OLA - Ensemble Learning

January 31, 2024

[1]:
```python
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
```

[2]:
```python
df = pd.read_csv("C:\\Users\\Lenovo\\Downloads\\ola_driver_scaler.csv")
```

[3]:
```python
df.head()
```

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

   Income Dateofjoining LastWorkingDate  Joining Designation  Grade  \
0   57387      24/12/18             NaN                    1      1
1   57387      24/12/18             NaN                    1      1
2   57387      24/12/18        03/11/19                    1      1
3   67016      11/06/20             NaN                    2      2
4   67016      11/06/20             NaN                    2      2
```

```
   Total Business Value  Quarterly Rating
0               2381060                 2
1               -665480                 2
2                     0                 2
3                     0                 1
4                     0                 1
```

[4]:
```
df.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
```

[5]:
```
df.describe()
```

[5]:
```
         Unnamed: 0     Driver_ID           Age        Gender  \
count  19104.000000  19104.000000  19043.000000  19052.000000
mean    9551.500000   1415.591133     34.668435      0.418749
std     5514.994107    810.705321      6.257912      0.493367
min        0.000000      1.000000     21.000000      0.000000
25%     4775.750000    710.000000     30.000000      0.000000
50%     9551.500000   1417.000000     34.000000      0.000000
75%    14327.250000   2137.000000     39.000000      1.000000
max    19103.000000   2788.000000     58.000000      1.000000

       Education_Level        Income  Joining Designation         Grade  \
count     19104.000000  19104.000000         19104.000000  19104.000000
mean          1.021671  65652.025126             1.690536      2.252670
std           0.800167  30914.515344             0.836984      1.026512
```

```
min            0.000000   10747.000000            1.000000       1.000000
25%            0.000000   42383.000000            1.000000       1.000000
50%            1.000000   60087.000000            1.000000       2.000000
75%            2.000000   83969.000000            2.000000       3.000000
max            2.000000  188418.000000            5.000000       5.000000

       Total Business Value  Quarterly Rating
count          1.910400e+04      19104.000000
mean           5.716621e+05          2.008899
std            1.128312e+06          1.009832
min           -6.000000e+06          1.000000
25%            0.000000e+00          1.000000
50%            2.500000e+05          2.000000
75%            6.997000e+05          3.000000
max            3.374772e+07          4.000000
```

[6]:

```python
df.drop(columns = 'Unnamed: 0', axis = 1, inplace = True)
```

[7]:

```python
df.nunique()
```

[7]:

```
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
```

[8]:

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

[8]:

```
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
```

0.0.1 Converting features to respective data-types

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

[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
```

0.0.2 Check for missing values and Prepare data for KNN Imputation

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

[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.

[12]:
```
num_vars = df.select_dtypes(np.number)
num_vars.columns
```

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

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

0.0.3 KNN Imputation

[14]:
```
imputer = KNNImputer(n_neighbors=5, weights='uniform', metric='nan_euclidean')
imputer.fit(num_vars)
df_new = imputer.transform(num_vars)
```

[15]:
```
df_new = pd.DataFrame(df_new)
```

[16]:
```
df_new
```

[16]:
```
          0    1    2        3    4    5          6    7
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
…      …   …   …        …   …   …          …   …
19099  30.0  0.0  2.0  70254.0  2.0  2.0   740280.0  3.0
19100  30.0  0.0  2.0  70254.0  2.0  2.0   448370.0  3.0
19101  30.0  0.0  2.0  70254.0  2.0  2.0        0.0  2.0
19102  30.0  0.0  2.0  70254.0  2.0  2.0   200420.0  2.0
19103  30.0  0.0  2.0  70254.0  2.0  2.0   411480.0  2.0

[19104 rows x 8 columns]
```

[17]:
```
df_new.columns = num_vars.columns
```

[18]:
```
df_new.isnull().sum()
```

[18]:
```
Age                     0
Gender                  0
Education_Level         0
Income                  0
Joining Designation     0
Grade                   0
Total Business Value    0
Quarterly Rating        0
dtype: int64
```

0.0.4 We have successfully imputed the missing values using KNNImputer

[19]:
```
df_new.nunique()
```

[19]:
```
Age                        70
Gender                      6
Education_Level             3
Income                   2383
Joining Designation         5
Grade                       5
Total Business Value    10181
Quarterly Rating            4
dtype: int64
```

0.0.5 Concatenating dataframes

[20]:
```
resultant_columns = list(set(df.columns).difference(set(num_vars)))
resultant_columns
```

[20]:
```
['Dateofjoining', 'Driver_ID', 'MMM-YY', 'City', 'LastWorkingDate']
```

[21]:
```
new_df = pd.concat([df_new, df[resultant_columns]], axis=1)
new_df.shape
```

[21]:
```
(19104, 13)
```

[22]:
```
new_df.head()
```

[22]:
```
    Age  Gender  Education_Level   Income  Joining Designation  Grade  \
0  28.0     0.0              2.0  57387.0                  1.0    1.0
1  28.0     0.0              2.0  57387.0                  1.0    1.0
2  28.0     0.0              2.0  57387.0                  1.0    1.0
3  31.0     0.0              2.0  67016.0                  2.0    2.0
4  31.0     0.0              2.0  67016.0                  2.0    2.0

   Total Business Value  Quarterly Rating Dateofjoining  Driver_ID     MMM-YY  \
0             2381060.0               2.0    2018-12-24          1 2019-01-01
1             -665480.0               2.0    2018-12-24          1 2019-02-01
```

```
2                    0.0                 2.0     2018-12-24           1 2019-03-01
3                    0.0                 1.0     2020-11-06           2 2020-11-01
4                    0.0                 1.0     2020-11-06           2 2020-12-01

  City LastWorkingDate
0  C23             NaT
1  C23             NaT
2  C23      2019-03-11
3   C7             NaT
4   C7             NaT
```

[23]:
```
new_df.columns
```

[23]:
```
Index(['Age', 'Gender', 'Education_Level', 'Income', 'Joining Designation',
       'Grade', 'Total Business Value', 'Quarterly Rating', 'Dateofjoining',
       'Driver_ID', 'MMM-YY', 'City', 'LastWorkingDate'],
      dtype='object')
```

0.0.6 Data Preprocessing

0.0.7 Feature Engineering

[24]:
```
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()
```

[24]:
```
                       Age  Gender  Education_Level   Income  \
Driver_ID MMM-YY
1         2019-01-01  28.0     0.0              2.0  57387.0
          2019-02-01  28.0     0.0              2.0  57387.0
          2019-03-01  28.0     0.0              2.0  57387.0
2         2020-11-01  31.0     0.0              2.0  67016.0
          2020-12-01  31.0     0.0              2.0  67016.0
```

```
                    Joining Designation  Grade  Total Business Value  \
Driver_ID MMM-YY
1         2019-01-01                1.0    1.0             2381060.0
          2019-02-01                1.0    1.0             -665480.0
          2019-03-01                1.0    1.0                   0.0
2         2020-11-01                2.0    2.0                   0.0
          2020-12-01                2.0    2.0                   0.0

                    Quarterly Rating LastWorkingDate City Dateofjoining
Driver_ID MMM-YY
1         2019-01-01             2.0             NaT  C23    2018-12-24
          2019-02-01             2.0             NaT  C23    2018-12-24
          2019-03-01             2.0      2019-03-11  C23    2018-12-24
2         2020-11-01             1.0             NaT   C7    2020-11-06
          2020-12-01             1.0             NaT   C7    2020-11-06
```

[25]:
```
final_data = pd.DataFrame()
```

[26]:
```
final_data
```

[26]:
```
Empty DataFrame
Columns: []
Index: []
```

[27]:
```
final_data["Driver_ID"] = new_df["Driver_ID"].unique()
```

[28]:
```
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'])
```

[29]:
```python
final_data.head()
```

[29]:
```
   Driver_ID   Age  Gender City  Education   Income  Joining_Designation  \
0          1  28.0     0.0  C23        2.0  57387.0                  1.0
1          2  31.0     0.0   C7        2.0  67016.0                  2.0
2          4  43.0     0.0  C13        2.0  65603.0                  2.0
3          5  29.0     0.0   C9        0.0  46368.0                  1.0
4          6  31.0     1.0  C11        1.0  78728.0                  3.0

   Grade  Total_Business_Value  Last_Quarterly_Rating
0    1.0             1715580.0                    2.0
1    2.0                   0.0                    1.0
2    2.0              350000.0                    1.0
3    1.0              120360.0                    1.0
4    3.0             1265000.0                    2.0
```

[30]:
```python
final_data.shape
```

[30]: (2381, 10)

0.0.8 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

[31]:
```python
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["Quarterly_Rating_Increased"] = qrl
```

[32]:
```python
final_data.head()
```

[32]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation
0	1	28.0	0.0	C23	2.0	57387.0	1.0
1	2	31.0	0.0	C7	2.0	67016.0	2.0
2	4	43.0	0.0	C13	2.0	65603.0	2.0
3	5	29.0	0.0	C9	0.0	46368.0	1.0
4	6	31.0	1.0	C11	1.0	78728.0	3.0

	Grade	Total_Business_Value	Last_Quarterly_Rating
0	1.0	1715580.0	2.0
1	2.0	0.0	1.0
2	2.0	350000.0	1.0
3	1.0	120360.0	1.0
4	3.0	1265000.0	2.0

	Quarterly_Rating_Increased
0	0
1	0
2	0
3	0
4	1

0.0.9 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

[33]:

```
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
```

[34]:

```
final_data.head()
```

[34]:

	Driver_ID	Age	Gender	City	Education	Income	Joining_Designation
0	1	28.0	0.0	C23	2.0	57387.0	1.0
1	2	31.0	0.0	C7	2.0	67016.0	2.0
2	4	43.0	0.0	C13	2.0	65603.0	2.0
3	5	29.0	0.0	C9	0.0	46368.0	1.0
4	6	31.0	1.0	C11	1.0	78728.0	3.0

```
   Grade  Total_Business_Value  Last_Quarterly_Rating  \
0    1.0             1715580.0                    2.0
1    2.0                   0.0                    1.0
2    2.0              350000.0                    1.0
3    1.0              120360.0                    1.0
4    3.0             1265000.0                    2.0

   Quarterly_Rating_Increased  target
0                           0       1
1                           0       0
2                           0       1
3                           0       1
4                           1       0
```

0.0.10 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

[35]:
```
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
```

[36]:
```
final_data.head()
```

[36]:
```
   Driver_ID   Age  Gender City  Education   Income  Joining_Designation  \
0          1  28.0     0.0  C23        2.0  57387.0                  1.0
1          2  31.0     0.0   C7        2.0  67016.0                  2.0
2          4  43.0     0.0  C13        2.0  65603.0                  2.0
3          5  29.0     0.0   C9        0.0  46368.0                  1.0
4          6  31.0     1.0  C11        1.0  78728.0                  3.0

   Grade  Total_Business_Value  Last_Quarterly_Rating  \
0    1.0             1715580.0                    2.0
1    2.0                   0.0                    1.0
```

```
2    2.0             350000.0                    1.0
3    1.0             120360.0                    1.0
4    3.0            1265000.0                    2.0

   Quarterly_Rating_Increased  target  Salary_Increased
0                           0       1                 0
1                           0       0                 0
2                           0       1                 0
3                           0       1                 0
4                           1       0                 0
```

[37]:
```
final_data["Salary_Increased"].value_counts(normalize=True)
```

[37]:
```
0    0.98194
1    0.01806
Name: Salary_Increased, dtype: float64
```

Around 1.8% drivers income have been increased.

0.0.11 Statistical Summary

[38]:
```
final_data.describe().T
```

[38]:
```
                             count          mean           std         min  \
Driver_ID                   2381.0  1.397559e+03  8.061616e+02         1.0
Age                         2381.0  3.377018e+01  5.933265e+00        21.0
Gender                      2381.0  4.105838e-01  4.914963e-01         0.0
Education                   2381.0  1.007560e+00  8.162900e-01         0.0
Income                      2381.0  5.933416e+04  2.838367e+04     10747.0
Joining_Designation         2381.0  1.820244e+00  8.414334e-01         1.0
Grade                       2381.0  2.096598e+00  9.415218e-01         1.0
Total_Business_Value        2381.0  4.586742e+06  9.127115e+06  -1385530.0
Last_Quarterly_Rating       2381.0  1.427971e+00  8.098389e-01         1.0
Quarterly_Rating_Increased  2381.0  1.503570e-01  3.574961e-01         0.0
target                      2381.0  6.787064e-01  4.670713e-01         0.0
Salary_Increased            2381.0  1.805964e-02  1.331951e-01         0.0

                                25%       50%        75%         max
Driver_ID                     695.0    1400.0     2100.0      2788.0
Age                            30.0      33.0       37.0        58.0
Gender                          0.0       0.0        1.0         1.0
Education                       0.0       1.0        2.0         2.0
Income                      39104.0   55315.0    75986.0    188418.0
Joining_Designation             1.0       2.0        2.0         5.0
Grade                           1.0       2.0        3.0         5.0
Total_Business_Value            0.0  817680.0  4173650.0  95331060.0
Last_Quarterly_Rating           1.0       1.0        2.0         4.0
Quarterly_Rating_Increased      0.0       0.0        0.0         1.0
```

```
target                          0.0        1.0         1.0          1.0
Salary_Increased                0.0        0.0         0.0          1.0
```

- There are total of 2831 different drivers data.
- Age of drivers range from 21years to 58years.
- 75% drivers monthly income is <= 75986.
- 75% drivers acquired 4173650 as total business values

[39]:
```
final_data.describe(include = 'object')
```

[39]:
```
        City
count   2381
unique    29
top      C20
freq     152
```

- Majority of drivers are coming from C20 city

[40]:
```
final_data["Gender"].value_counts()
```

[40]:
```
0.0    1400
1.0     975
0.6       3
0.2       2
0.4       1
Name: Gender, dtype: int64
```

- Majority of drivers are male

[41]:
```
final_data["Education"].value_counts()
```

[41]:
```
2.0    802
1.0    795
0.0    784
Name: Education, dtype: int64
```

- Majority of drivers have completed their graduation.

[42]:
```
final_data["target"].value_counts()
```

[42]:
```
1    1616
0     765
Name: target, dtype: int64
```

- Out of 2381 drivers 1616 have left the company.

[43]:
```
n =␣
↪['Gender','Education','Joining_Designation','Grade','Last_Quarterly_Rating','Quarterly_Rati

for i in n:
```

```python
    print("-----------------------------------------------------------")
    print(final_data[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
```

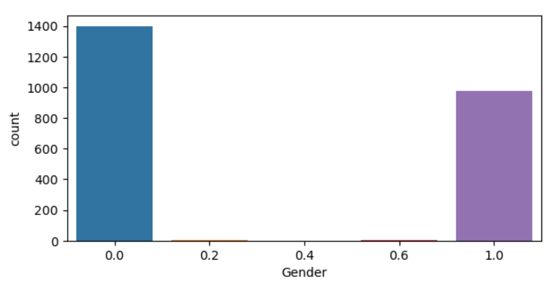
- 58% of drivers are male while female constitutes around 40%
- 33% of drivers have completed graduation and 12+ education
- 43% of drivers have 1 as joining_designation
- Around 36% of drivers graded as 2
- Around 73% of drivers rated as 1 on last quarter
- Only 15% of drivers rating has been increased on quarterly

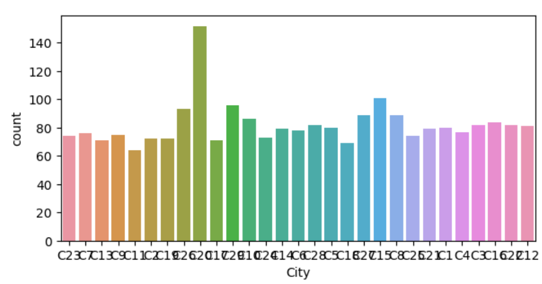
0.0.12 Univariate Analysis

[44]:
```
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="90")

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)
Cell In[44], line 8
      6 plt.subplot(422)
      7 sns.countplot(data=final_data, x="City")
----> 8 plt.xticks(rotation="90")
     10 plt.subplot(423)
     11 sns.countplot(data=final_data, x="Joining_Designation")

File ~\anaconda3\Lib\site-packages\matplotlib\pyplot.py:1891, in xticks(ticks,␣
↪labels, minor, **kwargs)
   1889     labels = ax.get_xticklabels(minor=minor)
   1890     for l in labels:
-> 1891         l._internal_update(kwargs)
   1892 else:
   1893     labels = ax.set_xticklabels(labels, minor=minor, **kwargs)
```

```
File ~\anaconda3\Lib\site-packages\matplotlib\artist.py:1223, in Artist.
↪_internal_update(self, kwargs)
   1216 def _internal_update(self, kwargs):
   1217     """
   1218     Update artist properties without prenormalizing them, but generating
   1219     errors as if calling `set`.
   1220 
   1221     The lack of prenormalization is to maintain backcompatibility.
   1222     """
-> 1223     return self._update_props(
   1224         kwargs, "{cls.__name__}.set() got an unexpected keyword argument␣
↪"
   1225         "{prop_name!r}")

File ~\anaconda3\Lib\site-packages\matplotlib\artist.py:1199, in Artist.
↪_update_props(self, props, errfmt)
   1196             if not callable(func):
   1197                 raise AttributeError(
   1198                     errfmt.format(cls=type(self), prop_name=k))
-> 1199             ret.append(func(v))
   1200 if ret:
   1201     self.pchanged()

File ~\anaconda3\Lib\site-packages\matplotlib\text.py:1234, in Text.
↪set_rotation(self, s)
   1232     self._rotation = 90.
   1233 else:
-> 1234     raise ValueError("rotation must be 'vertical', 'horizontal' or "
   1235                      f"a number, not {s}")
   1236 self.stale = True

ValueError: rotation must be 'vertical', 'horizontal' or a number, not 90
```





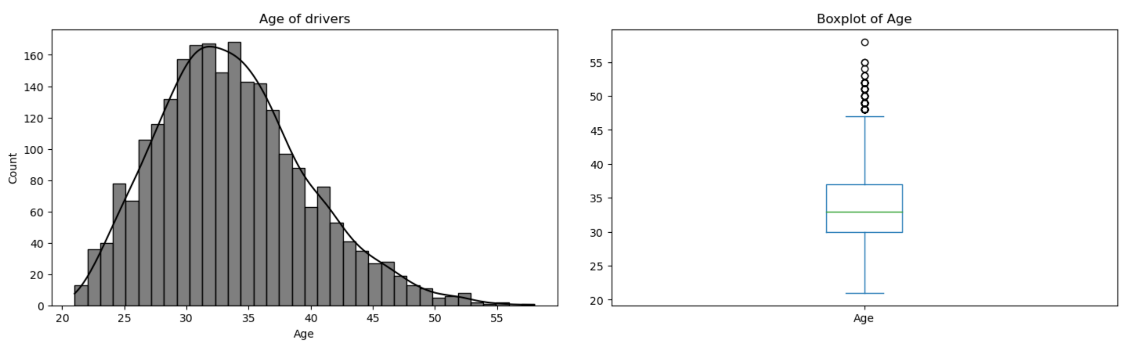
Insights

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

- 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.

[45]:
```
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)
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_16588\3593816054.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)
```



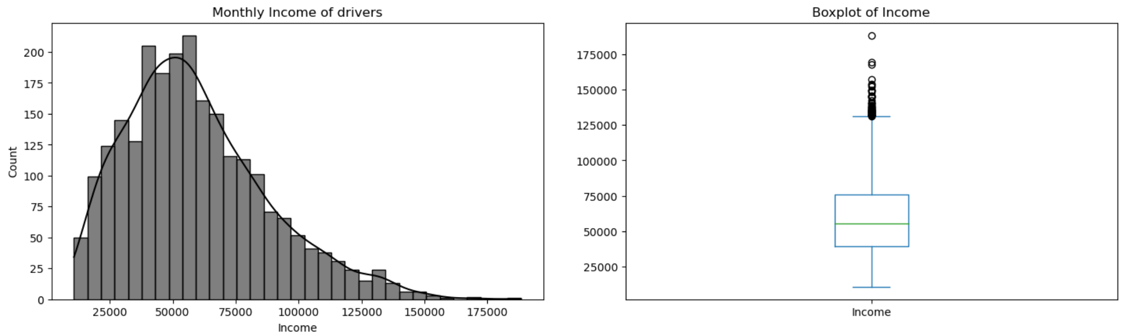
Insights

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

[46]:
```
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)
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_16588\2831293380.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)
```

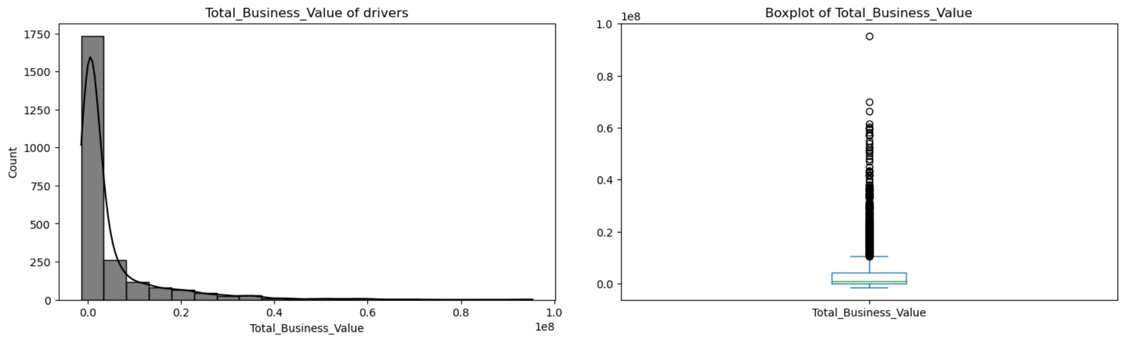


Insights

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

[47]:
```python
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)
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_16588\3219060859.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)
```



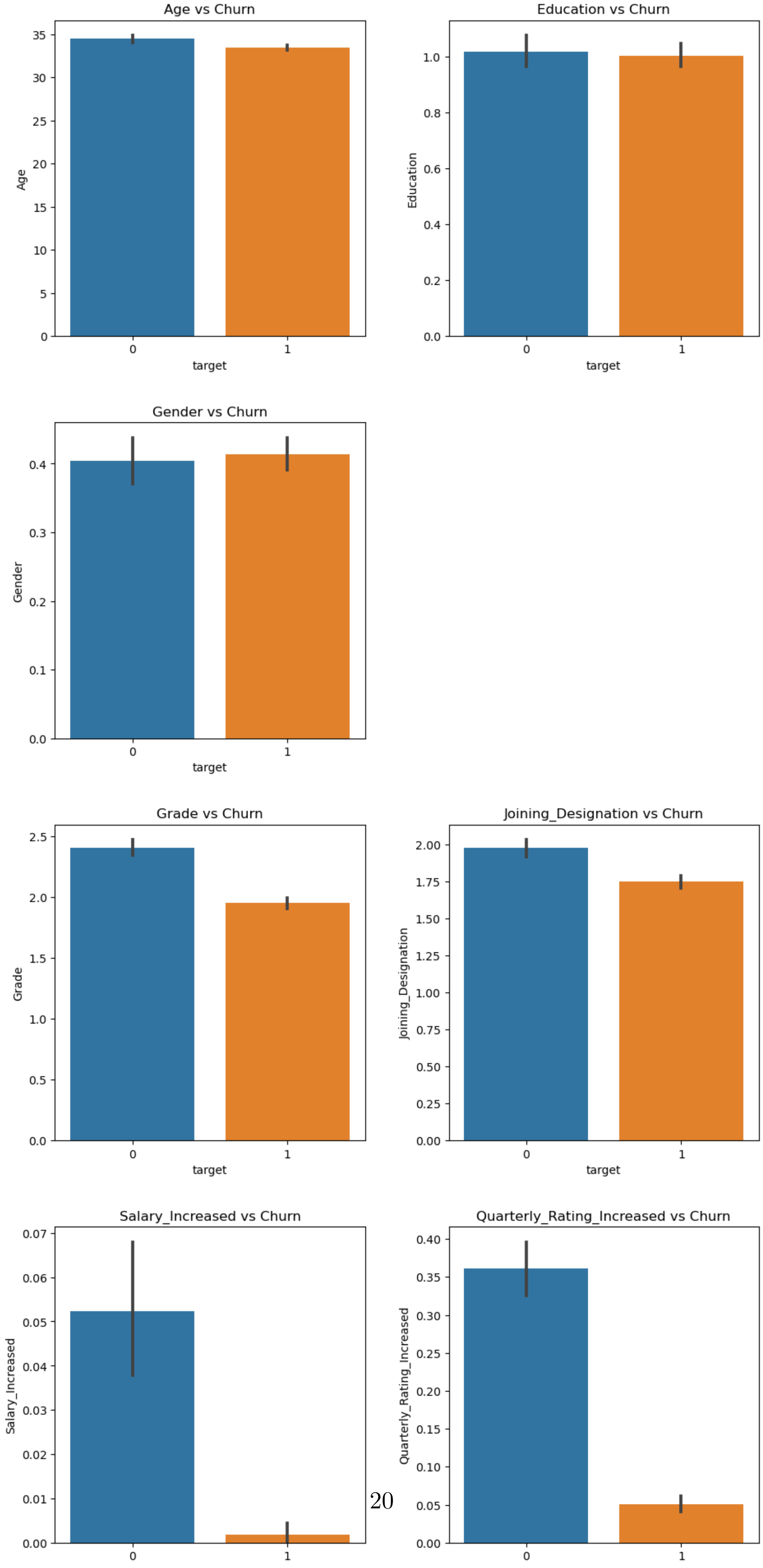
Insights

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

0.0.13 Bi-Variate Analysis

[48]:

```
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
Age
target
Education vs Churn
Education
target
Gender vs Churn
Gender
target
Grade vs Churn
Grade
target
Joining_Designation vs Churn
Joining_Designation
target
Salary_Increased vs Churn
Salary_Increased
target
Quarterly_Rating_Increased vs Churn
Quarterly_Rating_Increased
target

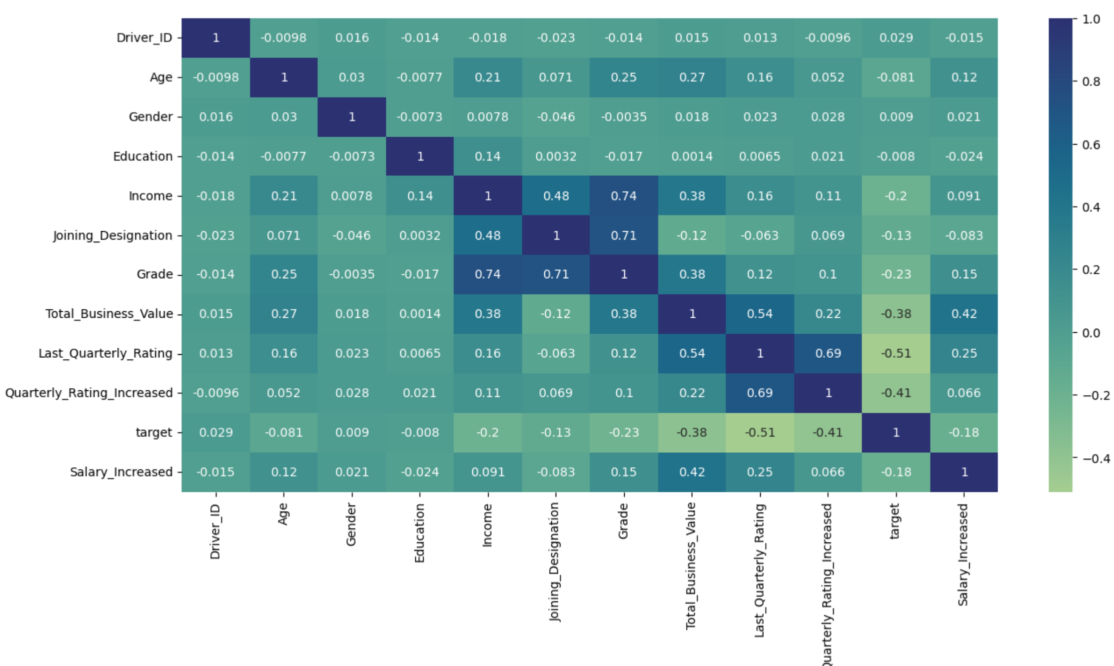
Insights

- 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.

0.0.14 Correlation Analysis

[49]:
```
plt.figure(figsize=(15, 7))

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

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_16588\617819596.py:3:
FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. 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(final_data.corr(method="pearson"), annot=True, cmap="crest")
```



Insights

- Income and Grade is highly correlated
- Joining Designation and Grade is highly correlated
- Total Business value and salary increament is correlated

0.0.15 One-Hot Encoding

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

[50]:
```
final_data = pd.concat([final_data, final_data['City']], axis=1)
```

[51]:
```
final_data.shape
```

[51]:
```
(2381, 14)
```

0.0.16 Standardization (for training data)

[52]:
```
X = final_data.drop(["Driver_ID", "target", "City"], axis = 1)
X_cols = X.columns
scaler = MinMaxScaler()

X = scaler.fit_transform(X)
```

[53]:
```
X = pd.DataFrame(X)
X.columns = X_cols
X
```

[53]:
```
           Age  Gender  Education    Income  Joining_Designation  Grade  \
0     0.189189     0.0        1.0  0.262508                 0.00   0.00
1     0.270270     0.0        1.0  0.316703                 0.25   0.25
2     0.594595     0.0        1.0  0.308750                 0.25   0.25
3     0.216216     0.0        0.0  0.200489                 0.00   0.00
4     0.270270     1.0        0.5  0.382623                 0.50   0.50
…          …     …        …     …                     …     …
2376  0.351351     0.0        0.0  0.405626                 0.25   0.50
2377  0.351351     1.0        0.0  0.007643                 0.00   0.00
2378  0.648649     0.0        0.0  0.138588                 0.25   0.25
2379  0.189189     1.0        1.0  0.330673                 0.00   0.00
2380  0.243243     0.0        1.0  0.334928                 0.25   0.25

      Total_Business_Value  Last_Quarterly_Rating  Quarterly_Rating_Increased  \
0                 0.032064               0.333333                         0.0
1                 0.014326               0.000000                         0.0
2                 0.017944               0.000000                         0.0
3                 0.015570               0.000000                         0.0
4                 0.027405               0.333333                         1.0
…                      …                      …                          …
2376              0.239197               1.000000                         1.0
```

```
2377            0.014326            0.000000                            0.0
2378            0.043432            0.000000                            0.0
2379            0.024436            0.000000                            0.0
2380            0.038088            0.333333                            1.0

      Salary_Increased
0                  0.0
1                  0.0
2                  0.0
3                  0.0
4                  0.0
…                …
2376               0.0
2377               0.0
2378               0.0
2379               0.0
2380               0.0

[2381 rows x 10 columns]
```

0.0.17 Train & Test Split

[54]:
```
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)
```

[55]:
```
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,)
```

0.0.18 Random Forest Classifier - Before Balancing

Keeping max_depth small to avoid overfitting

[56]:
```
params = {
    "max_depth": [2, 3, 4],
    "n_estimators": [50, 100, 150, 200],
}

start_time = time.time()
random_forest = RandomForestClassifier(class_weight="balanced")
```

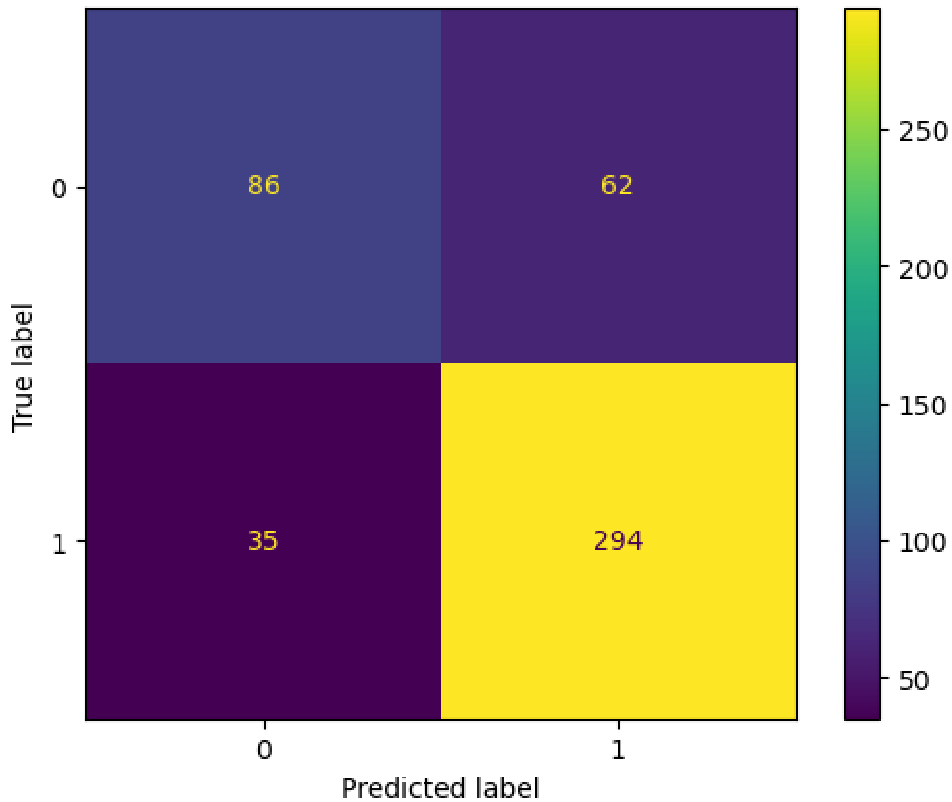
```
c = GridSearchCV(estimator=random_forest, param_grid=params, n_jobs=-1, cv=3,␣
↪verbose=True, scoring='f1')

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:  0.862567496519285

Elapsed Time:  5.831505060195923
```

[57]:
```
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      0.58      0.64       148
           1       0.83      0.89      0.86       329

    accuracy                           0.80       477
   macro avg       0.77      0.74      0.75       477
weighted avg       0.79      0.80      0.79       477
```

[57]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2c9368f9d90>



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

[58]:
```
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')
```

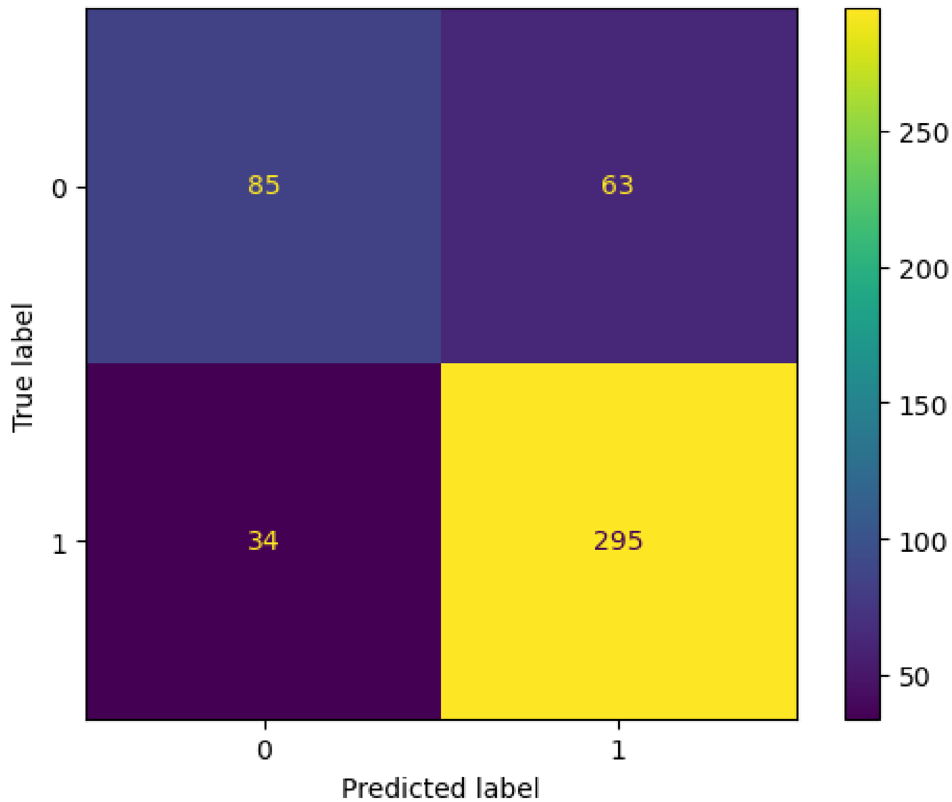
```
c.fit(X_train, y_train)

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:  0.8619369478383461

Elapsed Time:  8.724973440170288
```

[59]:
```
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      0.64       148
           1       0.82      0.90      0.86       329

    accuracy                           0.80       477
   macro avg       0.77      0.74      0.75       477
weighted avg       0.79      0.80      0.79       477
```

[59]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2c93836f090>



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

[60]:
```
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
```

0.0.19 Ensemble Learning: Bagging

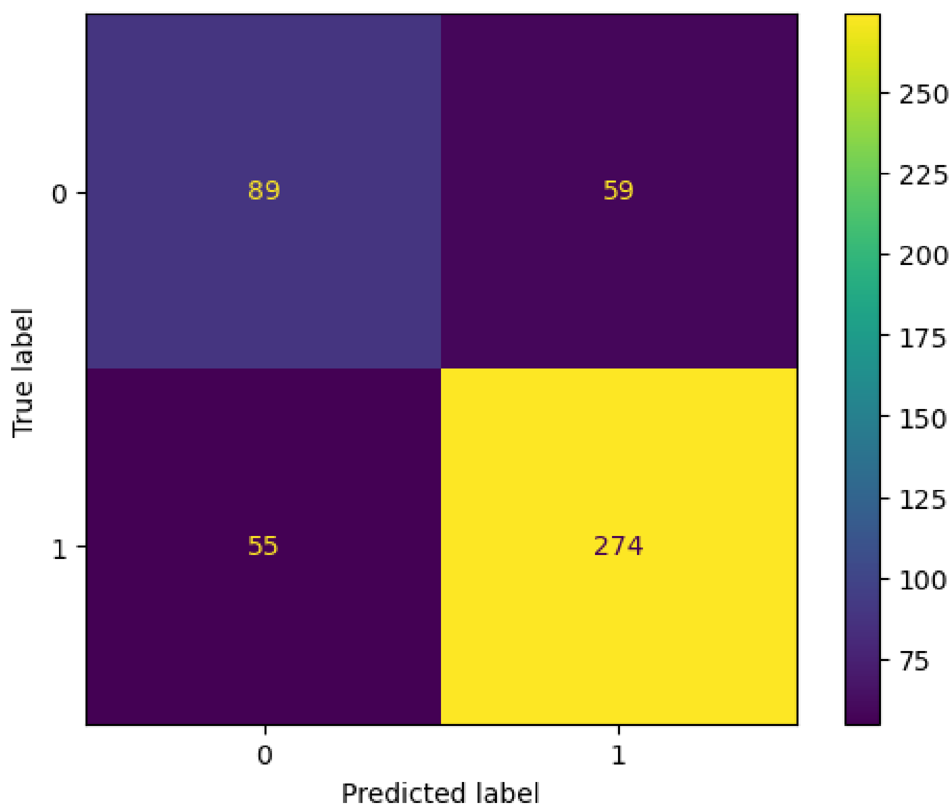
0.0.20 Gradient Boosting Classifier

[61]:
```
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_)

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': 0.2, 'loss': 'exponential', 'max_depth': 4,
'n_estimators': 150, 'subsample': 1}
Best Score:  0.8108003108003108

 Elapsed Time:  247.32427382469177
              precision    recall  f1-score   support

           0       0.62      0.60      0.61       148
           1       0.82      0.83      0.83       329

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

[61]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2c9387aae90>



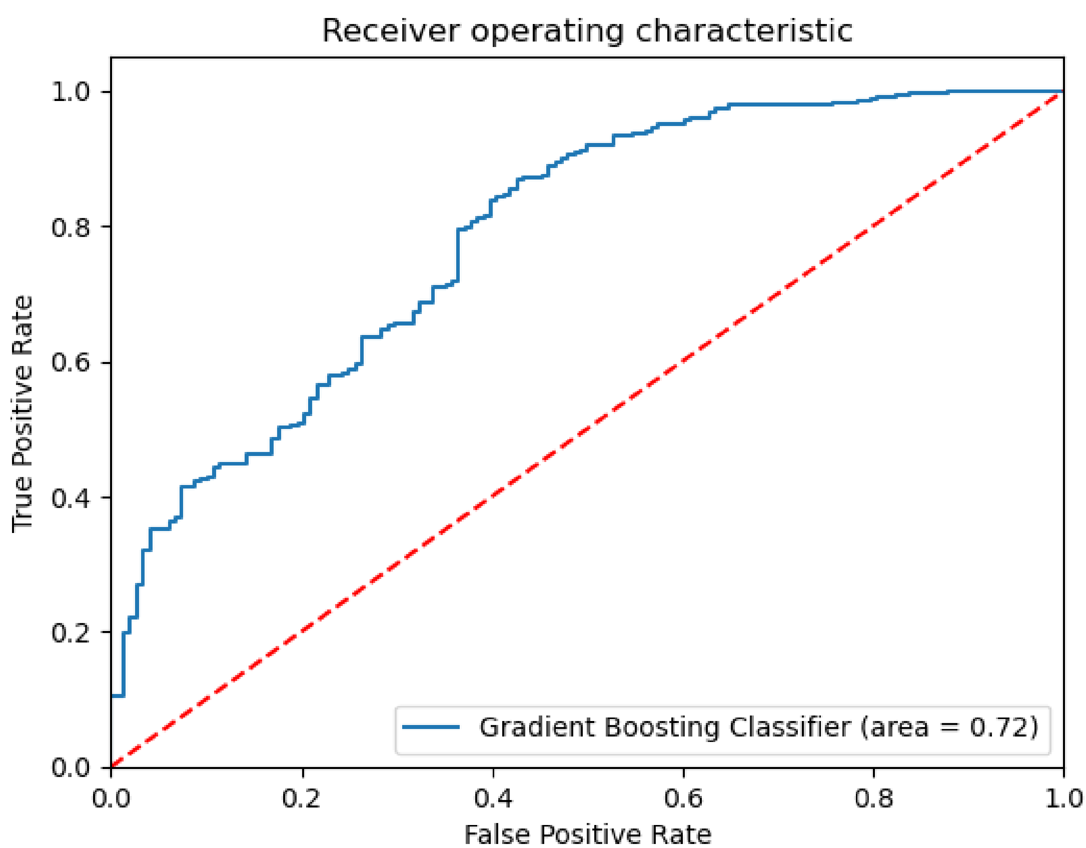
0.0.21 Gradient Boosting Classifier Metrics

- 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

0.0.22 F1 Score of 0 is 61%

0.0.23 F1 Score of 1 is 83%

[62]:
```
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()
```



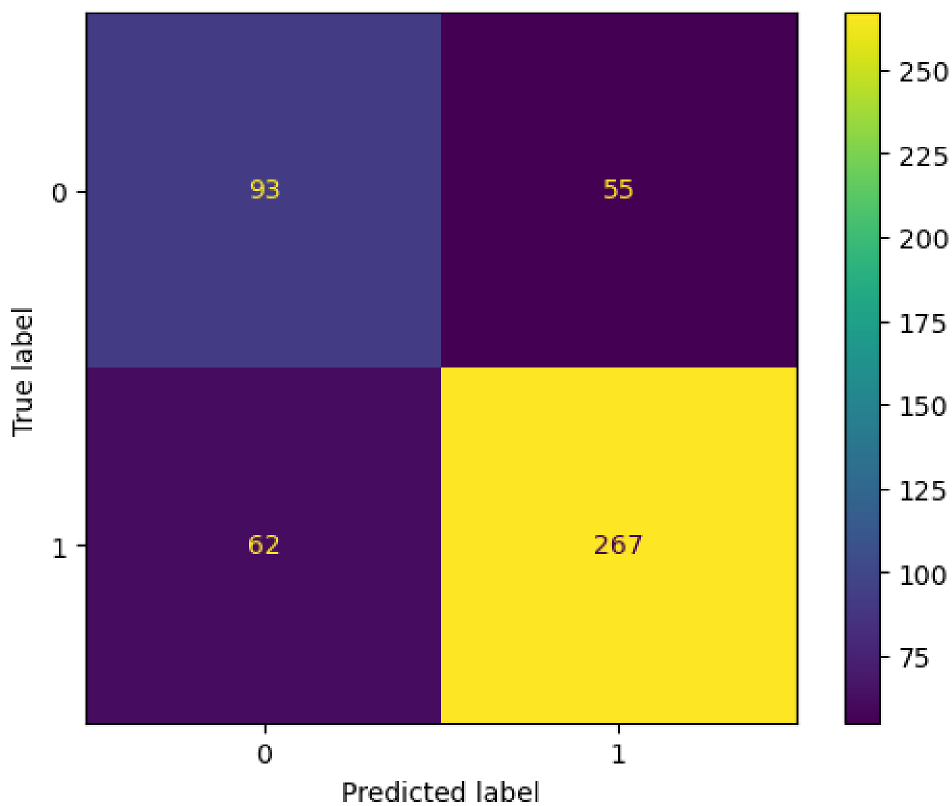
0.0.24 XGBoost Classifier

[63]:
```
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()
```

```
C:\Users\Lenovo\anaconda3\Lib\site-packages\xgboost\core.py:160: UserWarning:
[15:06:06] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0b3782d1791676daf-1\xgboost\xgboost-ci-windows\src\learner.cc:742:
Parameters: { "class_weight" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
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
```

[63]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2c9382b3710>



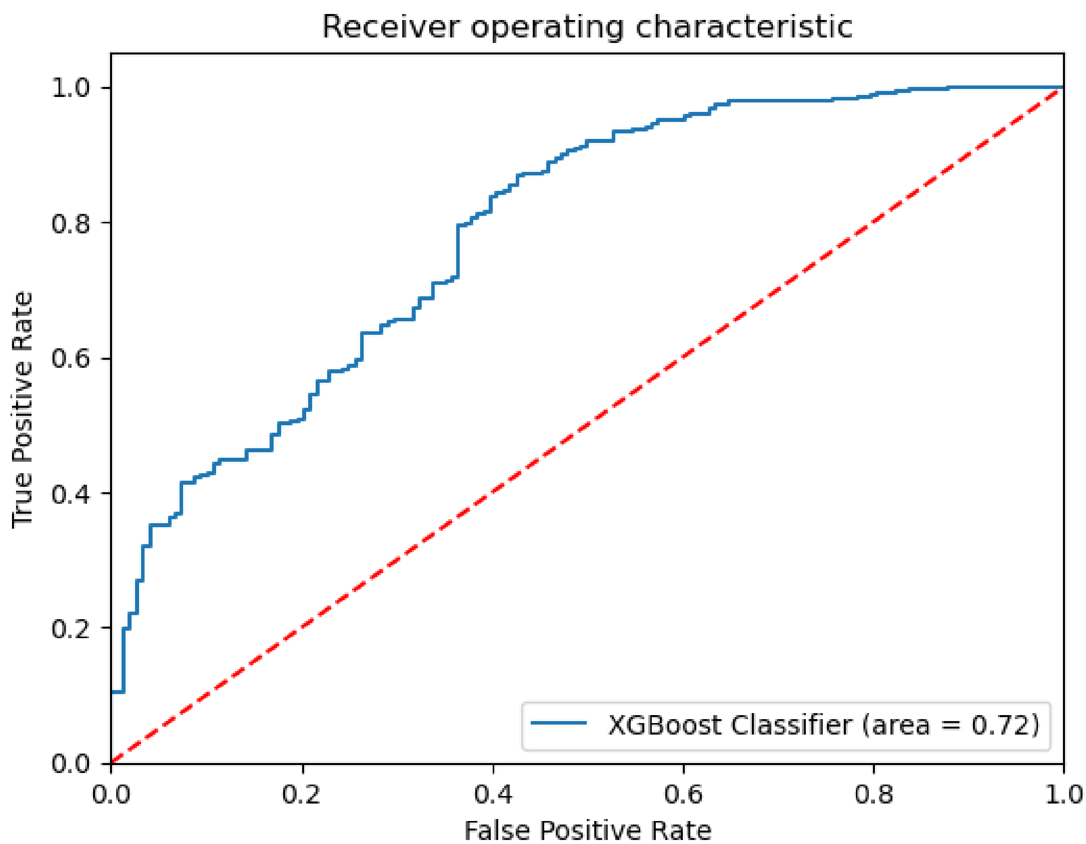
XGBoost Classifier with balanced class weight

- 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%

[64]:
```
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()
```



0.0.25 Final Result Evaluation

- 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 73%, and for 1 it is 82%(Precision).
  - The Random Forest method out of all actual 0 the measure of correctly predicted is 56%, 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 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)

- The 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 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)

- The ROC-AUC curve area for XGBoost Classifier is 0.71

Feature Importance of the best model so far

- Random Forest Classifier outperforms the rest of the modal.
- Best parameters

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

[65]:
```
rf = RandomForestClassifier(max_depth = 4, n_estimators= 50,␣
 ↪class_weight="balanced")

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

```
Score of RandomForestClassifier:  0.8050314465408805
```

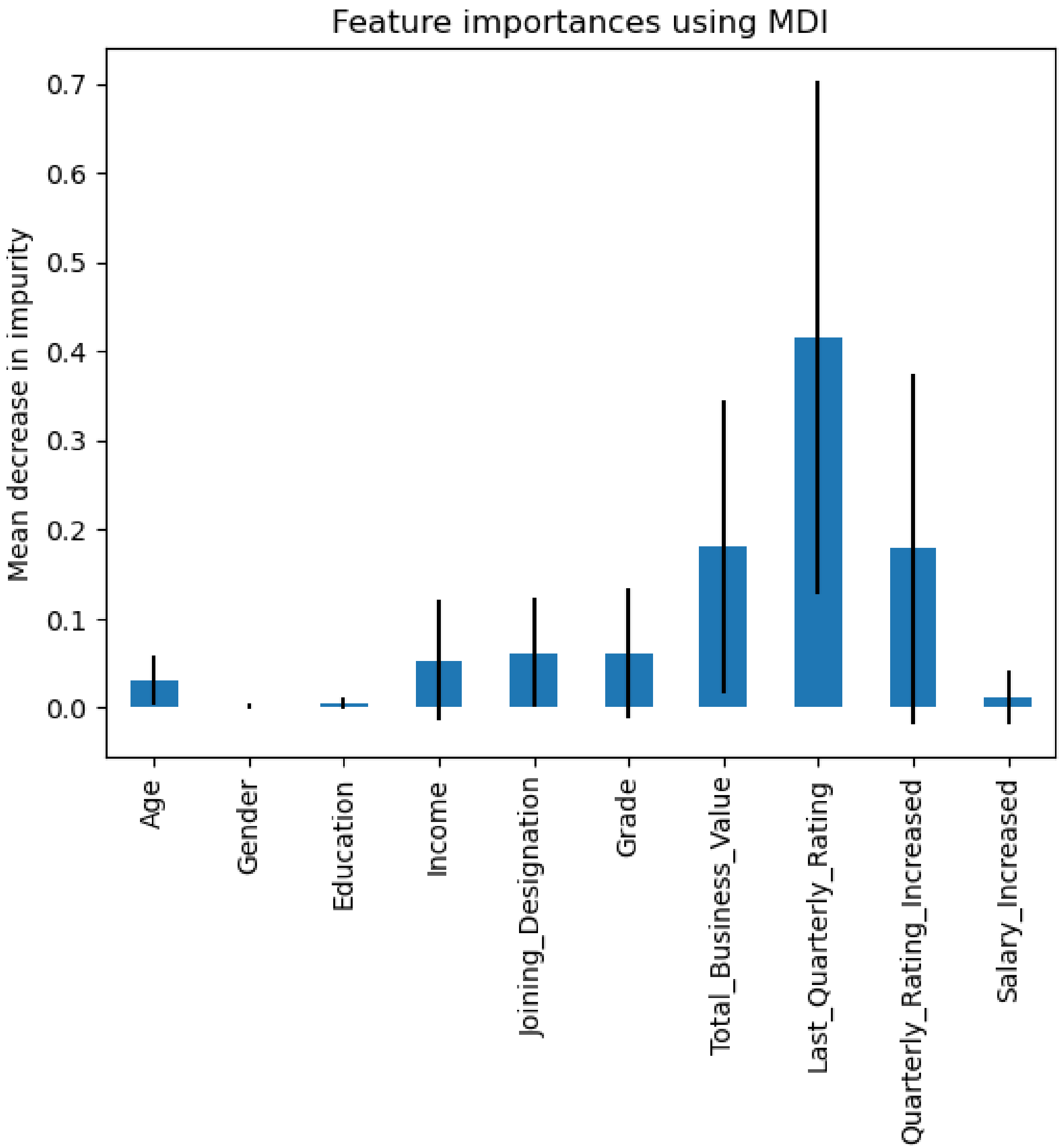
[66]:
```
importances = rf.feature_importances_
importances
```

[66]:
```
array([0.03115505, 0.00162143, 0.00479084, 0.05334725, 0.06185407,
       0.06109174, 0.1808501 , 0.41556439, 0.17832829, 0.01139683])
```

[67]:
```
std = np.std([tree.feature_importances_ for tree in rf.estimators_], axis=0)
```

[68]:
```
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>
```



Insights

- Last_Quarterly_Rating, Total_Business_Value & Quarterly_Rating_Increased are the most important features.

Actionable Insights and Recommendation

- Out of 2381 drivers 1616 have left the company.
- We need to incentivise the drivers overtime or other perks to overcome churning
- The employees whose quarterly rating has increased are less likely to leave the organization.
- Company needs to implement the reward system for the customer who provide the feedback and rate drivers

- The employees whose monthly salary has not increased are more likely to leave the organization.
- Company needs to get in touch with those drivers whose monthly salary has not increased and help them out to earn more by provider bonus and perks.
- Out of 2381 employees, 1744 employees had their last quarterly rating as 1.
- Out of 2381 employees, the quarterly rating has not increased for 2076 employees. This is red flag for the company which needs to regulate.
- Company needs to look why customers are not rating drivers.
- Last_Quarterly_Rating, Total_Business_Value & Quarterly_Rating_Increased are the most important features. Company needs to tracks these features as predicators
- We observe that we are not getting very high recall on target 0 which may be due to small unbalanced dataset. More data will overcome this issue.
- The Random Forest Classifier attains the Recall score of 91% for the driver who left the company. Which indicates that model is performing the decent job.

[ ]:
```

```