# Ola - Ensemble Learning Case Study

Problem Statement

Ola faces a tough competition from competitors as it is getting bigger as a company due to driver attrition. These drivers switch companies every now and then, which incurs higher costs to the company as they spend a lot in hiring of these drivers. The Analytics Department of Ola is responsible for understanding the possible reasons for driver churn beforehand, so that they can take appropriate steps to tackle it.

Objective of the Case Study : Develop ML Model that will predict whether a driver will be leaving the company or not based on their attributes like Demographics (City, Age, Gender etc.), Tenure Information (Joining Date, Last Date etc.), Historical data regarding the performance of the driver (Quarterly Rating, Monthly business acquired, grade, income).

# Importing the Libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('svg')
# Setting the theme for the plots to be drawn ahead
sns.set_theme(style="whitegrid")
```

In [3]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import KFold, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, roc_auc_score
from sklearn.metrics import precision_recall_curve, auc, f1_score, precision_score, recall_score, ConfusionMatrixDisplay
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
```

# Importing the Data

In [4]:
```
df = pd.read_csv(r"H:\Scaler\Machine Learning Projects\OLA\ola_driver_scaler.csv")
df.head()
```

Out[4]:

| | Unnamed: 0 | MMM-YY | Driver_ID | Age | Gender | City | Education_Level | Income | Dateofjoining | LastWorkingDate | Joining Designation | Grade | Total Business Value | Quarterly Rating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 01/01/19 | 1 | 28.0 | 0.0 | C23 | 2 | 57387 | 24/12/18 | NaN | 1 | 1 | 2381060 | 2 |
| **1** | 1 | 02/01/19 | 1 | 28.0 | 0.0 | C23 | 2 | 57387 | 24/12/18 | NaN | 1 | 1 | -665480 | 2 |
| **2** | 2 | 03/01/19 | 1 | 28.0 | 0.0 | C23 | 2 | 57387 | 24/12/18 | 03/11/19 | 1 | 1 | 0 | 2 |
| **3** | 3 | 11/01/20 | 2 | 31.0 | 0.0 | C7 | 2 | 67016 | 11/06/20 | NaN | 2 | 2 | 0 | 1 |
| **4** | 4 | 12/01/20 | 2 | 31.0 | 0.0 | C7 | 2 | 67016 | 11/06/20 | NaN | 2 | 2 | 0 | 1 |

In [5]:
```
# Checking the column names
df.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'MMM-YY', 'Driver_ID', 'Age', 'Gender', 'City',
       'Education_Level', 'Income', 'Dateofjoining', 'LastWorkingDate',
       'Joining Designation', 'Grade', 'Total Business Value',
       'Quarterly Rating'],
      dtype='object')
```

In [6]:
```
# Dropping the first column which is Unnecessary
df = df.drop("Unnamed: 0", axis = 1)
```

# Checking the Structure and Characteristics of Data

In [7]:
```
# Checking the Shape of the data
df.shape
```

Out[7]:
```
(19104, 13)
```

In [8]:
```
# Checking the Info of the Data
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  object 
 1   Driver_ID             19104 non-null  int64  
 2   Age                   19043 non-null  float64
 3   Gender                19052 non-null  float64
 4   City                  19104 non-null  object 
 5   Education_Level       19104 non-null  int64  
 6   Income                19104 non-null  int64  
 7   Dateofjoining         19104 non-null  object 
 8   LastWorkingDate       1616 non-null   object 
 9   Joining Designation   19104 non-null  int64  
 10  Grade                 19104 non-null  int64  
 11  Total Business Value  19104 non-null  int64  
 12  Quarterly Rating      19104 non-null  int64  
dtypes: float64(2), int64(7), object(4)
memory usage: 1.9+ MB
```

In [9]:
```
# Converting the datatypes of below mentioned columns to datetime type
dt_tm_cols = ['MMM-YY','Dateofjoining', 'LastWorkingDate']
for i in dt_tm_cols:
    df[i] = pd.to_datetime(df[i])
```

In [10]:
```
# Again Checking the info to verify if the data types have been correctly changed or not
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
```

In [11]:
```
# Checking the maximum and minimum datetime available in the dataset.
print(f"First Reporting Date and Time: {df['MMM-YY'].min()}, Last Reporting Date and time: {df['MMM-YY'].max()}")
```

```
First Reporting Date and Time: 2019-01-01 00:00:00, Last Reporting Date and time: 2020-12-01 00:00:00
```

In [12]:
```
# Checking the number of unique "Driver_ID"
print(f"Number of Unique 'Driver_ID' is: {df['Driver_ID'].nunique()}")
```

```
Number of Unique 'Driver_ID' is: 2381
```

In [13]:
```
the maximum and minimum age of drivers available in the dataset.
inimum Age of Driver available in dataset: {df['Age'].min()}, Maximum Age of Driver available in dataset: {df['Age'].max()
```

```
Minimum Age of Driver available in dataset: 21.0, Maximum Age of Driver available in dataset: 58.0
```

In [14]:
```
# Checking the percentage values of various "Gender" (Gender: 0 - Male, 1 - Female)
100 *(df['Gender'].value_counts()/len(df))
```

Out[14]:
```
0.0    57.966918
1.0    41.760888
Name: Gender, dtype: float64
```

In [15]:
```
# Checking the total number of unique cities avaialable in the dataset
df['City'].nunique()
```

Out[15]:
```
29
```

In [16]:
```
# Checking the top 5 percentage values of various "City"
100 *(df['City'].value_counts()/len(df))[:5]
```

Out[16]:
```
C20    5.276382
C29    4.711055
C26    4.548786
C22    4.234715
C27    4.114322
Name: City, dtype: float64
```

In [17]:
```
# Checking the percentage values of various "Education_Level" (0: 10+, 1: 12+, 2: Graduate)
100 *(df['Education_Level'].value_counts()/len(df))
```

Out[17]:
```
1    35.929648
2    33.118719
0    30.951633
Name: Education_Level, dtype: float64
```

In [18]:
```
and minimum income of drivers available in the dataset.
of Driver available in dataset: {df['Income'].min()}, Maximum Income of Driver available in dataset: {df['Income'].max()
```

```
Minimum Income of Driver available in dataset: 10747, Maximum Income of Driver available in dataset: 188418
```

In [19]:
```
# Checking the top 5 percentage values of various "Joining Designation"
100 *(df['Joining Designation'].value_counts()/len(df))
```

Out[19]:
```
1    51.460427
2    31.171482
3    14.902638
4     1.784966
5     0.680486
Name: Joining Designation, dtype: float64
```

In [20]:
```
# Checking the top 5 percentage values of various "Grade"
100 *(df['Grade'].value_counts()/len(df))
```

Out[20]:
```
2    34.689070
1    27.229899
3    25.261725
4    11.222781
5     1.596524
Name: Grade, dtype: float64
```

## Observations/Insights from Initial Analysis

1. There are a total of 19104 rows and 13 features in the given dataset.
2. Total number of unique drivers are 2381.
3. The age of drivers ranges from 21 to 58 years.
4. Around 58 % of total records are males while 42 % are females.
5. There are a total of 29 unique cities available in the dataset.
6. Maximum data is for city "C20" while all other city records are approximately same.
7. Maximum records i.e. around 36 % data are for education level 10+ category.
8. Around 51 % drivers join at joining designation 1.
9. Maximum drivers i.e. around 35 % are from Grade 1.

## Exploratory Data Analysis

## Checking Null Values and Their Treatment

In [22]:
```
# Checking the Null values available in the dataset
(100 * df.isnull().sum())/len(df)
```

Out[22]:
```
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
```

As we can see here that the columns "Age" and "Gender" have some amount of null values. But the column "LastWorkingDate" has a lot of missing value. Before treating these null values, first we need to perform some feature creation and then aggregation as per the column "Driver_ID".

After that we will again check if we actually have any missing values. If so, then we will use "KNN Imputation" to treat the null values.

# Feature Engineering

Feature Creation and Deletion of Unnecessary Features

In [23]:
```
# Creating a new column where we have a value of 1 if there is increment in Quarterly Rating of driver or else a value o
df['QR_change'] = df.groupby('Driver_ID')['Quarterly Rating'].transform('last') - df.groupby('Driver_ID')['Quarterly Rat
df['Increment_in_QR'] = df['QR_change'].apply(lambda x:1 if x > 0 else 0)
# Creating a new column where we have a value of 1 if there is increment in monthly income of driver or else a value of
df['Income_change'] = df.groupby('Driver_ID')['Income'].transform('last') - df.groupby('Driver_ID')['Income'].transform(
df['Increment_in_Income'] = df['Income_change'].apply(lambda x:1 if x > 0 else 0)
# Creating the Target Variable (Value of 1 if the driver has left the company or else a value of 0)
df['Churn'] = df['LastWorkingDate'].apply(lambda x:0 if pd.isnull(x) else 1)
# Dropping of newly created unnecessary columns
df = df.drop(['QR_change','Income_change'], axis = 1)
```

In [24]:
```
# Creating a dictionary for aggregation as per 'Driver_ID'
dict_driver = {
'MMM-YY' : 'first',
'Age' : 'first',
'Gender' : 'first',
'City' : 'first',
'Education_Level' : 'first',
'Income' : 'last',
'Dateofjoining' : 'first',
'LastWorkingDate' : 'last',
'Joining Designation' : 'first',
'Grade' : 'first',
'Total Business Value' : 'sum',
'Quarterly Rating' : 'last',
'Increment_in_QR' : 'last',
'Increment_in_Income' : 'last',
'Churn' : 'last'
}
```

In [25]:
```
# Creating the new dataframe with aggregation as per "Driver_ID"
new_df = df.groupby('Driver_ID').agg(dict_driver).reset_index()
```

In [26]:
```
# Dropping the datetime type columns as we have captured its importance by creating other columns using their given info
new_df = new_df.drop(['Driver_ID','MMM-YY','Dateofjoining','LastWorkingDate'], axis = 1)
```

In [27]:
```
# Changing the datatype of "Age" and "Gender" columns from float to int datatype
new_df['Age'] = new_df['Age'].astype('int64')
new_df['Gender'] = new_df['Gender'].astype('int64')
```

In [28]:
```
# Checking the info of the new dataframe
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Age                   2381 non-null   int64
 1   Gender                2381 non-null   int64
 2   City                  2381 non-null   object
 3   Education_Level       2381 non-null   int64
 4   Income                2381 non-null   int64
 5   Joining Designation   2381 non-null   int64
 6   Grade                 2381 non-null   int64
 7   Total Business Value  2381 non-null   int64
 8   Quarterly Rating      2381 non-null   int64
 9   Increment_in_QR       2381 non-null   int64
 10  Increment_in_Income   2381 non-null   int64
 11  Churn                 2381 non-null   int64
dtypes: int64(11), object(1)
memory usage: 223.3+ KB
```

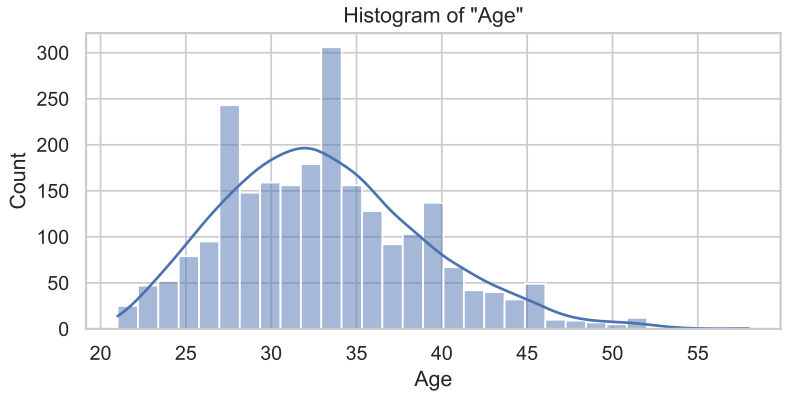
In [29]:
```
# Checking the head of the remaining dataframe
new_df.head()
```

Out[29]:

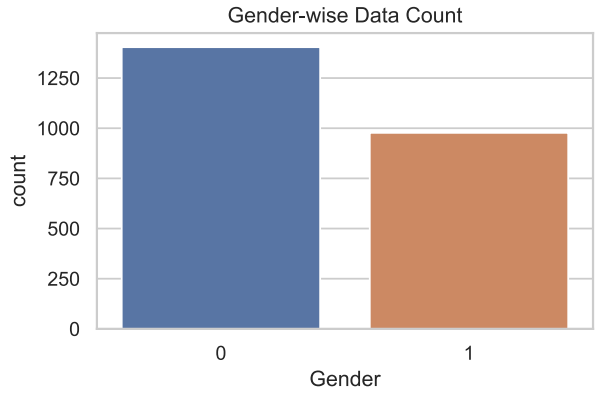
| | Age | Gender | City | Education_Level | Income | Joining Designation | Grade | Total Business Value | Quarterly Rating | Increment_in_QR | Increment_in_Income | Churn |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 28 | 0 | C23 | 2 | 57387 | 1 | 1 | 1715580 | 2 | 0 | 0 | 1 |
| **1** | 31 | 0 | C7 | 2 | 67016 | 2 | 2 | 0 | 1 | 0 | 0 | 0 |
| **2** | 43 | 0 | C13 | 2 | 65603 | 2 | 2 | 350000 | 1 | 0 | 0 | 1 |
| **3** | 29 | 0 | C9 | 0 | 46368 | 1 | 1 | 120360 | 1 | 0 | 0 | 1 |
| **4** | 31 | 1 | C11 | 1 | 78728 | 3 | 3 | 1265000 | 2 | 1 | 0 | 0 |

# Uni-Variate & Bi-Variate Visual Analysis

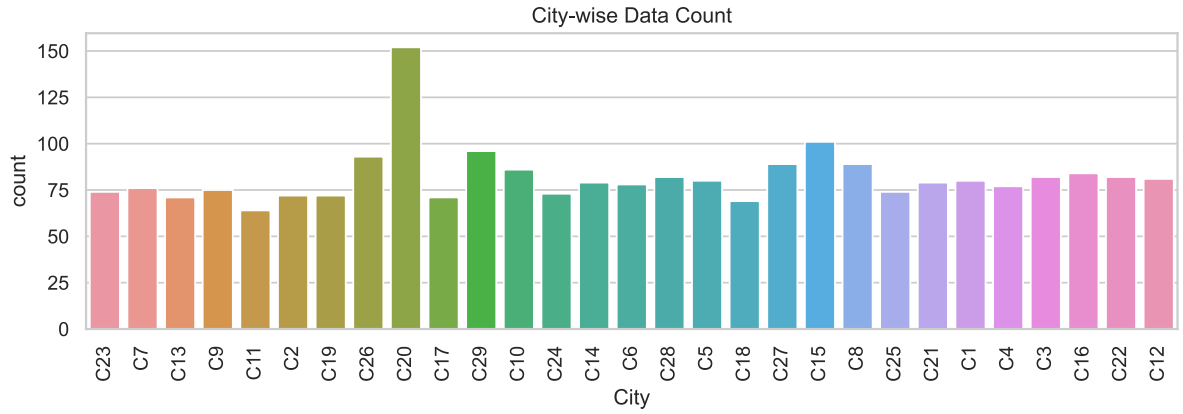
In [30]:
```
# Plot showing histogram of different 'Age' in dataset
fig, ax = plt.subplots(figsize = (7,3))
sns.histplot(data = new_df, x = 'Age', ax = ax, kde = True).set(title = 'Histogram of "Age"')
plt.show()
```



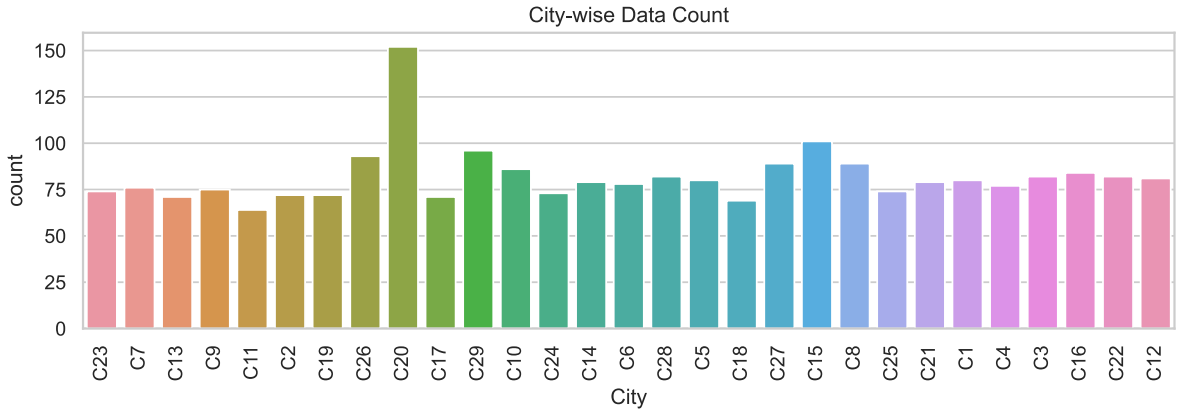

In [31]:
```
# Plot showing count of data of different "Gender"
fig, ax = plt.subplots(figsize = (5,3))
sns.countplot(data = new_df, x = 'Gender', ax = ax).set(title = 'Gender-wise Data Count')
plt.show()
```



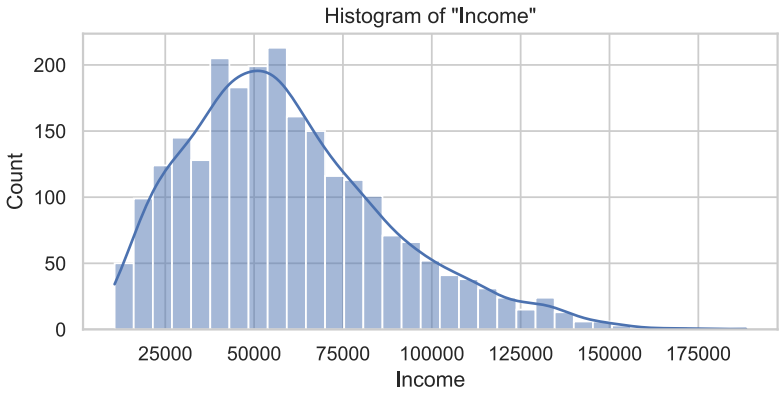

In [32]:
```
# Plot showing count of data of different "City"
fig, ax = plt.subplots(figsize = (11,3))
sns.countplot(data = new_df, x = 'City', ax = ax).set(title = 'City-wise Data Count')
plt.xticks(rotation = 90)
plt.show()
```



In [33]:
```
# Plot showing count of data of different "City"
fig, ax = plt.subplots(figsize = (11,3))
sns.countplot(data = new_df, x = 'City', ax = ax).set(title = 'City-wise Data Count')
plt.xticks(rotation = 90)
plt.show()
```

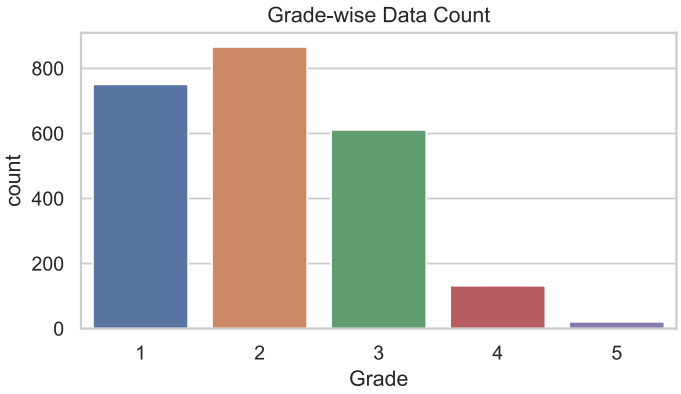



In [34]:
```
# Plot showing histogram of different 'Income' in dataset
fig, ax = plt.subplots(figsize = (7,3))
sns.histplot(data = new_df, x = 'Income', ax = ax, kde = True).set(title = 'Histogram of "Income"')
plt.show()
```

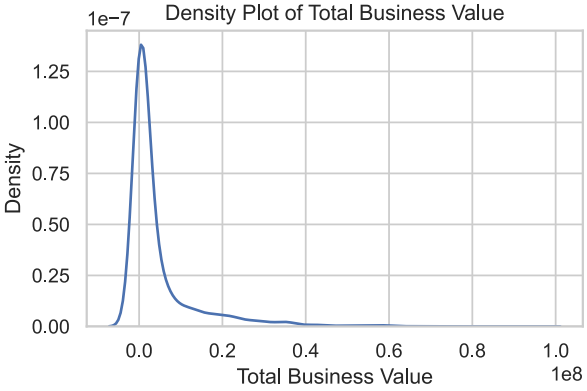



In [35]:
```
# Plot showing count of data of different "Joining Designation"
fig, ax = plt.subplots(figsize = (6,3))
sns.countplot(data = new_df, x = 'Joining Designation', ax = ax).set(title = 'Joining Designation-wise Data Count')
plt.show()
```

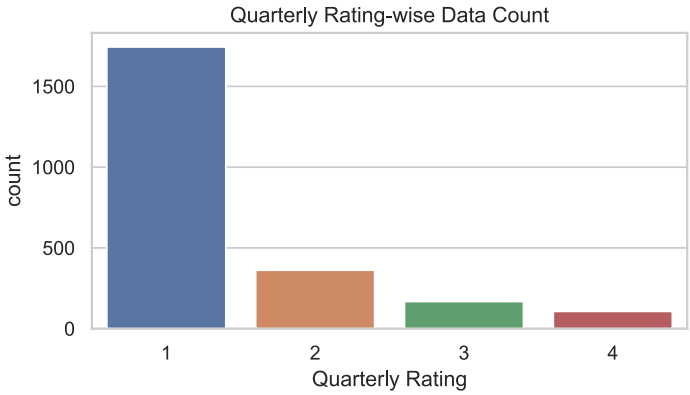
In [36]:
```
# Plot showing count of data of different "Grade"
fig, ax = plt.subplots(figsize = (6,3))
sns.countplot(data = new_df, x = 'Grade', ax = ax).set(title = 'Grade-wise Data Count')
plt.show()
```



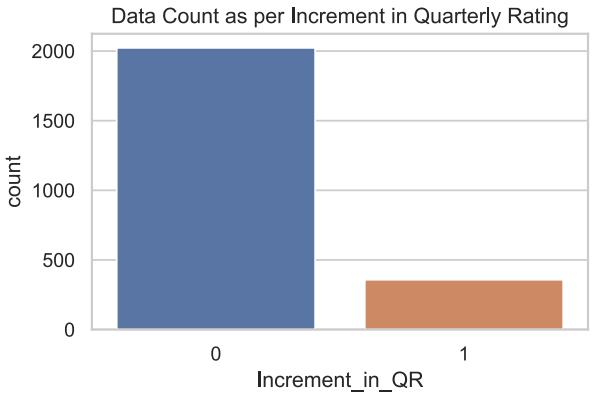

In [37]:
```
# Plot showing histogram of different 'Total Business Value' in dataset
fig, ax = plt.subplots(figsize = (5,3))
#sns.histplot(data = new_df, x = 'Total Business Value', ax = ax, kde = True).set(title = 'Histogram of "Total Business
sns.kdeplot(data = new_df, x = 'Total Business Value', ax = ax).set(title = 'Density Plot of Total Business Value')
plt.show()
```



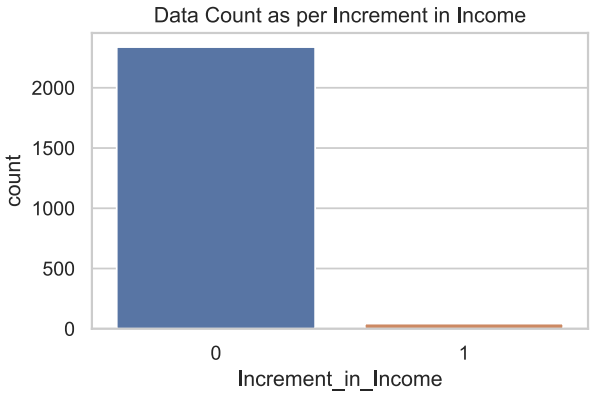

In [38]:
```
# Plot showing count of data of different "Quarterly Rating"
fig, ax = plt.subplots(figsize = (6,3))
sns.countplot(data = new_df, x = 'Quarterly Rating', ax = ax).set(title = 'Quarterly Rating-wise Data Count')
plt.show()
```



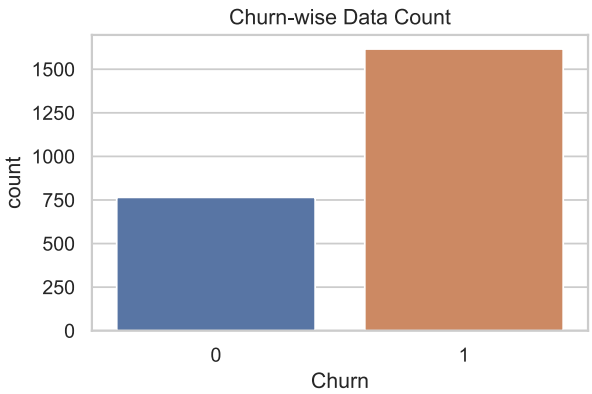
In [39]:
```
# Plot showing count of data of different "Increment_in_QR"
fig, ax = plt.subplots(figsize = (5,3))
sns.countplot(data = new_df, x = 'Increment_in_QR', ax = ax).set(title = 'Data Count as per Increment in Quarterly Ratir
plt.show()
```



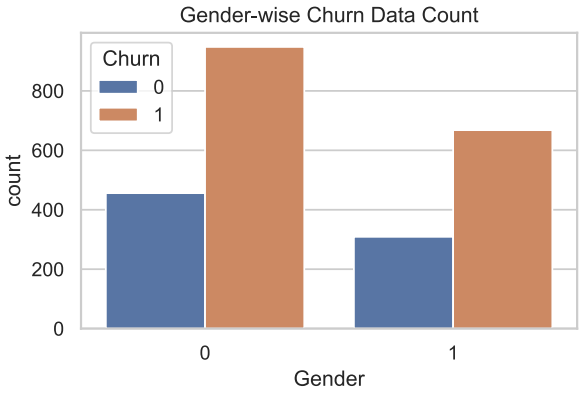

In [40]:
```
# Plot showing count of data of different "Increment_in_Income"
fig, ax = plt.subplots(figsize = (5,3))
sns.countplot(data = new_df, x = 'Increment_in_Income', ax = ax).set(title = 'Data Count as per Increment in Income')
plt.show()
```



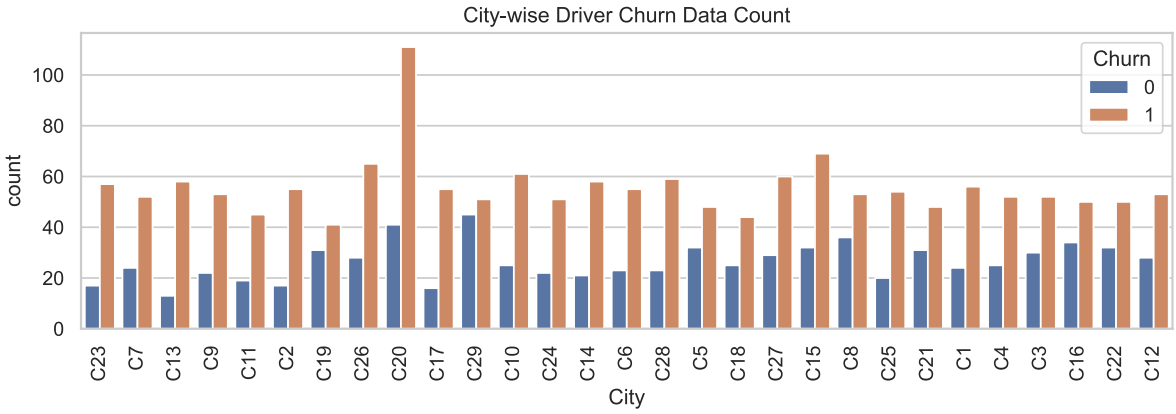

In [41]:
```
# Plot showing count of data of driver "Churn"
fig, ax = plt.subplots(figsize = (5,3))
sns.countplot(data = new_df, x = 'Churn', ax = ax).set(title = 'Churn-wise Data Count')
plt.show()
```



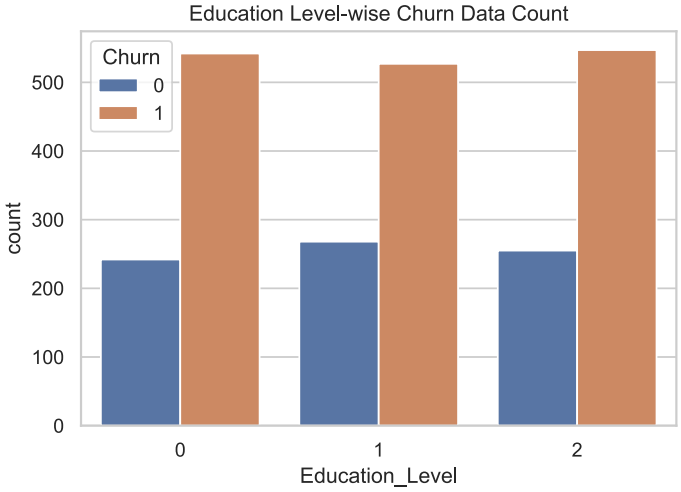
In [42]:
```
# Plot showing count of different "Gender" as per churn
fig, ax = plt.subplots(figsize = (5,3))
sns.countplot(data = new_df, x = 'Gender', hue = 'Churn', ax = ax).set(title = 'Gender-wise Churn Data Count')
plt.show()
```




In [43]:
```
# Plot showing count of driver churn for different "City"
fig, ax = plt.subplots(figsize = (11,3))
sns.countplot(data = new_df, x = 'City', hue = 'Churn', ax = ax).set(title = 'City-wise Driver Churn Data Count')
plt.xticks(rotation = 90)
plt.show()
```

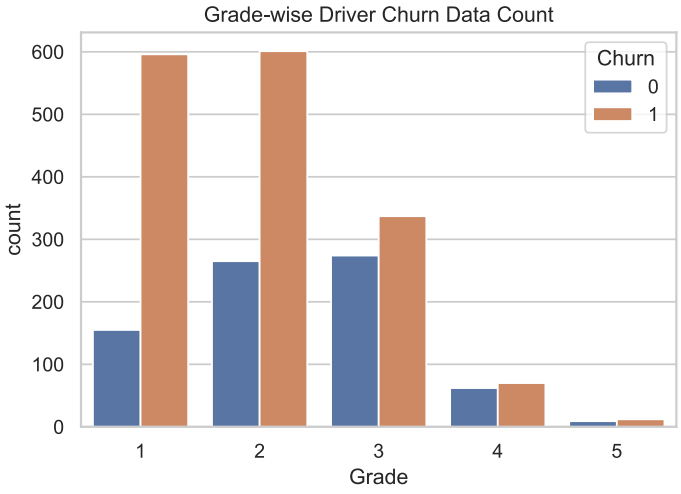



In [44]:
```
# Plot showing count of different "Education_Level" as per driver churn
fig, ax = plt.subplots(figsize = (6,4))
sns.countplot(data = new_df, x = 'Education_Level', hue = 'Churn', ax = ax).set(title = 'Education Level-wise Churn Data
plt.show()
```

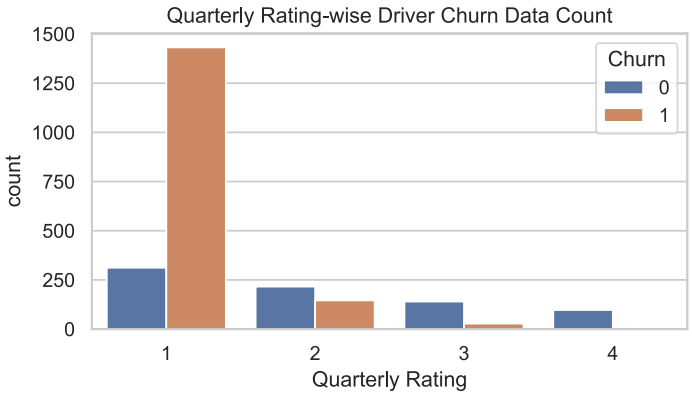


In [45]:
```
# Plot showing count of different "Joining Designation" as per driver churn
fig, ax = plt.subplots(figsize = (6,4))
sns.countplot(data = new_df, x = 'Joining Designation', hue = 'Churn', ax = ax).set(title = 'Joining Designation-wise Ch
plt.show()
```

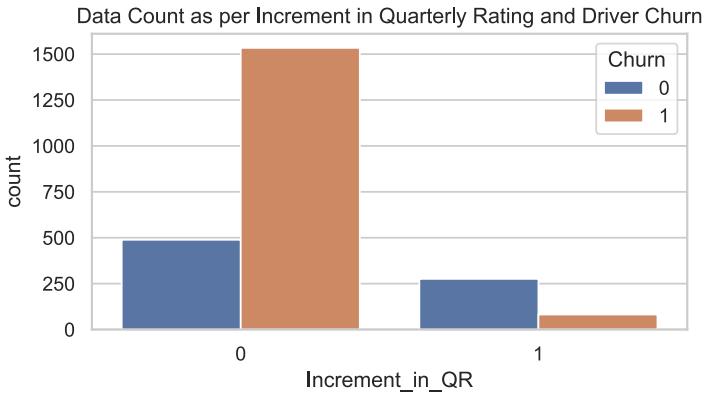

In [46]:
```
# Plot showing count of different "Grade" as per driver churn
fig, ax = plt.subplots(figsize = (6,4))
sns.countplot(data = new_df, x = 'Grade', hue = 'Churn', ax = ax).set(title = 'Grade-wise Driver Churn Data Count')
plt.show()
```



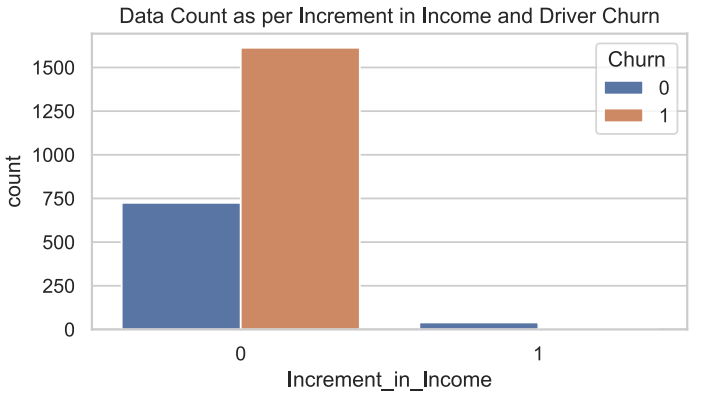

In [47]:
```
# Plot showing count of different "Quarterly Rating" as per Driver Churn
fig, ax = plt.subplots(figsize = (6,3))
sns.countplot(data = new_df, x = 'Quarterly Rating', hue = 'Churn', ax = ax).set(title = 'Quarterly Rating-wise Driver C
plt.show()
```



In [48]:
```
# Plot showing count of "Increment_in_QR" as per Driver Churn
fig, ax = plt.subplots(figsize = (6,3))
sns.countplot(data = new_df, x = 'Increment_in_QR', hue = 'Churn', ax = ax).set(title = 'Data Count as per Increment in
plt.show()
```

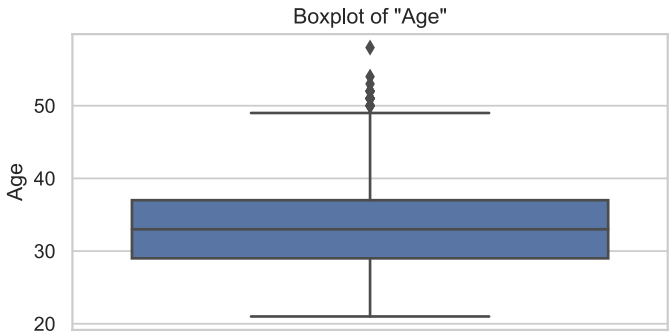


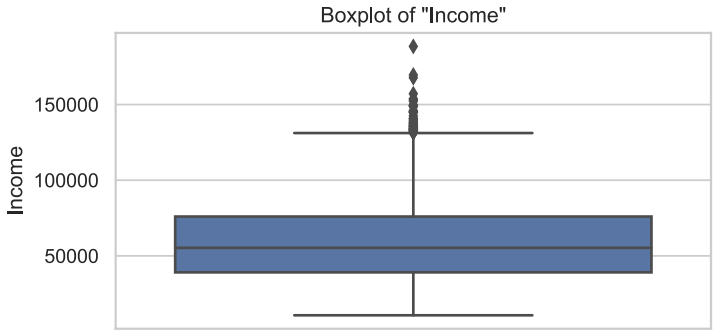

In [49]:
```
# Plot showing count of "Increment_in_Income" as per driver churn
fig, ax = plt.subplots(figsize = (6,3))
sns.countplot(data = new_df, x = 'Increment_in_Income', hue = 'Churn', ax = ax).set(title = 'Data Count as per Increment
plt.show()
```

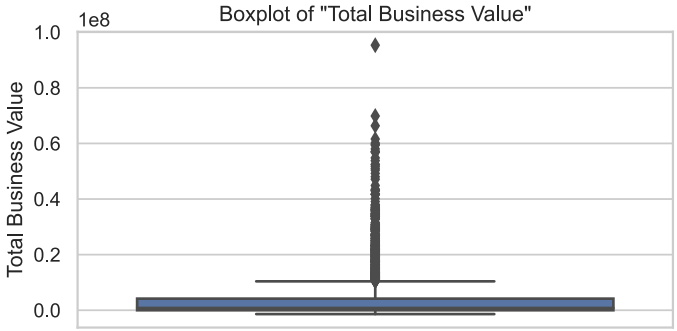



## Outlier Detection using Boxplots

In [50]:
```
# Boxplots of 'Age', 'Income' and 'Total Business Value'
plt.figure(figsize = (6,3))
sns.boxplot(data = new_df, y = 'Age').set(title = 'Boxplot of "Age"')
plt.show()
plt.figure(figsize = (6,3))
sns.boxplot(data = new_df, y = 'Income').set(title = 'Boxplot of "Income"')
plt.show()
plt.figure(figsize = (6,3))
sns.boxplot(data = new_df, y = 'Total Business Value').set(title = 'Boxplot of "Total Business Value"')
plt.show()
```

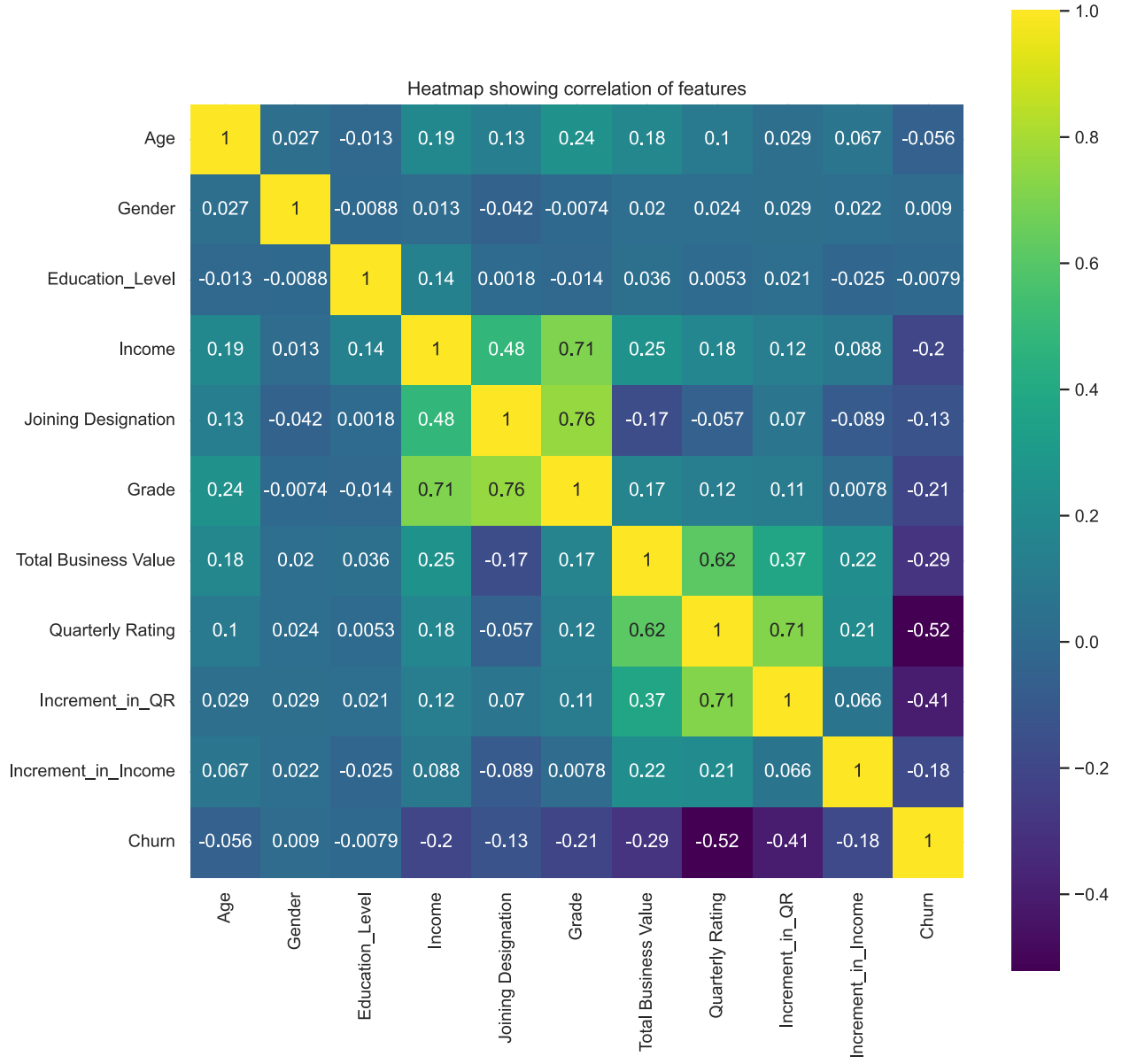


In [51]:
```
# Checking the correlation between different features through Heatmap
fig, ax = plt.subplots(figsize = (11,11))
sns.heatmap(new_df.corr(method = 'spearman'), square = True, annot = True, cmap = 'viridis').set(title = 'Heatmap showir
plt.show()
```

```
C:\Users\india\AppData\Local\Temp\ipykernel_17104\2043829239.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(new_df.corr(method = 'spearman'), square = True, annot = True, cmap = 'viridis').set(title = 'Heatmap sh
owing correlation of features')
```




## Description of Numerical and Categorical Features

In [52]:
```
# Checking the statistical description for numerical features
new_df.describe()
```

Out[52]:

| | Age | Gender | Education_Level | Income | Joining Designation | Grade | Total Business Value | Quarterly Rating | Increment_in_QR | Increment_i |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 2381.000000 | 2381.000000 | 2381.00000 | 2381.000000 | 2381.000000 | 2381.000000 | 2.381000e+03 | 2381.000000 | 2381.000000 | 238 |
| mean | 33.090718 | 0.410332 | 1.00756 | 59334.157077 | 1.820244 | 2.078538 | 4.586742e+06 | 1.427971 | 0.150357 | |
| std | 5.840686 | 0.491997 | 0.81629 | 28383.666384 | 0.841433 | 0.931321 | 9.127115e+06 | 0.809839 | 0.357496 | |
| min | 21.000000 | 0.000000 | 0.00000 | 10747.000000 | 1.000000 | 1.000000 | -1.385530e+06 | 1.000000 | 0.000000 | |
| 25% | 29.000000 | 0.000000 | 0.00000 | 39104.000000 | 1.000000 | 1.000000 | 0.000000e+00 | 1.000000 | 0.000000 | |
| 50% | 33.000000 | 0.000000 | 1.00000 | 55315.000000 | 2.000000 | 2.000000 | 8.176800e+05 | 1.000000 | 0.000000 | |
| 75% | 37.000000 | 1.000000 | 2.00000 | 75986.000000 | 2.000000 | 3.000000 | 4.173650e+06 | 2.000000 | 0.000000 | |
| max | 58.000000 | 1.000000 | 2.00000 | 188418.000000 | 5.000000 | 5.000000 | 9.533106e+07 | 4.000000 | 1.000000 | |

# Observations/Insights from Visual Analysis

1. Majority of the drivers in the dataset have their age around 34 years.
2. There is minor imbalance in the count of male and female drivers, with male drivers being in majority.
3. Majority drivers are for city "C20" while all other cities have more or less similar number of drivers.
4. The count of drivers are almost evenly distributed among the three categories of education level.
5. The histogram of income of drivers is a bit right-skewed, thereby indicating that very few drivers are having very high income.
6. Majority of the drivers join at designation 1, followed by 2, 3, 4 and 5.
7. Maximum drivers are in grade 2, followed by grade 1, grade 3, grade 4 and grade 5.
8. Huge number of drivers are having a quarterly rating of 1, while few of them also having grade 2, 3 and 4.
9. There is huge imbalance in the data depicting increment in quarterly rating, with majority of drivers not having any increment in quarterly rating.
10. Again, there is a huge imbalace in the data depicting increment in income of drivers, with very few of them who have got their income increased.
11. The data for driver churn is also imbalanced, with majority of the drivers leaving the company.
12. The churn data for male and female drivers are very similar as per imbalance in the gender data.
13. Most Male drivers are from City C20, while most female drivers are from City C29. Minimum number of female drivers are from City C13.
14. The churn data for drivers of all education levels are very similar.
15. The churn data for drivers having Joining Designation 1 is maximum, followed by those having Joining Designation 2, 3, 4 and 5. This data also follows similar pattern for drivers who are not leaving the company.
16. Majority of the drivers who leave the company are from Grade 1 and 2 while the drivers who do not leave the company are from Grade 3, followed by Grade 2.
17. The quaretly rating for the drivers who are leaving the company is 1 while from among those who are not leaving the company, majority have quarterly rating 1, followed by 2, 3 and 4.
18. Driver churn is high among those who have got no increment in their quarterly rating.
19. Attritiion is high for those drivers who have got no increment in income.
20. There are a few outliers in features "Age" and "Income" while it is maximum for feature "Total Business Value".
21. None of the features have a very high positive correlation with each other.
22. The mean and median value of "Age" is approximately similar i.e. 33 years.
23. Similarly, mean and median income of drivers is somewhat close indicating there are not much outliers in this feature.

# Encoding

In [54]:
```
# Performing One-Hot Encoding of features
city_dummy = pd.get_dummies(new_df["City"], prefix="City")
city_dummy = city_dummy[city_dummy.columns[1:]]
new_df = pd.concat([new_df, city_dummy], axis=1)
education_level_dummy = pd.get_dummies(new_df["Education_Level"], prefix="Education_Level")
education_level_dummy = education_level_dummy[education_level_dummy.columns[1:]]
new_df = pd.concat([new_df, education_level_dummy], axis=1)

joining_designation_dummy = pd.get_dummies(new_df["Joining Designation"], prefix="Joining Designation")
joining_designation_dummy = joining_designation_dummy[joining_designation_dummy.columns[1:]]
new_df = pd.concat([new_df, joining_designation_dummy], axis=1)
grade_dummy = pd.get_dummies(new_df["Grade"], prefix="Grade")
grade_dummy = grade_dummy[grade_dummy.columns[1:]]
new_df = pd.concat([new_df, grade_dummy], axis=1)
quarterly_rating_dummy = pd.get_dummies(new_df["Quarterly Rating"], prefix="Quarterly Rating")
quarterly_rating_dummy = quarterly_rating_dummy[quarterly_rating_dummy.columns[1:]]
new_df = pd.concat([new_df, quarterly_rating_dummy], axis=1)
```

In [55]:
```
# Dropping the original columns that were encoded
new_df = new_df.drop(['City','Education_Level','Joining Designation','Grade','Quarterly Rating'], axis = 1)
```

In [56]:
```
# Checking the shape of the data
new_df.shape
```

Out[56]: (2381, 48)

## Train and Test Splitting of Data

In [57]:
```
# Separating the target variable
X = new_df.drop('Churn',axis=1)
y = new_df['Churn']
```

In [58]:
```
# Splitting the dataset into training data and testing data in the ratio of 80:20 respectively
XTrain, XTest, YTrain, YTest = train_test_split(X, y, test_size = 0.20, stratify = y, random_state = 42)
```

In [59]:
```
print("Shape of Training Data : ", XTrain.shape)
print("Shape of Testing Data : ", XTest.shape)
print("Shape of Training Target Data : ", YTrain.shape)
print("Shape of Testing Target Data : ", YTest.shape)
```

```
Shape of Training Data :  (1904, 47)
Shape of Testing Data :  (477, 47)
Shape of Training Target Data :  (1904,)
Shape of Testing Target Data :  (477,)
```

In [60]:
```
# Keeping list of Original column headers
original_features = list(XTrain.columns)
```

## Standardization of Features using StandardScaler()

We will perform scaling of the data using StandardScaler(). This removes the mean and scales each feature to unit variance.

In [61]:
```
# Instantiating the Standard Scaler and fitting on training and testing data
scaler = StandardScaler()
XTrain = scaler.fit_transform(XTrain)
XTest = scaler.transform(XTest)
```

## Class Imbalance Treatment by Oversampling using SMOTE

There is a huge class imbalance in the driver churn data. To tackle this issue, we will do oversampling using SMOTE.

In [62]:
```
sm = SMOTE(random_state=42)
XTrain_sm, YTrain_sm = sm.fit_resample(XTrain,YTrain.ravel())
print('After OverSampling, XTrain shape: {}'.format(XTrain_sm.shape))
print('After OverSampling, YTrain shape: {} \n'.format(YTrain_sm.shape))
print("After OverSampling, counts of label '1': {}".format(sum(YTrain_sm == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(YTrain_sm == 0)))
```

```
After OverSampling, XTrain shape: (2584, 47)
After OverSampling, YTrain shape: (2584,)

After OverSampling, counts of label '1': 1292
After OverSampling, counts of label '0': 1292
```

## Model Evaluation Metrics

To assess the performance of the models, we will use metrics like Classification Report, Confusion Matrix, ROC-AUC Curve, Precision-Recall Curve. For this, different functions have been defined below which would plot confusion matrix, classification report, ROC Curve and Precision-Recall Curve.

In [63]:
```
# Defining a function that will plot the Confusion matrix and show the Train & Test accuracy scores
def conf_matrix_plot(XTrain, YTrain, XTest, YTest, Classifier, YPred, Classifier_name):
#fig, ax = plt.subplots(figsize = (5,4))
    cm = confusion_matrix(YTest, YPred, labels = Classifier.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = Classifier.classes_)
    #disp.set(title = f'{Classifier_name} - Confusion Matrix')
    disp.plot()
    plt.show()
    print(f'Train -> Accuracy Score : {Classifier.score(XTrain, YTrain)}')
    print(f'Test -> Accuracy Score : {accuracy_score(YTest, YPred)}')
    return print("")
```

In [64]:
```
# Defining a function that will plot the ROC-Curve and show the AUC-Score
def roc_auc_curve(XTest, YTest, YPred_probabilities, Classifier):
    YPred_prob = YPred_probabilities[:,1]
    fpr, tpr, thresholds = roc_curve(YTest, YPred_prob)
    plt.plot([0,1],[0,1],'k--')
    plt.plot(fpr,tpr, label = f'{Classifier}')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f'{Classifier} - ROC Curve')
    plt.show()
    return print(f'AUC Score: {roc_auc_score(YTest, YPred_prob)}')
```

In [65]:
```
# Defining a function that will plot the Precision-Recall Curve and show the F1 and AUC Scores
def precision_recall(XTest, YTest, YPred, YPred_probabilities, Classifier) :
    YPred_prob = YPred_probabilities[:,1]
    precision, recall, thresholds = precision_recall_curve(YTest, YPred_prob)
    plt.plot(recall, precision, label = f'{Classifier}')
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title(f'{Classifier} - Precision-Recall Curve')
    plt.show()
    f1, auc_score = f1_score(YTest, YPred), auc(recall, precision)
    precision, recall = precision_score(YTest, YPred), recall_score(YTest, YPred)
    return print(f'Precision: {precision} \nRecall: {recall} \nf1 Score: {f1} \nAUC Score: {auc_score}')
```

# Training & Predicting the Models

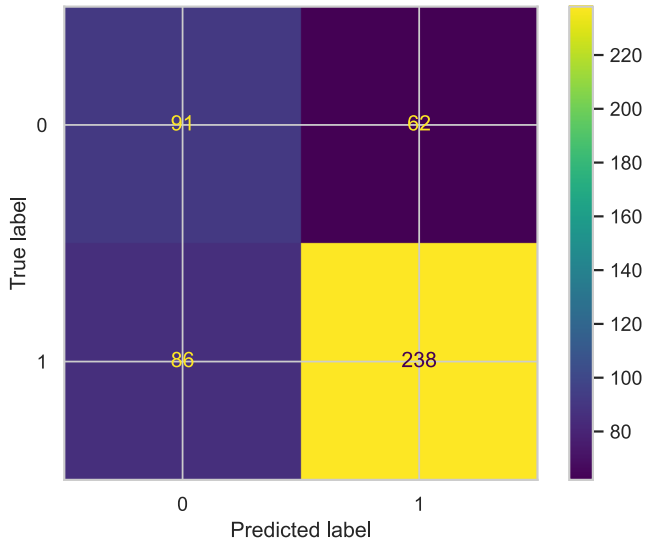
K Nearest Neighbours

In [66]:
```
# Instantiating the KNN Classifier
knn = KNeighborsClassifier()
# Fitting the instantiated Classifier on the training data
knn.fit(XTrain_sm, YTrain_sm)
```

Out[66]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

In [67]:
```
# Predicting the Classes and Probabilities for KNN Model
YPred_knn = knn.predict(XTest)
YPred_knn_prob = knn.predict_proba(XTest)
```

In [68]:
```
# Printing Confusion Matrix for KNN Model
conf_matrix_plot(XTrain_sm, YTrain_sm, XTest, YTest, knn, YPred_knn, 'KNN')
```

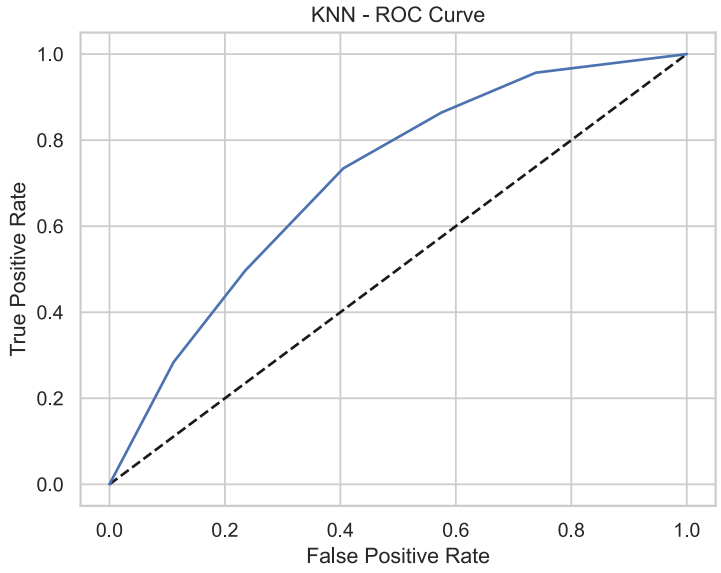



```
Train -> Accuracy Score : 0.8459752321981424
Test -> Accuracy Score : 0.689727463312369
```

In [69]:
```
# Printing the Classification Report for KNN Model
print(classification_report(YTest, YPred_knn))
```

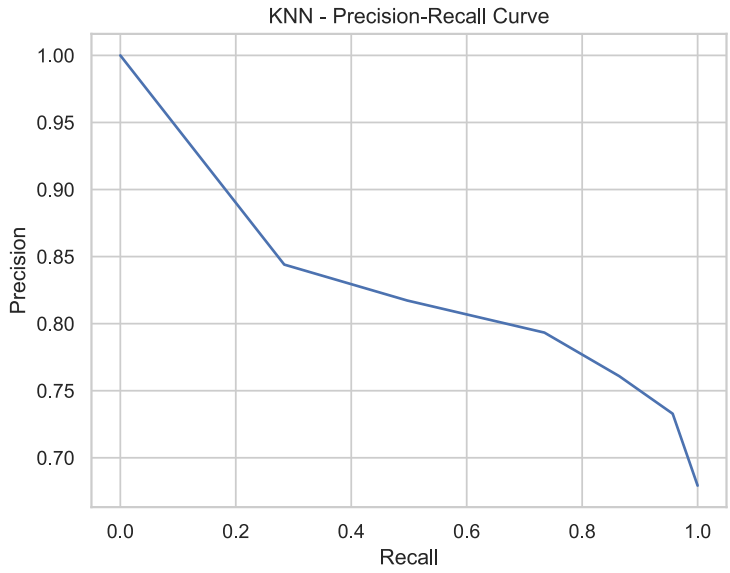
```
              precision    recall  f1-score   support

           0       0.51      0.59      0.55       153
           1       0.79      0.73      0.76       324

    accuracy                           0.69       477
   macro avg       0.65      0.66      0.66       477
weighted avg       0.70      0.69      0.70       477
```

In [70]:
```
# Printing the ROC Curve for KNN Model
roc_auc_curve(XTest, YTest, YPred_knn_prob, 'KNN')
```




```
AUC Score: 0.7093016218833212
```

In [71]:
```
# Printing the precision-recall curve for KNN Model
precision_recall(XTest, YTest, YPred_knn, YPred_knn_prob, 'KNN')
```




```
Precision: 0.7933333333333333
Recall: 0.7345679012345679
f1 Score: 0.7628205128205129
AUC Score: 0.8304849960184246
```

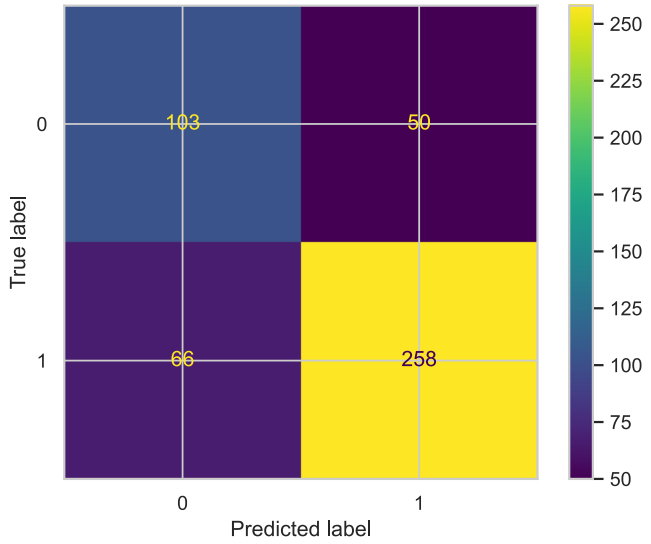
# Decision Tree

In [72]:
```
# Instantiating the Decision Tree Classifier
dtc = DecisionTreeClassifier()
# Fitting the instantiated Classifier on the training data
dtc.fit(XTrain_sm, YTrain_sm)
```

Out[72]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [73]:
```
# Predicting the Classes and Probabilities for Decision Tree Model
YPred_dtc = dtc.predict(XTest)
YPred_dtc_prob = dtc.predict_proba(XTest)
```

In [74]:
```
# Printing Confusion Matrix for Decision Tree Model
conf_matrix_plot(XTrain_sm, YTrain_sm, XTest, YTest, dtc, YPred_dtc, 'Decision Tree')
```

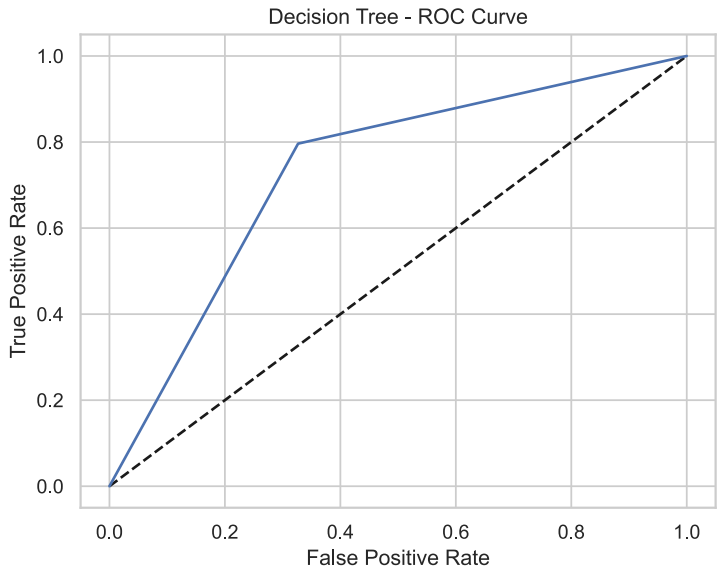



```
Train -> Accuracy Score : 1.0
Test -> Accuracy Score : 0.7568134171907757
```

In [75]:
```
# Printing the Classification Report for Decision Tree Model
print(classification_report(YTest, YPred_dtc))
```

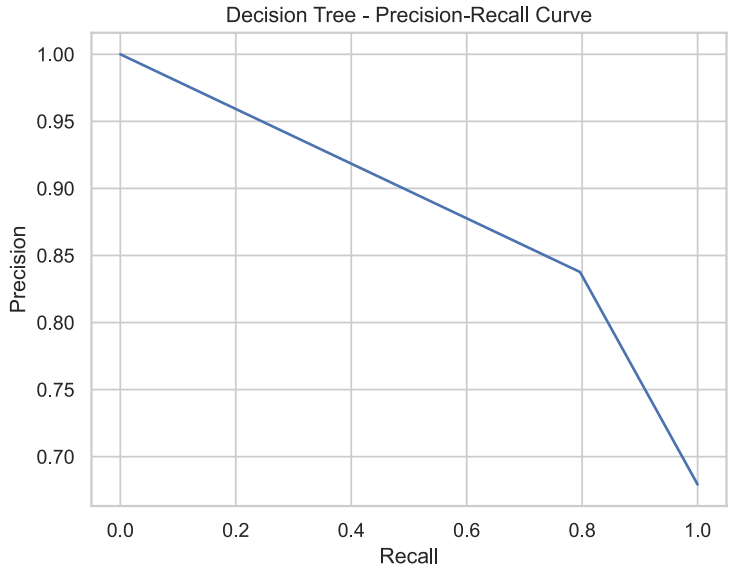
```
              precision    recall  f1-score   support

           0       0.61      0.67      0.64       153
           1       0.84      0.80      0.82       324

    accuracy                           0.76       477
   macro avg       0.72      0.73      0.73       477
weighted avg       0.76      0.76      0.76       477
```

In [76]:
```
# Printing the ROC Curve for Decision Tree Model
roc_auc_curve(XTest, YTest, YPred_dtc_prob, 'Decision Tree')
```




```
AUC Score: 0.7347494553376908
```

In [77]:
```
# Printing the precision-recall curve for Decision Tree Model
precision_recall(XTest, YTest, YPred_dtc, YPred_dtc_prob, 'Decision Tree')
```




```
Precision: 0.8376623376623377
Recall: 0.7962962962962963
f1 Score: 0.8164556962025317
AUC Score: 0.8861617069164238
```

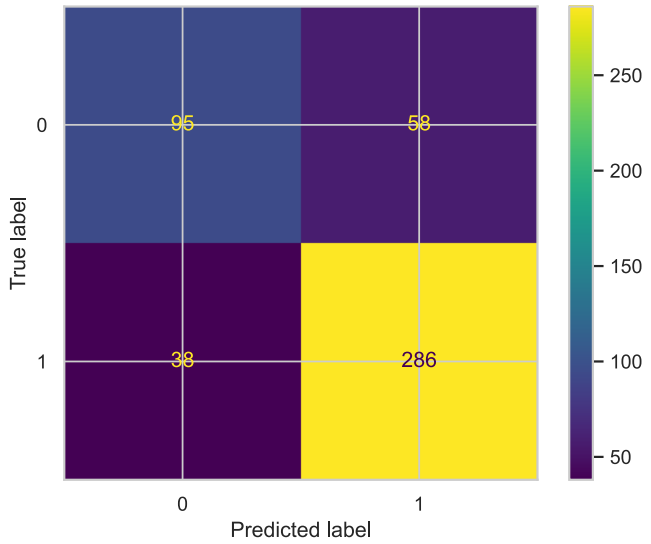
# Random Forest

In [78]:
```
# Instantiating the Random Forest Classifier
rfc = RandomForestClassifier()
# Fitting the instantiated Classifier on the training data
rfc.fit(XTrain_sm, YTrain_sm)
```

Out[78]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [79]:
```
# Predicting the Classes and Probabilities for Random Forest Model
YPred_rfc = rfc.predict(XTest)
YPred_rfc_prob = rfc.predict_proba(XTest)
```

In [80]:
```
# Printing Confusion Matrix for Random Forest Model
conf_matrix_plot(XTrain_sm, YTrain_sm, XTest, YTest, rfc, YPred_rfc, 'Random Forest')
```

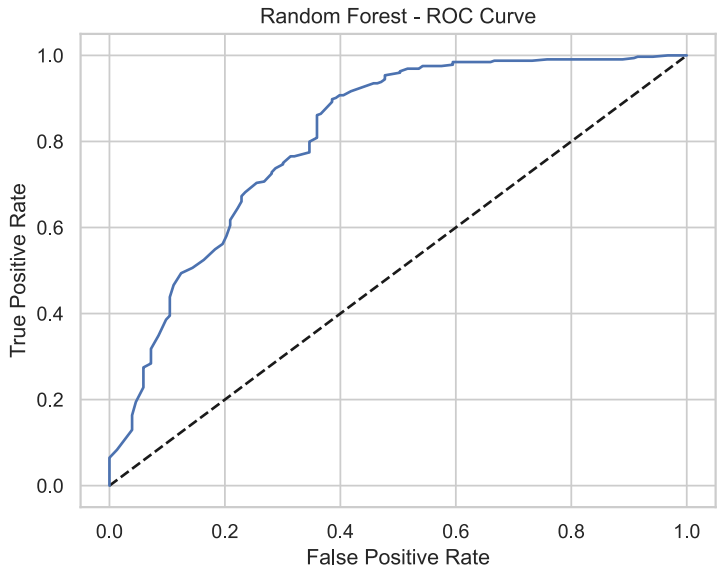



```
Train -> Accuracy Score : 1.0
Test -> Accuracy Score : 0.7987421383647799
```

In [81]:
```
# Printing the Classification Report for Random Forest Model
print(classification_report(YTest, YPred_rfc))
```

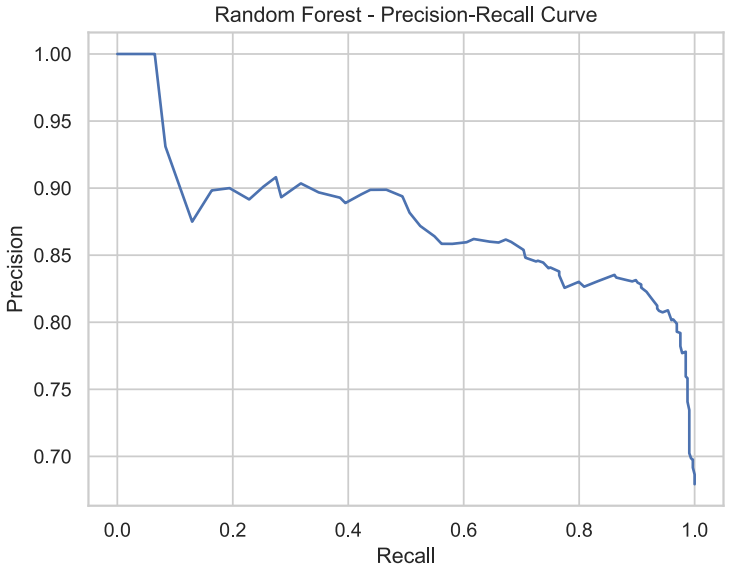
```
              precision    recall  f1-score   support

           0       0.71      0.62      0.66       153
           1       0.83      0.88      0.86       324

    accuracy                           0.80       477
   macro avg       0.77      0.75      0.76       477
weighted avg       0.79      0.80      0.79       477
```

In [82]:
```
# Printing the ROC Curve for Random Forest Model
roc_auc_curve(XTest, YTest, YPred_rfc_prob, 'Random Forest')
```




```
AUC Score: 0.8094892277898813
```

In [83]:
```
# Printing the precision-recall curve for Random Forest Model
precision_recall(XTest, YTest, YPred_rfc, YPred_rfc_prob, 'Random Forest')
```



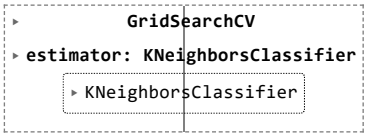

```
Precision: 0.8313953488372093
Recall: 0.8827160493827161
f1 Score: 0.8562874251497007
AUC Score: 0.8755057222259814
```

# Hyperparameter Tuning and Model Improvement

K Nearest Neighbours (Optimized)

In [84]:
```
# Defining parameters to explore for Grid Search for tuning of KNN Model
params2explore_knn = {'n_neighbors' : np.arange(1,20)}
# Instantiating the KNN Classifier
knn = KNeighborsClassifier()
# Hyperparameter tuning to find the best possible parameters for KNN Model
knn_tuned = GridSearchCV(knn, params2explore_knn, cv = 5)
# Fitting the instantiated classifier on the training data
knn_tuned.fit(XTrain_sm, YTrain_sm)
```

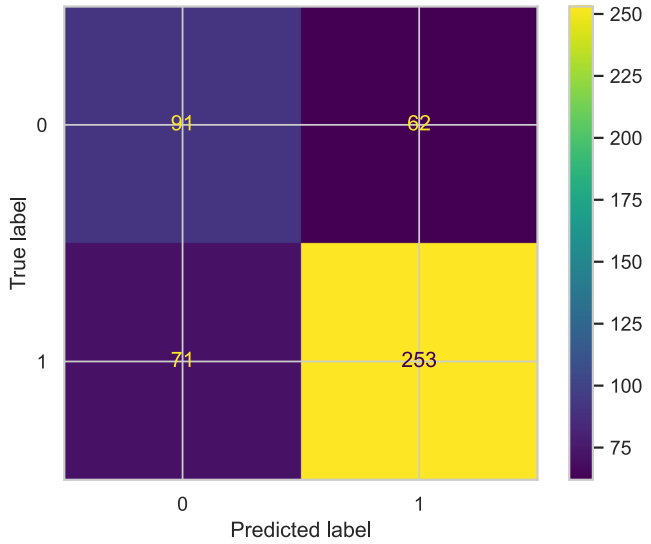
Out[84]:




In [85]:
```
# Predicting the Classes and Probabilities for the Tuned KNN Model
YPred_knn_tuned = knn_tuned.predict(XTest)
YPred_knn_tuned_prob = knn_tuned.predict_proba(XTest)
# Printing the best possible number of Neighbors
print(f'Best possible number of Neighbors : {knn_tuned.best_params_}')
```

```
Best possible number of Neighbors : {'n_neighbors': 1}
```

In [86]:
```
# Printing Confusion Matrix for the Tuned KNN Model
conf_matrix_plot(XTrain_sm, YTrain_sm, XTest, YTest, knn_tuned, YPred_knn_tuned, 'Tuned KNN')
```

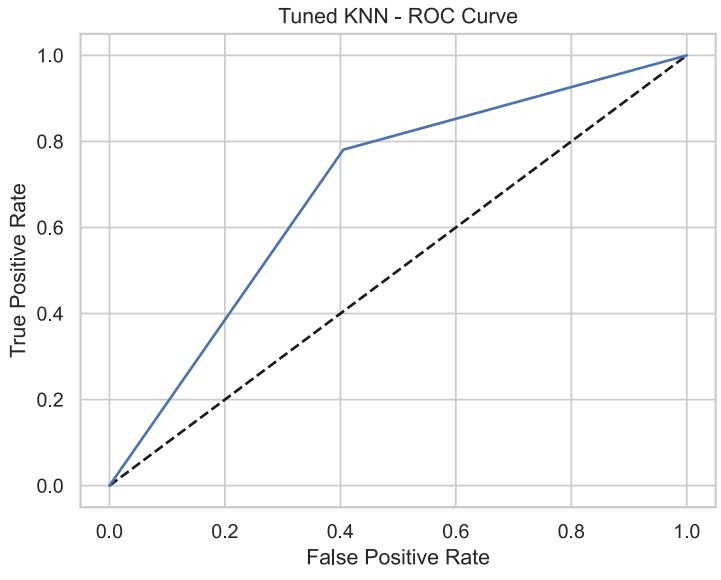



```
Train -> Accuracy Score : 1.0
Test -> Accuracy Score : 0.7211740041928721
```

In [87]:
```
# Printing the Classification Report for the Tuned KNN Model
print(classification_report(YTest, YPred_knn_tuned))
```

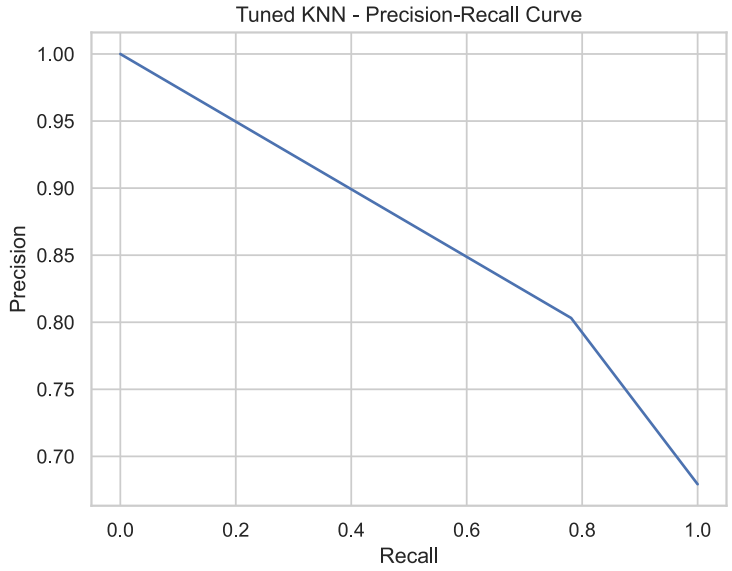
```
              precision    recall  f1-score   support

           0       0.56      0.59      0.58       153
           1       0.80      0.78      0.79       324

    accuracy                           0.72       477
   macro avg       0.68      0.69      0.68       477
weighted avg       0.73      0.72      0.72       477
```

In [88]:
```
# Printing the ROC Curve for the Tuned KNN Model
roc_auc_curve(XTest, YTest, YPred_knn_tuned_prob, 'Tuned KNN')
```




```
AUC Score: 0.6878177196804648
```

In [89]:
```
# Printing the precision-recall curve for the Tuned KNN Model
precision_recall(XTest, YTest, YPred_knn_tuned, YPred_knn_tuned_prob, 'Tuned KNN')
```

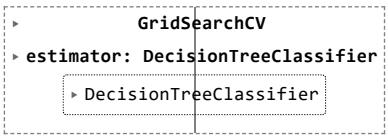



```
Precision: 0.8031746031746032
Recall: 0.7808641975308642
f1 Score: 0.7918622848200314
AUC Score: 0.8664428804365911
```

## Decision Tree (Optimized)

In [90]:
```
# Defining parameters to explore for Grid Search for Tuning of Decision Tree
params2explore_dtc = {'max_depth' : np.arange(1,21),
'min_samples_leaf' : np.arange(1,21),
'criterion' : ["gini","entropy"]
}
# Instantiating the Decision Tree Classifier
dtc = DecisionTreeClassifier()
# Hyperparameter tuning to find the best possible parameters for Decision Tree Model
dtc_tuned = GridSearchCV(dtc, params2explore_dtc, cv = 5)
# Fitting the instantiated classifier on the training data
dtc_tuned.fit(XTrain_sm, YTrain_sm)
```

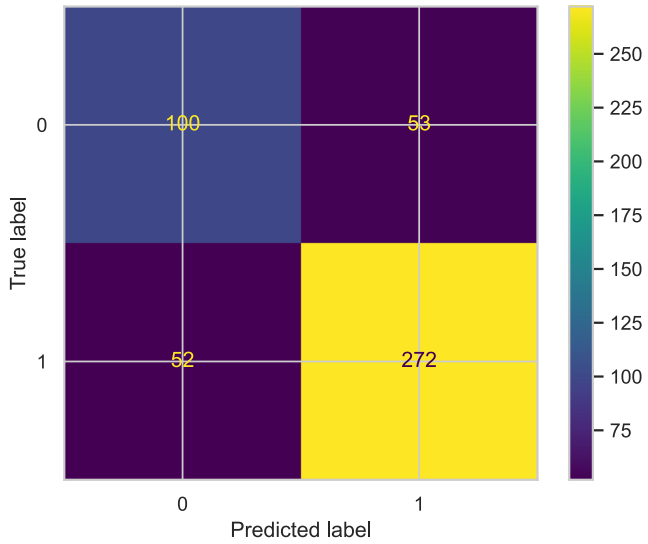
Out[90]:




In [91]:
```
# Predicting the Classes and Probabilities for the Tuned Decision Tree Model
YPred_dtc_tuned = dtc_tuned.predict(XTest)
YPred_dtc_tuned_prob = dtc_tuned.predict_proba(XTest)
# Printing the best possible hyperparameters
print(f'Best possible hyperparameters for the tuned Decision Tree : {dtc_tuned.best_params_}')
```

```
Best possible hyperparameters for the tuned Decision Tree : {'criterion': 'entropy', 'max_depth': 8, 'min_samples_lea
f': 10}
```

In [92]:
```
# Printing Confusion Matrix for the Tuned Decision Tree Model
conf_matrix_plot(XTrain_sm, YTrain_sm, XTest, YTest, dtc_tuned, YPred_dtc_tuned, 'Tuned Decision Tree')
```

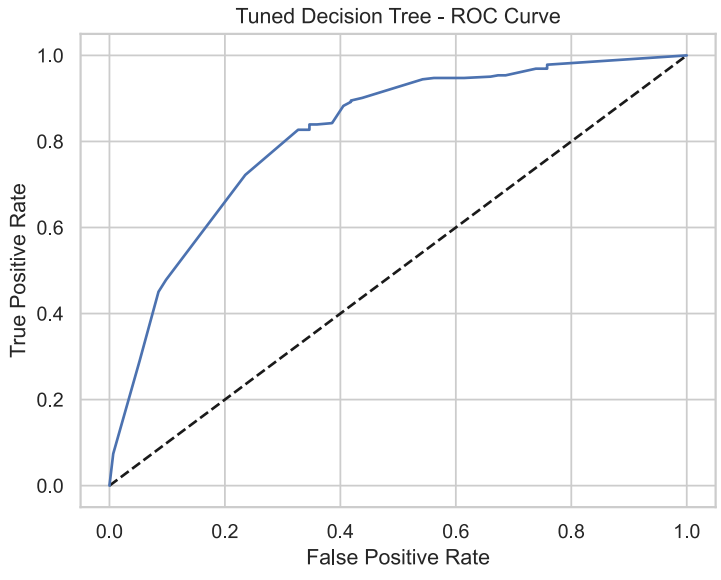



```
Train -> Accuracy Score : 0.8223684210526315
Test -> Accuracy Score : 0.779874213836478
```

In [93]:
```
# Printing the Classification Report for the Tuned Decision Tree Model
print(classification_report(YTest, YPred_dtc_tuned))
```

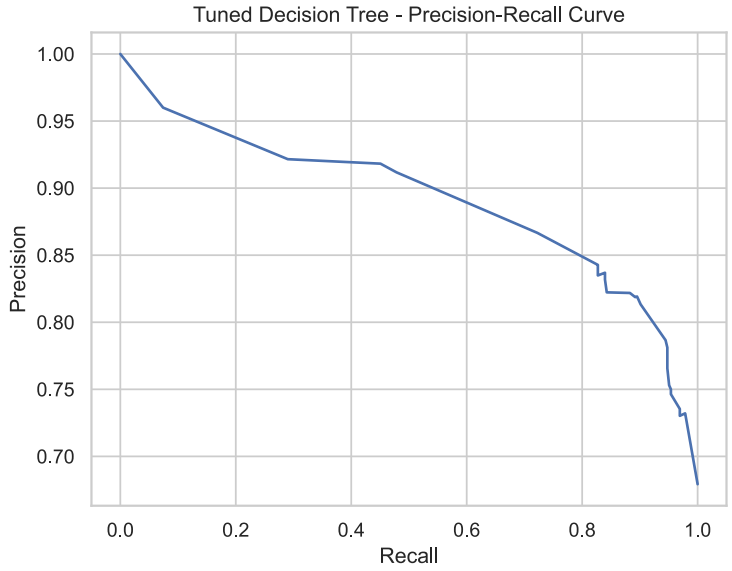
```
              precision    recall  f1-score   support

           0       0.66      0.65      0.66       153
           1       0.84      0.84      0.84       324

    accuracy                           0.78       477
   macro avg       0.75      0.75      0.75       477
weighted avg       0.78      0.78      0.78       477
```

In [94]:
```
# Printing the ROC Curve for the Tuned Decision Tree Model
roc_auc_curve(XTest, YTest, YPred_dtc_tuned_prob, 'Tuned Decision Tree')
```




```
AUC Score: 0.8167614782538529
```

In [95]:
```
# Printing the precision-recall curve for the Tuned Decision Tree Model
precision_recall(XTest, YTest, YPred_dtc_tuned, YPred_dtc_tuned_prob, 'Tuned Decision Tree')
```

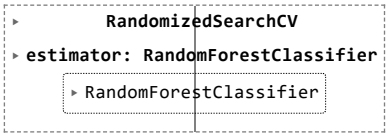



```
Precision: 0.8369230769230769
Recall: 0.8395061728395061
f1 Score: 0.8382126348228043
AUC Score: 0.8915293634976915
```

## Random Forest (Optimized)

In [96]:
```
# Defining parameters to explore for Randomized Search for Tuning of Decision Tree
params2explore_rfc = {'n_estimators' : np.arange(10,2000,10),
'max_features' : ['log2','sqrt'],
'max_depth' : np.arange(10,100,2),
'criterion' : ['gini','entropy'],
'bootstrap' : [True,False]
}
# Instantiating the Random Forest Classifier
rfc = RandomForestClassifier()
# Hyperparameter tuning to find the best possible parameters for Random Forest Model
rfc_tuned = RandomizedSearchCV(rfc, params2explore_rfc, cv = 5)
# Fitting the instantiated classifier on the training data
rfc_tuned.fit(XTrain_sm, YTrain_sm)
```

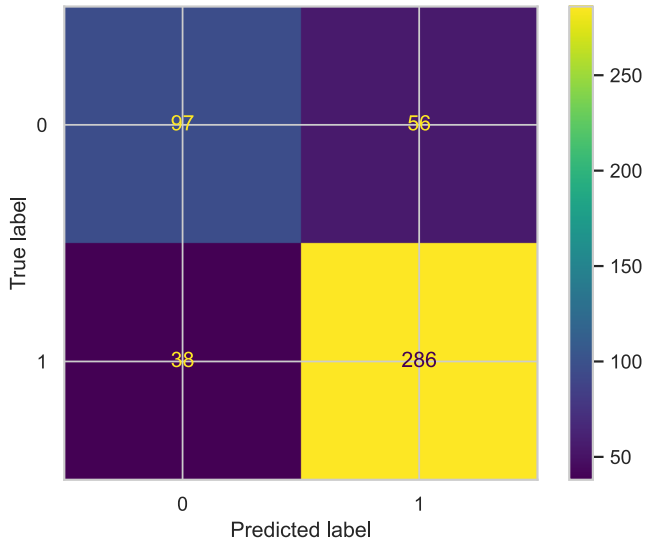
Out[96]:




In [97]:
```
# Predicting the Classes and Probabilities for the Tuned Random Forest Model
YPred_rfc_tuned = rfc_tuned.predict(XTest)
YPred_rfc_tuned_prob = rfc_tuned.predict_proba(XTest)
# Printing the best possible hyperparameters
print(f'Best possible hyperparameters for the tuned Random Forest : {rfc_tuned.best_params_}')
```

```
Best possible hyperparameters for the tuned Random Forest : {'n_estimators': 770, 'max_features': 'log2', 'max_depth':
80, 'criterion': 'entropy', 'bootstrap': True}
```

In [98]:
```
# Printing Confusion Matrix for the Tuned Random Forest Model
conf_matrix_plot(XTrain_sm, YTrain_sm, XTest, YTest, rfc_tuned, YPred_rfc_tuned, 'Tuned Random Forest')
```

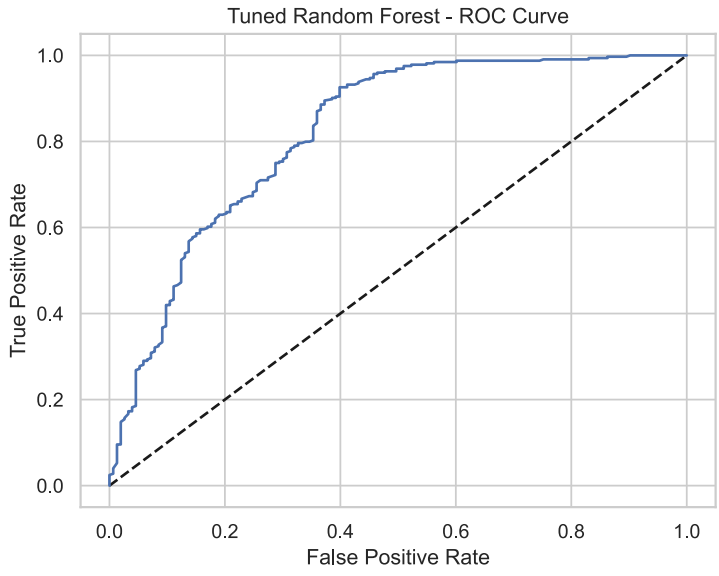



```
Train -> Accuracy Score : 1.0
Test -> Accuracy Score : 0.8029350104821803
```

In [99]:
```
# Printing the Classification Report for the Tuned Random Forest Model
print(classification_report(YTest, YPred_rfc_tuned))
```

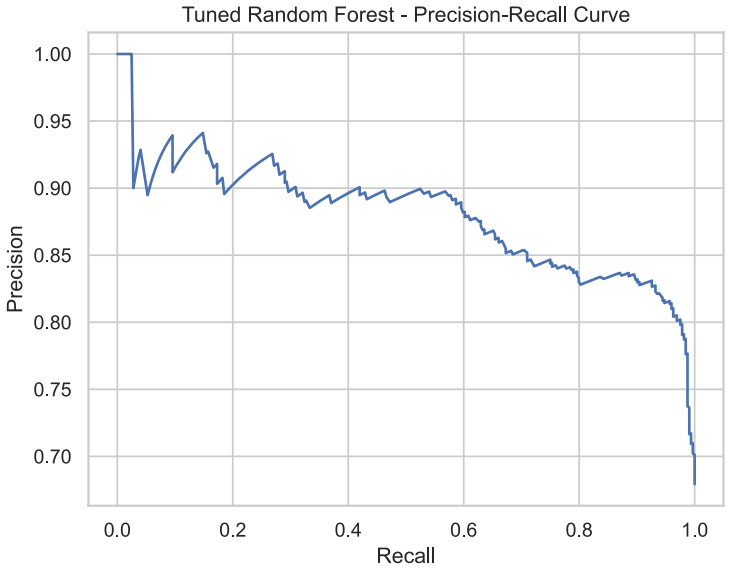
```
              precision    recall  f1-score   support

           0       0.72      0.63      0.67       153
           1       0.84      0.88      0.86       324

    accuracy                           0.80       477
   macro avg       0.78      0.76      0.77       477
weighted avg       0.80      0.80      0.80       477
```

In [100]:
```
# Printing the ROC Curve for the Tuned Random Forest Model
roc_auc_curve(XTest, YTest, YPred_rfc_tuned_prob, 'Tuned Random Forest')
```




```
AUC Score: 0.8200899701444364
```

In [101]:
```
    # Printing the precision-recall curve for the Tuned Random Forest Model
precision_recall(XTest, YTest, YPred_rfc_tuned, YPred_rfc_tuned_prob, 'Tuned Random Forest')
```

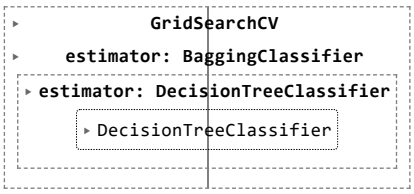



```
Precision: 0.8362573099415205
Recall: 0.8827160493827161
f1 Score: 0.8588588588588588
AUC Score: 0.8799733497415428
```

## Bagging Classifier

In [102]:
```
# Defining parameters to explore for Grid Search for Bagging Model
params2explore_bag = {'n_estimators' : [100,200,300,400,500,600,700,800,900,1000]}
bagging_grid = GridSearchCV(BaggingClassifier(estimator = DecisionTreeClassifier(),
bootstrap_features = True),
param_grid = params2explore_bag,
cv = 5)
# Fitting the instantiated Classifier to train the model
bagging_grid.fit(XTrain_sm, YTrain_sm)
```

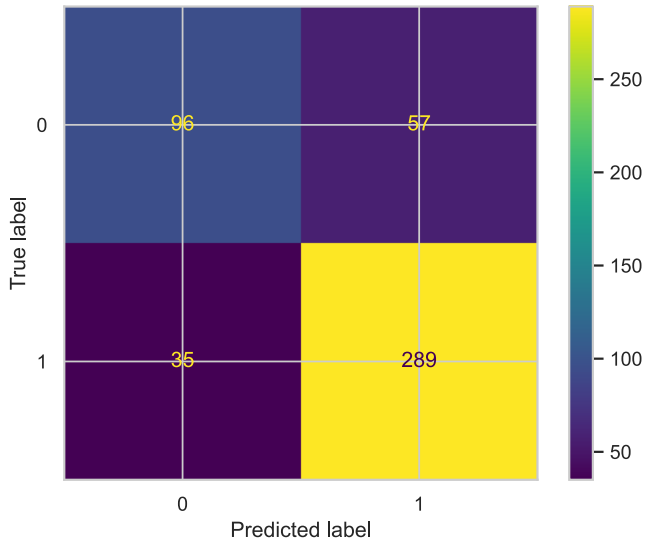
Out[102]:




In [103]:
```
# Predicting the Classes and Probabilities for the Bagging Model
YPred_bag = bagging_grid.predict(XTest)
YPred_bag_prob = bagging_grid.predict_proba(XTest)
# Printing the best possible hyperparameters
print(f'Best possible hyperparameters for the Bagging Model : {bagging_grid.best_params_}')
```

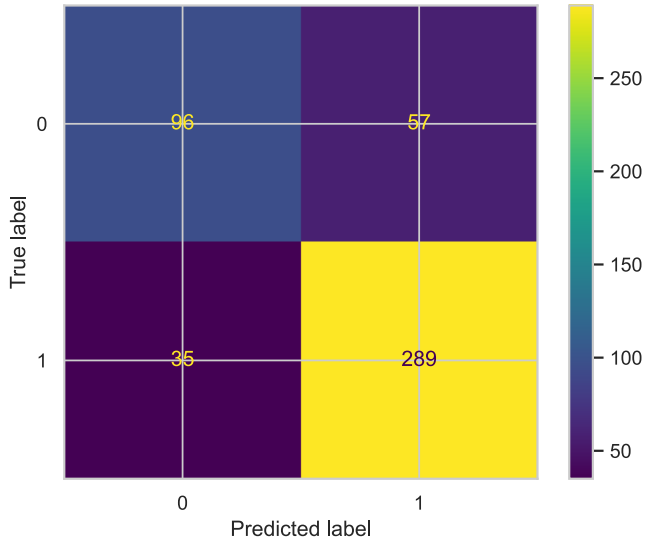
```
Best possible hyperparameters for the Bagging Model : {'n_estimators': 400}
```

In [104]:
```
# Printing Confusion Matrix for Bagging Model
conf_matrix_plot(XTrain_sm, YTrain_sm, XTest, YTest, bagging_grid, YPred_bag, 'Bagging Model (DT)')
```



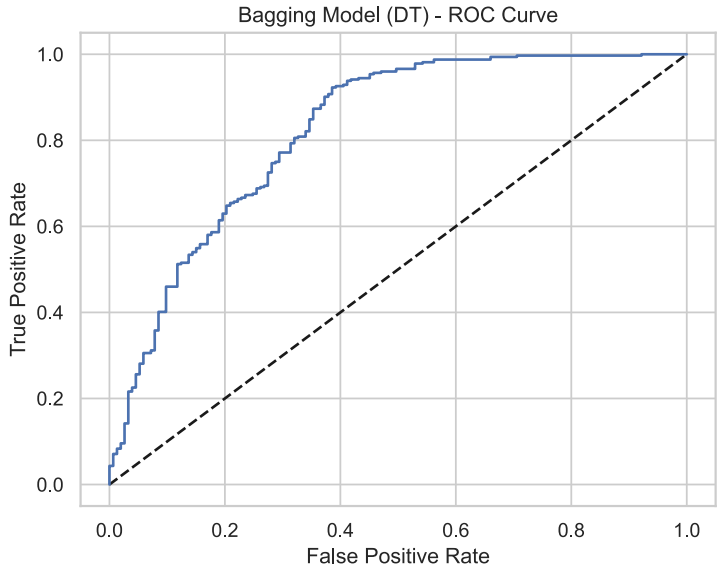

```
Train -> Accuracy Score : 1.0
Test -> Accuracy Score : 0.8071278825995807
```

In [105]:
```
# Printing Confusion Matrix for Bagging Model
conf_matrix_plot(XTrain_sm, YTrain_sm, XTest, YTest, bagging_grid, YPred_bag, 'Bagging Model (DT)')
```



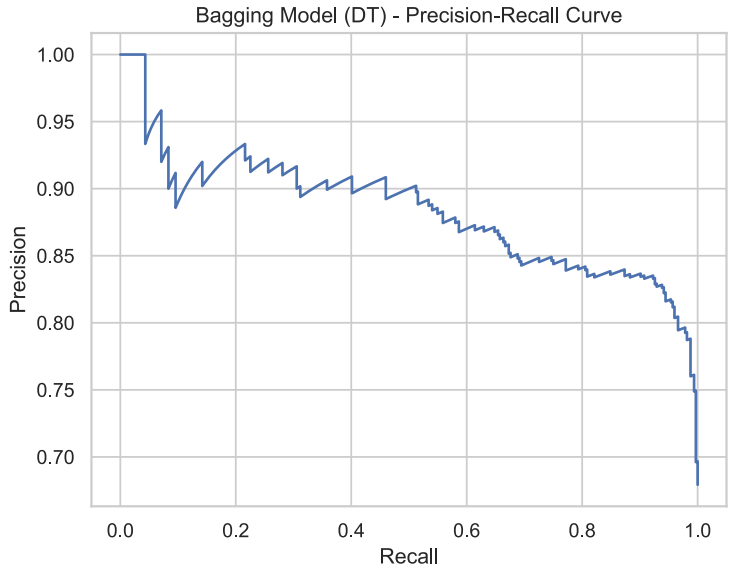

```
Train -> Accuracy Score : 1.0
Test -> Accuracy Score : 0.8071278825995807
```

In [106]:
```
# Printing the ROC Curve for Bagging Model
roc_auc_curve(XTest, YTest, YPred_bag_prob, 'Bagging Model (DT)')
```




```
AUC Score: 0.8226821592834664
```

In [107]:
```
# Printing the precision-recall curve for Bagging Model
precision_recall(XTest, YTest, YPred_bag, YPred_bag_prob, 'Bagging Model (DT)')
```




```
Precision: 0.8352601156069365
Recall: 0.8919753086419753
f1 Score: 0.862686567164179
AUC Score: 0.8829275315716176
```

## Gradient Boosting Classifier

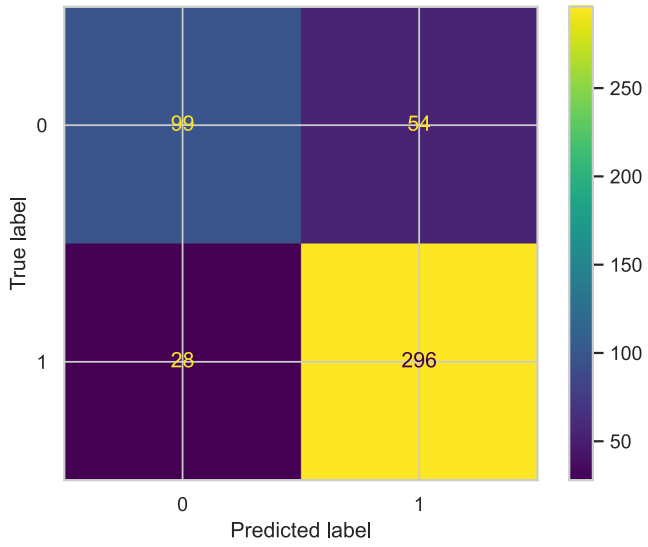
In [108]:
```
# Training with Gradient Boosting Algorithm
learning_rates = [0.01,0.05,0.10,0.25,0.50,0.75,1.0]
```

In [109]:
```
for lr in learning_rates:
    gbc = GradientBoostingClassifier(n_estimators = 150, learning_rate = lr, max_depth = 3, loss = 'log_loss')
    gbc.fit(XTrain_sm, YTrain_sm)
    print("Learning Rate : ", lr)
    print("Train -> Accuracy Score : ", round(gbc.score(XTrain_sm, YTrain_sm),3))
    print("Test -> Accuracy Score : ", round(gbc.score(XTest, YTest),3))
```

```
Learning Rate :  0.01
Train -> Accuracy Score :  0.795
Test -> Accuracy Score :  0.824
Learning Rate :  0.05
Train -> Accuracy Score :  0.86
Test -> Accuracy Score :  0.828
Learning Rate :  0.1
Train -> Accuracy Score :  0.88
Test -> Accuracy Score :  0.811
Learning Rate :  0.25
Train -> Accuracy Score :  0.926
Test -> Accuracy Score :  0.813
Learning Rate :  0.5
Train -> Accuracy Score :  0.975
Test -> Accuracy Score :  0.803
Learning Rate :  0.75
Train -> Accuracy Score :  0.991
Test -> Accuracy Score :  0.776
Learning Rate :  1.0
Train -> Accuracy Score :  0.998
Test -> Accuracy Score :  0.774
```

In [110]:
```
gbc_final = GradientBoostingClassifier(n_estimators = 150, learning_rate = 0.05, max_depth = 3, loss = 'log_loss')
# Fitting the instantiated Classifier to train the model
gbc_final.fit(XTrain_sm, YTrain_sm)
# Predicting the Classes and Probabilities for the Gradient Boosting Classifier
YPred_gbc_final = gbc_final.predict(XTest)
YPred_gbc_final_prob = gbc_final.predict_proba(XTest)
```

In [111]:
```
# Printing Confusion Matrix for Gradient Boosting Classifier
conf_matrix_plot(XTrain_sm, YTrain_sm, XTest, YTest, gbc_final, YPred_gbc_final, 'Gradient Boosting Classifier')
```

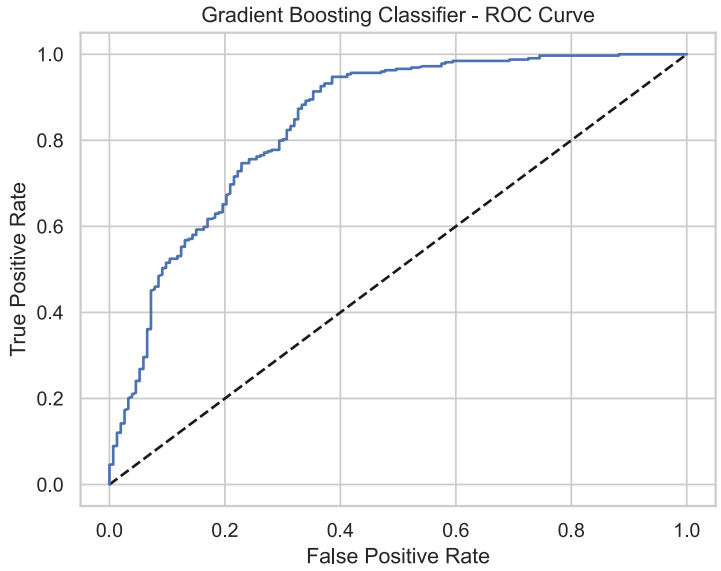



```
Train -> Accuracy Score : 0.8595201238390093
Test -> Accuracy Score : 0.8280922431865828
```

In [112]:
```
# Printing the Classification Report for Bradient Boosting Classifier
print(classification_report(YTest, YPred_gbc_final))
```

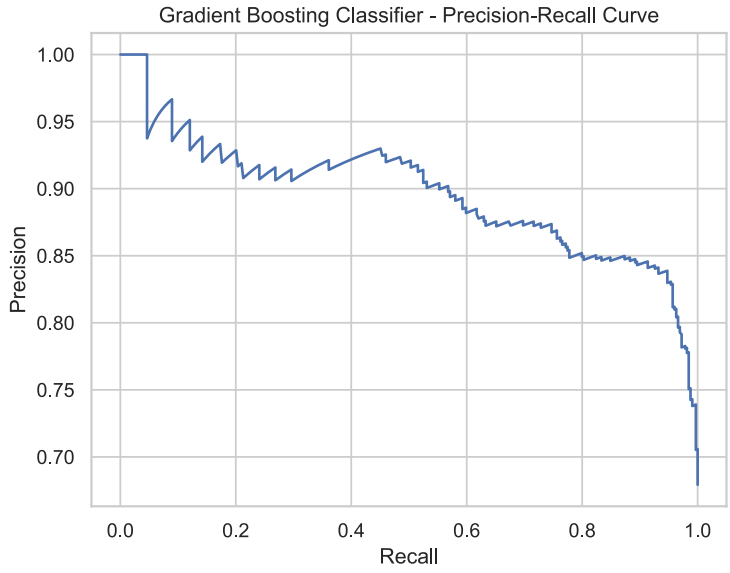
```
              precision    recall  f1-score   support

           0       0.78      0.65      0.71       153
           1       0.85      0.91      0.88       324

    accuracy                           0.83       477
   macro avg       0.81      0.78      0.79       477
weighted avg       0.82      0.83      0.82       477
```

In [113]:
```
# Printing the ROC Curve for Gradient Boosting Classifier
roc_auc_curve(XTest, YTest, YPred_gbc_final_prob, 'Gradient Boosting Classifier')
```




```
AUC Score: 0.8399499717582506
```

In [114]:
```
# Printing the precision-recall curve for Gradient Boosting Classifier
precision_recall(XTest, YTest, YPred_gbc_final, YPred_gbc_final_prob, 'Gradient Boosting Classifier')
```

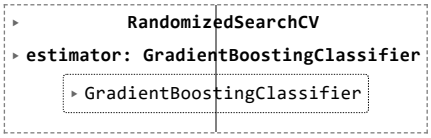



```
Precision: 0.8457142857142858
Recall: 0.9135802469135802
f1 Score: 0.8783382789317508
AUC Score: 0.8959157711948444
```

## Gradient Boosting Decision Tree (GBDT)

In [115]:
```
# Defining parameters to explore for Randomized Search for Tuning GBDT
params2explore_gbdt = {'n_estimators' : np.arange(10,200,10),
'max_depth' : np.arange(10,50,2),
'max_leaf_nodes' : [20,40,80,90,100],
'learning_rate' : [0.01,0.05,0.10,0.25,0.50,0.75,1.0]
}
# Instantiating the GBDT
gbdt = GradientBoostingClassifier()
# Hyperparameter tuning to find the best possible parameters for GBDT
gbdt_tuned = RandomizedSearchCV(gbdt, params2explore_gbdt, cv = 5)
# Fitting the instantiated classifier on the training data
gbdt_tuned.fit(XTrain_sm, YTrain_sm)
```

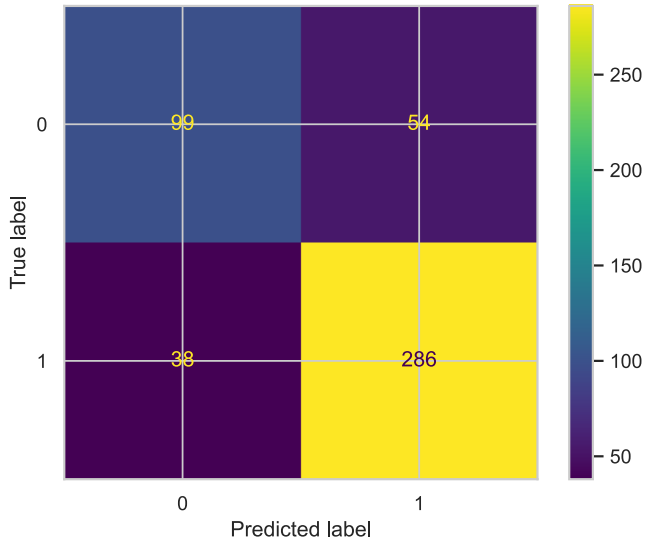
Out[115]:



In [116]:
```
# Predicting the Classes and Probabilities for GBDT
YPred_gbdt_tuned = gbdt_tuned.predict(XTest)
YPred_gbdt_tuned_prob = gbdt_tuned.predict_proba(XTest)
# Printing the best possible hyperparameters
print(f'Best possible hyperparameters for the GBDT : {gbdt_tuned.best_params_}')
```

```
Best possible hyperparameters for the GBDT : {'n_estimators': 110, 'max_leaf_nodes': 100, 'max_depth': 28, 'learning_r
ate': 0.05}
```

In [117]:
```
# Printing Confusion Matrix for GBDT
conf_matrix_plot(XTrain_sm, YTrain_sm, XTest, YTest, gbdt_tuned, YPred_gbdt_tuned, 'GBDT')
```

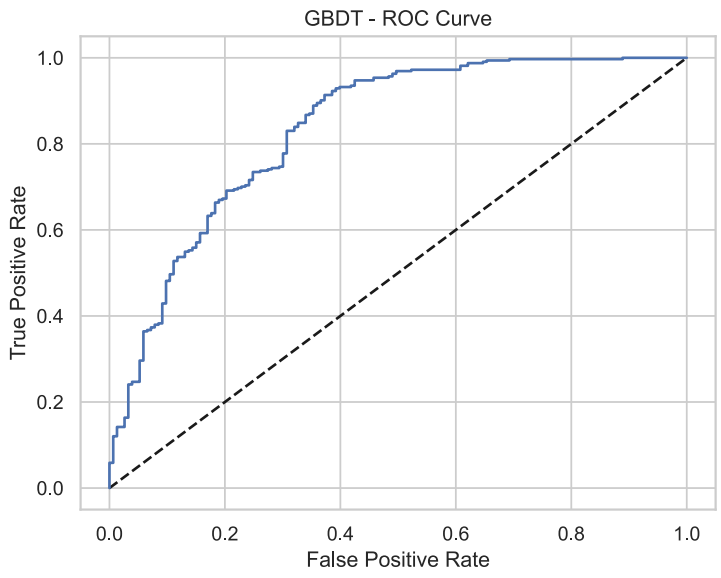



```
Train -> Accuracy Score : 0.9996130030959752
Test -> Accuracy Score : 0.8071278825995807
```

In [118]:
```
# Printing the Classification Report for GBDT
print(classification_report(YTest, YPred_gbdt_tuned))
```

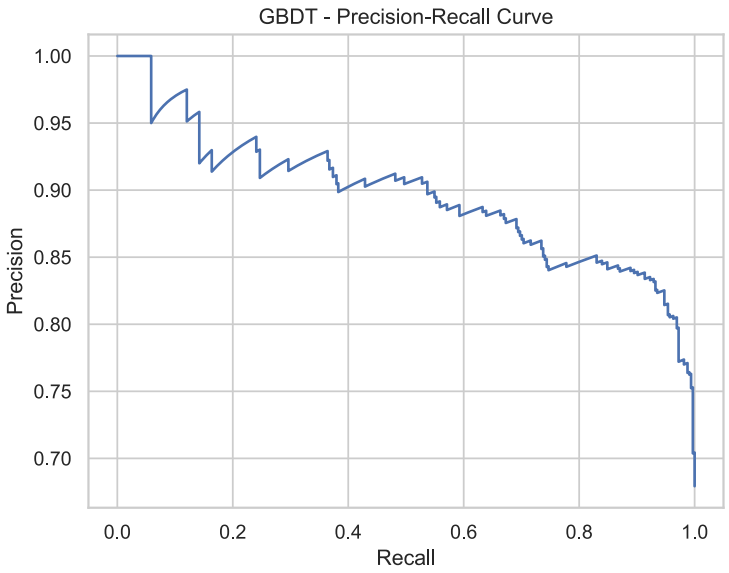
```
              precision    recall  f1-score   support

           0       0.72      0.65      0.68       153
           1       0.84      0.88      0.86       324

    accuracy                           0.81       477
   macro avg       0.78      0.76      0.77       477
weighted avg       0.80      0.81      0.80       477
```

In [119]:
```
# Printing the ROC Curve for GBDT
roc_auc_curve(XTest, YTest, YPred_gbdt_tuned_prob, 'GBDT')
```




```
AUC Score: 0.834079722423949
```

In [120]:
```
# Printing the precision-recall curve for GBDT
precision_recall(XTest, YTest, YPred_gbdt_tuned, YPred_gbdt_tuned_prob, 'GBDT')
```

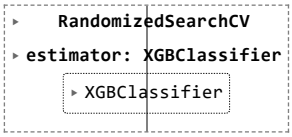



```
Precision: 0.8411764705882353
Recall: 0.8827160493827161
f1 Score: 0.8614457831325302
AUC Score: 0.8950493604410096
```

## XGBoost

In [121]:
```
# Defining parameters to explore for Randomized Search for Tuning GBDT
params2explore_xgb = {'n_estimators' : np.arange(10,2000,10),
'max_depth' : np.arange(10,100,2),
'learning_rate' : [0.01,0.05,0.10,0.25,0.50,0.75,1.0],
'subsample' : [0.6,0.8,1.0],
'colsample_bytree' : [0.6,0.8,1.0]
}
# Instantiating the XGBoost
xgb = XGBClassifier(objective = 'multi:softmax', num_class = 20)
# Hyperparameter tuning to find the best possible parameters for XGBoost
xgb_tuned = RandomizedSearchCV(xgb, params2explore_xgb, n_iter = 10, scoring = 'accuracy', cv = 5)
# Fitting the instantiated classifier on the training data
xgb_tuned.fit(XTrain_sm, YTrain_sm)
```

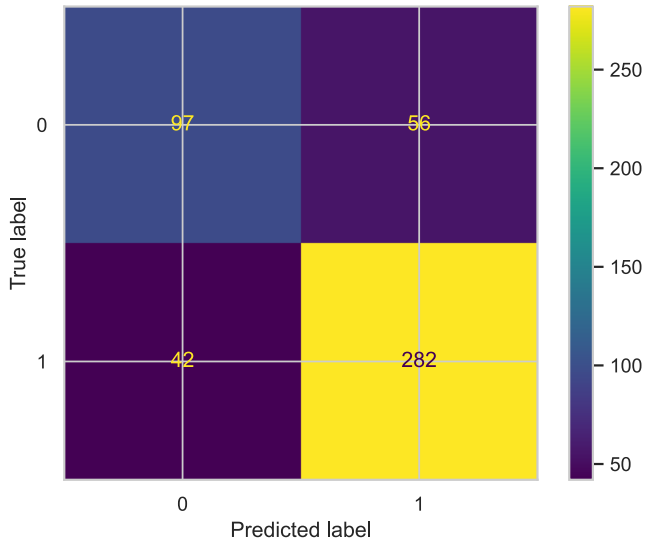
Out[121]:




In [122]:
```
# Predicting the Classes and Probabilities for XGBoost
YPred_xgb_tuned = xgb_tuned.predict(XTest)
YPred_xgb_tuned_prob = xgb_tuned.predict_proba(XTest)
# Printing the best possible hyperparameters
print(f'Best possible hyperparameters for the XGBoost : {xgb_tuned.best_params_}')
```

```
Best possible hyperparameters for the XGBoost : {'subsample': 0.6, 'n_estimators': 1070, 'max_depth': 90, 'learning_ra
te': 0.01, 'colsample_bytree': 1.0}
```

In [123]:
```
# Printing Confusion Matrix for XGBoost
conf_matrix_plot(XTrain_sm, YTrain_sm, XTest, YTest, xgb_tuned, YPred_xgb_tuned, 'XGBoost')
```

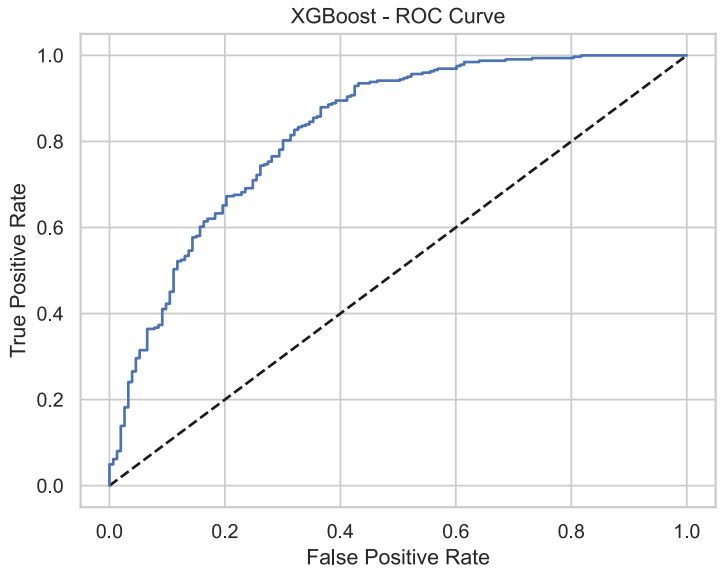



```
Train -> Accuracy Score : 1.0
Test -> Accuracy Score : 0.7945492662473794
```

In [124]:
```
# Printing the Classification Report for XGBoost
print(classification_report(YTest, YPred_xgb_tuned))
```

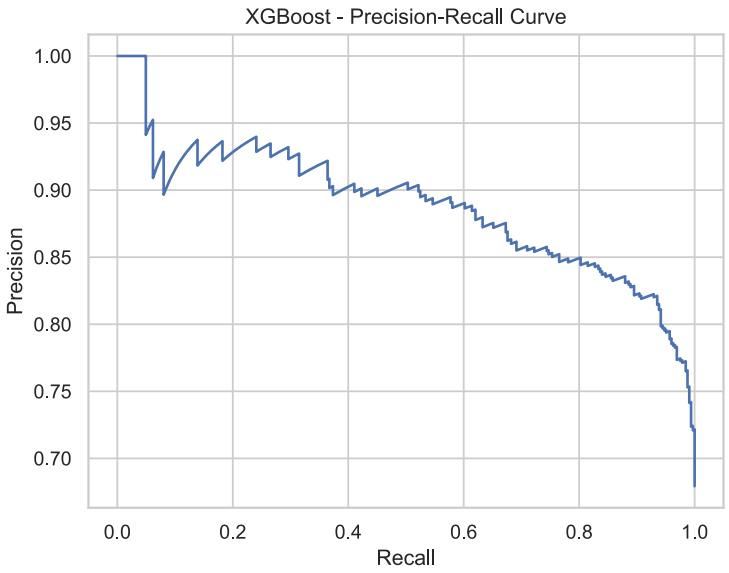
```
              precision    recall  f1-score   support

           0       0.70      0.63      0.66       153
           1       0.83      0.87      0.85       324

    accuracy                           0.79       477
   macro avg       0.77      0.75      0.76       477
weighted avg       0.79      0.79      0.79       477
```

In [125]:
```
# Printing the ROC Curve for XGBoost
roc_auc_curve(XTest, YTest, YPred_xgb_tuned_prob, 'XGBoost')
```




```
AUC Score: 0.8254054708303074
```

In [126]:
```
# Printing the precision-recall curve for XGBoost
precision_recall(XTest, YTest, YPred_xgb_tuned, YPred_xgb_tuned_prob, 'XGBoost')
```




```
Precision: 0.834319526627219
Recall: 0.8703703703703703
f1 Score: 0.8519637462235565
AUC Score: 0.888125051212945
```

## Observations/Insights from Model Assessment

1.Accuracy for Training Data:

K Nearest Neighbors - 0.8459752321981424

Decision Tree - 1.0

Random Forest - 1.0

K Nearest Neighbors (Optimized) - 1.0

Decision Tree (Optimized) - 0.8223684210526315

Random Forest (Optimized) - 1.0

Bagging Classifier - 1.0

Gradient Boosting Classifier - 0.8595201238390093

Gradient Boosting Decision Tree - 1.0

XGBoost - 0.9562693498452013

1.Accuracy for Testing Data:

K Nearest Neighbors - 0.689727463312369

Decision Tree - 0.7526205450733753

Random Forest - 0.7819706498951782

K Nearest Neighbors (Optimized) - 0.7211740041928721

Decision Tree (Optimized) - 0.779874213836478

Random Forest (Optimized) - 0.8029350104821803

Bagging Classifier - 0.8092243186582809

Gradient Boosting Classifier - 0.8280922431865828

Gradient Boosting Decision Tree - 0.7840670859538784

XGBoost - 0.8134171907756813

1. It can be seen from the above mentioned accuracies that Gradient Boosting Classifier has the maximum accuracy for the test data, which is 0.828.

## Multi-Collinearity check using Variance Inflation Factor

In [128]:
```
# Function for VIF Calculation
def calculate_VIF(X):
    vif = pd.DataFrame()
    X_t = X
    vif['Features'] = X_t.columns
    vif['VIF'] = [variance_inflation_factor(X_t.values, i) for i in range(X_t.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    return vif
```

In [129]:
```
# Using the calculate_VIF function to check the VIFs
calculate_VIF(X)[:10]
```

Out[129]:

| | Features | VIF |
|---|---|---|
| 0 | Age | 19.47 |
| 2 | Income | 12.99 |
| 41 | Grade_3 | 9.00 |
| 37 | Joining Designation_3 | 6.39 |
| 40 | Grade_2 | 6.28 |
| 36 | Joining Designation_2 | 5.00 |
| 3 | Total Business Value | 4.01 |
| 42 | Grade_4 | 3.77 |
| 43 | Grade_5 | 2.80 |
| 4 | Increment_in_QR | 2.75 |

Dropping the 'Age' column since it has VIF greater than 5 and proceeding to calculate the VIF again.

In [130]:
```
X.drop(columns=['Age'],axis=1,inplace=True)
calculate_VIF(X)[:10]
```

Out[130]:

| | Features | VIF |
|---|---|---|
| 1 | Income | 12.56 |
| 40 | Grade_3 | 8.93 |
| 36 | Joining Designation_3 | 6.36 |
| 39 | Grade_2 | 6.16 |
| 35 | Joining Designation_2 | 4.98 |
| 2 | Total Business Value | 4.01 |
| 41 | Grade_4 | 3.76 |
| 42 | Grade_5 | 2.80 |
| 3 | Increment_in_QR | 2.74 |
| 38 | Joining Designation_5 | 2.33 |

Dropping the 'Income' column since it has VIF greater than 5 and proceeding to calculate the VIF again.

In [131]:
```
X.drop(columns=['Income'],axis=1,inplace=True)
calculate_VIF(X)[:5]
```

Out[131]:

| | Features | VIF |
|---|---|---|
| 39 | Grade_3 | 7.31 |
| 35 | Joining Designation_3 | 6.35 |
| 38 | Grade_2 | 5.70 |
| 34 | Joining Designation_2 | 4.98 |
| 1 | Total Business Value | 4.00 |

Dropping the 'Grade_3' and 'Joining Designation_3' columns since it has VIF greater than 5 and proceeding to calculate the VIF again.

In [132]:
```
X.drop(columns=['Grade_3','Joining Designation_3'],axis=1,inplace=True)
calculate_VIF(X)[:5]
```

Out[132]:

| | Features | VIF |
|---|---|---|
| 37 | Grade_2 | 4.40 |
| 34 | Joining Designation_2 | 4.21 |
| 1 | Total Business Value | 3.46 |
| 2 | Increment_in_QR | 2.71 |
| 39 | Grade_5 | 2.46 |

Now all the VIFs are less than 5. So, we will proceed with Standardization and Cross-Validation of X. After that, we will calculate the model accuracy on the finalized dataset (after dropping of certain columns due to multicollinearity), using those models which yielded better results while training.

In [133]:
```
X = scaler.fit_transform(X)
kfold = KFold(n_splits = 5)
# Calculating the model accuracy for Bagging
accuracy_bagging = np.mean(cross_val_score(bagging_grid, X, y, cv = kfold, scoring = 'accuracy', n_jobs = -1))
print("Cross Validation accuracy for Bagging : ", round(accuracy_bagging,3))
# Calculating the model accuracy for Gradient Boosting Classification
accuracy_gbc_final=np.mean(cross_val_score(gbc_final,X,y,cv=kfold,scoring='accuracy',n_jobs=-1))
print("Cross Validation accuracy for Gradient Boosting Classifier : ",round(accuracy_gbc_final,3))
# Calculating the model accuracy for GBDT
accuracy_gbdt_tuned=np.mean(cross_val_score(gbdt_tuned,X,y,cv=kfold,scoring='accuracy',n_jobs=-1))
print("Cross Validation accuracy for GBDT : ",round(accuracy_gbdt_tuned,3))
# Calculating the model accuracy for XGBoost
accuracy_xgb_tuned=np.mean(cross_val_score(xgb_tuned,X,y,cv=kfold,scoring='accuracy',n_jobs=-1))
print("Cross Validation accuracy for XGBoost : ",round(accuracy_xgb_tuned,3))
```

```
Cross Validation accuracy for Bagging :  0.797
Cross Validation accuracy for Gradient Boosting Classifier :  0.81
Cross Validation accuracy for GBDT :  0.793
Cross Validation accuracy for XGBoost :  0.763
```

# Observations/Insights from Multicollinearity check (using VIF) and Cross-validation of various models

1.After doing multi-collinearity check using Variance Inflation Factor (VIF), we were able to drop features "Age", "Income", "Grade_3", "Joining Designation_3" from our dataset.

2.Cross-Validation Accuracy for four best performing models:

Bagging - 0.797

Gradient Bossting Classifier - 0.81

GBDT - 0.793

XGBoost - 0.763

# Recommendations

```
1.Since we saw from our ML models that increment of quarterly rating turned out to be an important feature for
deciding driver churn, thereby signifying that the company shall focus on improving customer experiencewhich would
make them provide good ratings to drivers.

2.As we also saw that the churn rate is higher among the drivers having Joining Designation 1, certain incentives like
incentive after completing minimum number of rides in a specified time, can be provided by thecompany to such drivers
which will make them think twice before leaving the company.

3.The company can be pro-active in dealing with the drivers in order to understand their requirements and problems
they face while providing any service to customers.4.
Drivers can be provided with some training while onboarding with the company, about what can be the possible ways to
provide excellent services to customers. This might help them earn better ratings andsimultaneously better income.
This would also help in getting a better business value for the company.
```