In [460…
```
# Pursope: To help OLA to address driver attrition by bringing the data driven insights / decision strategies
```

In [461…
```
# Approach:
```

In [462…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.impute import KNNImputer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report, roc_auc_score, roc_curve
from imblearn.over_sampling import SMOTE
```

In [463…
```
df1 = pd.read_csv('ola_driver_scaler.csv')
df=df1.copy()
print(df.head())
```

	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 [464…
```
print(df.shape)
```

```
(19104, 14)
```

In [465…
```
print(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
None
```

In [466…
```
df.rename(columns={'MMM-YY': 'Reporting Date'}, inplace=True)
```

In [467…
```
df.head()
```

Out[467…

	Unnamed: 0	Reporting Date	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	B
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	2
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	03/11/19	1	1	
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	NaN	2	2	

In [ ]:

In [468…

```
print(df.describe())
```

```
         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
```

In [469…
```
df['Dateofjoining'] = pd.to_datetime(df['Dateofjoining'], format='%d/%m/%y')
```

In [470…
```
df['LastWorkingDate'] = pd.to_datetime(df['LastWorkingDate'], format='%d/%m/%y')
```

In [471…
```
df['Reporting Date'] = pd.to_datetime(df['Reporting Date'], format='%d/%m/%y')
```

In [472…
```
# Extract year and month from the Reporting Date
df['ReportingYear'] = df['Reporting Date'].dt.year
```

```
df['ReportingMonth'] = df['Reporting Date'].dt.month
df['ReportingDay'] = df['Reporting Date'].dt.day
```

In [473…
```
df.head()
```

Out[473…

	Unnamed: 0	Reporting Date	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	B
0	0	2019-01-01	1	28.0	0.0	C23	2	57387	2018-12-24	NaT	1	1	2
1	1	2019-01-02	1	28.0	0.0	C23	2	57387	2018-12-24	NaT	1	1	
2	2	2019-01-03	1	28.0	0.0	C23	2	57387	2018-12-24	2019-11-03	1	1	
3	3	2020-01-11	2	31.0	0.0	C7	2	67016	2020-06-11	NaT	2	2	
4	4	2020-01-12	2	31.0	0.0	C7	2	67016	2020-06-11	NaT	2	2	

In [474…
```
df.drop(columns=['Reporting Date'], inplace=True)
```

In [475…
```
df.head()
```

Out[475...

	Unnamed: 0	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Qu
0	0	1	28.0	0.0	C23	2	57387	2018-12-24	NaT	1	1	2381060	
1	1	1	28.0	0.0	C23	2	57387	2018-12-24	NaT	1	1	-665480	
2	2	1	28.0	0.0	C23	2	57387	2018-12-24	2019-11-03	1	1	0	
3	3	2	31.0	0.0	C7	2	67016	2020-06-11	NaT	2	2	0	
4	4	2	31.0	0.0	C7	2	67016	2020-06-11	NaT	2	2	0	

In [476...

```
df['Target'] = df['LastWorkingDate'].notnull().astype(int)
```

In [477...

```
missing_values = df.isnull().sum()
print(missing_values)
```

```
Unnamed: 0                 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
ReportingYear              0
ReportingMonth             0
ReportingDay               0
Target                     0
dtype: int64
```

In [478...

```
df.head()
```

Out[478...

	Unnamed: 0	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Qu
0	0	1	28.0	0.0	C23	2	57387	2018-12-24	NaT	1	1	2381060	
1	1	1	28.0	0.0	C23	2	57387	2018-12-24	NaT	1	1	-665480	
2	2	1	28.0	0.0	C23	2	57387	2018-12-24	2019-11-03	1	1	0	
3	3	2	31.0	0.0	C7	2	67016	2020-06-11	NaT	2	2	0	
4	4	2	31.0	0.0	C7	2	67016	2020-06-11	NaT	2	2	0	

In [ ]:
```
# Target encoding of City
```

In [479...
```
# Calculate the mean of the target for each city
city_target_mean = df.groupby('City')['Target'].mean()
```

In [480...
```
# Map the calculated mean to the City column
df['City_Target_Encoded'] = df['City'].map(city_target_mean)
```

In [481...
```
df.head()
```

Out[481...

	Unnamed: 0	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Qu
0	0	1	28.0	0.0	C23	2	57387	2018-12-24	NaT	1	1	2381060	
1	1	1	28.0	0.0	C23	2	57387	2018-12-24	NaT	1	1	-665480	
2	2	1	28.0	0.0	C23	2	57387	2018-12-24	2019-11-03	1	1	0	
3	3	2	31.0	0.0	C7	2	67016	2020-06-11	NaT	2	2	0	
4	4	2	31.0	0.0	C7	2	67016	2020-06-11	NaT	2	2	0	

In [482…
```
df['City'].nunique()
```

Out[482…
```
29
```

In [483…
```
df['City'].value_counts()
```

Out[483…
```
City
C20    1008
C29     900
C26     869
C22     809
C27     786
C15     761
C10     744
C12     727
C8      712
C16     709
C28     683
C1      677
C6      660
C5      656
C14     648
C3      637
C24     614
C7      609
C21     603
C25     584
C19     579
C4      578
C13     569
C18     544
C23     538
C9      520
C2      472
C11     468
C17     440
Name: count, dtype: int64
```

In [484…
```
df['City_Target_Encoded']
```

Out[484…
```
0        0.105948
1        0.105948
2        0.105948
3        0.085386
4        0.085386
           ...
19099    0.076336
19100    0.076336
19101    0.076336
19102    0.076336
19103    0.076336
Name: City_Target_Encoded, Length: 19104, dtype: float64
```

In [485…
```
df['City_Target_Encoded'].nunique() # There are 29 unique cities; these have been target encoded
```

Out[485…
```
29
```

In [486…
```
df['City_Target_Encoded'].value_counts()
```

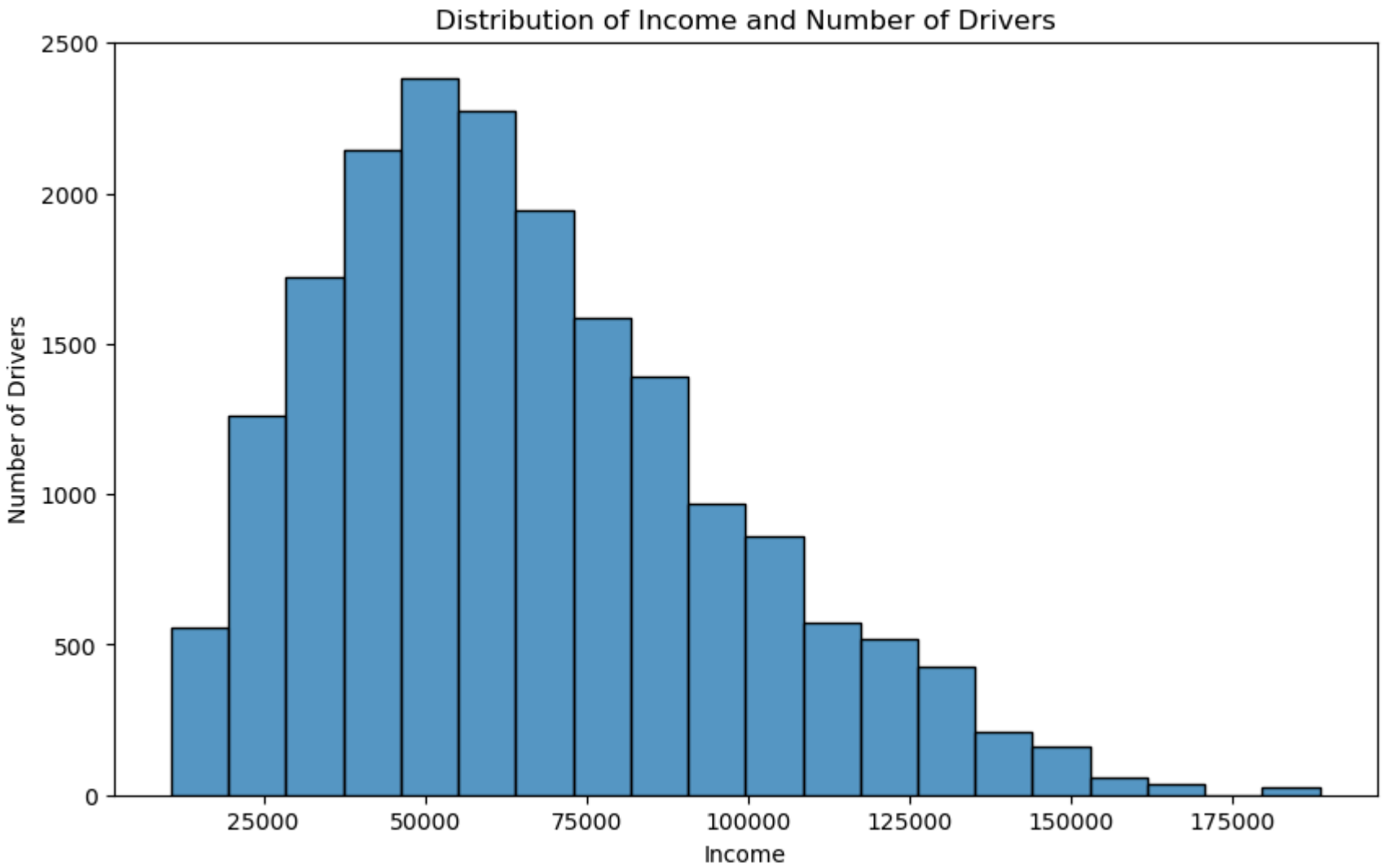
Out[486…

```
City_Target_Encoded
0.110119    1008
0.056667     900
0.074799     869
0.061805     809
0.076336     786
0.090670     761
0.081989     744
0.072902     727
0.074438     712
0.070522     709
0.086384     683
0.082718     677
0.083333     660
0.073171     656
0.089506     648
0.081633     637
0.083062     614
0.085386     609
0.079602     603
0.092466     584
0.070812     579
0.089965     578
0.101933     569
0.080882     544
0.105948     538
0.101923     520
0.116525     472
0.096154     468
0.125000     440
Name: count, dtype: int64
```

In [ ]:

In [529…

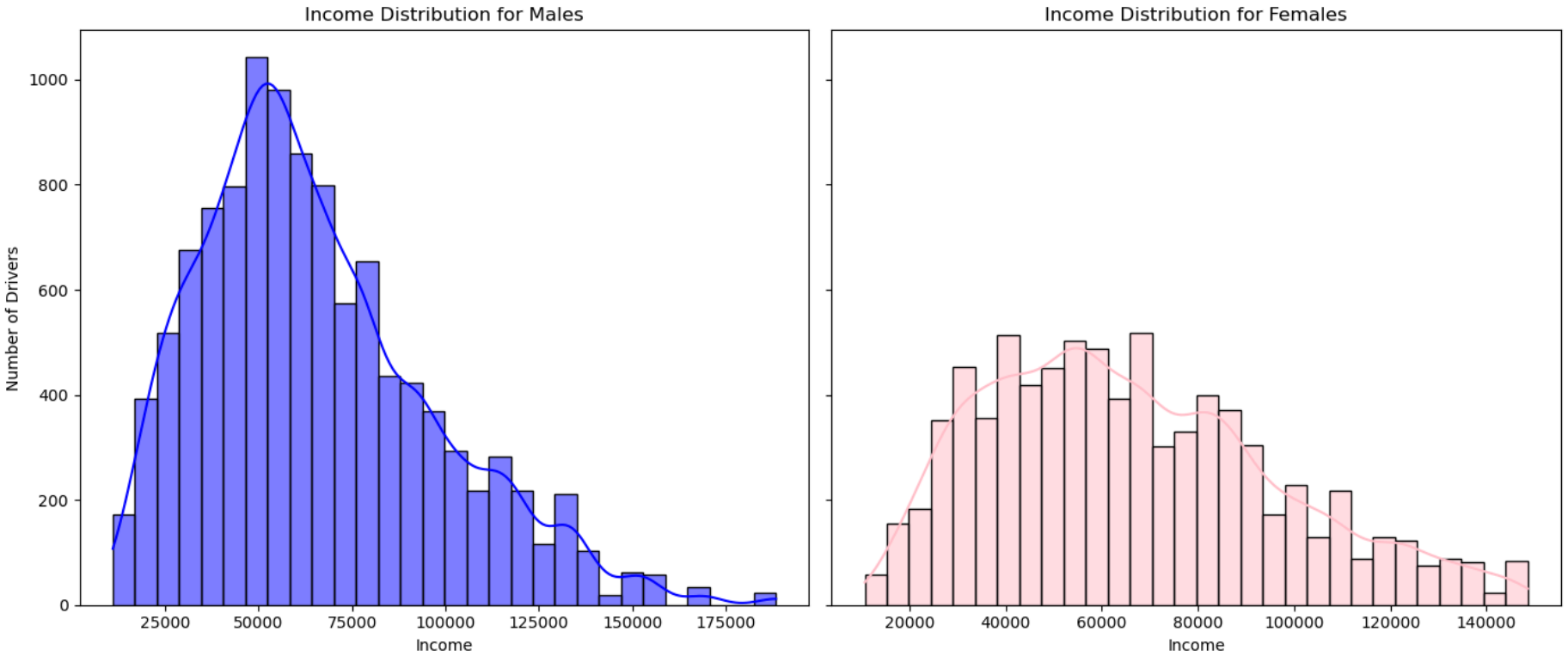
```
plt.figure(figsize=(10, 6))  # Set the figure size for better visibility
sns.histplot(data=df, x='Income', bins=20, kde=False)  # kde=False means we won't add a kernel density estimate
plt.title('Distribution of Income and Number of Drivers')
plt.xlabel('Income')
plt.ylabel('Number of Drivers')
plt.show()
```



In [686…

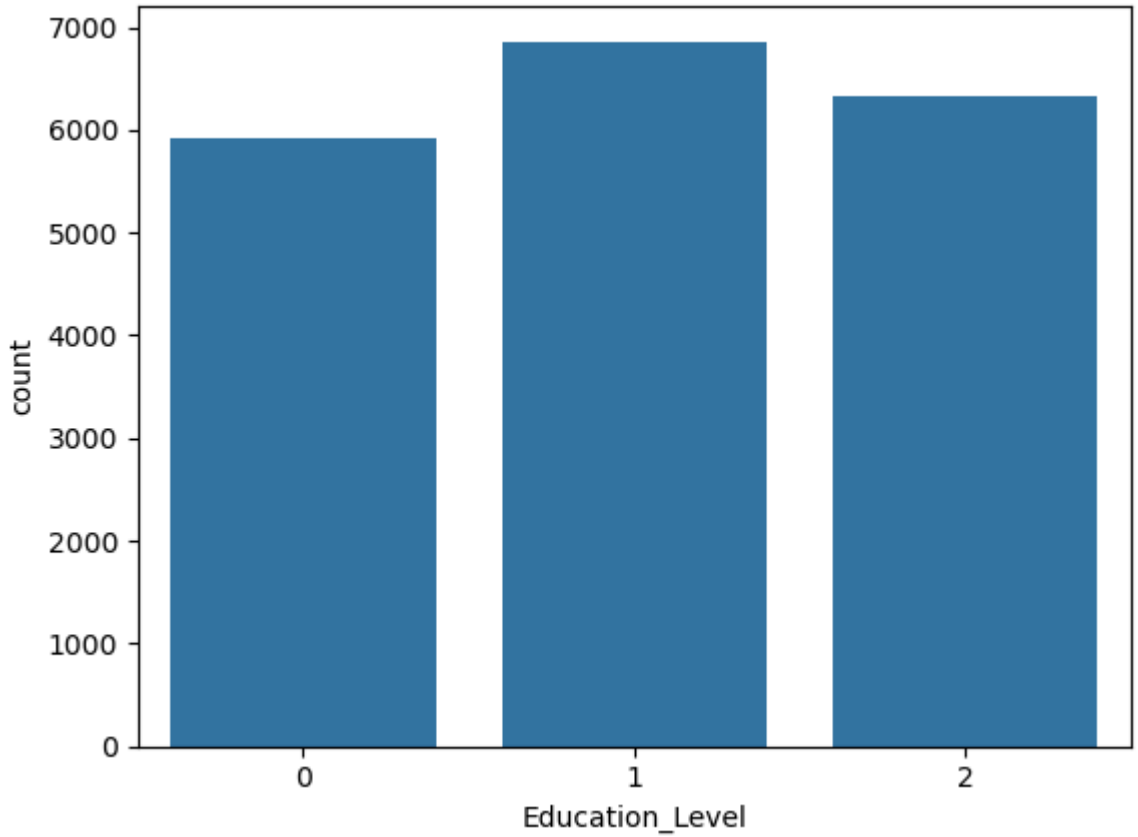
```
# Filter data for males and females
df_male = df[df['Gender'] == 0]
df_female = df[df['Gender'] == 1]

# Set up the figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=True)
```

```
# Plot for Males
sns.histplot(df_male['Income'], bins=30, kde=True, ax=axes[0], color='blue')
axes[0].set_title('Income Distribution for Males')
axes[0].set_xlabel('Income')
axes[0].set_ylabel('Number of Drivers')

# Plot for Females
sns.histplot(df_female['Income'], bins=30, kde=True, ax=axes[1], color='pink')
axes[1].set_title('Income Distribution for Females')
axes[1].set_xlabel('Income')
axes[1].set_ylabel('Number of Drivers')

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



Female drivers have relatively lower income

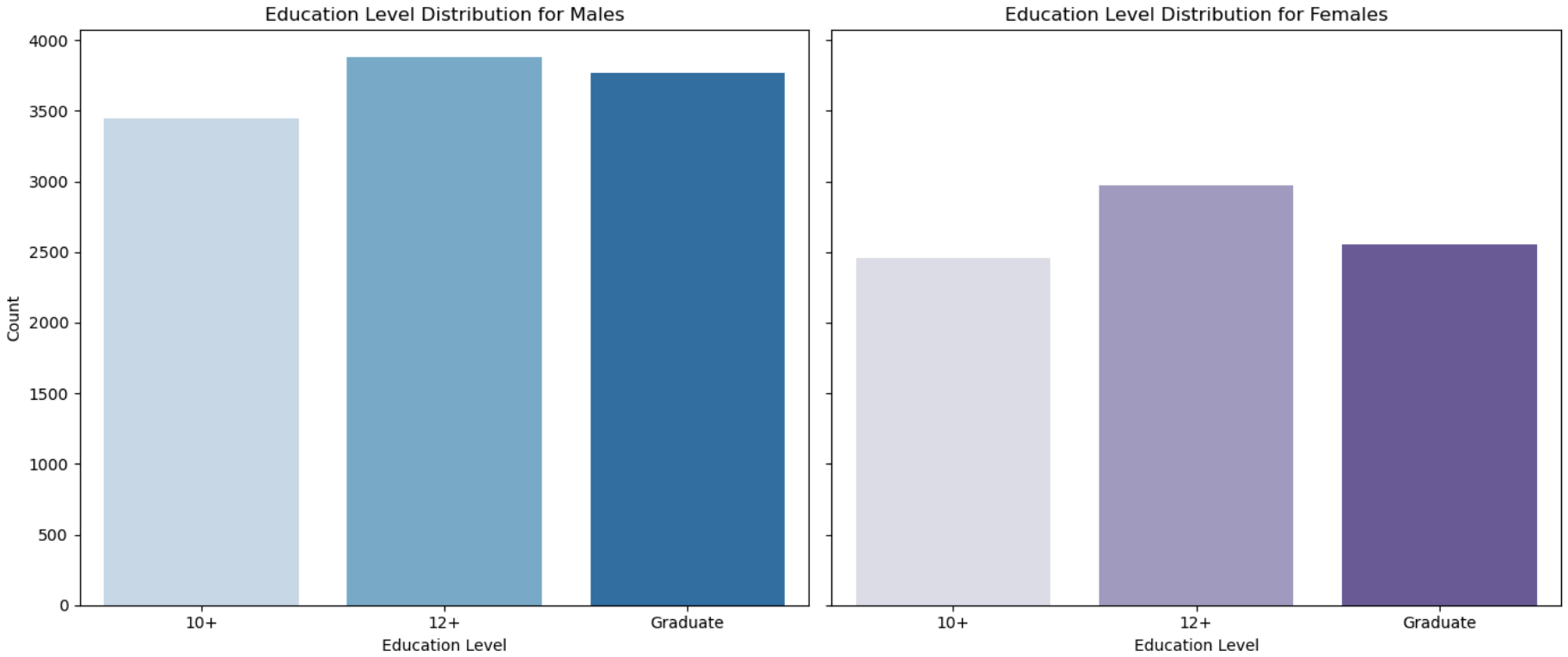
In [487…
```
sns.countplot(x='Education_Level', data=df)
plt.show()
```



In [689…
```
# Set up the figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=True)

# Plot for Males
sns.countplot(x='Education_Level', data=df_male, ax=axes[0], palette='Blues')
axes[0].set_title('Education Level Distribution for Males')
axes[0].set_xlabel('Education Level')
axes[0].set_ylabel('Count')
axes[0].set_xticklabels(['10+', '12+', 'Graduate'])

# Plot for Females
```

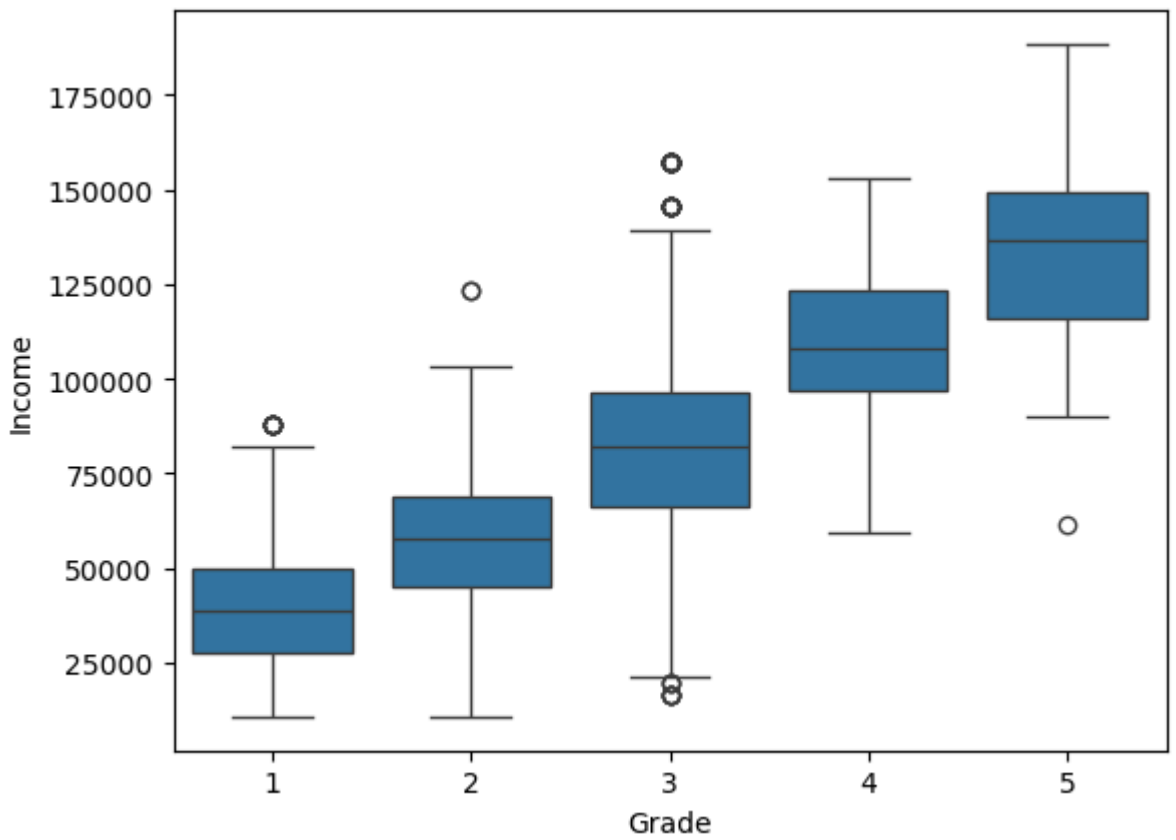
```
sns.countplot(x='Education_Level', data=df_female, ax=axes[1], palette='Purples')
axes[1].set_title('Education Level Distribution for Females')
axes[1].set_xlabel('Education Level')
axes[1].set_ylabel('Count')
axes[1].set_xticklabels(['10+', '12+', 'Graduate'])

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

```
C:\Users\Dell\AppData\Local\Temp\ipykernel_25160\1920163869.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se
t `legend=False` for the same effect.

  sns.countplot(x='Education_Level', data=df_male, ax=axes[0], palette='Blues')
C:\Users\Dell\AppData\Local\Temp\ipykernel_25160\1920163869.py:9: UserWarning: set_ticklabels() should only be used with a fixe
d number of ticks, i.e. after set_ticks() or using a FixedLocator.
  axes[0].set_xticklabels(['10+', '12+', 'Graduate'])
C:\Users\Dell\AppData\Local\Temp\ipykernel_25160\1920163869.py:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se
t `legend=False` for the same effect.

  sns.countplot(x='Education_Level', data=df_female, ax=axes[1], palette='Purples')
C:\Users\Dell\AppData\Local\Temp\ipykernel_25160\1920163869.py:16: UserWarning: set_ticklabels() should only be used with a fix
ed number of ticks, i.e. after set_ticks() or using a FixedLocator.
  axes[1].set_xticklabels(['10+', '12+', 'Graduate'])
```



In [488…

```
sns.boxplot(x='Grade', y='Income', data=df)
plt.show()
```

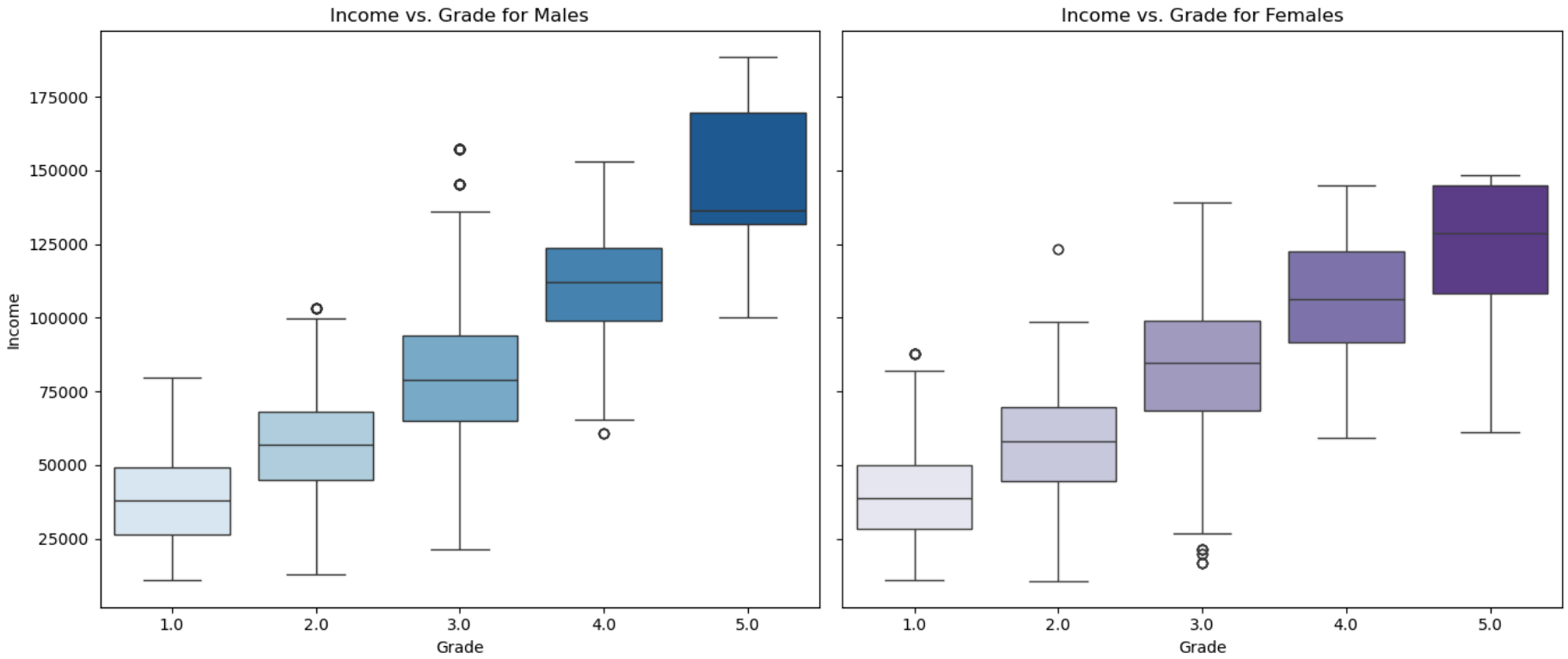


In [690…

```
# Set up the figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=True)

# Boxplot for Males
sns.boxplot(x='Grade', y='Income', data=df_male, ax=axes[0], palette='Blues')
axes[0].set_title('Income vs. Grade for Males')
axes[0].set_xlabel('Grade')
axes[0].set_ylabel('Income')

# Boxplot for Females
sns.boxplot(x='Grade', y='Income', data=df_female, ax=axes[1], palette='Purples')
axes[1].set_title('Income vs. Grade for Females')
axes[1].set_xlabel('Grade')
axes[1].set_ylabel('Income')
```

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

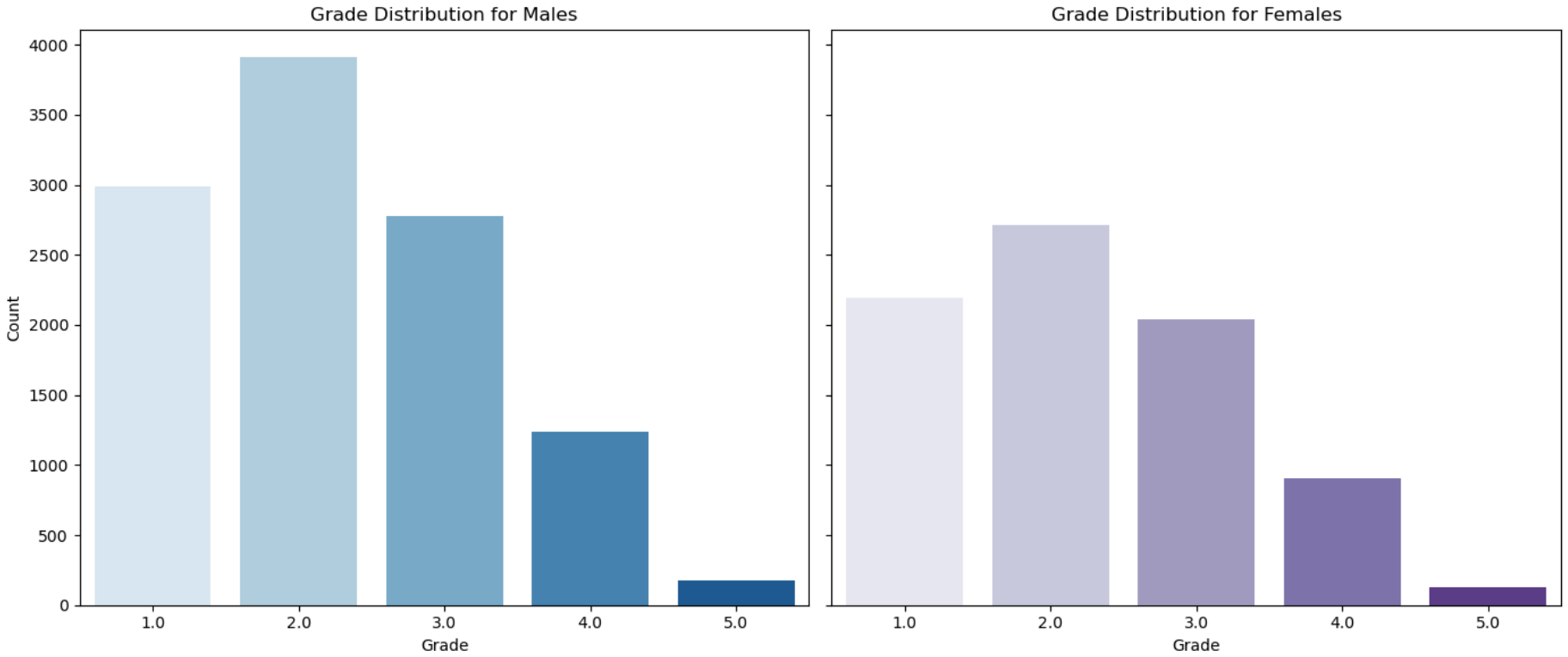
```
C:\Users\Dell\AppData\Local\Temp\ipykernel_25160\2679726086.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se
t `legend=False` for the same effect.

  sns.boxplot(x='Grade', y='Income', data=df_male, ax=axes[0], palette='Blues')
C:\Users\Dell\AppData\Local\Temp\ipykernel_25160\2679726086.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se
t `legend=False` for the same effect.

  sns.boxplot(x='Grade', y='Income', data=df_female, ax=axes[1], palette='Purples')
```



In [691…

```
# Set up the figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(14, 6), sharey=True)

# Plot for Males
sns.countplot(x='Grade', data=df_male, ax=axes[0], palette='Blues')
axes[0].set_title('Grade Distribution for Males')
axes[0].set_xlabel('Grade')
axes[0].set_ylabel('Count')

# Plot for Females
sns.countplot(x='Grade', data=df_female, ax=axes[1], palette='Purples')
axes[1].set_title('Grade Distribution for Females')
axes[1].set_xlabel('Grade')
axes[1].set_ylabel('Count')

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

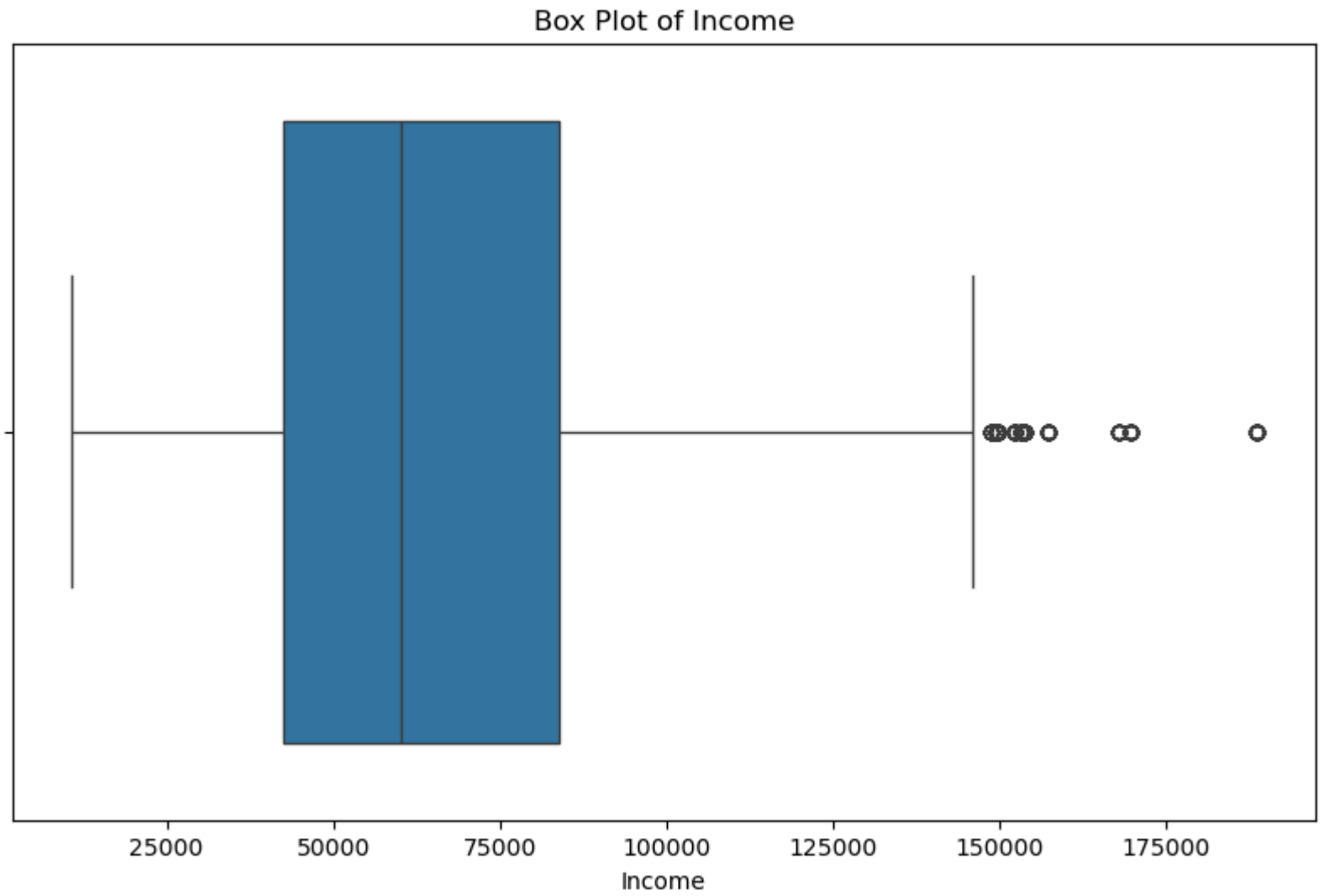
```
C:\Users\Dell\AppData\Local\Temp\ipykernel_25160\4190791469.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se
t `legend=False` for the same effect.

  sns.countplot(x='Grade', data=df_male, ax=axes[0], palette='Blues')
C:\Users\Dell\AppData\Local\Temp\ipykernel_25160\4190791469.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se
t `legend=False` for the same effect.

  sns.countplot(x='Grade', data=df_female, ax=axes[1], palette='Purples')
```



In [692…
```
# Count the number of male and female drivers
gender_counts = df['Gender'].value_counts()

# Print the counts
print(f"Number of male drivers: {gender_counts[0]}")
print(f"Number of female drivers: {gender_counts[1]}")
```

```
Number of male drivers: 11087
Number of female drivers: 7981
```

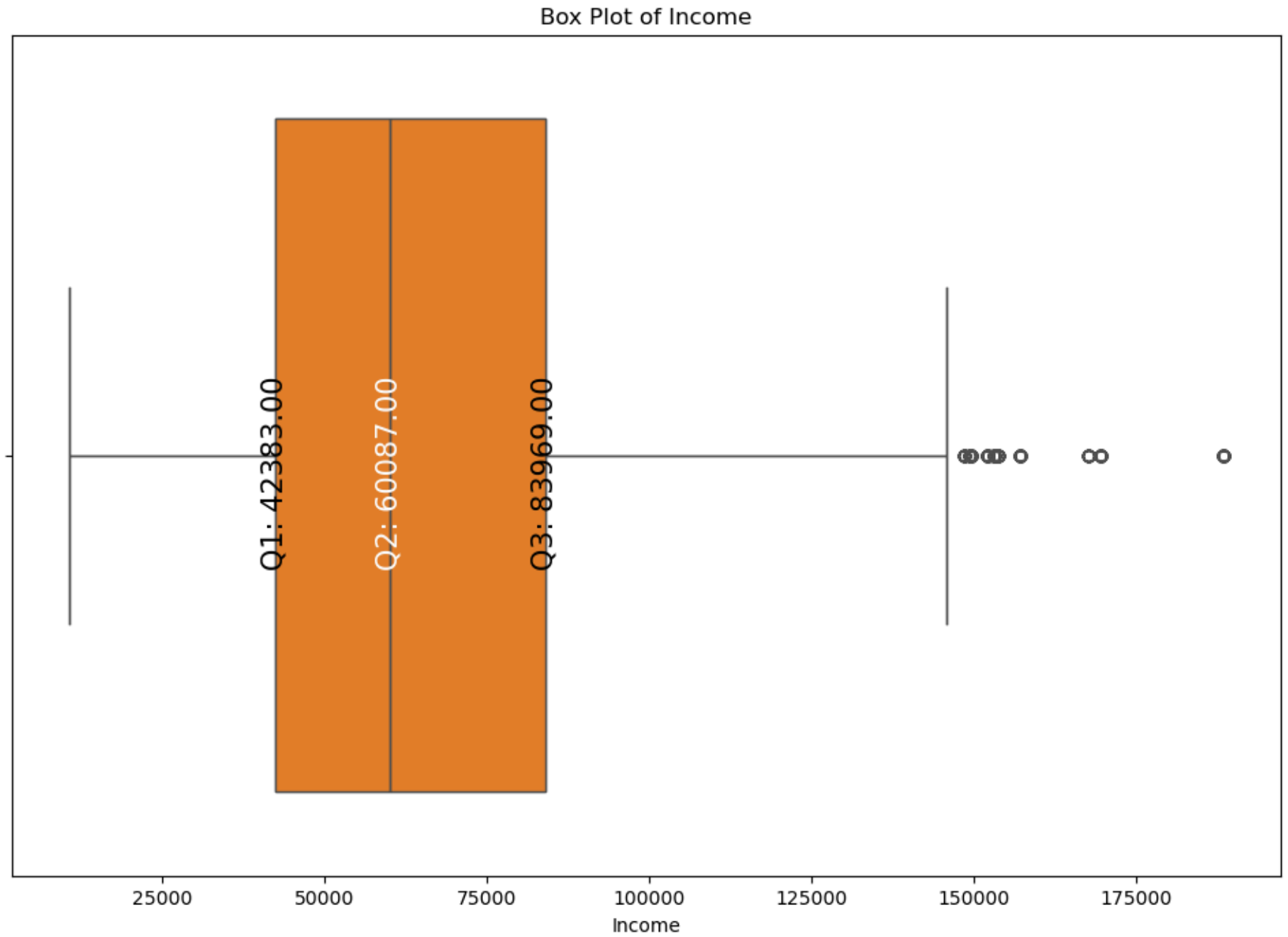
In [530…
```
plt.figure(figsize=(10, 6))  # Set the figure size for better visibility
sns.boxplot(data=df, x='Income')
plt.title('Box Plot of Income')
plt.xlabel('Income')
plt.show()
```



In [549…

```
plt.figure(figsize=(12, 8))
sns.boxplot(data=df, x='Income')
plt.title('Box Plot of Income')
plt.xlabel('Income')

boxplot = sns.boxplot(data=df, x='Income')
```

```
# Calculate Q1, Q2 (median), and Q3
Q1 = df['Income'].quantile(0.25)
Q2 = df['Income'].median()
Q3 = df['Income'].quantile(0.75)

# Adding the annotations for Q1, Q2, and Q3
plt.text(Q1, 0.02, f'Q1: {Q1:.2f}', ha='center', va='center', color='black', fontsize=15, rotation=90)
plt.text(Q2, 0.02, f'Q2: {Q2:.2f}', ha='center', va='center', color='white', fontsize=15, rotation=90)
plt.text(Q3, 0.02, f'Q3: {Q3:.2f}', ha='center', va='center', color='black', fontsize=15, rotation=90)


# Display the plot
plt.show()
```


Box Plot of Income
Q1: 42383.00
Q2: 60087.00
Q3: 83969.00
25000
50000
75000
100000
125000
150000
175000
Income

median income of drivers is 60087.00; there are a few drivers who earn as much as 150000.00 and more

better the grade more the income

we have almost same number of drivers across different education levels

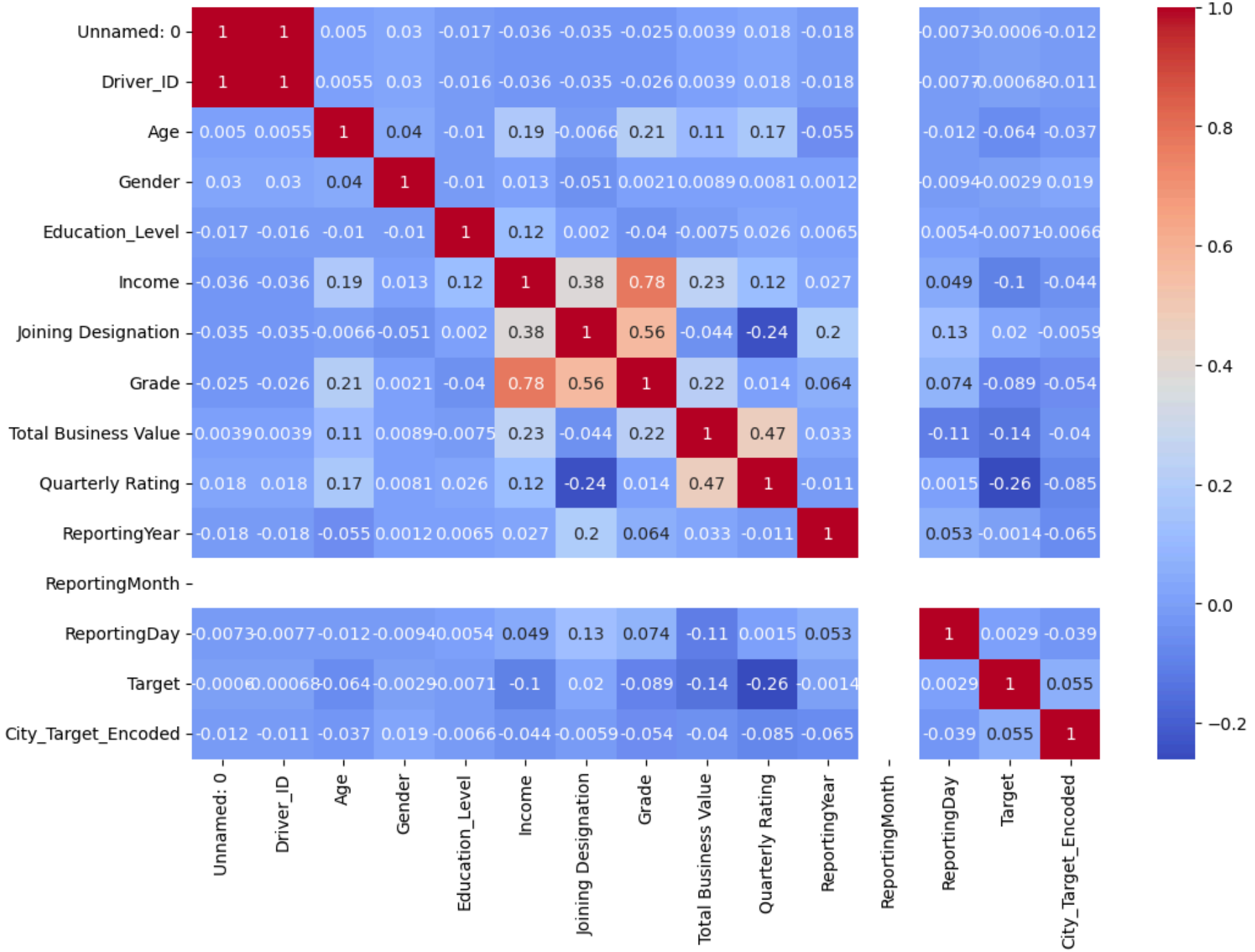
number of drivers who earn more than 100000.00 is also significant

missing value imputation is done. however outlier treatment is not done...

In [ ]:

In [489…

```
# Select only the numeric columns
df_numeric = df.select_dtypes(include=[np.number])

# Now calculate the correlation matrix
corr_matrix = df_numeric.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```


Unnamed: 0
Driver_ID
Age
Gender
Education_Level
Income
Joining Designation
Grade
Total Business Value
Quarterly Rating
ReportingYear
ReportingMonth
ReportingDay
Target
City_Target_Encoded
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

In [ ]:

In [490…
```
missing_values = df.isnull().sum()
print(missing_values)
```

```
Unnamed: 0                 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
ReportingYear              0
ReportingMonth             0
ReportingDay               0
Target                     0
City_Target_Encoded        0
dtype: int64
```

52 missing values in Gender column; this may not be accidental or by mistake; they could be 3rd gender people - it makes sense to study only these records seperately (anyway, not done in this notebook); neverthless we are going to use knn imputer to impute missing values.

In [491…
```
# Perform KNN imputation on numerical features to handle missing values.
```

In [492…
```
imputer = KNNImputer(n_neighbors=5)
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
df[numerical_cols] = imputer.fit_transform(df[numerical_cols])
```

In [493…
```
df.head()
```

Out[493...

	Unnamed: 0	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Q
0	0.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	NaT	1.0	1.0	2381060.0	
1	1.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	NaT	1.0	1.0	-665480.0	
2	2.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	2019-11-03	1.0	1.0	0.0	
3	3.0	2.0	31.0	0.0	C7	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0	
4	4.0	2.0	31.0	0.0	C7	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0	

In [494...

```
missing_values = df.isnull().sum()
print(missing_values)
```

```
Unnamed: 0                 0
Driver_ID                  0
Age                        0
Gender                     0
City                       0
Education_Level            0
Income                     0
Dateofjoining              0
LastWorkingDate        17488
Joining Designation        0
Grade                      0
Total Business Value       0
Quarterly Rating           0
ReportingYear              0
ReportingMonth             0
ReportingDay               0
Target                     0
City_Target_Encoded        0
dtype: int64
```

In [495...

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            19104 non-null  float64
 1   Driver_ID             19104 non-null  float64
 2   Age                   19104 non-null  float64
 3   Gender                19104 non-null  float64
 4   City                  19104 non-null  object
 5   Education_Level       19104 non-null  float64
 6   Income                19104 non-null  float64
 7   Dateofjoining         19104 non-null  datetime64[ns]
 8   LastWorkingDate       1616 non-null   datetime64[ns]
 9   Joining Designation   19104 non-null  float64
 10  Grade                 19104 non-null  float64
 11  Total Business Value  19104 non-null  float64
 12  Quarterly Rating      19104 non-null  float64
 13  ReportingYear         19104 non-null  int32
 14  ReportingMonth        19104 non-null  int32
 15  ReportingDay          19104 non-null  int32
 16  Target                19104 non-null  int32
 17  City_Target_Encoded   19104 non-null  float64
dtypes: datetime64[ns](2), float64(11), int32(4), object(1)
memory usage: 2.3+ MB
```

In [ ]:

In [693…

```
# Calculate total number of male and female drivers
total_males = df[df['Gender'] == 0].shape[0]
total_females = df[df['Gender'] == 1].shape[0]

# Calculate number of male and female drivers who have left (attrition)
attrition_males = df[(df['Gender'] == 0) & (df['Target'] == 1)].shape[0]
attrition_females = df[(df['Gender'] == 1) & (df['Target'] == 1)].shape[0]

# Calculate attrition percentages
attrition_percentage_males = (attrition_males / total_males) * 100
attrition_percentage_females = (attrition_females / total_females) * 100
```

```
# Print the results
print(f"Percentage of attrition for male drivers: {attrition_percentage_males:.2f}%")
print(f"Percentage of attrition for female drivers: {attrition_percentage_females:.2f}%")
```

```
Percentage of attrition for male drivers: 8.54%
Percentage of attrition for female drivers: 8.36%
```

In [694…

```
# Calculate total number of drivers
total_drivers = df.shape[0]

# Calculate total number of drivers who have left (attrition)
total_attrition = df[df['Target'] == 1].shape[0]

# Calculate overall attrition percentage
overall_attrition_percentage = (total_attrition / total_drivers) * 100

# Print the result
print(f"Overall percentage of attrition: {overall_attrition_percentage:.2f}%")
```

```
Overall percentage of attrition: 8.46%
```

In [ ]:

In [695…

```
print("total_males   : ", total_males)
print("total_females : ", total_females)
```

```
total_males   :  11087
total_females :  7981
```

In [496…

```
# Feature Engineering:
```

In [497…

```
df['Rating_Increase'] = (df['Quarterly Rating'] > df['Quarterly Rating'].shift(1)).astype(int)
```

In [498…

```
df['Income_Increase'] = (df['Income'] > df['Income'].shift(1)).astype(int)
```

In [ ]:

In [499…

```
df.head()
```

Out[499…

	Unnamed: 0	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Q
0	0.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	NaT	1.0	1.0	2381060.0	
1	1.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	NaT	1.0	1.0	-665480.0	
2	2.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	2019-11-03	1.0	1.0	0.0	
3	3.0	2.0	31.0	0.0	C7	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0	
4	4.0	2.0	31.0	0.0	C7	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0	

In [ ]:

In [500…
```
df.head()
```

Out[500…

	Unnamed: 0	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Q
0	0.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	NaT	1.0	1.0	2381060.0	
1	1.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	NaT	1.0	1.0	-665480.0	
2	2.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	2019-11-03	1.0	1.0	0.0	
3	3.0	2.0	31.0	0.0	C7	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0	
4	4.0	2.0	31.0	0.0	C7	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0	

In [501…
```
missing_values = df.isnull().sum()
print(missing_values)
```

```
Unnamed: 0                 0
Driver_ID                  0
Age                        0
Gender                     0
City                       0
Education_Level            0
Income                     0
Dateofjoining              0
LastWorkingDate        17488
Joining Designation        0
Grade                      0
Total Business Value       0
Quarterly Rating           0
ReportingYear              0
ReportingMonth             0
ReportingDay               0
Target                     0
City_Target_Encoded        0
Rating_Increase            0
Income_Increase            0
dtype: int64
```

In [502…
```
df['LastWorkingDate'].isna().sum()
```

Out[502…
```
17488
```

In [ ]:
```

```

In [503…
```
df.head(100)
```

Out[503...

	Unnamed: 0	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value
0	0.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	NaT	1.0	1.0	2381060.0
1	1.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	NaT	1.0	1.0	-665480.0
2	2.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	2019-11-03	1.0	1.0	0.0
3	3.0	2.0	31.0	0.0	C7	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0
4	4.0	2.0	31.0	0.0	C7	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
95	95.0	22.0	40.0	0.0	C10	2.0	31224.0	2018-05-25	NaT	1.0	1.0	1120560.0
96	96.0	22.0	40.0	0.0	C10	2.0	31224.0	2018-05-25	NaT	1.0	1.0	696920.0
97	97.0	22.0	34.2	0.0	C10	2.0	31224.0	2018-05-25	NaT	1.0	1.0	200000.0
98	98.0	22.0	41.0	0.0	C10	2.0	31224.0	2018-05-25	NaT	1.0	1.0	306410.0
99	99.0	22.0	41.0	0.0	C10	2.0	31224.0	2018-05-25	NaT	1.0	1.0	499480.0

100 rows × 20 columns

In [504...

```
df.tail(100)
```

Out[504...

	Unnamed: 0	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value
19004	19004.0	2771.0	40.0	0.0	C12	0.0	91456.0	2015-08-29	NaT	2.0	4.0	770090.0
19005	19005.0	2771.0	40.0	0.0	C12	0.0	91456.0	2015-08-29	NaT	2.0	4.0	660230.0
19006	19006.0	2771.0	40.0	0.0	C12	0.0	91456.0	2015-08-29	NaT	2.0	4.0	401530.0
19007	19007.0	2771.0	40.0	0.0	C12	0.0	91456.0	2015-08-29	NaT	2.0	4.0	120800.0
19008	19008.0	2771.0	40.0	0.0	C12	0.0	91456.0	2015-08-29	NaT	2.0	4.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
19099	19099.0	2788.0	30.0	0.0	C27	2.0	70254.0	2020-08-06	NaT	2.0	2.0	740280.0
19100	19100.0	2788.0	30.0	0.0	C27	2.0	70254.0	2020-08-06	NaT	2.0	2.0	448370.0
19101	19101.0	2788.0	30.0	0.0	C27	2.0	70254.0	2020-08-06	NaT	2.0	2.0	0.0
19102	19102.0	2788.0	30.0	0.0	C27	2.0	70254.0	2020-08-06	NaT	2.0	2.0	200420.0
19103	19103.0	2788.0	30.0	0.0	C27	2.0	70254.0	2020-08-06	NaT	2.0	2.0	411480.0

100 rows × 20 columns

In [508...

```
df.head()
```

Out[508...

	Unnamed: 0	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Q
0	0.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	NaT	1.0	1.0	2381060.0	
1	1.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	NaT	1.0	1.0	-665480.0	
2	2.0	1.0	28.0	0.0	C23	2.0	57387.0	2018-12-24	2019-11-03	1.0	1.0	0.0	
3	3.0	2.0	31.0	0.0	C7	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0	
4	4.0	2.0	31.0	0.0	C7	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0	

In [509...
```
df.drop(columns=['City'],inplace=True)
```

In [510...
```
df.head()
```

Out[510...

	Unnamed: 0	Driver_ID	Age	Gender	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value	Quarter Ratir
0	0.0	1.0	28.0	0.0	2.0	57387.0	2018-12-24	NaT	1.0	1.0	2381060.0	2
1	1.0	1.0	28.0	0.0	2.0	57387.0	2018-12-24	NaT	1.0	1.0	-665480.0	2
2	2.0	1.0	28.0	0.0	2.0	57387.0	2018-12-24	2019-11-03	1.0	1.0	0.0	2
3	3.0	2.0	31.0	0.0	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0	1
4	4.0	2.0	31.0	0.0	2.0	67016.0	2020-06-11	NaT	2.0	2.0	0.0	1

In [ ]:
```
```

In [514...
```
df.drop(columns=['Driver_ID', 'Dateofjoining', 'LastWorkingDate'],inplace=True)
```

In [515...
```
df.head()
```

Out[515...

	Unnamed: 0	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	ReportingYear	ReportingMonth	Reporti
0	0.0	28.0	0.0	2.0	57387.0	1.0	1.0	2381060.0	2.0	2019	1	
1	1.0	28.0	0.0	2.0	57387.0	1.0	1.0	-665480.0	2.0	2019	1	
2	2.0	28.0	0.0	2.0	57387.0	1.0	1.0	0.0	2.0	2019	1	
3	3.0	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2020	1	
4	4.0	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2020	1	

In [511...
```
y = df['Target']
```

In [524...
```
y.value_counts()
```

Out[524...
```
Target
0    17488
1     1616
Name: count, dtype: int64
```

As we see, there is class imbalance problem in the dataset ; we will use SMOTE for balancing the dataset

In [516...
```
# Drop column target
X = df.drop(columns=['Target'])
```

In [517...
```
X.head()
```

Out[517…

	Unnamed: 0	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	ReportingYear	ReportingMonth	Reporti
0	0.0	28.0	0.0	2.0	57387.0	1.0	1.0	2381060.0	2.0	2019	1	
1	1.0	28.0	0.0	2.0	57387.0	1.0	1.0	-665480.0	2.0	2019	1	
2	2.0	28.0	0.0	2.0	57387.0	1.0	1.0	0.0	2.0	2019	1	
3	3.0	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2020	1	
4	4.0	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2020	1	

In [521…
```
X.shape
```

Out[521… (19104, 15)

In [522…
```
missing_values = X.isnull().sum()
print(missing_values)
```

```
Unnamed: 0             0
Age                    0
Gender                 0
Education_Level        0
Income                 0
Joining Designation    0
Grade                  0
Total Business Value   0
Quarterly Rating       0
ReportingYear          0
ReportingMonth         0
ReportingDay           0
City_Target_Encoded    0
Rating_Increase        0
Income_Increase        0
dtype: int64
```

In [518…
```
smote = SMOTE(random_state=42)
```

In [525…
```
# Apply SMOTE to create balanced X and y
X_resampled, y_resampled = smote.fit_resample(X, y)
```

In [526…
```
X_resampled.shape
```

Out[526… (34976, 15)

In [528…
```
y_resampled.value_counts()
```

Out[528…
```
Target
0    17488
1    17488
Name: count, dtype: int64
```

In [ ]:
```
# Splitting the data into train-val-test data sets
```

In [552…
```
# Split the data into 70% train and 30% temporary dataset (which will be split further into val and test)
X_train, X_temp, y_train, y_temp = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)

# Split the temporary dataset into 50% validation and 50% test (15% each of the original dataset)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

In [ ]:
```

```

In [553…
```
# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit the scaler on the training data and transform it
X_train_scaled = scaler.fit_transform(X_train)

# Transform the validation and test sets using the fitted scaler
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

Decision Trees

In [554…
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
```

In [555…
```
# Initialize the decision tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Train the classifier on the training data
dt_classifier.fit(X_train_scaled, y_train)
```

Out[555…
```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [556…
```
# Predict on the validation set
y_val_pred = dt_classifier.predict(X_val_scaled)

# Predict on the test set
y_test_pred = dt_classifier.predict(X_test_scaled)
```

In [557…
```
# Validation set performance
print("Validation Set Performance:")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Classification Report:\n", classification_report(y_val, y_val_pred))
```

```
Validation Set Performance:
Accuracy: 0.9096454441479223
Confusion Matrix:
 [[2420  233]
 [ 241 2352]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91      2653
           1       0.91      0.91      0.91      2593

    accuracy                           0.91      5246
   macro avg       0.91      0.91      0.91      5246
weighted avg       0.91      0.91      0.91      5246
```

In [558…

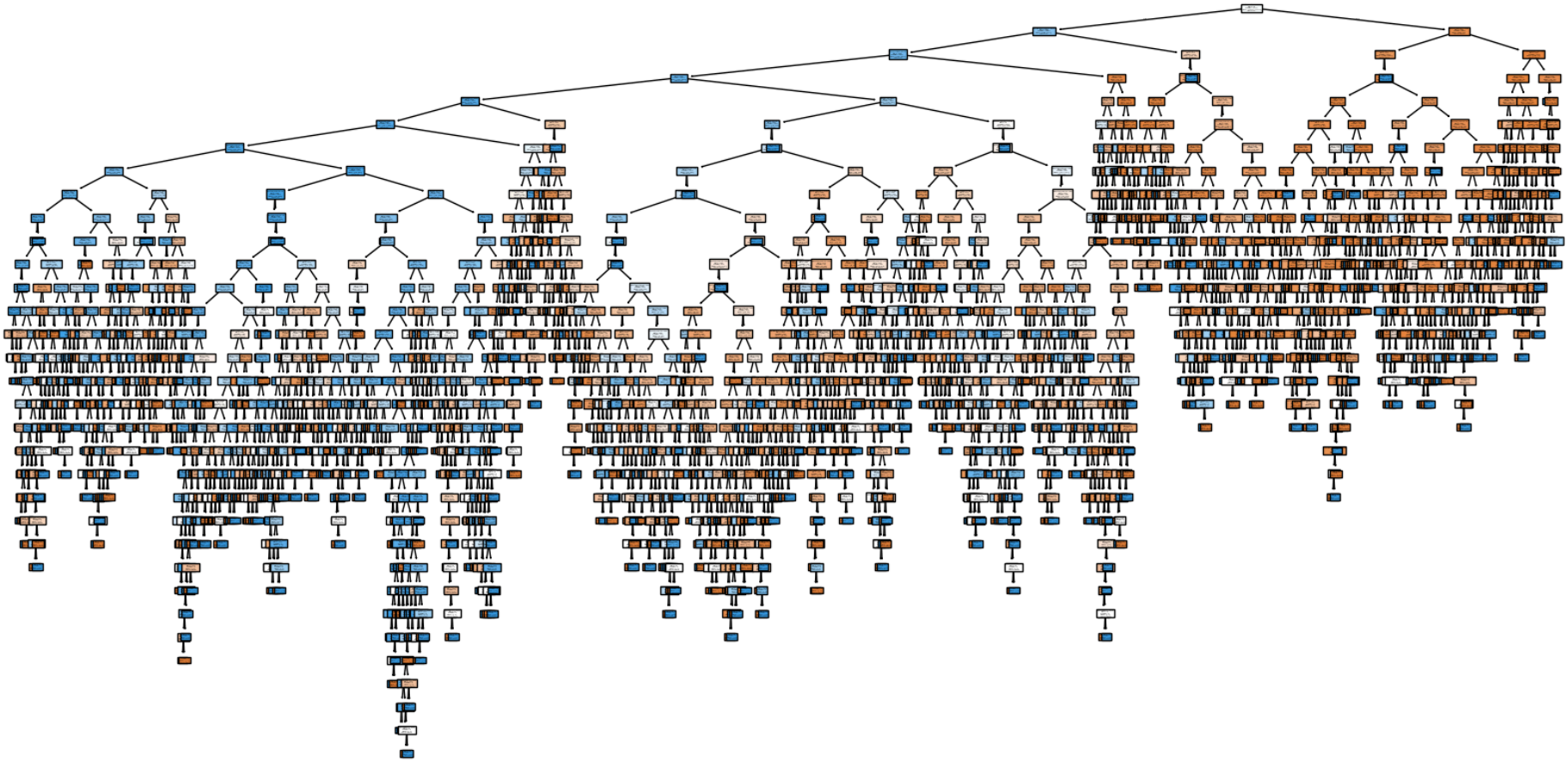
```
# Test set performance
print("Test Set Performance:")
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
Test Set Performance:
Accuracy: 0.9212883552506194
Confusion Matrix:
 [[2409  214]
 [ 199 2425]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      2623
           1       0.92      0.92      0.92      2624

    accuracy                           0.92      5247
   macro avg       0.92      0.92      0.92      5247
weighted avg       0.92      0.92      0.92      5247
```

In [559…

```
from sklearn import tree
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(20,10))
tree.plot_tree(dt_classifier, filled=True, feature_names=df.columns[:-1], class_names=["Stayed", "Left"])
plt.show()
```



plain vanilla decision tree classifier with default parameters and hyperparameters gives Accuracy: 0.92128and f1-score of 0.92

from the visualization of tree diagram we can see that it is an overfit model

In [594…

```
# Initialize the decision tree classifier
dtc1 = DecisionTreeClassifier(random_state=42,
                              max_depth=10,
```

```
                                min_samples_split=8,
                                min_samples_leaf=4,
                                max_features=None,
                                max_leaf_nodes=None,
                                min_impurity_decrease=0.0,
                                class_weight=None,
                                ccp_alpha=0.0,
                                monotonic_cst=None)
```

In [595…

```
# Train the decision tree classifier on the training data
dtc1.fit(X_train_scaled, y_train)
```

Out[595…

DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=4, min_samples_split=8,
                       random_state=42)
```

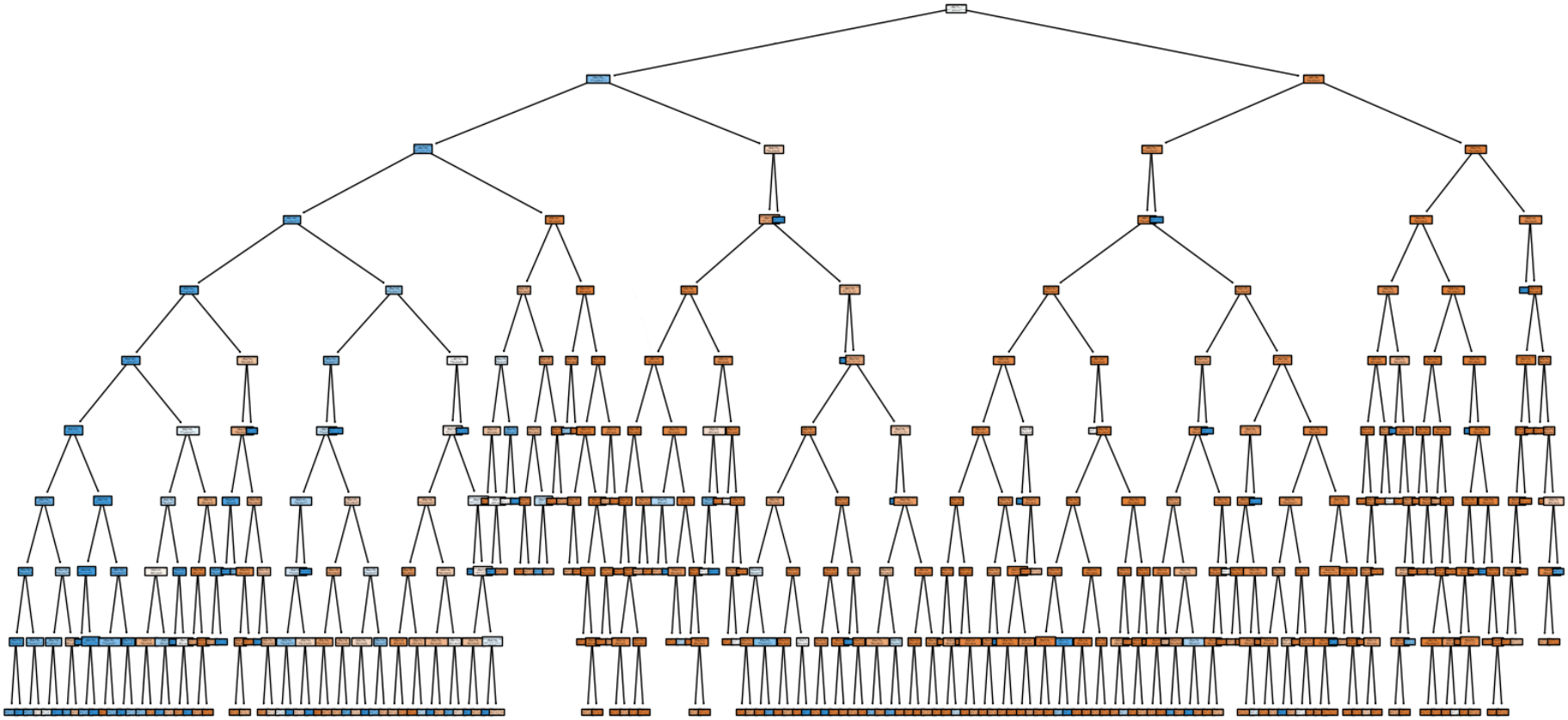
In [596…

```
# Predict on the validation set
y_val_pred = dtc1.predict(X_val_scaled)

# Predict on the test set
y_test_pred = dtc1.predict(X_test_scaled)

# Evaluate performance on the validation set
print("Validation Set Performance:")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Classification Report:\n", classification_report(y_val, y_val_pred))

# Evaluate performance on the test set
print("Test Set Performance:")
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
Validation Set Performance:
Accuracy: 0.9018299656881433
Confusion Matrix:
 [[2403  250]
 [ 265 2328]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.91      0.90      2653
           1       0.90      0.90      0.90      2593

    accuracy                           0.90      5246
   macro avg       0.90      0.90      0.90      5246
weighted avg       0.90      0.90      0.90      5246

Test Set Performance:
Accuracy: 0.9108061749571184
Confusion Matrix:
 [[2370  253]
 [ 215 2409]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.90      0.91      2623
           1       0.90      0.92      0.91      2624

    accuracy                           0.91      5247
   macro avg       0.91      0.91      0.91      5247
weighted avg       0.91      0.91      0.91      5247
```

In [597…

```
from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
tree.plot_tree(dtc1, filled=True, feature_names=df.columns[:-1], class_names=["Stayed", "Left"])
plt.show()
```



In [ ]:

In [590…

```
# Initialize the decision tree classifier
dtc2 = DecisionTreeClassifier(random_state=42,
                              max_depth=8,
                              min_samples_split=4,
                              min_samples_leaf=2,
                              max_features=None,
                              max_leaf_nodes=None,
                              min_impurity_decrease=0,
                              class_weight=None,
```

```
                                ccp_alpha=0.0,
                                monotonic_cst=None)
```

In [591…

```
# Train the decision tree classifier on the training data
dtc2.fit(X_train_scaled, y_train)
```

Out[591…

DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=8, min_impurity_decrease=0, min_samples_leaf=2,
                       min_samples_split=4, random_state=42)
```

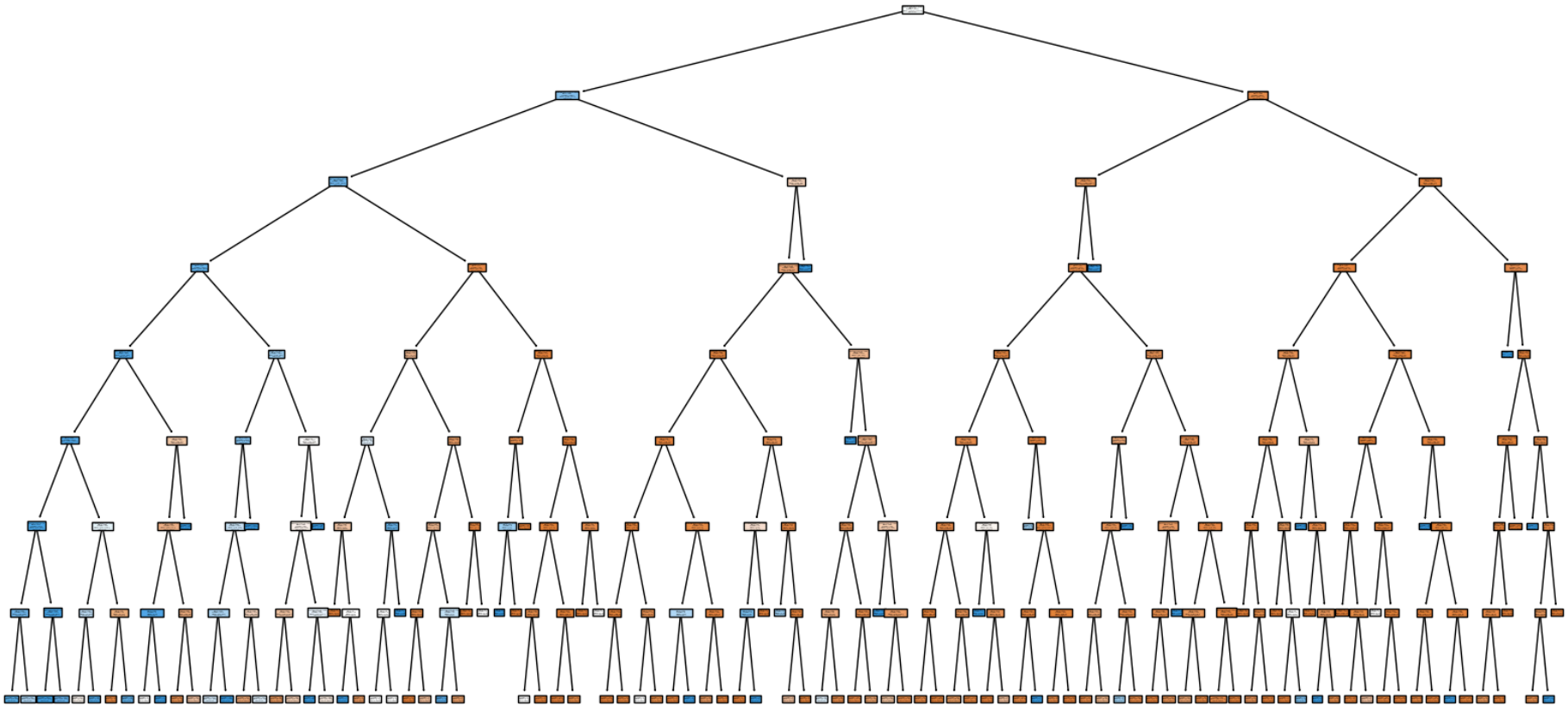
In [592…

```
# Predict on the validation set
y_val_pred = dtc2.predict(X_val_scaled)

# Predict on the test set
y_test_pred = dtc2.predict(X_test_scaled)

# Evaluate performance on the validation set
print("Validation Set Performance:")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Classification Report:\n", classification_report(y_val, y_val_pred))

# Evaluate performance on the test set
print("Test Set Performance:")
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
Validation Set Performance:
Accuracy: 0.8898208158597026
Confusion Matrix:
 [[2321  332]
 [ 246 2347]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.87      0.89      2653
           1       0.88      0.91      0.89      2593

    accuracy                           0.89      5246
   macro avg       0.89      0.89      0.89      5246
weighted avg       0.89      0.89      0.89      5246

Test Set Performance:
Accuracy: 0.8944158566800077
Confusion Matrix:
 [[2278  345]
 [ 209 2415]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.87      0.89      2623
           1       0.88      0.92      0.90      2624

    accuracy                           0.89      5247
   macro avg       0.90      0.89      0.89      5247
weighted avg       0.90      0.89      0.89      5247
```

In [593…

```
from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
tree.plot_tree(dtc2, filled=True, feature_names=df.columns[:-1], class_names=["Stayed", "Left"])
plt.show()
```



In [ ]:

In [586…

```
# Initialize the decision tree classifier
dtc3 = DecisionTreeClassifier(random_state=42,
                              max_depth=6,
                              min_samples_split=4,
                              min_samples_leaf=2,
                              max_features=None,
                              max_leaf_nodes=None,
                              min_impurity_decrease=0,
                              class_weight=None,
```

```
                                ccp_alpha=0.0,
                                monotonic_cst=None)
```

In [587…

```
# Train the decision tree classifier on the training data
dtc3.fit(X_train_scaled, y_train)
```

Out[587…

DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=6, min_impurity_decrease=0, min_samples_leaf=2,
                       min_samples_split=4, random_state=42)
```

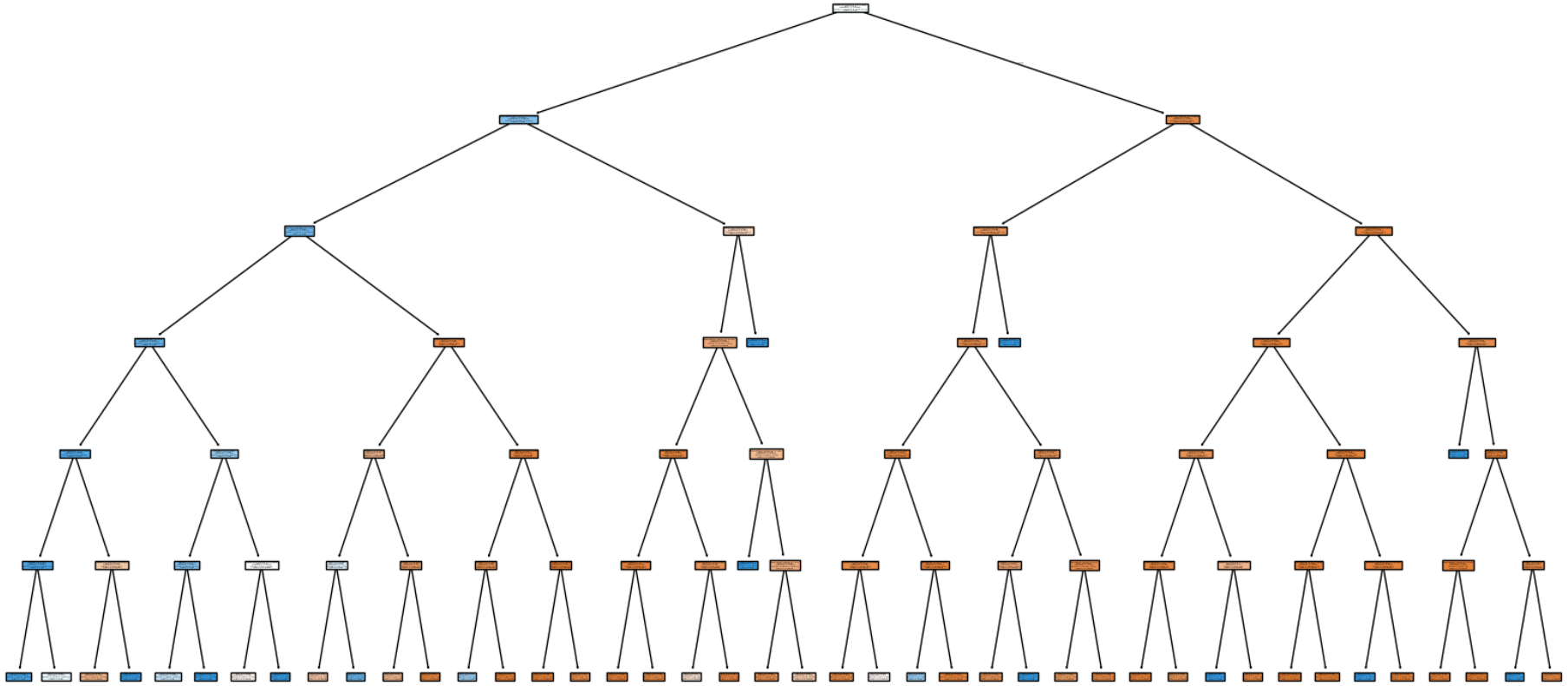
In [588…

```
# Predict on the validation set
y_val_pred = dtc3.predict(X_val_scaled)

# Predict on the test set
y_test_pred = dtc3.predict(X_test_scaled)

# Evaluate performance on the validation set
print("Validation Set Performance:")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Classification Report:\n", classification_report(y_val, y_val_pred))

# Evaluate performance on the test set
print("Test Set Performance:")
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
Validation Set Performance:
Accuracy: 0.8715211589782692
Confusion Matrix:
 [[2255  398]
 [ 276 2317]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.85      0.87      2653
           1       0.85      0.89      0.87      2593

    accuracy                           0.87      5246
   macro avg       0.87      0.87      0.87      5246
weighted avg       0.87      0.87      0.87      5246

Test Set Performance:
Accuracy: 0.8726891557080236
Confusion Matrix:
 [[2212  411]
 [ 257 2367]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.84      0.87      2623
           1       0.85      0.90      0.88      2624

    accuracy                           0.87      5247
   macro avg       0.87      0.87      0.87      5247
weighted avg       0.87      0.87      0.87      5247
```

In [589…

```
from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
tree.plot_tree(dtc3, filled=True, feature_names=df.columns[:-1], class_names=["Stayed", "Left"])
plt.show()
```



In [ ]:

In [598…

```
# Initialize the decision tree classifier
dtc4 = DecisionTreeClassifier(random_state=42,
                              max_depth=None,
                              min_samples_split=8,
                              min_samples_leaf=4,
                              max_features=None,
                              max_leaf_nodes=None,
                              min_impurity_decrease=0.05,
                              class_weight=None,
```

```
                                ccp_alpha=0.0,
                                monotonic_cst=None)
```

In [599…

```
# Train the decision tree classifier on the training data
dtc4.fit(X_train_scaled, y_train)
```

Out[599…

DecisionTreeClassifier

```
DecisionTreeClassifier(min_impurity_decrease=0.05, min_samples_leaf=4,
                       min_samples_split=8, random_state=42)
```

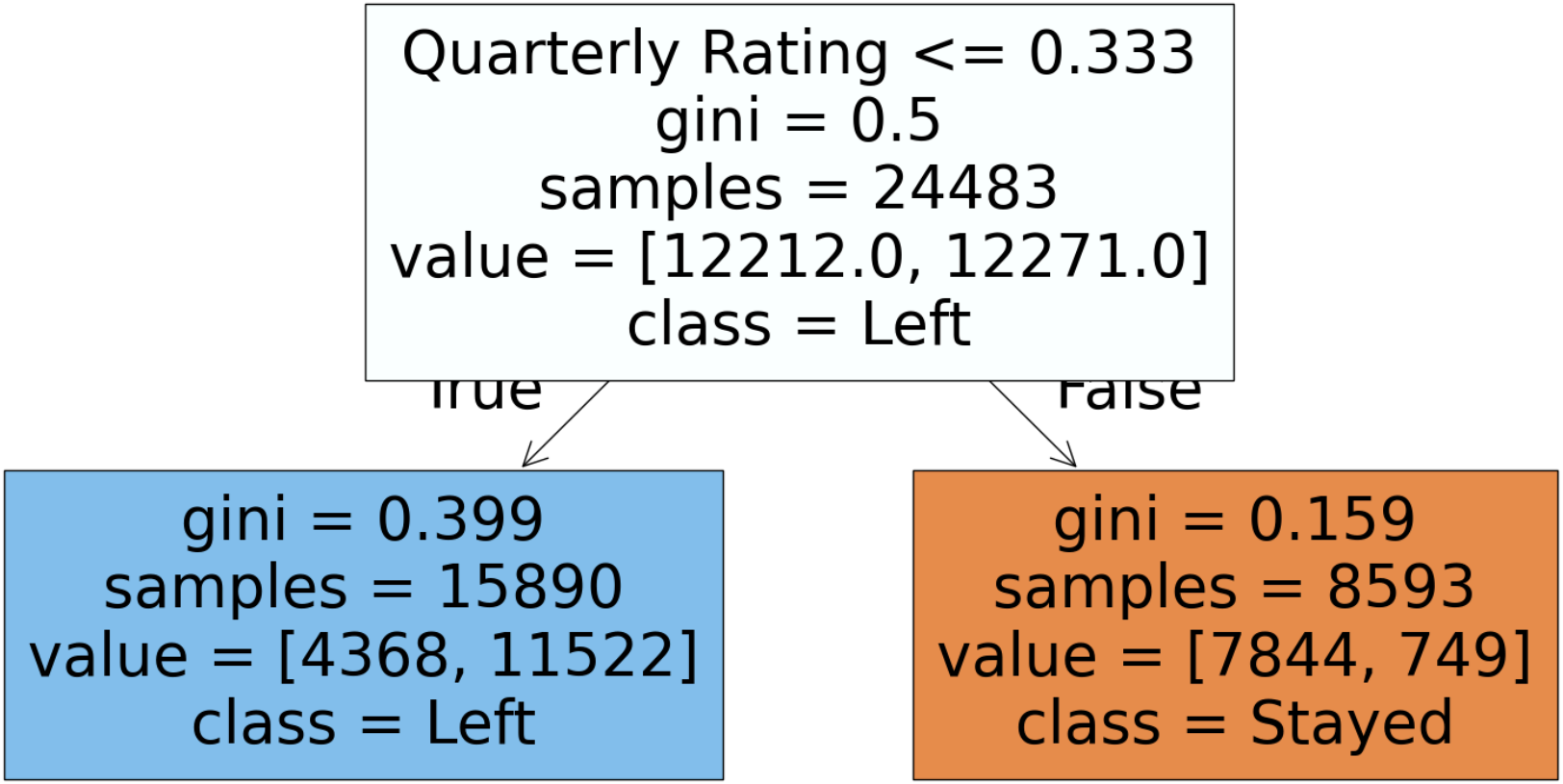
In [600…

```
# Predict on the validation set
y_val_pred = dtc4.predict(X_val_scaled)

# Predict on the test set
y_test_pred = dtc4.predict(X_test_scaled)

# Evaluate performance on the validation set
print("Validation Set Performance:")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Classification Report:\n", classification_report(y_val, y_val_pred))

# Evaluate performance on the test set
print("Test Set Performance:")
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
Validation Set Performance:
Accuracy: 0.7889820815859703
Confusion Matrix:
 [[1699  954]
 [ 153 2440]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.64      0.75      2653
           1       0.72      0.94      0.82      2593

    accuracy                           0.79      5246
   macro avg       0.82      0.79      0.78      5246
weighted avg       0.82      0.79      0.78      5246

Test Set Performance:
Accuracy: 0.7953116066323613
Confusion Matrix:
 [[1698  925]
 [ 149 2475]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.65      0.76      2623
           1       0.73      0.94      0.82      2624

    accuracy                           0.80      5247
   macro avg       0.82      0.80      0.79      5247
weighted avg       0.82      0.80      0.79      5247
```

In [601…

```
from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
tree.plot_tree(dtc4, filled=True, feature_names=df.columns[:-1], class_names=["Stayed", "Left"])
plt.show()
```



In [ ]:

In [602…

```
# Initialize the decision tree classifier
dtc5 = DecisionTreeClassifier(random_state=42,
                              max_depth=None,
                              min_samples_split=8,
                              min_samples_leaf=4,
                              max_features=None,
                              max_leaf_nodes=None,
                              min_impurity_decrease=0.02,
                              class_weight=None,
```

```
                                ccp_alpha=0.0,
                                monotonic_cst=None)
```

In [603…

```
# Train the decision tree classifier on the training data
dtc5.fit(X_train_scaled, y_train)
```

Out[603…

DecisionTreeClassifier

```
DecisionTreeClassifier(min_impurity_decrease=0.02, min_samples_leaf=4,
                       min_samples_split=8, random_state=42)
```

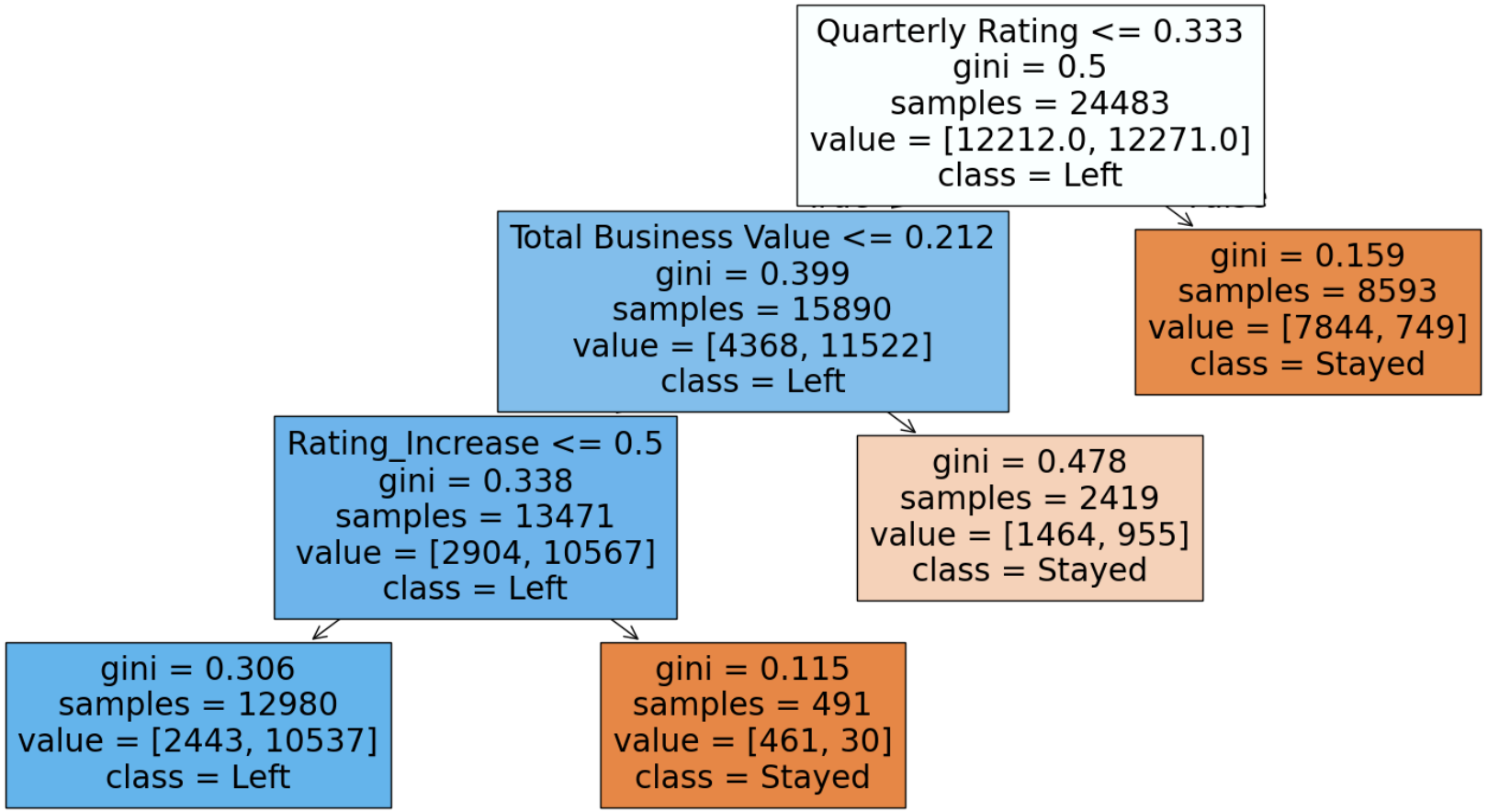
In [604…

```
# Predict on the validation set
y_val_pred = dtc5.predict(X_val_scaled)

# Predict on the test set
y_test_pred = dtc5.predict(X_test_scaled)

# Evaluate performance on the validation set
print("Validation Set Performance:")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Classification Report:\n", classification_report(y_val, y_val_pred))

# Evaluate performance on the test set
print("Test Set Performance:")
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
Validation Set Performance:
Accuracy: 0.8272969881814716
Confusion Matrix:
 [[2109  544]
 [ 362 2231]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.79      0.82      2653
           1       0.80      0.86      0.83      2593

    accuracy                           0.83      5246
   macro avg       0.83      0.83      0.83      5246
weighted avg       0.83      0.83      0.83      5246

Test Set Performance:
Accuracy: 0.8328568705927196
Confusion Matrix:
 [[2091  532]
 [ 345 2279]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.80      0.83      2623
           1       0.81      0.87      0.84      2624

    accuracy                           0.83      5247
   macro avg       0.83      0.83      0.83      5247
weighted avg       0.83      0.83      0.83      5247
```

In [605…

```
from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
tree.plot_tree(dtc5, filled=True, feature_names=df.columns[:-1], class_names=["Stayed", "Left"])
plt.show()
```



In [ ]:

In [606…

```
# Initialize the decision tree classifier
dtc6 = DecisionTreeClassifier(random_state=42,
                              max_depth=None,
                              min_samples_split=8,
                              min_samples_leaf=4,
                              max_features=None,
                              max_leaf_nodes=None,
                              min_impurity_decrease=0.005,
                              class_weight=None,
```

```
                                ccp_alpha=0.0,
                                monotonic_cst=None)
```

In [607…

```
# Train the decision tree classifier on the training data
dtc6.fit(X_train_scaled, y_train)
```

Out[607…

DecisionTreeClassifier

```
DecisionTreeClassifier(min_impurity_decrease=0.005, min_samples_leaf=4,
                       min_samples_split=8, random_state=42)
```

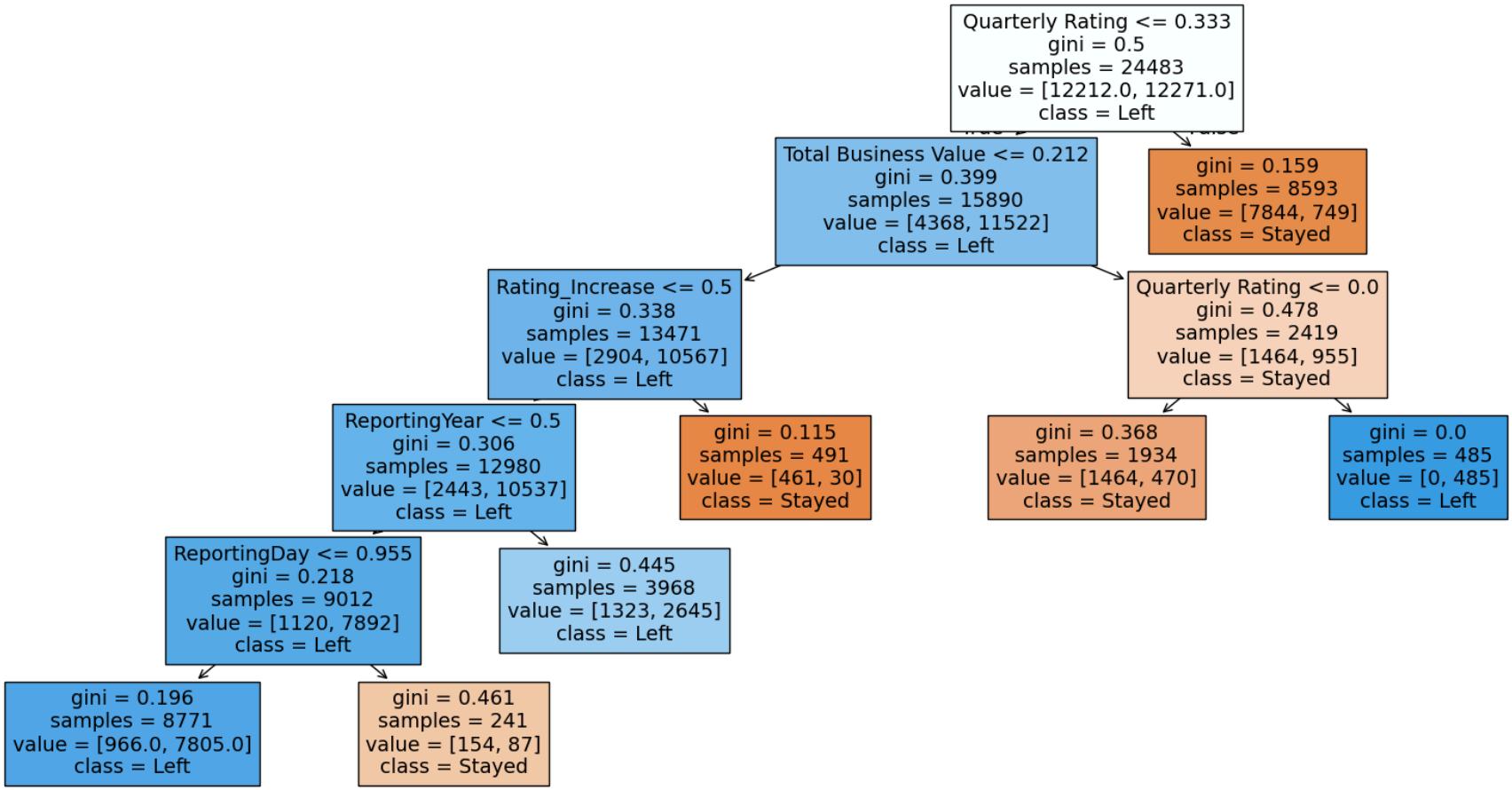
In [608…

```
# Predict on the validation set
y_val_pred = dtc6.predict(X_val_scaled)

# Predict on the test set
y_test_pred = dtc6.predict(X_test_scaled)

# Evaluate performance on the validation set
print("Validation Set Performance:")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Classification Report:\n", classification_report(y_val, y_val_pred))

# Evaluate performance on the test set
print("Test Set Performance:")
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
Validation Set Performance:
Accuracy: 0.848455966450629
Confusion Matrix:
 [[2142  511]
 [ 284 2309]]
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.81      0.84      2653
           1       0.82      0.89      0.85      2593

    accuracy                           0.85      5246
   macro avg       0.85      0.85      0.85      5246
weighted avg       0.85      0.85      0.85      5246

Test Set Performance:
Accuracy: 0.8517247951210215
Confusion Matrix:
 [[2116  507]
 [ 271 2353]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.81      0.84      2623
           1       0.82      0.90      0.86      2624

    accuracy                           0.85      5247
   macro avg       0.85      0.85      0.85      5247
weighted avg       0.85      0.85      0.85      5247
```

In [609…

```
from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
tree.plot_tree(dtc6, filled=True, feature_names=df.columns[:-1], class_names=["Stayed", "Left"])
plt.show()
```



In [ ]:

In [610…

```
# Initialize the decision tree classifier
dtc7 = DecisionTreeClassifier(random_state=42,
                              max_depth=None,
                              min_samples_split=8,
                              min_samples_leaf=4,
                              max_features=None,
                              max_leaf_nodes=None,
                              min_impurity_decrease=0.0005,
                              class_weight=None,
```

```
                                ccp_alpha=0.0,
                                monotonic_cst=None)
```

In [611…

```
# Train the decision tree classifier on the training data
dtc7.fit(X_train_scaled, y_train)
```

Out[611…

DecisionTreeClassifier

```
DecisionTreeClassifier(min_impurity_decrease=0.0005, min_samples_leaf=4,
                       min_samples_split=8, random_state=42)
```

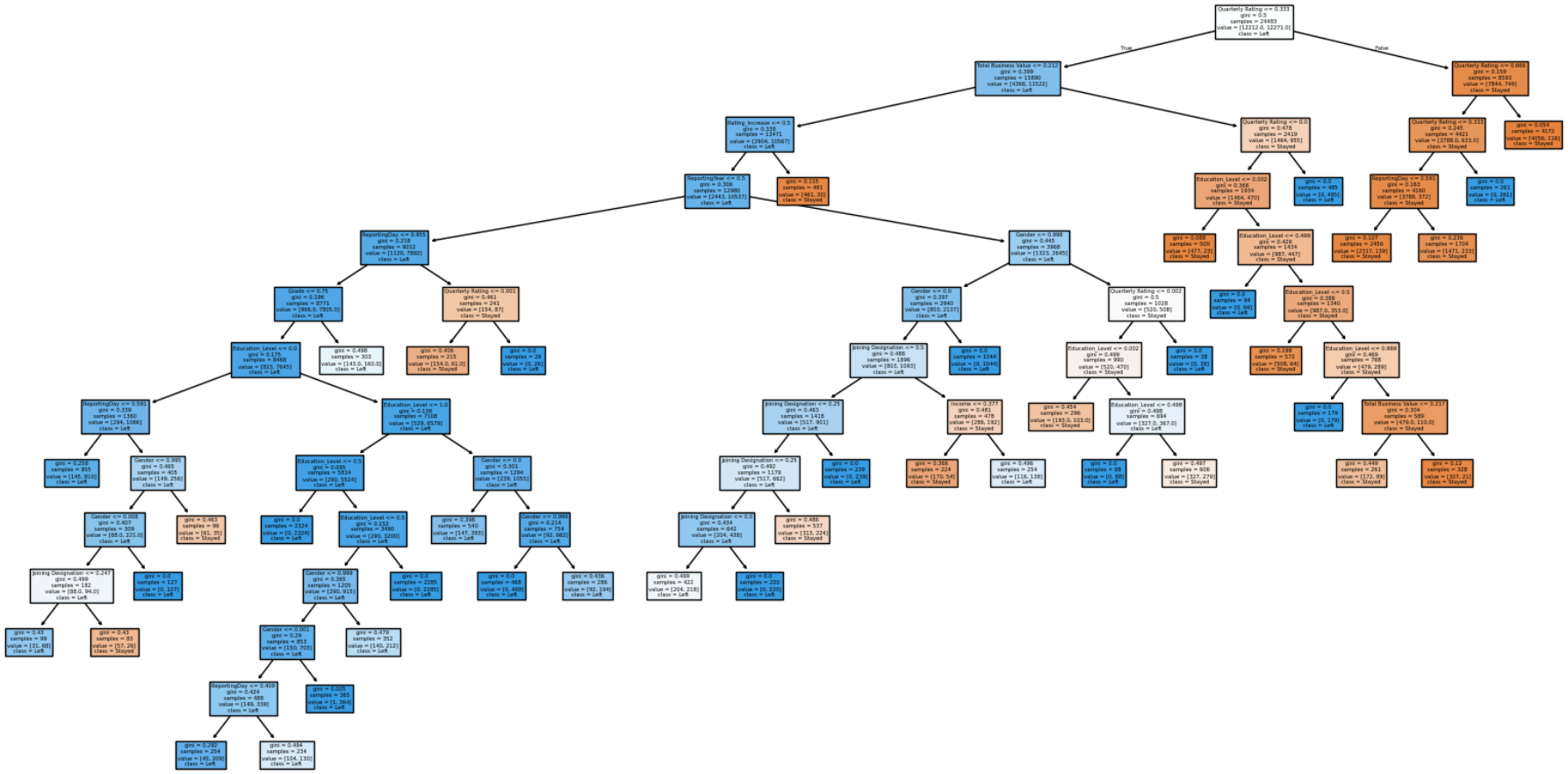
In [612…

```
# Predict on the validation set
y_val_pred = dtc7.predict(X_val_scaled)

# Predict on the test set
y_test_pred = dtc7.predict(X_test_scaled)

# Evaluate performance on the validation set
print("Validation Set Performance:")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Classification Report:\n", classification_report(y_val, y_val_pred))

# Evaluate performance on the test set
print("Test Set Performance:")
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
Validation Set Performance:
Accuracy: 0.8867708730461303
Confusion Matrix:
 [[2391  262]
 [ 332 2261]]
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.90      0.89      2653
           1       0.90      0.87      0.88      2593

    accuracy                           0.89      5246
   macro avg       0.89      0.89      0.89      5246
weighted avg       0.89      0.89      0.89      5246

Test Set Performance:
Accuracy: 0.8871736230226797
Confusion Matrix:
 [[2343  280]
 [ 312 2312]]
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.89      0.89      2623
           1       0.89      0.88      0.89      2624

    accuracy                           0.89      5247
   macro avg       0.89      0.89      0.89      5247
weighted avg       0.89      0.89      0.89      5247
```

In [613…

```
from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
tree.plot_tree(dtc7, filled=True, feature_names=df.columns[:-1], class_names=["Stayed", "Left"])
plt.show()
```



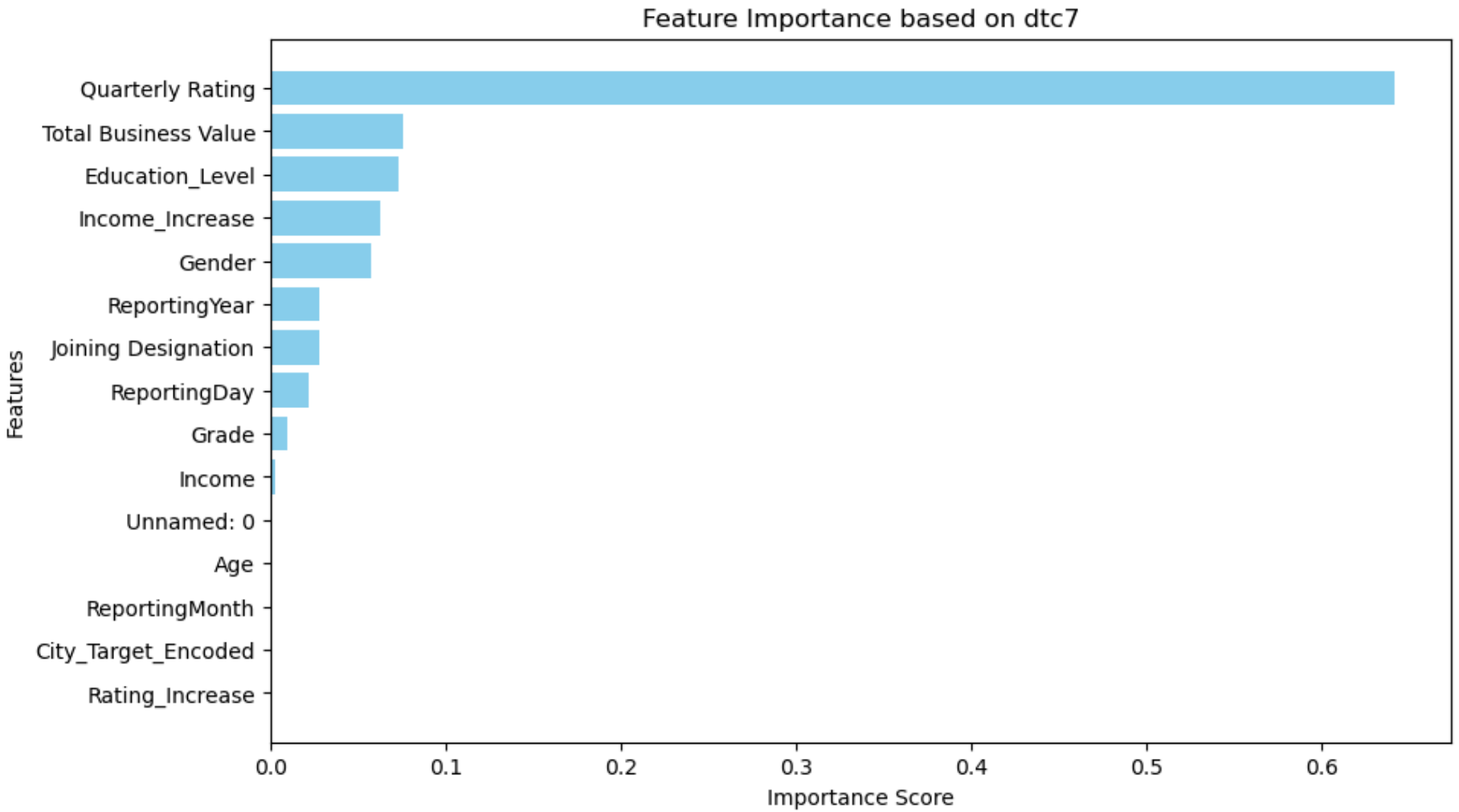
In [687…

```
features = df.drop(columns=["Target"]).columns

# Get feature importances from the decision tree
importances = dtc7.feature_importances_

# Create a DataFrame for better visualization
importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': importances
})
```

```
# Sort the DataFrame by importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)
```

In [688…

```
# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.title('Feature Importance based on dtc7')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.gca().invert_yaxis()  # Highest importance at the top
plt.show()
```



"min_impurity_decrease" looks like a better hyperparameter compared to others (like max_depth, min_samples_split, min_samples_leaf etc)

dtc7 looks to be a good choice with accuracy=0.887 and f1-score=0.88 and also explainability will be good

In [675…

```
!pip install pydotplus
```

```
Collecting pydotplus
  Downloading pydotplus-2.0.2.tar.gz (278 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: pyparsing>=2.0.1 in c:\users\dell\anaconda3\lib\site-packages (from pydotplus) (3.0.9)
Building wheels for collected packages: pydotplus
  Building wheel for pydotplus (setup.py): started
  Building wheel for pydotplus (setup.py): finished with status 'done'
  Created wheel for pydotplus: filename=pydotplus-2.0.2-py3-none-any.whl size=24578 sha256=cccd2582f82d559bb77dbfc58fbb2e7752a8
032ac847558090ab31faf43318eb
  Stored in directory: c:\users\dell\appdata\local\pip\cache\wheels\bd\ce\e8\ff9d9c699514922f57caa22fbd55b0a32761114b4c4acc9e03
Successfully built pydotplus
Installing collected packages: pydotplus
Successfully installed pydotplus-2.0.2
```

In [678…
```
!pip install graphviz pydotplus
```

```
Requirement already satisfied: graphviz in c:\users\dell\anaconda3\lib\site-packages (0.20.3)
Requirement already satisfied: pydotplus in c:\users\dell\anaconda3\lib\site-packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in c:\users\dell\anaconda3\lib\site-packages (from pydotplus) (3.0.9)
```

In [680…
```
!choco install graphviz
```

```
'choco' is not recognized as an internal or external command,
operable program or batch file.
```

In [681…
```
!dot -version
```

```
'dot' is not recognized as an internal or external command,
operable program or batch file.
```

In [ ]:
```

```

In [ ]:
```

```

In [684…
```
from sklearn.tree import export_graphviz
import pydotplus
from IPython.display import Image
```

In [685…
```
# Export the decision tree to a DOT file
dot_data = export_graphviz(
    dtc7,
    out_file=None,  # Output as a string
    filled=True,
    rounded=True,
    special_characters=True,
    feature_names=df.drop(columns=["Target"]).columns,
    class_names=["Class 0", "Class 1"]  # Replace with actual class names
)

# Use pydotplus to create a graph from the DOT data
graph = pydotplus.graph_from_dot_data(dot_data)

# Save the graph as a PDF
graph.write_pdf("decision_tree_dtc7.pdf")
```

Out[685… True

In [ ]:
```
# pdf exported file of this decision tree can help in defining the decision rules... This pdf will be shared in teh same folde
```

In [ ]:
```

```

In [ ]:
```

```

Bagging (Random Forest)

In [614…
```
# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)
```

In [615…
```
# Train the Random Forest classifier on the training data
rf_classifier.fit(X_train_scaled, y_train)
```

Out[615…

RandomForestClassifier

RandomForestClassifier(random_state=42)

In [616…

```
# Predict on the validation set
y_val_pred = rf_classifier.predict(X_val_scaled)
```

In [617…

```
# Predict on the test set
y_test_pred = rf_classifier.predict(X_test_scaled)
```

In [618…

```
# Validation set performance
print("Validation Set Performance:")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Classification Report:\n", classification_report(y_val, y_val_pred))
```

```
Validation Set Performance:
Accuracy: 0.9365230651925276
Confusion Matrix:
 [[2546  107]
 [ 226 2367]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94      2653
           1       0.96      0.91      0.93      2593

    accuracy                           0.94      5246
   macro avg       0.94      0.94      0.94      5246
weighted avg       0.94      0.94      0.94      5246
```

In [619…

```
# Test set performance
print("Test Set Performance:")
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
print("Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
Test Set Performance:
Accuracy: 0.942062130741376
Confusion Matrix:
 [[2513  110]
 [ 194 2430]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.96      0.94      2623
           1       0.96      0.93      0.94      2624

    accuracy                           0.94      5247
   macro avg       0.94      0.94      0.94      5247
weighted avg       0.94      0.94      0.94      5247
```

plain vanilla Random Forest classifier: accuracy: 0.942062130741376; f1-score:0.94 (Test set based); scores are already good and even after hyperparameter tuning and using xgboost , lightGBM and stacking did not result in significant improvement in the model performance.

In [ ]:

In [620…
```
from sklearn.model_selection import GridSearchCV
```

In [621…
```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

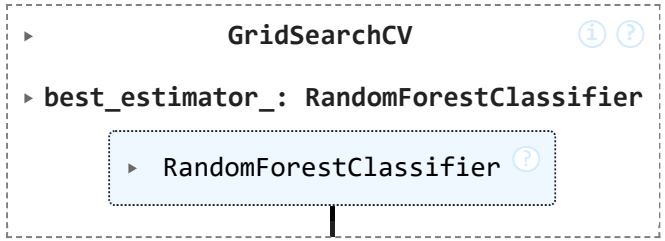
In [622…
```
# Initialize the GridSearchCV object
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2, scoring='accuracy')

# Fit Grid Search on the training data
grid_search.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
```

```
C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py:540: FitFailedWarning:
540 fits failed out of a total of 1620.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
540 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\base.py", line 1466, in wrapper
    estimator._validate_params()
  File "C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\utils\_param_validation.py", line 95, in validate_parameter_constrain
ts
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of RandomForestClassifier must be an int in
the range [1, inf), a float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:1102: UserWarning: One or more of the test scores
are non-finite: [       nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.93468926 0.93526111 0.9368132
 0.93542455 0.93628228 0.93730345 0.93419925 0.93505693 0.93538367
 0.93562888 0.93615985 0.93607804 0.93473018 0.93493439 0.93587381
 0.93407672 0.93436254 0.9354654  0.93187108 0.93321905 0.93399503
 0.93187108 0.93321905 0.93399503 0.9318711  0.93281053 0.93338237
 0.93468926 0.93526111 0.9368132  0.93542455 0.93628228 0.93730345
 0.93419925 0.93505693 0.93538367 0.93562888 0.93615985 0.93607804
 0.93473018 0.93493439 0.93587381 0.93407672 0.93436254 0.9354654
 0.93187108 0.93321905 0.93399503 0.93187108 0.93321905 0.93399503
 0.9318711  0.93281053 0.93338237        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
```

```
       nan        nan        nan        nan        nan        nan
0.91238827 0.91287835 0.91336854 0.9102644  0.91169391 0.91255169
0.91144869 0.91238831 0.91332769 0.9102642  0.91202063 0.91181652
0.91169396 0.91218413 0.91320528 0.91140794 0.91214313 0.91340935
0.90912076 0.91022355 0.9115714  0.90912076 0.91022355 0.9115714
0.91116279 0.91112211 0.91210236 0.91238827 0.91287835 0.91336854
0.9102644  0.91169391 0.91255169 0.91144869 0.91238831 0.91332769
0.9102642  0.91202063 0.91181652 0.91169396 0.91218413 0.91320528
0.91140794 0.91214313 0.91340935 0.90912076 0.91022355 0.9115714
0.90912076 0.91022355 0.9115714  0.91116279 0.91112211 0.91210236
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan 0.93366823 0.93428084 0.93436258
0.9340767  0.93468941 0.93554712 0.93362741 0.93477105 0.93485276
0.93370909 0.9346894  0.93473023 0.93330068 0.9337908  0.93477105
0.93338238 0.93436269 0.93452601 0.93170782 0.93305559 0.93399502
0.93170782 0.93305559 0.93399502 0.93097259 0.93248381 0.93293305
0.93366823 0.93428084 0.93436258 0.9340767  0.93468941 0.93554712
0.93362741 0.93477105 0.93485276 0.93370909 0.9346894  0.93473023
0.93330068 0.9337908  0.93477105 0.93338238 0.93436269 0.93452601
0.93170782 0.93305559 0.93399502 0.93170782 0.93305559 0.93399502
0.93097259 0.93248381 0.93293305        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.93468936 0.9355879  0.93652737 0.93562875 0.93693585 0.93738518
0.93436263 0.93489356 0.93562877 0.93522043 0.93628239 0.9358738
0.93534288 0.93513857 0.93603721 0.93403594 0.93493441 0.93575131
0.93183023 0.93330073 0.93391337 0.93183023 0.93330073 0.93391337
0.93183025 0.93285136 0.93334153 0.93468936 0.9355879  0.93652737
0.93562875 0.93693585 0.93738518 0.93436263 0.93489356 0.93562877
0.93522043 0.93628239 0.9358738  0.93534288 0.93513857 0.93603721
0.93403594 0.93493441 0.93575131 0.93183023 0.93330073 0.93391337
0.93183023 0.93330073 0.93391337 0.93183025 0.93285136 0.93334153]
 warnings.warn(
```

Out[622…



In [623…

```
# Get the best parameters found by GridSearchCV
best_params = grid_search.best_params_
print("Best Parameters found by Grid Search:", best_params)

# Evaluate on the validation set
y_val_pred = grid_search.predict(X_val_scaled)
print("Validation Set Performance After Grid Search:")
print("Accuracy:", accuracy_score(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))
print("Classification Report:\n", classification_report(y_val, y_val_pred))
```

```
Best Parameters found by Grid Search: {'max_depth': 30, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5,
'n_estimators': 200}
Validation Set Performance After Grid Search:
Accuracy: 0.9363324437666793
Confusion Matrix:
 [[2545  108]
 [ 226 2367]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94      2653
           1       0.96      0.91      0.93      2593

    accuracy                           0.94      5246
   macro avg       0.94      0.94      0.94      5246
weighted avg       0.94      0.94      0.94      5246
```

Random Forest classifier: gridsearchCV accuracy: 0.9363324437666793; f1-score:0.94 (slightly better than randomisedsearchCV)

In [ ]:

Hyper parameter tuning using randomizedsearchCV

In [624…
```
from sklearn.model_selection import RandomizedSearchCV
```

In [625…
```
import time
```

In [626…
```
param_dist = {
    'n_estimators': np.arange(50, 300, 50),
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': np.arange(2, 11),
    'min_samples_leaf': np.arange(1, 5),
    'max_features': ['auto', 'sqrt', 'log2']
}
```

In [627…
```
# Start the timer
start_time = time.time()
```

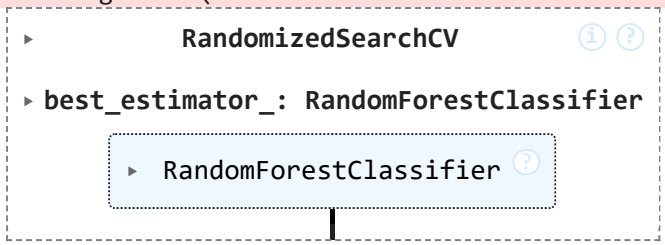
In [628…
```
# Initialize the RandomizedSearchCV object
random_search = RandomizedSearchCV(estimator=rf_classifier, param_distributions=param_dist, n_iter=100, cv=5, n_jobs=-1, verbo

# Fit Random Search on the training data
random_search.fit(X_train_scaled, y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py:540: FitFailedWarning:
165 fits failed out of a total of 500.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
165 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\base.py", line 1466, in wrapper
    estimator._validate_params()
  File "C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\utils\_param_validation.py", line 95, in validate_parameter_constrain
ts
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of RandomForestClassifier must be an int in
the range [1, inf), a float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:1102: UserWarning: One or more of the test scores
are non-finite: [       nan 0.93554712 0.93309649 0.93321905        nan 0.9113263
 0.93473023        nan 0.93473023        nan 0.93615975 0.93591476
 0.93370911        nan 0.93424005 0.91206146 0.93456681        nan
 0.93240202 0.91136721 0.93346406 0.93399503 0.93411754 0.93321905
        nan 0.93509782        nan        nan 0.9349345  0.91185727
 0.93517949 0.93342316        nan        nan        nan 0.93660901
        nan 0.93183023        nan 0.93379083        nan 0.93513864
        nan 0.93513864 0.93338237 0.93379078 0.9113263  0.93456682
 0.93444428 0.93321905 0.93399503 0.93558797 0.9368132  0.93538367
 0.93583302        nan 0.93187108 0.93481185 0.93693587        nan
        nan 0.93615973        nan 0.93436267        nan 0.91136721
        nan 0.93456689 0.93485278 0.93424005 0.93615975 0.93330073
 0.91238821 0.93697667        nan        nan        nan        nan
        nan 0.93603721 0.91214318        nan 0.91238828 0.9115306
 0.93444442 0.93334144 0.91291923        nan        nan 0.93501607
 0.93607804 0.93603721        nan 0.93473023 0.93485276        nan
```

```
0.91226577 0.91022355 0.93399503        nan]
 warnings.warn(
```

Out[628…



In [629…

```
# Stop the timer
end_time = time.time()
```

In [630…

```
# Calculate the elapsed time
elapsed_time = end_time - start_time
print(f"Random Search completed in: {elapsed_time:.2f} seconds")
```

```
Random Search completed in: 919.48 seconds
```

In [631…

```
# Get the best parameters found by RandomizedSearchCV
best_params_random = random_search.best_params_
print("Best Parameters found by Random Search:", best_params_random)

# Evaluate on the validation set
y_val_pred_random = random_search.predict(X_val_scaled)
print("Validation Set Performance After Random Search:")
print("Accuracy:", accuracy_score(y_val, y_val_pred_random))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred_random))
print("Classification Report:\n", classification_report(y_val, y_val_pred_random))
```

```
Best Parameters found by Random Search: {'n_estimators': 250, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'l
og2', 'max_depth': 50}
Validation Set Performance After Random Search:
Accuracy: 0.9361418223408311
Confusion Matrix:
 [[2543  110]
 [ 225 2368]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94      2653
           1       0.96      0.91      0.93      2593

    accuracy                           0.94      5246
   macro avg       0.94      0.94      0.94      5246
weighted avg       0.94      0.94      0.94      5246
```

Random Forest classifier: RandomisedsearchCV Accuracy: 0.9361418223408311; f1-score: 0.94

Boosting using xgboost

In [632…
```
import xgboost as xgb
```

In [633…
```
# Initialize the XGBoost classifier
xgb_classifier = xgb.XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='mlogloss')
```

In [634…
```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 6, 10],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.2]
}
```

In [635…

```
# Initialize GridSearchCV
grid_search_xgb = GridSearchCV(estimator=xgb_classifier, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2, scoring='accuracy'

# Start the timer
start_time = time.time()

# Fit Grid Search on the training data
grid_search_xgb.fit(X_train_scaled, y_train)

# Stop the timer
end_time = time.time()

# Calculate the elapsed time
elapsed_time_grid = end_time - start_time
print(f"GridSearchCV completed in: {elapsed_time_grid:.2f} seconds")

# Get the best parameters and evaluate
best_params_grid = grid_search_xgb.best_params_
print("Best Parameters found by Grid Search:", best_params_grid)

y_val_pred_grid = grid_search_xgb.predict(X_val_scaled)
print("Validation Set Performance After Grid Search:")
print("Accuracy:", accuracy_score(y_val, y_val_pred_grid))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred_grid))
print("Classification Report:\n", classification_report(y_val, y_val_pred_grid))
```

```
Fitting 5 folds for each of 729 candidates, totalling 3645 fits
C:\Users\Dell\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning: [02:00:28] WARNING: C:\buildkite-agent\builds\build
kite-windows-cpu-autoscaling-group-i-0015a694724fa8361-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
GridSearchCV completed in: 1463.44 seconds
Best Parameters found by Grid Search: {'colsample_bytree': 0.8, 'gamma': 0.2, 'learning_rate': 0.2, 'max_depth': 3, 'n_estimato
rs': 200, 'subsample': 1.0}
Validation Set Performance After Grid Search:
Accuracy: 0.9441479222264583
Confusion Matrix:
 [[2603   50]
 [ 243 2350]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.98      0.95      2653
           1       0.98      0.91      0.94      2593

    accuracy                           0.94      5246
   macro avg       0.95      0.94      0.94      5246
weighted avg       0.95      0.94      0.94      5246
```

In [ ]:

In [636…
```
param_dist = {
    'n_estimators': [int(x) for x in np.linspace(start=50, stop=200, num=10)],
    'max_depth': [int(x) for x in np.linspace(3, 10, num=4)],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.2, 0.3]
}
```

In [637…
```
# Initialize RandomizedSearchCV
random_search_xgb = RandomizedSearchCV(estimator=xgb_classifier, param_distributions=param_dist, n_iter=100, cv=5, n_jobs=-1,

# Start the timer
start_time = time.time()

# Fit Randomized Search on the training data
random_search_xgb.fit(X_train_scaled, y_train)

# Stop the timer
```

```
end_time = time.time()

# Calculate the elapsed time
elapsed_time_random = end_time - start_time
print(f"RandomizedSearchCV completed in: {elapsed_time_random:.2f} seconds")

# Get the best parameters and evaluate
best_params_random = random_search_xgb.best_params_
print("Best Parameters found by Random Search:", best_params_random)

y_val_pred_random = random_search_xgb.predict(X_val_scaled)
print("Validation Set Performance After Random Search:")
print("Accuracy:", accuracy_score(y_val, y_val_pred_random))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred_random))
print("Classification Report:\n", classification_report(y_val, y_val_pred_random))
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
C:\Users\Dell\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning: [02:03:30] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-0015a694724fa8361-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
RandomizedSearchCV completed in: 181.46 seconds
Best Parameters found by Random Search: {'subsample': 1.0, 'n_estimators': 116, 'max_depth': 5, 'learning_rate': 0.2, 'gamma': 0.3, 'colsample_bytree': 0.6}
Validation Set Performance After Random Search:
Accuracy: 0.9435760579489134
Confusion Matrix:
 [[2595   58]
 [ 238 2355]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      2653
           1       0.98      0.91      0.94      2593

    accuracy                           0.94      5246
   macro avg       0.95      0.94      0.94      5246
weighted avg       0.95      0.94      0.94      5246
```

In [638…

```
# Evaluate on the test set for final comparison

# Vanilla XGBoost
xgb_classifier.fit(X_train_scaled, y_train)
y_test_pred_vanilla = xgb_classifier.predict(X_test_scaled)
print("Test Set Performance for Vanilla XGBoost:")
print("Accuracy:", accuracy_score(y_test, y_test_pred_vanilla))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred_vanilla))
print("Classification Report:\n", classification_report(y_test, y_test_pred_vanilla))

# Best model from GridSearchCV
y_test_pred_grid = grid_search_xgb.predict(X_test_scaled)
print("\nTest Set Performance for GridSearchCV XGBoost:")
print("Accuracy:", accuracy_score(y_test, y_test_pred_grid))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred_grid))
print("Classification Report:\n", classification_report(y_test, y_test_pred_grid))

# Best model from RandomizedSearchCV
y_test_pred_random = random_search_xgb.predict(X_test_scaled)
print("\nTest Set Performance for RandomizedSearchCV XGBoost:")
print("Accuracy:", accuracy_score(y_test, y_test_pred_random))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred_random))
print("Classification Report:\n", classification_report(y_test, y_test_pred_random))

# Print timing information
print(f"\nTiming Summary:")
print(f"GridSearchCV Time: {elapsed_time_grid:.2f} seconds")
print(f"RandomizedSearchCV Time: {elapsed_time_random:.2f} seconds")
```

```
C:\Users\Dell\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning: [02:03:31] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-0015a694724fa8361-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
Test Set Performance for Vanilla XGBoost:
Accuracy: 0.9483514389174766
Confusion Matrix:
 [[2544   79]
 [ 192 2432]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      2623
           1       0.97      0.93      0.95      2624

    accuracy                           0.95      5247
   macro avg       0.95      0.95      0.95      5247
weighted avg       0.95      0.95      0.95      5247


Test Set Performance for GridSearchCV XGBoost:
Accuracy: 0.9483514389174766
Confusion Matrix:
 [[2559   64]
 [ 207 2417]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      2623
           1       0.97      0.92      0.95      2624

    accuracy                           0.95      5247
   macro avg       0.95      0.95      0.95      5247
weighted avg       0.95      0.95      0.95      5247


Test Set Performance for RandomizedSearchCV XGBoost:
Accuracy: 0.9481608538212312
Confusion Matrix:
 [[2550   73]
 [ 199 2425]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      2623
```

```
           1       0.97      0.92      0.95      2624

    accuracy                           0.95      5247
   macro avg       0.95      0.95      0.95      5247
weighted avg       0.95      0.95      0.95      5247


Timing Summary:
GridSearchCV Time: 1463.44 seconds
RandomizedSearchCV Time: 181.46 seconds
```

Test Set Performance for GridSearchCV XGBoost: Accuracy: 0.9483514389174766; f1-score: 0.95 slightly better than RandomizedSearchCV XGBoost Test Set Performance for Vanilla XGBoost: Accuracy: 0.948351438917476; f1-score: 0.95

Vanilla xgboost is already optimised to the extent such that gridsearchCV and randomisedsearchCV are not showing any better performance indicators6

In [ ]:

light boost (lightGBM)

In [639…
```
import lightgbm as lgb
```

In [640…
```
# Initialize the LightGBM classifier
lgb_classifier = lgb.LGBMClassifier(random_state=42)
```

In [641…
```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 6, 10, -1],  # -1 indicates no limit on depth
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'num_leaves': [31, 50, 100],
    'min_child_samples': [20, 50, 100]
}
```

In [ ]:

In [642…

```
# Initialize GridSearchCV
grid_search_lgb = GridSearchCV(estimator=lgb_classifier, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2, scoring='accuracy'

# Start the timer
start_time = time.time()

# Fit Grid Search on the training data
grid_search_lgb.fit(X_train_scaled, y_train)

# Stop the timer
end_time = time.time()

# Calculate the elapsed time
elapsed_time_grid = end_time - start_time
print(f"GridSearchCV completed in: {elapsed_time_grid:.2f} seconds")

# Get the best parameters and evaluate
best_params_grid = grid_search_lgb.best_params_
print("Best Parameters found by Grid Search:", best_params_grid)

y_val_pred_grid = grid_search_lgb.predict(X_val_scaled)
print("Validation Set Performance After Grid Search:")
print("Accuracy:", accuracy_score(y_val, y_val_pred_grid))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred_grid))
print("Classification Report:\n", classification_report(y_val, y_val_pred_grid))
```

```
Fitting 5 folds for each of 2916 candidates, totalling 14580 fits
[LightGBM] [Info] Number of positive: 12271, number of negative: 12212
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.006669 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2568
[LightGBM] [Info] Number of data points in the train set: 24483, number of used features: 14
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.501205 -> initscore=0.004820
[LightGBM] [Info] Start training from score 0.004820
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
GridSearchCV completed in: 5474.31 seconds
Best Parameters found by Grid Search: {'colsample_bytree': 0.6, 'learning_rate': 0.2, 'max_depth': 3, 'min_child_samples': 20,
'n_estimators': 200, 'num_leaves': 31, 'subsample': 0.6}
Validation Set Performance After Grid Search:
Accuracy: 0.9481509721692718
Confusion Matrix:
 [[2612   41]
 [ 231 2362]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      2653
           1       0.98      0.91      0.95      2593

    accuracy                           0.95      5246
   macro avg       0.95      0.95      0.95      5246
weighted avg       0.95      0.95      0.95      5246
```

In [ ]:

In [ ]:

In [643…
```
param_dist = {
    'n_estimators': np.arange(50, 201, 50),
    'max_depth': [-1, 3, 6, 10],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'num_leaves': np.arange(31, 101, 10),
    'min_child_samples': np.arange(20, 101, 20)
}
```

In [644…
```
# Initialize RandomizedSearchCV
random_search_lgb = RandomizedSearchCV(estimator=lgb_classifier, param_distributions=param_dist, n_iter=100, cv=5, n_jobs=-1,

# Start the timer
```

```
start_time = time.time()

# Fit Randomized Search on the training data
random_search_lgb.fit(X_train_scaled, y_train)

# Stop the timer
end_time = time.time()

# Calculate the elapsed time
elapsed_time_random = end_time - start_time
print(f"RandomizedSearchCV completed in: {elapsed_time_random:.2f} seconds")

# Get the best parameters and evaluate
best_params_random = random_search_lgb.best_params_
print("Best Parameters found by Random Search:", best_params_random)

y_val_pred_random = random_search_lgb.predict(X_val_scaled)
print("Validation Set Performance After Random Search:")
print("Accuracy:", accuracy_score(y_val, y_val_pred_random))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred_random))
print("Classification Report:\n", classification_report(y_val, y_val_pred_random))
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
[LightGBM] [Info] Number of positive: 12271, number of negative: 12212
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.006065 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2568
[LightGBM] [Info] Number of data points in the train set: 24483, number of used features: 14
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.501205 -> initscore=0.004820
[LightGBM] [Info] Start training from score 0.004820
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
RandomizedSearchCV completed in: 191.93 seconds
Best Parameters found by Random Search: {'subsample': 1.0, 'num_leaves': 41, 'n_estimators': 150, 'min_child_samples': 60, 'max
_depth': 3, 'learning_rate': 0.2, 'colsample_bytree': 0.6}
Validation Set Performance After Random Search:
Accuracy: 0.9437666793747618
Confusion Matrix:
```

```
[[2595   58]
 [ 237 2356]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      2653
           1       0.98      0.91      0.94      2593

    accuracy                           0.94      5246
   macro avg       0.95      0.94      0.94      5246
weighted avg       0.95      0.94      0.94      5246
```

In [ ]:

In [645…

```
# Evaluate on the test set for final comparison

# Vanilla LightGBM
lgb_classifier.fit(X_train_scaled, y_train)
y_test_pred_vanilla = lgb_classifier.predict(X_test_scaled)
print("Test Set Performance for Vanilla LightGBM:")
print("Accuracy:", accuracy_score(y_test, y_test_pred_vanilla))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred_vanilla))
print("Classification Report:\n", classification_report(y_test, y_test_pred_vanilla))

# Best model from GridSearchCV
y_test_pred_grid = grid_search_lgb.predict(X_test_scaled)
print("\nTest Set Performance for GridSearchCV LightGBM:")
print("Accuracy:", accuracy_score(y_test, y_test_pred_grid))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred_grid))
print("Classification Report:\n", classification_report(y_test, y_test_pred_grid))

# Best model from RandomizedSearchCV
y_test_pred_random = random_search_lgb.predict(X_test_scaled)
print("\nTest Set Performance for RandomizedSearchCV LightGBM:")
print("Accuracy:", accuracy_score(y_test, y_test_pred_random))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred_random))
print("Classification Report:\n", classification_report(y_test, y_test_pred_random))

# Print timing information
```

```
print(f"\nTiming Summary:")
print(f"GridSearchCV Time: {elapsed_time_grid:.2f} seconds")
print(f"RandomizedSearchCV Time: {elapsed_time_random:.2f} seconds")
```

```
[LightGBM] [Info] Number of positive: 12271, number of negative: 12212
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.035256 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2568
[LightGBM] [Info] Number of data points in the train set: 24483, number of used features: 14
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.501205 -> initscore=0.004820
[LightGBM] [Info] Start training from score 0.004820
Test Set Performance for Vanilla LightGBM:
Accuracy: 0.9498761196874405
Confusion Matrix:
 [[2559   64]
 [ 199 2425]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      2623
           1       0.97      0.92      0.95      2624

    accuracy                           0.95      5247
   macro avg       0.95      0.95      0.95      5247
weighted avg       0.95      0.95      0.95      5247


Test Set Performance for GridSearchCV LightGBM:
Accuracy: 0.9510196302649133
Confusion Matrix:
 [[2569   54]
 [ 203 2421]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      2623
           1       0.98      0.92      0.95      2624

    accuracy                           0.95      5247
   macro avg       0.95      0.95      0.95      5247
weighted avg       0.95      0.95      0.95      5247


Test Set Performance for RandomizedSearchCV LightGBM:
```

```
Accuracy: 0.9500667047836859
Confusion Matrix:
 [[2561   62]
 [ 200 2424]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      2623
           1       0.98      0.92      0.95      2624

    accuracy                           0.95      5247
   macro avg       0.95      0.95      0.95      5247
weighted avg       0.95      0.95      0.95      5247


Timing Summary:
GridSearchCV Time: 5474.31 seconds
RandomizedSearchCV Time: 191.93 seconds
```

Test Set Performance for GridSearchCV LightGBM: Accuracy: 0.9510196302649133 f1 score: 0.95 slightly better than RandomizedSearchCV LightGBM

Test Set Performance for RandomizedSearchCV LightGBM: Accuracy: 0.0.9500667047836859 f1 score: 0.95 slightly better than RandomizedSearchCV LightGBM

In [ ]:

Stacking

In [ ]:

In [646…
```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
```

In [647…
```
# Define the base models for stacking
base_models = [
    ('rf', rf_classifier),  # Random Forest
```

```
    ('xgb', grid_search_xgb.best_estimator_),  # XGBoost with best params from GridSearchCV
    ('lgbm', grid_search_lgb.best_estimator_)  # LightGBM with best params from GridSearchCV
]
```

In [648…

```
# Define the meta-model
meta_model = LogisticRegression(random_state=42)
```

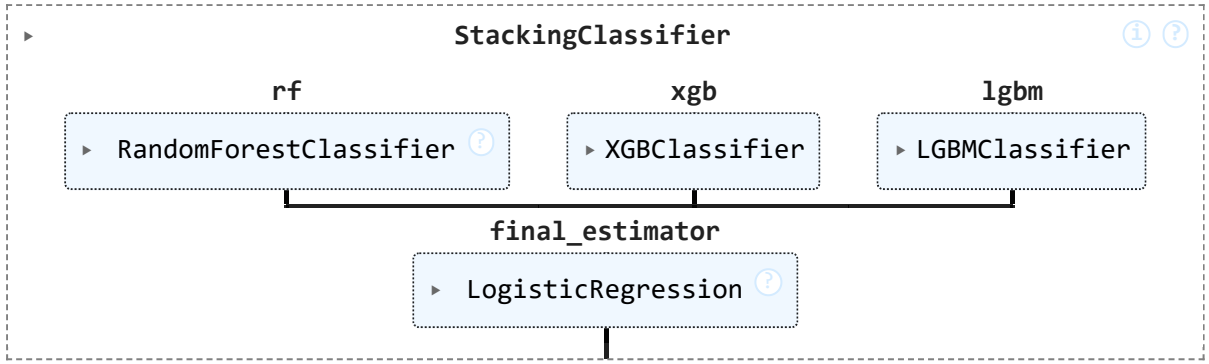
In [649…

```
# Create the stacking classifier
stacking_clf = StackingClassifier(estimators=base_models, final_estimator=meta_model, cv=5, n_jobs=-1)
```

In [650…

```
# Fit the stacking classifier
stacking_clf.fit(X_train_scaled, y_train)
```

Out[650…



In [651…

```
# Predict on the validation set
y_val_pred_stacking = stacking_clf.predict(X_val_scaled)
print("Validation Set Performance for Stacking Classifier:")
print("Accuracy:", accuracy_score(y_val, y_val_pred_stacking))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred_stacking))
print("Classification Report:\n", classification_report(y_val, y_val_pred_stacking))

# Predict on the test set
y_test_pred_stacking = stacking_clf.predict(X_test_scaled)
print("Test Set Performance for Stacking Classifier:")
print("Accuracy:", accuracy_score(y_test, y_test_pred_stacking))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred_stacking))
print("Classification Report:\n", classification_report(y_test, y_test_pred_stacking))
```

```
Validation Set Performance for Stacking Classifier:
Accuracy: 0.9431948150972169
Confusion Matrix:
 [[2571   82]
 [ 216 2377]]
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.97      0.95      2653
           1       0.97      0.92      0.94      2593

    accuracy                           0.94      5246
   macro avg       0.94      0.94      0.94      5246
weighted avg       0.94      0.94      0.94      5246

Test Set Performance for Stacking Classifier:
Accuracy: 0.9466361730512673
Confusion Matrix:
 [[2522  101]
 [ 179 2445]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.96      0.95      2623
           1       0.96      0.93      0.95      2624

    accuracy                           0.95      5247
   macro avg       0.95      0.95      0.95      5247
weighted avg       0.95      0.95      0.95      5247
```

In [ ]:

In [652…
```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

In [653…
```
# Define the base models for stacking, including simpler models
base_models = [
    ('rf', rf_classifier),  # Random Forest
    ('xgb', grid_search_xgb.best_estimator_),  # XGBoost with best params from GridSearchCV
```

```
    ('lgbm', grid_search_lgb.best_estimator_),  # LightGBM with best params from GridSearchCV
    ('dt', DecisionTreeClassifier(random_state=42)),  # Simple Decision Tree
    ('svm', SVC(kernel='linear', probability=True, random_state=42)),  # SVM with linear kernel
    ('knn', KNeighborsClassifier(n_neighbors=5))  # kNN with 5 neighbors
]
```

In [654…
```
# Define the meta-model
meta_model = LogisticRegression(random_state=42)
```

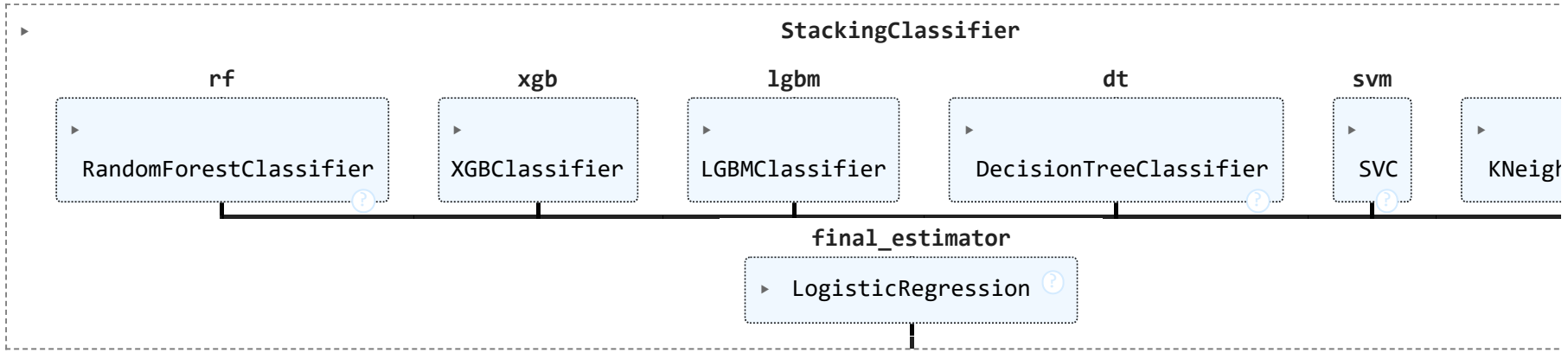
In [655…
```
# Create the stacking classifier
stacking_clf = StackingClassifier(estimators=base_models, final_estimator=meta_model, cv=5, n_jobs=-1)
```

In [662…
```
# Start the timer
start_time = time.time()
print("start_time:", start_time)
```

```
start_time: 1724323344.8975704
```

In [663…
```
# Fit the stacking classifier
stacking_clf.fit(X_train_scaled, y_train)
```

Out[663…



In [664…
```
# Stop the timer
end_time = time.time()
print("end_time:", end_time)
```

```
end_time: 1724332961.0732303
```

In [665…
```
print("time taken for fitting stacking classifier:",end_time-start_time)
```

```
time taken for fitting stacking classifier: 9616.175659894943
```

In [666…
```
# Predict on the validation set
y_val_pred_stacking = stacking_clf.predict(X_val_scaled)
print("Validation Set Performance for Stacking Classifier with Simple Models:")
print("Accuracy:", accuracy_score(y_val, y_val_pred_stacking))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred_stacking))
print("Classification Report:\n", classification_report(y_val, y_val_pred_stacking))

# Predict on the test set
y_test_pred_stacking = stacking_clf.predict(X_test_scaled)
print("Test Set Performance for Stacking Classifier with Simple Models:")
print("Accuracy:", accuracy_score(y_test, y_test_pred_stacking))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred_stacking))
print("Classification Report:\n", classification_report(y_test, y_test_pred_stacking))
```

```
Validation Set Performance for Stacking Classifier with Simple Models:
Accuracy: 0.9456728936332444
Confusion Matrix:
 [[2566   87]
 [ 198 2395]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95      2653
           1       0.96      0.92      0.94      2593

    accuracy                           0.95      5246
   macro avg       0.95      0.95      0.95      5246
weighted avg       0.95      0.95      0.95      5246

Test Set Performance for Stacking Classifier with Simple Models:
Accuracy: 0.9479702687249857
Confusion Matrix:
 [[2514  109]
 [ 164 2460]]
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      2623
           1       0.96      0.94      0.95      2624

    accuracy                           0.95      5247
   macro avg       0.95      0.95      0.95      5247
weighted avg       0.95      0.95      0.95      5247
```

In [ ]:

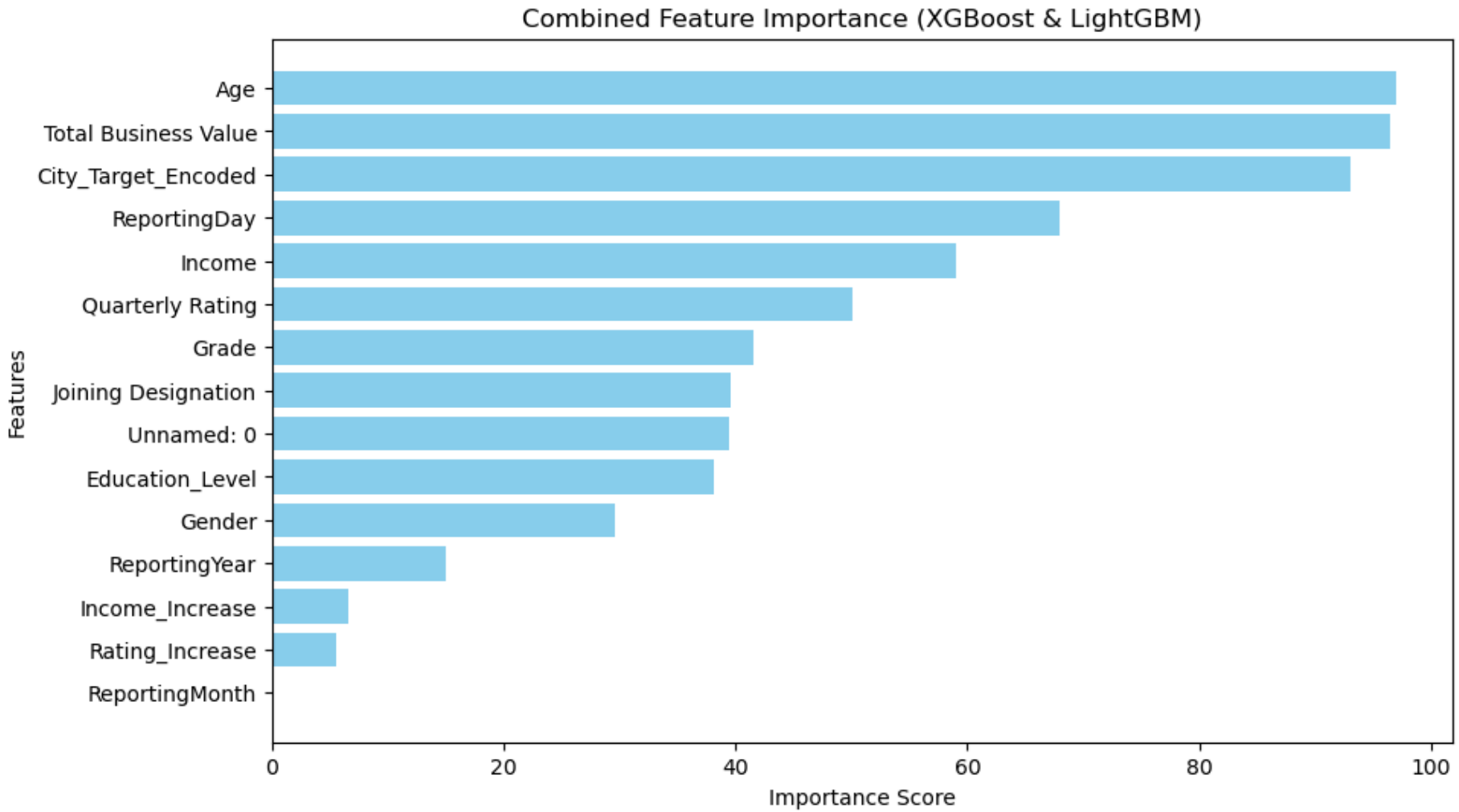
In [ ]:

Get feature importance based on xgboost anf lightGBM combined

In [673…

```
# Get feature importances from both models
xgb_importance = xgb_model.feature_importances_
lgb_importance = lgb_model.feature_importances_
```

```
# Average the importances (you could also sum them)
combined_importance = (xgb_importance + lgb_importance) / 2

# Create a DataFrame, excluding the 'Target' column
features = df.drop(columns=["Target"]).columns  # Exclude the 'Target' column
importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': combined_importance
})

# Sort by importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot the combined feature importance with feature names
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.title('Combined Feature Importance (XGBoost & LightGBM)')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.gca().invert_yaxis()  # Highest importance at the top
plt.show()
```



median income of drivers is 60087.00; there are a few drivers who earn as much as 150000.00 and more

drivers having better grade are earning more income

we have almost same number of drivers across different education leve

Percentage of attrition for male drivers: 8.54% ; Percentage of attrition for female drivers: 8.36

Overall percentage of attrition: 8.46%

drivers: total_males = 11087 ; total_females=: 7981%ls

number of drivers who earn more than 100000.00 is also significant

missing value imputation is done. however outlier treatment is not done...

As per vanilla decision tree dtc7, Quarterly rating, Total business value, Education Level, Income increase (engineered feature), Gender are the top features

based on combined (xgboost and lightgbm) feature importances: age, business value, city, reporting day, income, quarterly rating are the top factors

Gender based EDXA would have given more insights (which is not done in this notebook)

plain vanilla decision tree is already giving good scores (accuracy=0.92 and f1-score 0.92)

"min_impurity_decrease" looks like a better hyperparameter compared to others (like max_depth, min_samples_split, min_samples_leaf etc)

dtc7 looks to be a good choice with accuracy=0.887 and f1-score=0.88 and also explainability will be good

plain vanilla Random Forest classifier: accuracy: 0.942062130741376; f1-score:0.94 (Test set based); scores are already good and even after hyperparameter tuning and using xgboost , lightGBM and stacking did not result in significant improvement in the model performance.

Random Forest classifier: gridsearchCV accuracy: 0.9363324437666793; f1-score:0.94 (slightly better than randomisedsearchCV)

Random Forest classifier: RandomisedsearchCV Accuracy: 0.9361418223408311; f1-score: 0.94

Test Set Performance for GridSearchCV XGBoost: Accuracy: 0.9483514389174766; f1-score: 0.95 slightly better than RandomizedSearchCV XGBoost

Test Set Performance for Vanilla XGBoost: Accuracy: 0.9483514389174766; f1-score: 0.95

Vanilla xgboost is already optimised to the extent such that gridsearchCV and randomisedsearchCV are not showing any better performance indicators

Test Set Performance for GridSearchCV LightGBM: Accuracy: 0.9510196302649133 f1 score: 0.95 slightly better than RandomizedSearchCV LightGBM

Test Set Performance for RandomizedSearchCV LightGBM: Accuracy: 0.0.9500667047836859 f1 score: 0.95 slightly better than RandomizedSearchCV LightGBM

Test Set Performance for Stacking Classifier with Simple Models: Accuracy: 0.947970268724985; f1-score=0.95

stacking did not result any significant improvement in the performance

I have not done one hot encoding of categorical features where ordinal data was available.

(I gave more importance to model building and comparing the models rather than on bringing out the business insights. I know at the end of teh day business insights are important. But for me , right-now building models (practicing ML hands on) is more important... I would appreciate valuable suggestions for improvement)

In [ ]: