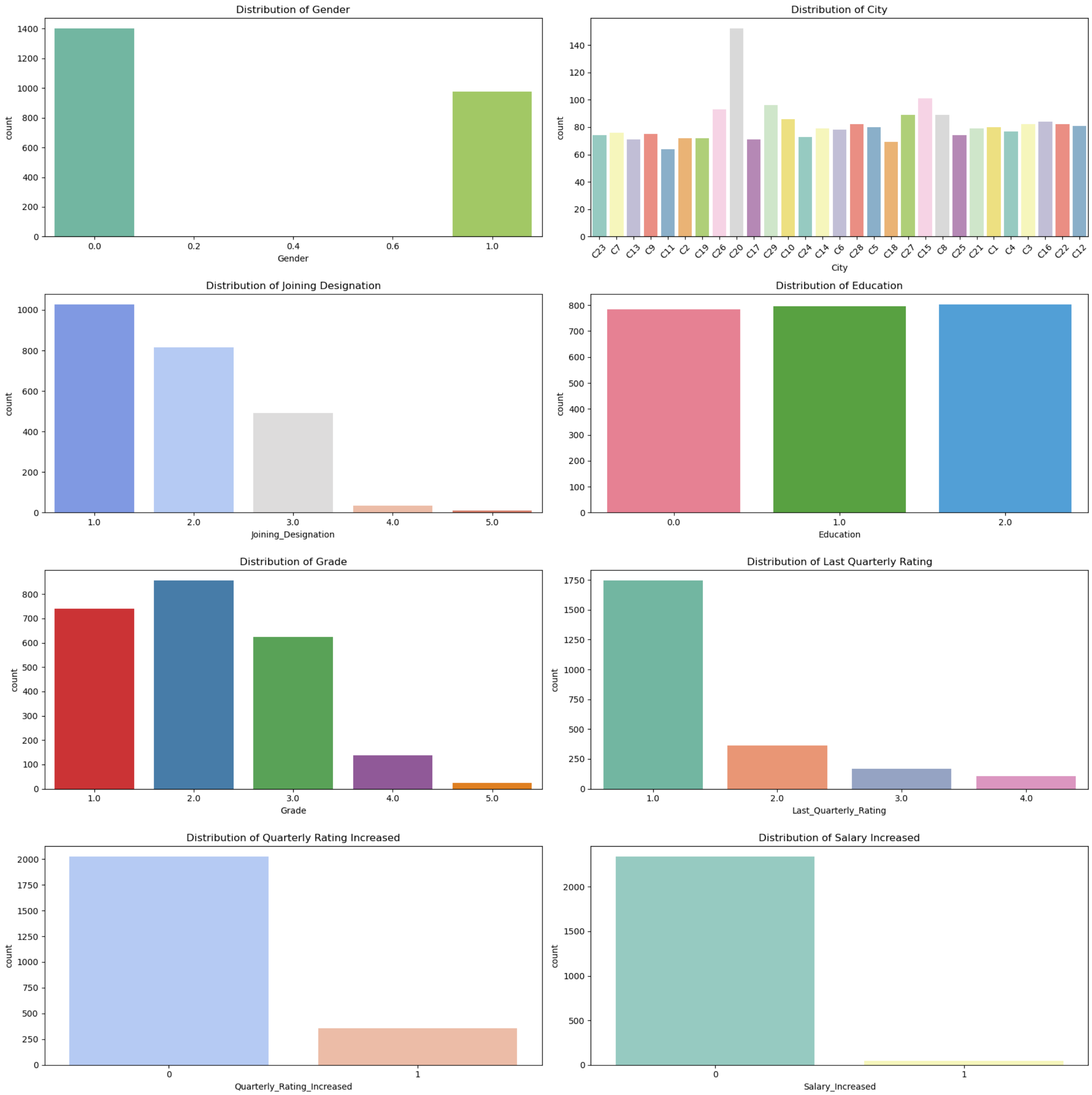
Around 36% of drivers graded as 2.

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

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

Univariate Analysis

In [34]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(18, 18))

# Subplot 1: Gender
plt.subplot(421)
sns.countplot(data=df_OLA_final, x="Gender", palette="Set2")
plt.title("Distribution of Gender")

# Subplot 2: City
plt.subplot(422)
sns.countplot(data=df_OLA_final, x="City", palette="Set3")
plt.xticks(rotation=45)
plt.title("Distribution of City")

# Subplot 3: Joining Designation
plt.subplot(423)
sns.countplot(data=df_OLA_final, x="Joining_Designation", palette="coolwarm"
plt.title("Distribution of Joining Designation")

# Subplot 4: Education
plt.subplot(424)
sns.countplot(data=df_OLA_final, x="Education", palette="husl")
plt.title("Distribution of Education")

# Subplot 5: Grade
plt.subplot(425)
sns.countplot(data=df_OLA_final, x="Grade", palette="Set1")
plt.title("Distribution of Grade")

# Subplot 6: Last Quarterly Rating
plt.subplot(426)
sns.countplot(data=df_OLA_final, x="Last_Quarterly_Rating", palette="Set2")
plt.title("Distribution of Last Quarterly Rating")

# Subplot 7: Quarterly Rating Increased
plt.subplot(427)
sns.countplot(data=df_OLA_final, x="Quarterly_Rating_Increased", palette="cc
plt.title("Distribution of Quarterly Rating Increased")

# Subplot 8: Salary Increased
plt.subplot(428)
sns.countplot(data=df_OLA_final, x="Salary_Increased", palette="Set3")
plt.title("Distribution of Salary Increased")

plt.tight_layout()
plt.show()
```



Observation:

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

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

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

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

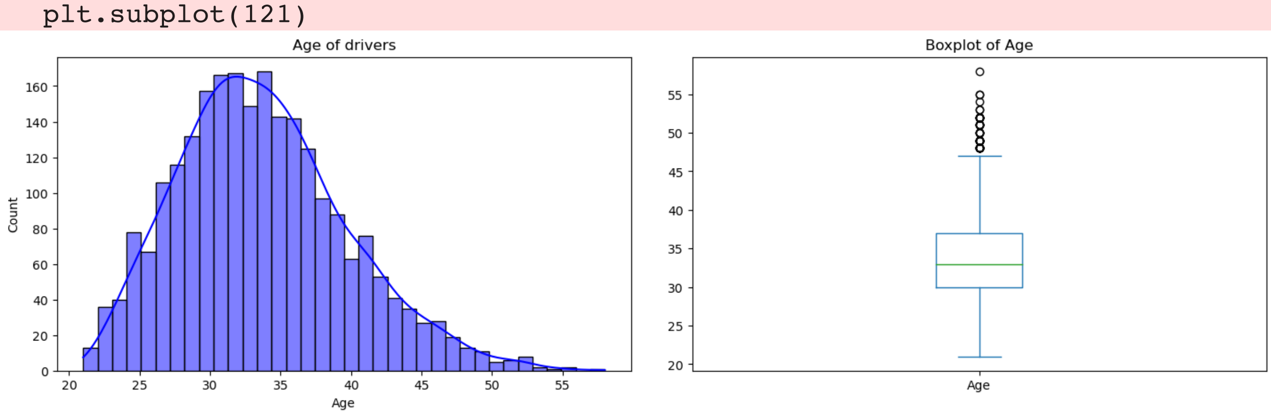
Out of 2381 employees, 855 employees had their designation as 2 at the time of reporting.

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.

In [35]:
```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.histplot(df_OLA_final['Age'],color='blue', kde=True)
plt.title("Age of drivers")
plt.subplot(122)
df_OLA_final['Age'].plot.box(title='Boxplot of Age')
plt.tight_layout(pad=3)
```

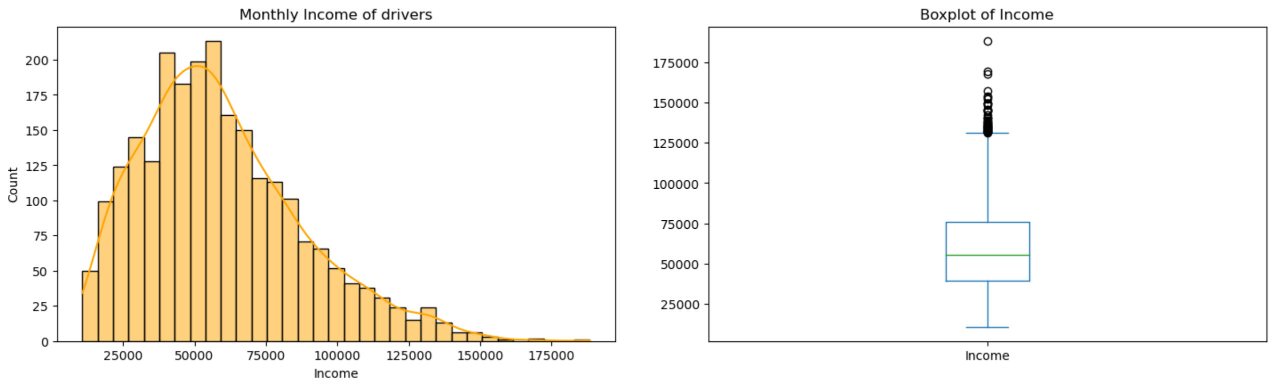
```
/var/folders/fx/1km2ndm10xxcmn5xy9fsdksm0000gn/T/ipykernel_86133/2461973495.py:2: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.
  plt.subplot(121)
```



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

In [36]:
```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.histplot(df_OLA_final['Income'],color='orange', kde=True)
plt.title("Monthly Income of drivers")
plt.subplot(122)
df_OLA_final['Income'].plot.box(title='Boxplot of Income')
plt.tight_layout(pad=3)
```

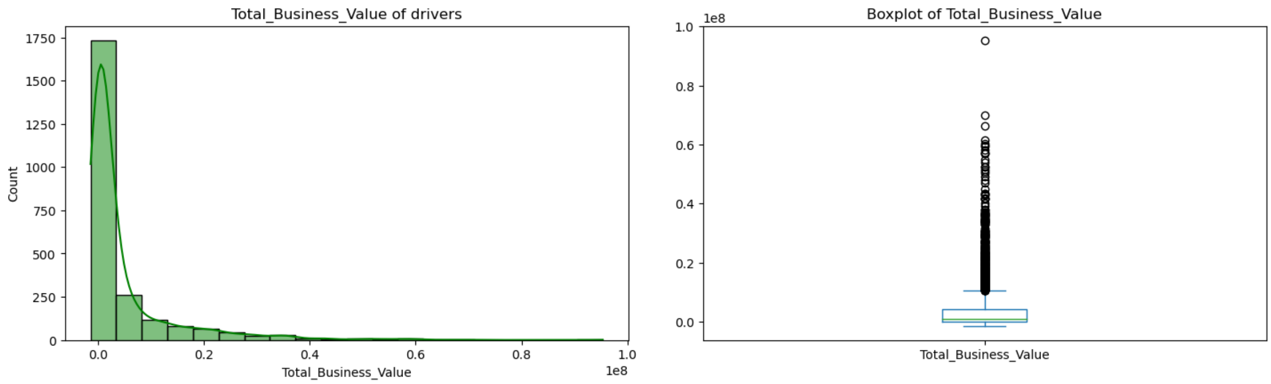
```
/var/folders/fx/1km2ndm10xxcmn5xy9fsdksm0000gn/T/ipykernel_86133/3346899482.
py:2: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is depr
ecated since 3.6 and will be removed two minor releases later; explicitly ca
ll ax.remove() as needed.
  plt.subplot(121)
```



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

In [37]:
```
plt.subplots(figsize=(15,5))
plt.subplot(121)
sns.histplot(df_OLA_final['Total_Business_Value'],color='green', kde=True, b
plt.title("Total_Business_Value of drivers")
plt.subplot(122)
df_OLA_final['Total_Business_Value'].plot.box(title='Boxplot of Total_Busine
plt.tight_layout(pad=3)
```

```
/var/folders/fx/1km2ndm10xxcmn5xy9fsdksm0000gn/T/ipykernel_86133/2853607533.
py:2: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is depr
ecated since 3.6 and will be removed two minor releases later; explicitly ca
ll ax.remove() as needed.
  plt.subplot(121)
```

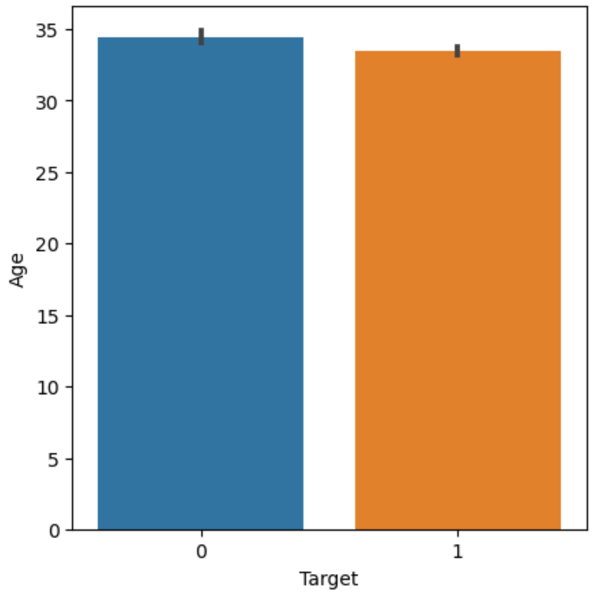


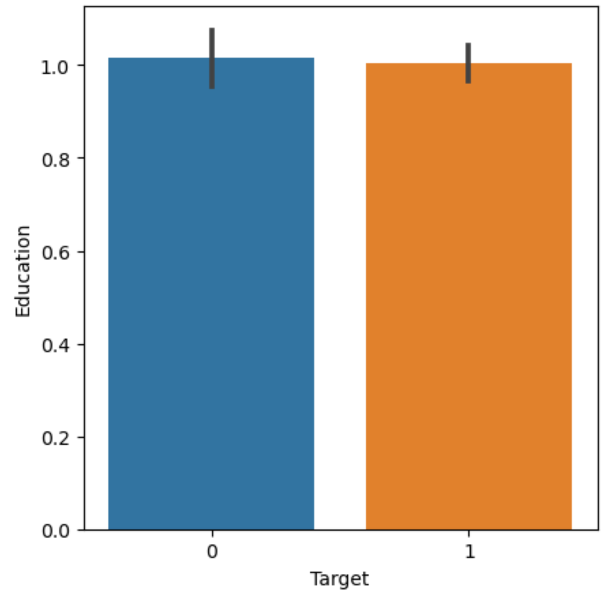
Observation: The distribution of total business value highly skewed on right which might indicate the outliers in the data.

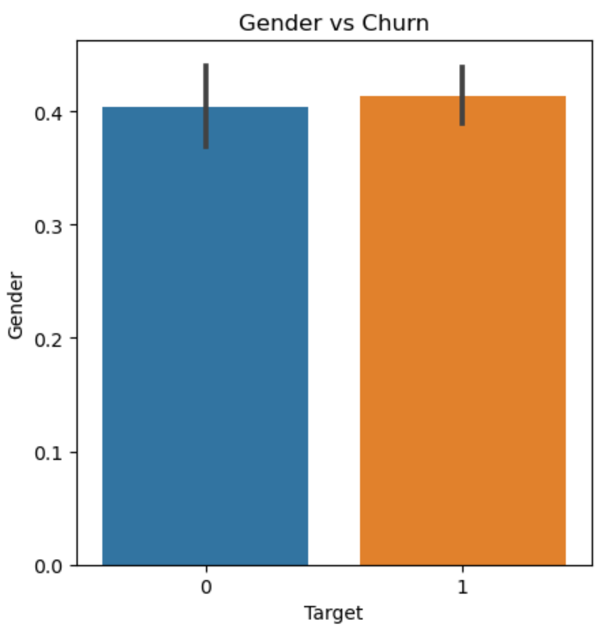
Bi-variate Analysis

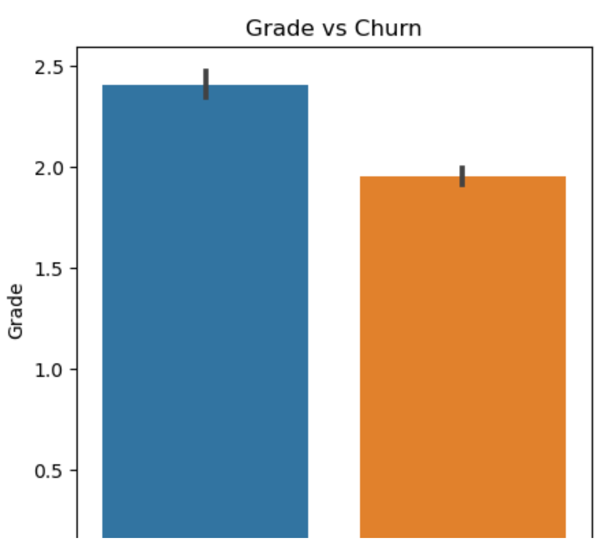
In [38]:
```
plt.figure(figsize=(10,20))

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

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

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

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

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

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

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

plt.tight_layout(pad=3)
```

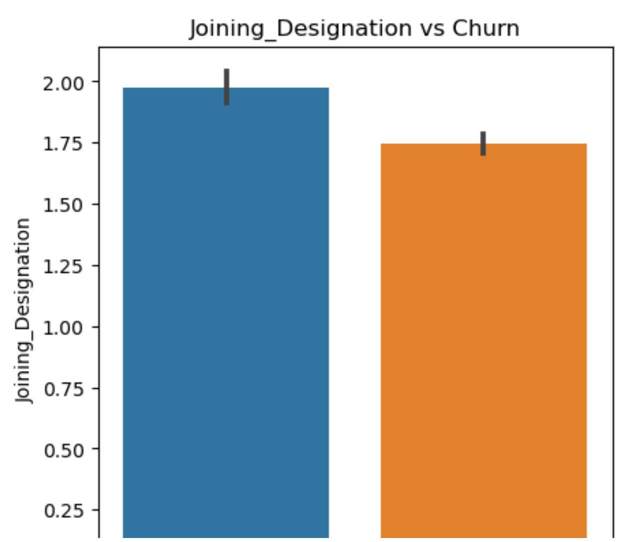
Age vs Churn Education vs Churn

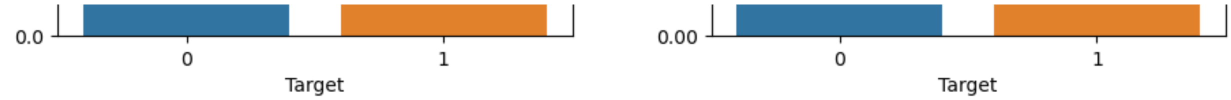

Age
Target
0
1

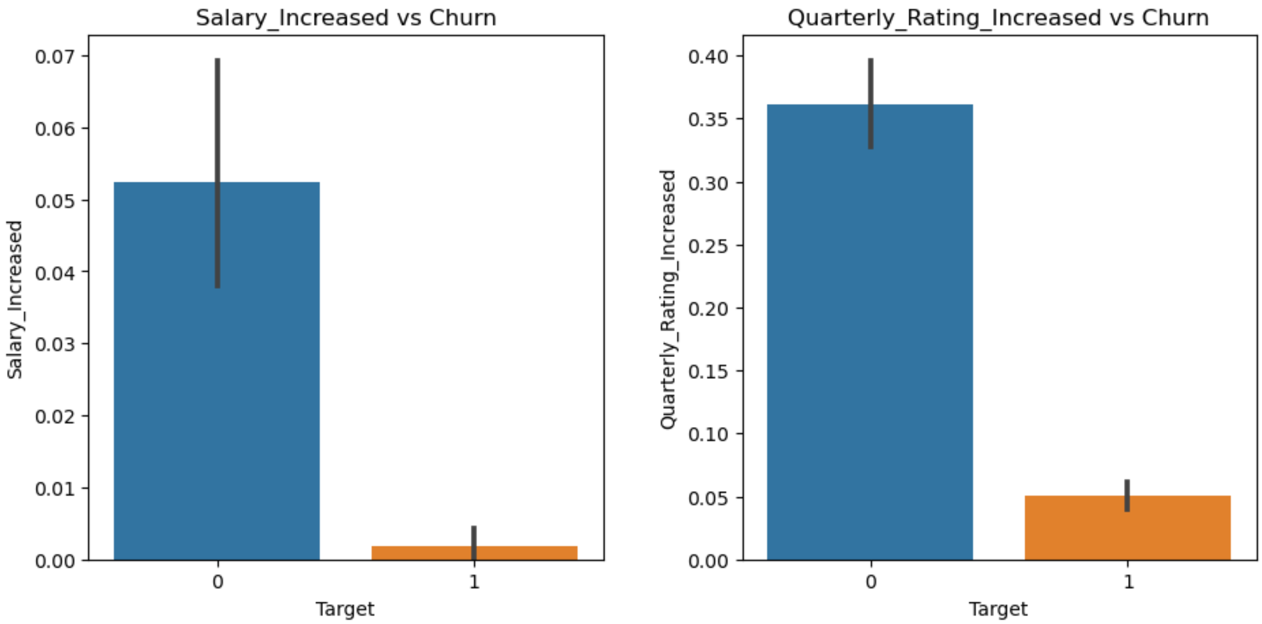

Education
Target
0
1


Gender vs Churn
Gender
Target
0
1


Grade vs Churn
Grade


Joining_Designation vs Churn
Joining_Designation





Observation:

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.

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

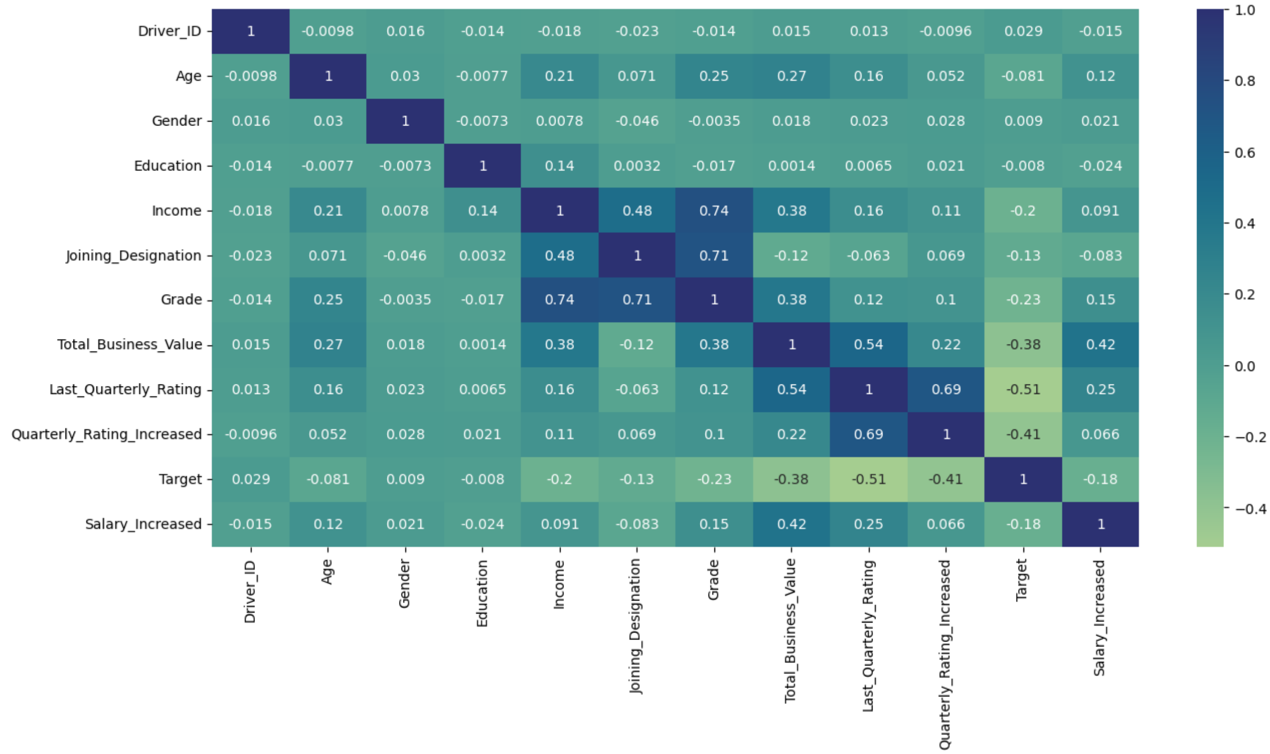
The employees whose quarterly rating has increased are less likely to leave the organization.

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

Correlation Analysis

In [39]:
```
plt.figure(figsize=(15, 7))

sns.heatmap(df_OLA_final.corr(method="pearson"), annot=True, cmap="crest")
plt.show()
```

```
/var/folders/fx/1km2ndm10xxcmn5xy9fsdksm0000gn/T/ipykernel_86133/902824261.p
y:3: FutureWarning: The default value of numeric_only in DataFrame.corr is d
eprecated. In a future version, it will default to False. Select only valid
columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(df_OLA_final.corr(method="pearson"), annot=True, cmap="crest")
```



Observation:

Income and Grade is highly correlated.

Joining Designation and Grade is highly correlated.

Total Business value and salary increament is also correlated.

One-Hot Coding

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

In [40]:
```
df_OLA_final = pd.concat([df_OLA_final, df_OLA_final['City']], axis=1)
df_OLA_final.shape
```

Out[40]: (2381, 14)

Standardiazation - Train Data

In [41]:
```
X = df_OLA_final.drop(["Driver_ID", "Target", "City"], axis = 1)
X_cols = X.columns
scaler = MinMaxScaler()

X =scaler.fit_transform(X)
```

In [42]:
```
X = pd.DataFrame(X)
X.columns = X_cols
X
```

Out[42]:

	Age	Gender	Education	Income	Joining_Designation	Grade	Total_Business_
0	0.189189	0.0	1.0	0.262508	0.00	0.00	0.03
1	0.270270	0.0	1.0	0.316703	0.25	0.25	0.01
2	0.594595	0.0	1.0	0.308750	0.25	0.25	0.01
3	0.216216	0.0	0.0	0.200489	0.00	0.00	0.01
4	0.270270	1.0	0.5	0.382623	0.50	0.50	0.02
...	...	...	...	...	...	...	
2376	0.351351	0.0	0.0	0.405626	0.25	0.50	0.23
2377	0.351351	1.0	0.0	0.007643	0.00	0.00	0.01
2378	0.648649	0.0	0.0	0.138588	0.25	0.25	0.04
2379	0.189189	1.0	1.0	0.330673	0.00	0.00	0.02
2380	0.243243	0.0	1.0	0.334928	0.25	0.25	0.03

2381 rows × 10 columns

Train & Test Split

In [43]:
```
y = df_OLA_final["Target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

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

keeping max_depth small to avoid overfitting

In [44]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassif
from sklearn.tree import DecisionTreeClassifier
import xgboost as xgb
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.metrics import classification_report, accuracy_score, confusion
from sklearn.metrics import roc_auc_score, roc_curve
```

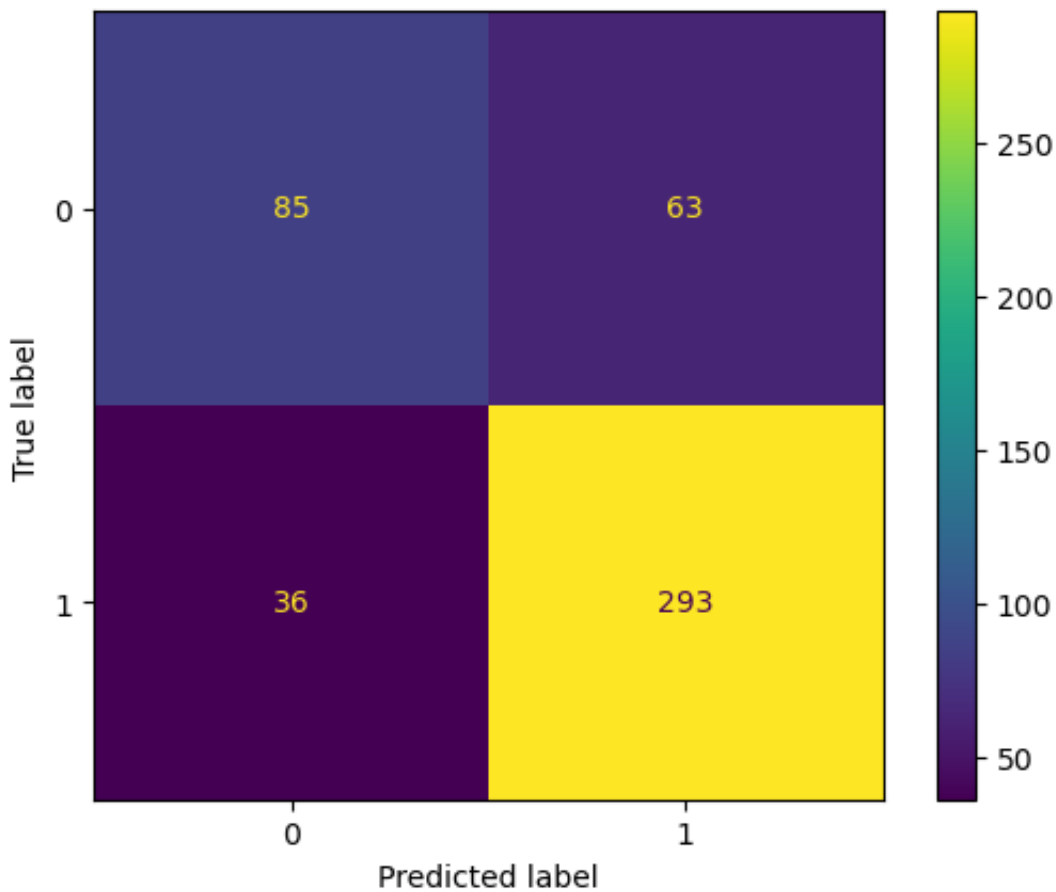
In [45]:
```
params = {
    "max_depth": [2, 3, 4],
    "n_estimators": [50, 100, 150, 200],
}

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

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:  0.8620357982229242

Elapsed Time:  2.1533491611480713
```

In [46]:
```
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       0.70      0.57      0.63       148
           1       0.82      0.89      0.86       329

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

Out[46]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x16428656 0>



Random Forest Classifier with balanced class weight

Out of all prediction, the measure for correctly predicted 0 is 70% and for 1 is 82% (Precision) Out of all actual 0, the measure for correctly predicted is 57% and for 1 is 89% (Recall) As this is imbalanced dataset.

We give importance to F1-Score metrics

F1 Score of 0 is 63%

F1 Score of 1 is 86%

Bootstrapped Random Forest using Subsample

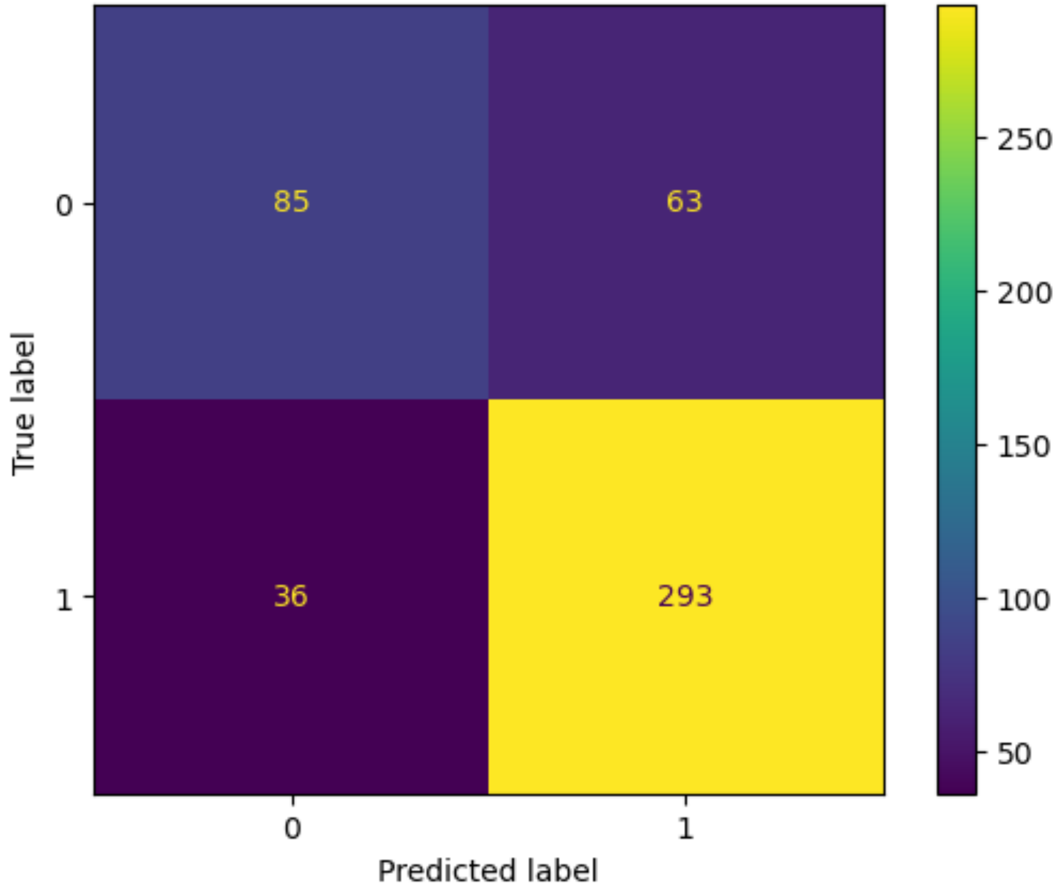
In [47]:
```
params = {
    "max_depth": [2, 3, 4],
    "n_estimators": [50, 100, 150, 200],
}

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)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best Params:  {'max_depth': 4, 'n_estimators': 200}
Best Score:  0.8625438077473855

Elapsed Time:  1.0272979736328125
```

In [48]:
```
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       0.70      0.57      0.63       148
           1       0.82      0.89      0.86       329

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

Out[48]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x16428725
0>
```



Random Forest Classifier with balanced class weight

Out of all prediction, the measure for correctly predicted 0 is 70% and for 1 is 82% (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 63%

F! Score of 1 is 86%

Observation: There is hardly any difference in the results.

Balancing Dataset using SMOTE

note: The Target variable is imbalanced towards 1.

Let's see SMOTE method rectifies this issue.

In [49]:
```
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)
```

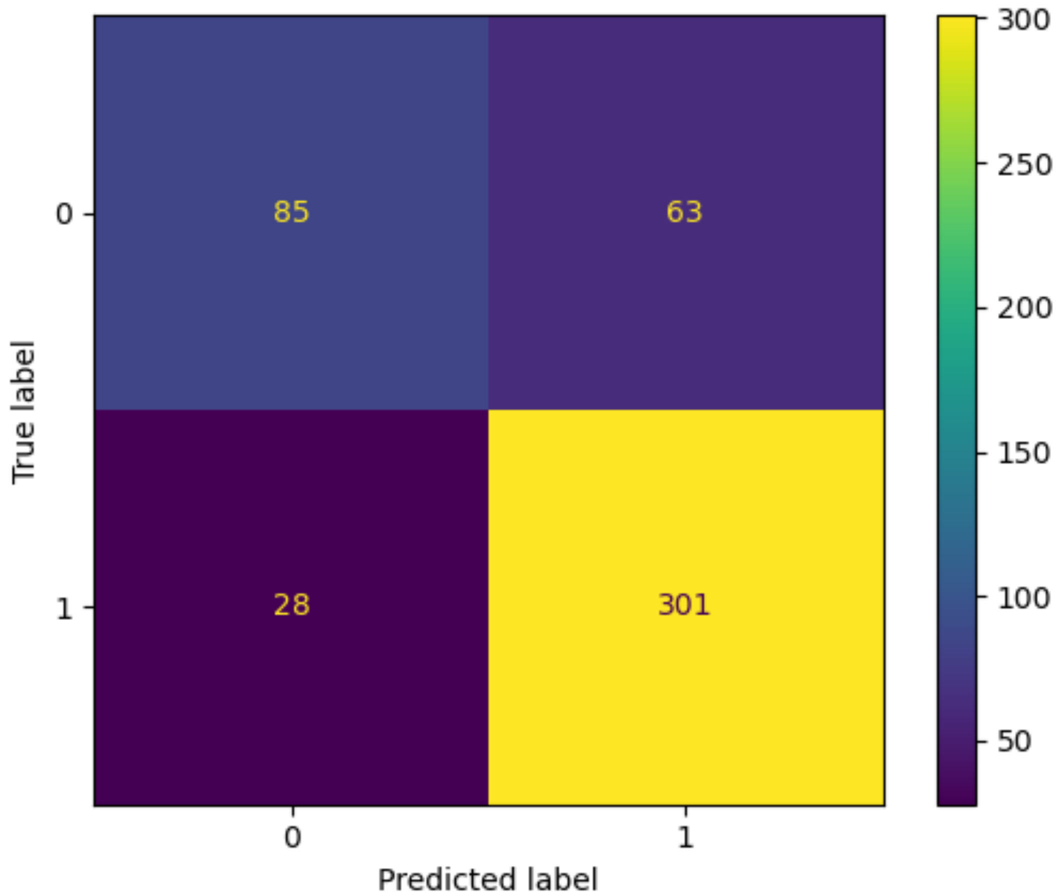
```
Before OverSampling, counts of label '1': 1287
Before OverSampling, counts of label '0': 617

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
```

Ensemble Learning - Bagging method

In [51]:
```
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

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': 50}
Best Score: 0.7828403987674123

Elapsed Time: 0.9582688808441162
              precision    recall  f1-score   support

           0       0.75      0.57      0.65       148
           1       0.83      0.91      0.87       329

    accuracy                           0.81       477
   macro avg       0.79      0.74      0.76       477
weighted avg       0.80      0.81      0.80       477
```

Out[51]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x10272067 0>



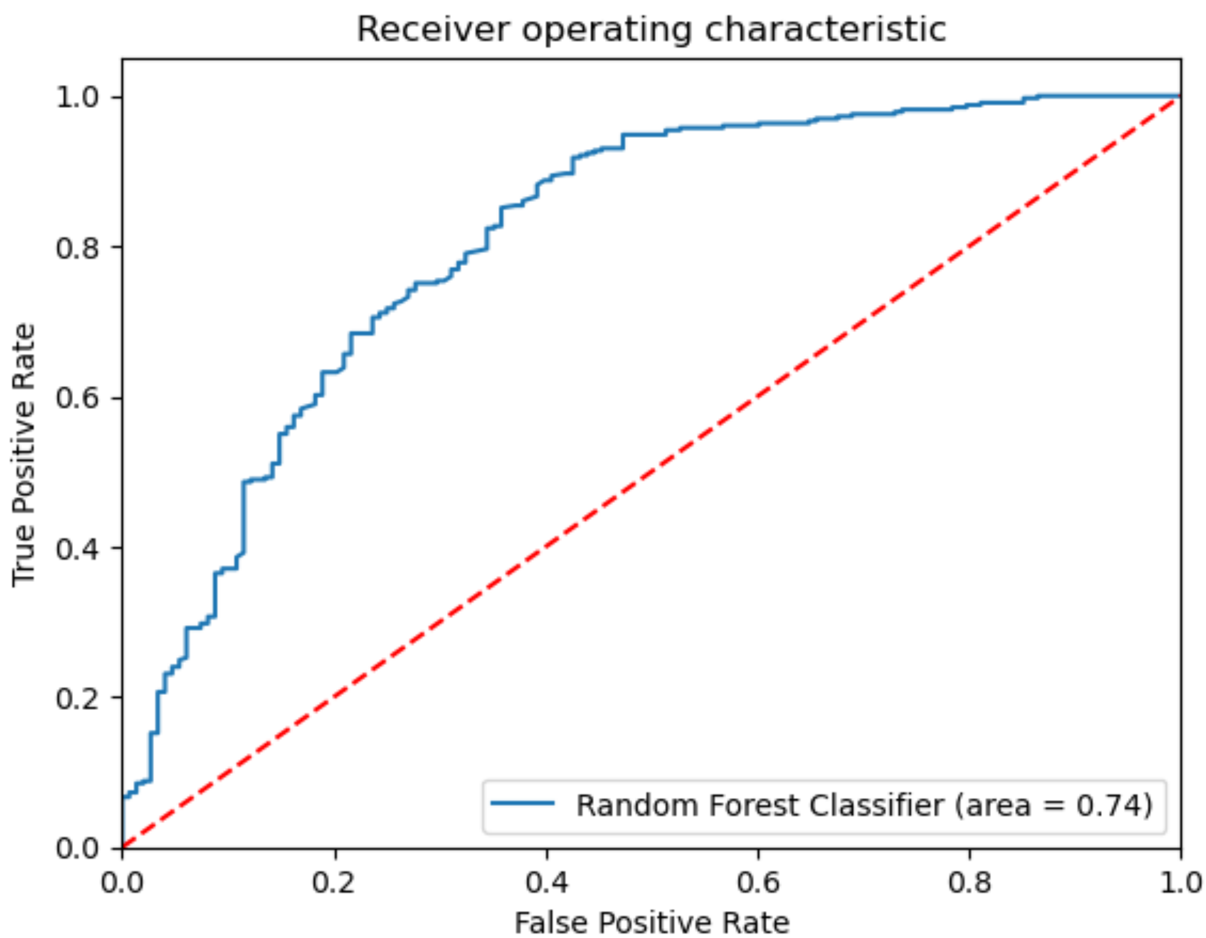
Random Forest Classifier with balanced class weight

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% F1 Score of 1 is 87%

ROC-AUC Curve

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



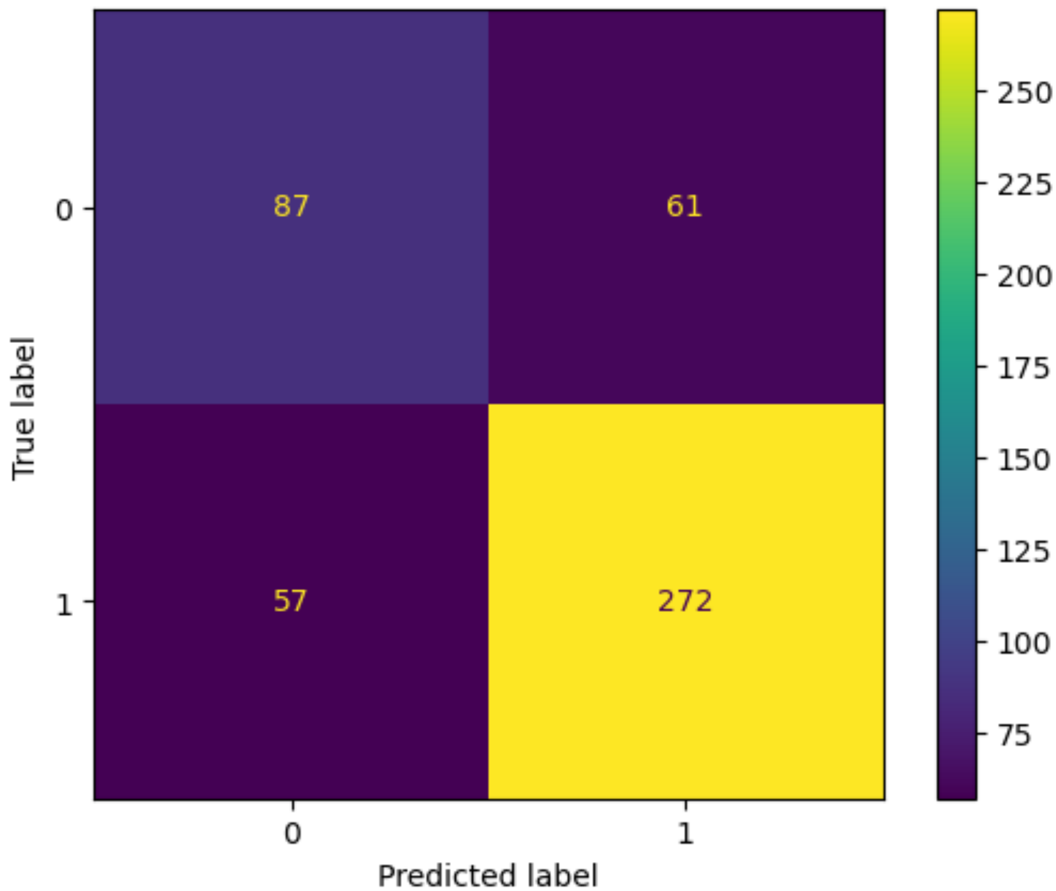
Ensemble Learning: Boosting

Gradient Boosting Classifier

In [53]:
```
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': 1}
Best Score:  0.8135198135198135

 Elapsed Time:  23.1019070148468
              precision    recall  f1-score   support

           0       0.60      0.59      0.60       148
           1       0.82      0.83      0.82       329

    accuracy                           0.75       477
   macro avg       0.71      0.71      0.71       477
weighted avg       0.75      0.75      0.75       477
```

Out[53]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x169128e2
0>
```



Gradient Boosting Classifier Metrics

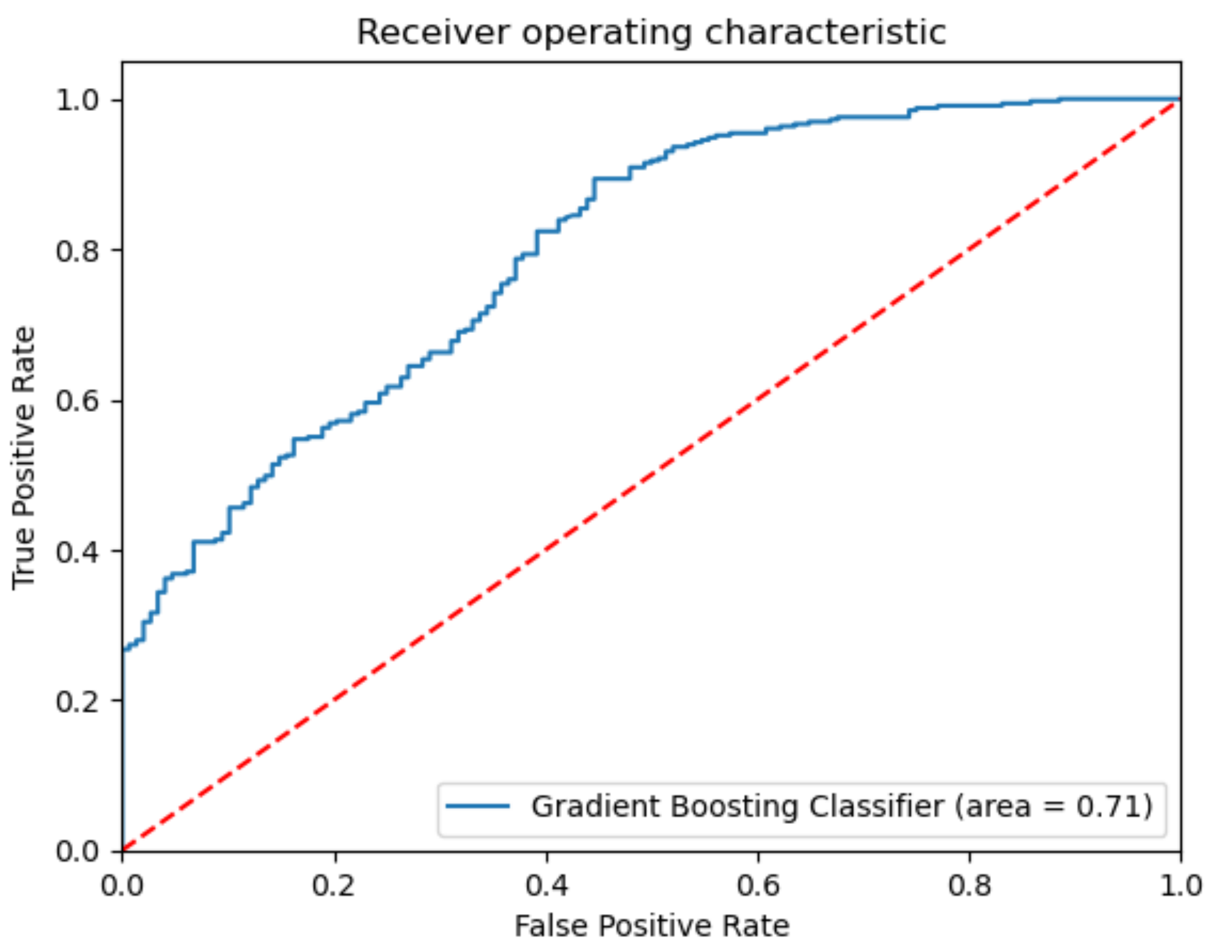
Out of all prediction, the measure for correctly predicted 0 is 60% and for 1 is 82% (Precision) Out of all actual 0, the measure for correctly predicted is 59% and for 1 is 83% (Recall) As this is imbalanced dataset, We give importance to F1-Score metrics

F1 Score of 0 is 60%

F1 Score of 1 is 82%

ROC-AUC 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='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()
```



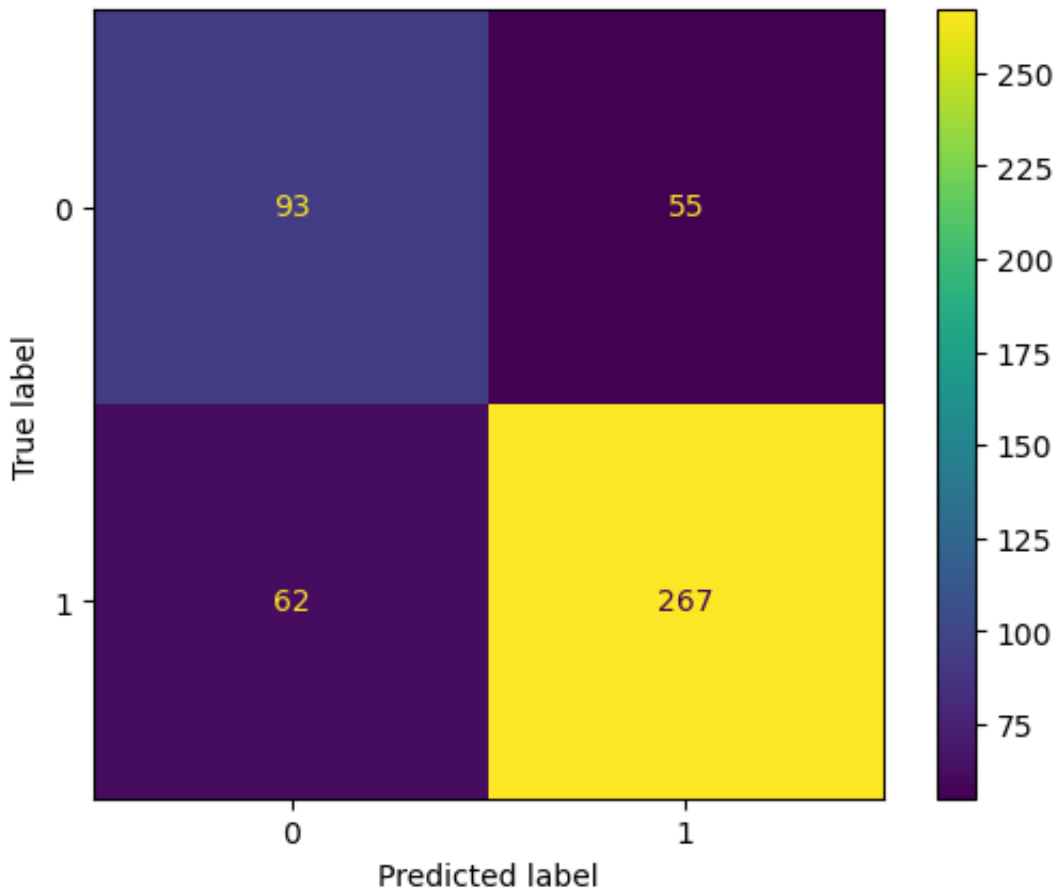
XGBoost Classifier

In [56]:
```
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:  0.7547169811320755

              precision    recall  f1-score   support

           0       0.60      0.63      0.61       148
           1       0.83      0.81      0.82       329

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

```
/Users/ramv/anaconda3/lib/python3.10/site-packages/xgboost/core.py:158: User
Warning: [15:55:51] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.
cc:740:
Parameters: { "class_weight" } are not used.

  warnings.warn(smsg, UserWarning)
```

Out[56]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x168fcab0
0>
```



XGBoost Classifier with balanced class weight

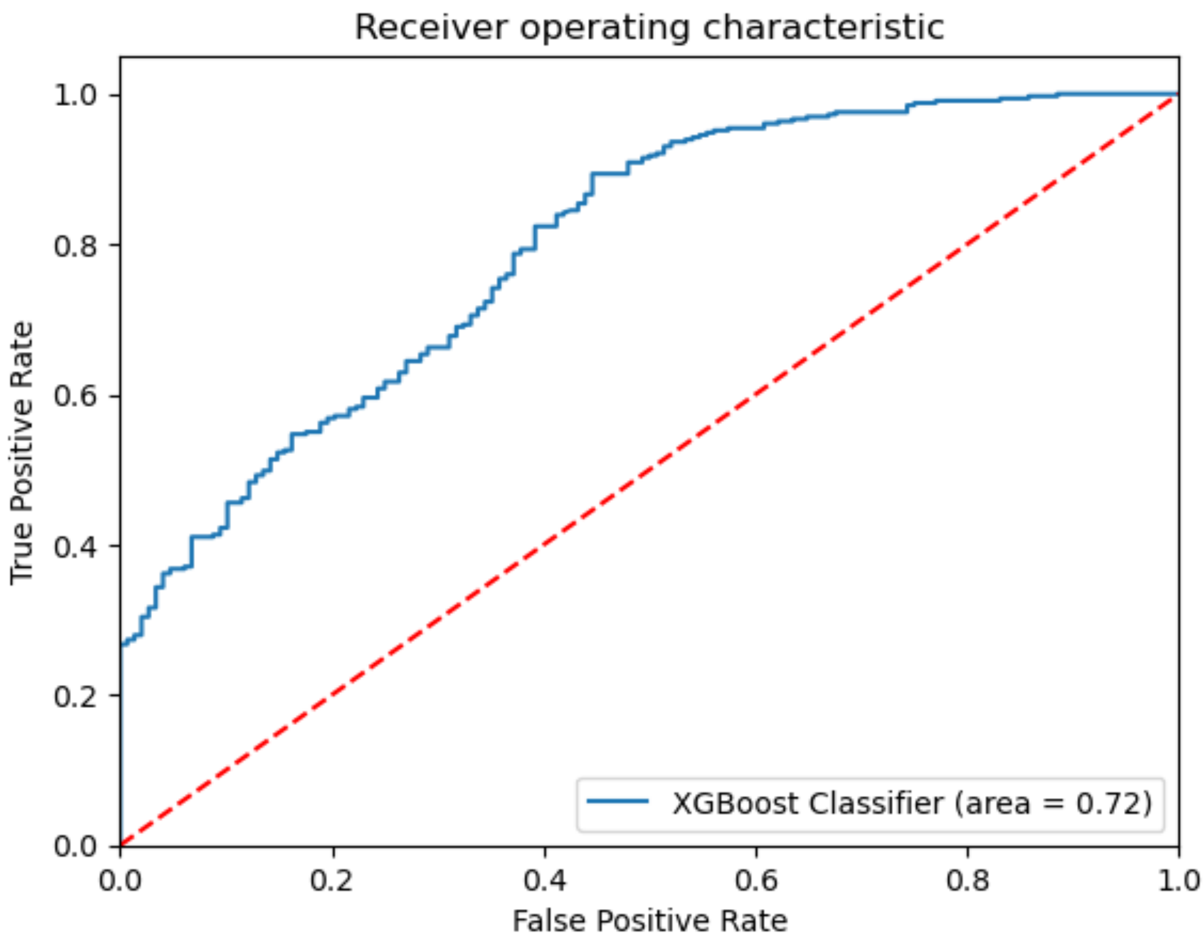
Out of all prediction, the measure for correctly predicted 0 is 60% and for 1 is 83% (Precision) Out of all actual 0, the measure for correctly predicted is 63% and for 1 is 81% (Recall) As this is imbalanced dataset, We give importance to F1-Score metrics

F1 Score of 0 is 61%

F1 Score of 1 is 82%

ROC-AUC Curve

In [57]:
```
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()
```



Final Result Summary:

We observe that we are not getting very high recall on target 0 which may be due to small unbalanced dataset. Higher precision means that an algorithm returns more relevant results than irrelevant ones, and high recall means that an algorithm returns most of the relevant results (whether or not irrelevant ones are also returned).

We observe that Random Forest with SMOTE outperforms rest of the models and has higher recall and precision values.

The Random Forest method out of all predicted 0 the measure of correctly predicted is 70%, and for 1 it is 82%(Precision). The Random Forest method out of all actual 0 the measure of correctly predicted is 57%, and for 1 it is 91%(Recall). The ROC-AUC curve area for Random Forest Classifier is 0.74

Gradient Boosting Classifier Result:

Out of all prediction, the measure for correctly predicted 0 is 60% and for 1 is 82% (Precision) Out of all actual 0, the measure for correctly predicted is 59% and for 1 is 83% (Recall) ROC-AUC curve area for Gradient Boosting Decision Tree Classifier is 0.71

XGBoost Classifier Result:

Out of all prediction, the measure for correctly predicted 0 is 60% and for 1 is 83% (Precision) Out of all actual 0, the measure for correctly predicted is 63% and for 1 is 81% (Recall) ROC-AUC curve area for XGBoost Classifier is 0.72

Random Forest Classifier outperforms all the other models.

Best Parameters: Best Params: {'max_depth': 4, 'n_estimators': 50}

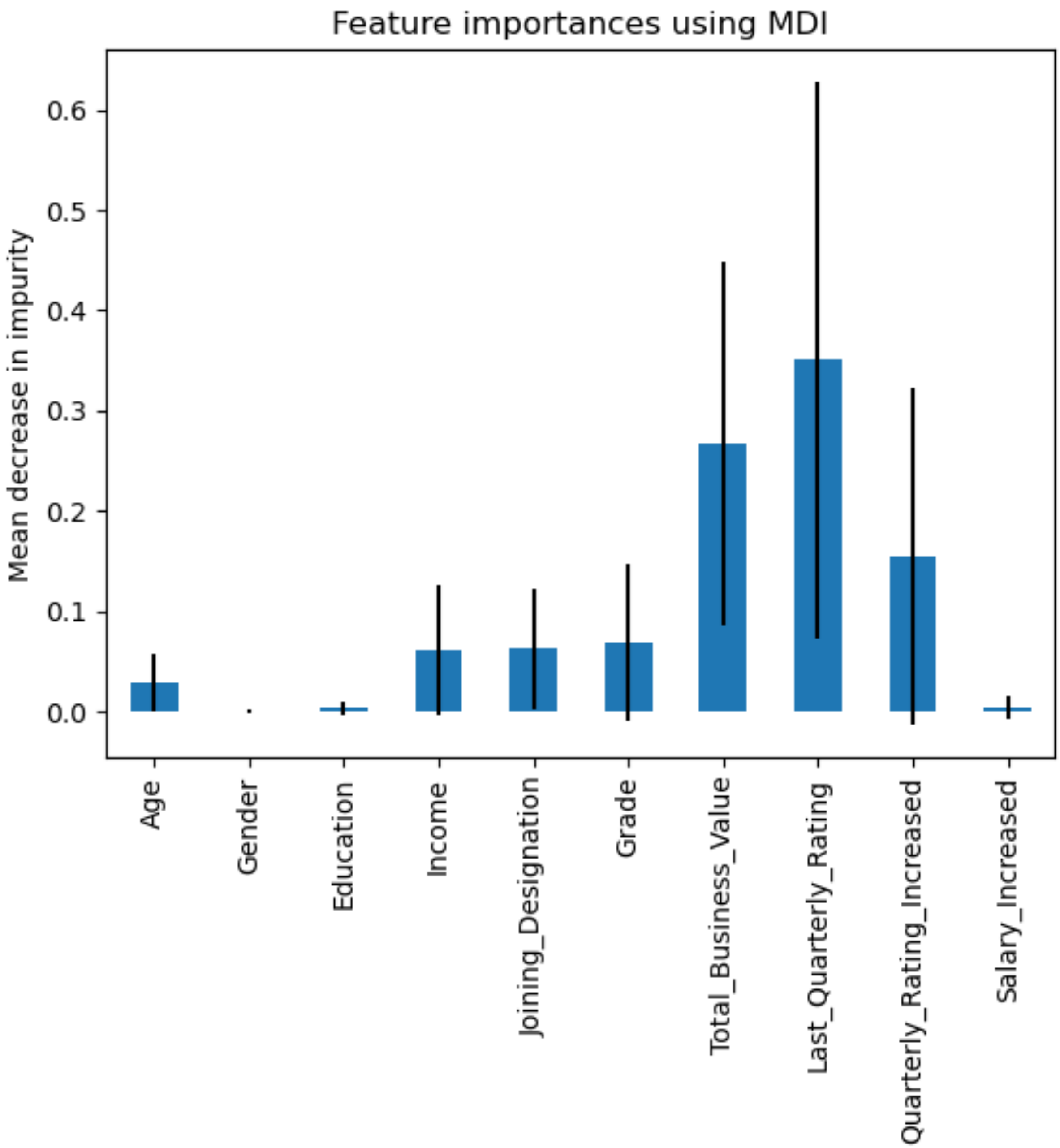
In [58]:
```
rf = RandomForestClassifier(max_depth = 4, n_estimators= 50, class_weight="b

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

```
Score of RandomForestClassifier:  0.7945492662473794
```

In [60]:
```
importances = rf.feature_importances_
importances

std = np.std([tree.feature_importances_ for tree in rf.estimators_], axis=0)
```

In [61]:
```
feature_importances = pd.Series(importances, X_train.columns)

plt.figure(figsize=(15,7))
fig, ax = plt.subplots()
feature_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")

plt.show()
```

```
<Figure size 1500x700 with 0 Axes>
```



Observation: Last_Quarterly_Rating, Total_Business_Value & Quarterly_Rating_Increased are the most important features.

Insights & Recommendations:

Out of a total of 2,381 drivers, 1,616 have left the company. To address driver churn, the company should consider offering overtime incentives or additional perks.

Drivers whose quarterly ratings have improved are less likely to leave the organization. Implementing a reward system for customers who provide feedback and rate drivers can help improve driver performance and retention.

Employees who have not received an increase in their monthly salary are more likely to leave the company. The organization should engage with these drivers, offering bonuses and perks to help them increase their earnings.

Among the 2,381 drivers, 1,744 had a last quarterly rating of 1. Furthermore, the quarterly rating has not improved for 2,076 drivers, which is a concerning trend that the company needs to address. Investigating why customers are not providing ratings could also help identify areas for improvement.

The features Last_Quarterly_Rating, Total_Business_Value, and Quarterly_Rating_Increased are critical indicators of driver retention and performance. The company should closely monitor these metrics as key predictors.

It has been observed that the model's recall for drivers who stayed (target = 0) is not very high, which could be due to the small and imbalanced dataset. Acquiring more data would likely improve this performance.

The Random Forest Classifier achieves a recall score of 91% for identifying drivers who left the company, indicating that the model is performing well.

In [ ]: