```
import pandas as pd

df = pd.read_csv("ola_driver_scaler.csv")

df.head()
```

```
/usr/local/lib/python3.10/dist-packages/google/colab/_dataframe_summarizer.py:88: UserWarning: Could not infer format, so each eleme
  cast_date_col = pd.to_datetime(column, errors="coerce")
```

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Bus
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	23
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	-6
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	

```
df.shape
```

```
(19104, 14)
```

```
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
```

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

```
df.head()
```

```
/usr/local/lib/python3.10/dist-packages/google/colab/_dataframe_summarizer.py:88: UserWarning: Could not infer format, so each eleme
  cast_date_col = pd.to_datetime(column, errors="coerce")
```

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

```
df.columns
```

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

```
df['Reporting_Date'] = pd.to_datetime(df['MMM-YY'])

df['Date_Of_Joining'] = pd.to_datetime(df['Dateofjoining'])
```

```
df['Last_Working_Date'] = pd.to_datetime(df['LastWorkingDate'])

df.head()
```

```
<ipython-input-107-555294bdd4d7>:1: UserWarning: Could not infer format, so each element will be parsed individually, falling back t
  df['Reporting_Date'] = pd.to_datetime(df['MMM-YY'])
<ipython-input-107-555294bdd4d7>:3: UserWarning: Could not infer format, so each element will be parsed individually, falling back t
  df['Date_Of_Joining'] = pd.to_datetime(df['Dateofjoining'])
<ipython-input-107-555294bdd4d7>:5: UserWarning: Could not infer format, so each element will be parsed individually, falling back t
  df['Last_Working_Date'] = pd.to_datetime(df['LastWorkingDate'])
/usr/local/lib/python3.10/dist-packages/google/colab/_dataframe_summarizer.py:88: UserWarning: Could not infer format, so each eleme
  cast_date_col = pd.to_datetime(column, errors="coerce")
```

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

```
df.drop(columns=['MMM-YY','Dateofjoining','LastWorkingDate'],inplace=True)

df.head()
```

	Driver_ID	Age	Gender	City	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	Reporting_Date	Date_Of_Joining
0	1	28.0	0.0	C23	2	57387	1	1	2381060	2	2019-01-01	2018-12-24
1	1	28.0	0.0	C23	2	57387	1	1	-665480	2	2019-02-01	2018-12-24
2	1	28.0	0.0	C23	2	57387	1	1	0	2	2019-03-01	2018-12-24
3	2	31.0	0.0	C7	2	67016	2	2	0	1	2020-11-01	2020-11-06
4	2	31.0	0.0	C7	2	67016	2	2	0	1	2020-12-01	2020-11-06

```
numerical_columns = df.select_dtypes(include=['int64','float64']).columns

numerical_columns
```

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

```
df[numerical_columns].isna().sum()
```

	0
Driver_ID	0
Age	61
Gender	52
Education_Level	0
Income	0
Joining Designation	0
Grade	0
Total Business Value	0
Quarterly Rating	0

dtype: int64

```
import matplotlib.pyplot as plt
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=5,weights='distance').set_output(transform='pandas')

imputer.fit(df[numerical_columns])

imputed_numerical_columns = imputer.transform(df[numerical_columns])
```

```
imputed_numerical_columns.isna().sum()
```

	0
Driver_ID	0
Age	0
Gender	0
Education_Level	0
Income	0
Joining Designation	0
Grade	0
Total Business Value	0
Quarterly Rating	0

dtype: int64

```
df['Age'] = imputed_numerical_columns['Age']
df['Gender'] = imputed_numerical_columns['Gender']

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

	0
Driver_ID	0
Age	0
Gender	0
City	0
Education_Level	0
Income	0
Joining Designation	0
Grade	0
Total Business Value	0
Quarterly Rating	0
Reporting_Date	0
Date_Of_Joining	0
Last_Working_Date	17488

dtype: int64

```
df = df.sort_values(by=['Driver_ID', 'Reporting_Date']).reset_index(drop=True)
```

```
df
```

	Driver_ID	Age	Gender	City	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	Reporting_Date	Date_Of_Joini
0	1	28.0	0.0	C23	2	57387	1	1	2381060	2	2019-01-01	2018-12-
1	1	28.0	0.0	C23	2	57387	1	1	-665480	2	2019-02-01	2018-12-
2	1	28.0	0.0	C23	2	57387	1	1	0	2	2019-03-01	2018-12-
3	2	31.0	0.0	C7	2	67016	2	2	0	1	2020-11-01	2020-11-
4	2	31.0	0.0	C7	2	67016	2	2	0	1	2020-12-01	2020-11-
...	...	...	...	...	...	...	...	...	...	...	...	
19099	2788	30.0	0.0	C27	2	70254	2	2	740280	3	2020-08-01	2020-06-
19100	2788	30.0	0.0	C27	2	70254	2	2	448370	3	2020-09-01	2020-06-
19101	2788	30.0	0.0	C27	2	70254	2	2	0	2	2020-10-01	2020-06-
19102	2788	30.0	0.0	C27	2	70254	2	2	200420	2	2020-11-01	2020-06-
19103	2788	30.0	0.0	C27	2	70254	2	2	411480	2	2020-12-01	2020-06-

19104 rows × 13 columns

```
# Create a column to check if the rating increased compared to the previous row
df['Rating_Increased'] = df.groupby('Driver_ID')['Quarterly Rating'].diff().apply(lambda x: 1 if x > 0 else 0)
```

```
df
```

	Driver_ID	Age	Gender	City	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	Reporting_Date	Date_Of_Joini
0	1	28.0	0.0	C23	2	57387	1	1	2381060	2	2019-01-01	2018-12-
1	1	28.0	0.0	C23	2	57387	1	1	-665480	2	2019-02-01	2018-12-
2	1	28.0	0.0	C23	2	57387	1	1	0	2	2019-03-01	2018-12-
3	2	31.0	0.0	C7	2	67016	2	2	0	1	2020-11-01	2020-11-
4	2	31.0	0.0	C7	2	67016	2	2	0	1	2020-12-01	2020-11-
...	...	...	...	...	...	...	...	...	...	...	...	
19099	2788	30.0	0.0	C27	2	70254	2	2	740280	3	2020-08-01	2020-06-
19100	2788	30.0	0.0	C27	2	70254	2	2	448370	3	2020-09-01	2020-06-
19101	2788	30.0	0.0	C27	2	70254	2	2	0	2	2020-10-01	2020-06-
19102	2788	30.0	0.0	C27	2	70254	2	2	200420	2	2020-11-01	2020-06-
19103	2788	30.0	0.0	C27	2	70254	2	2	411480	2	2020-12-01	2020-06-

19104 rows × 14 columns

```
df['Rating_Increased'].value_counts()
```

	count
Rating_Increased	
0	17859
1	1245

dtype: int64

```
# Sort the dataframe by Driver_ID (we don't need Reporting_Date here)
df = df.sort_values(by=['Driver_ID']).reset_index(drop=True)
```

```
# Create a column to check if Monthly Income has increased compared to the previous row
df['Income_Increased'] = df.groupby('Driver_ID')['Income'].diff().apply(lambda x: 1 if x > 0 else 0)
```

```
df['Income_Increased'].value_counts()
```

	count
Income_Increased	
0	19049
1	55

dtype: int64

```
aggregation_rules = {
    'Age': 'mean',
    'Gender': 'first',
    'City': 'first',
    'Education_Level': 'max',
    'Income': 'mean',
    'Date_Of_Joining': 'min',
    'Last_Working_Date': 'max',
    'Joining Designation': 'first',
    'Grade': 'max',
    'Total Business Value': 'sum',
    'Quarterly Rating': 'mean',
    'Income_Increased': 'max'  # Did the income increase at any point (1 if yes, 0 otherwise)
}

aggregated_data = df.groupby('Driver_ID').agg(aggregation_rules).reset_index()
```

```
aggregated_data
```

	Driver_ID	Age	Gender	City	Education_Level	Income	Date_Of_Joining	Last_Working_Date	Joining Designation	Grade	Total Business Value
0	1	28.000000	0.0	C23	2	57387.0	2018-12-24	2019-03-11	1	1	1715580
1	2	31.000000	0.0	C7	2	67016.0	2020-11-06	NaT	2	2	0
2	4	43.000000	0.0	C13	2	65603.0	2019-12-07	2020-04-27	2	2	350000
3	5	29.000000	0.0	C9	0	46368.0	2019-01-09	2019-03-07	1	1	120360
4	6	31.000000	1.0	C11	1	78728.0	2020-07-31	NaT	3	3	1265000
...	...	...	...	...	...	...	...	...	...	...	...
2376	2784	33.500000	0.0	C24	0	82815.0	2015-10-15	NaT	2	3	21748820
2377	2785	34.000000	1.0	C9	0	12105.0	2020-08-28	2020-10-28	1	1	0
2378	2786	44.888889	0.0	C19	0	35370.0	2018-07-31	2019-09-22	2	2	2815090
2379	2787	28.000000	1.0	C20	2	69498.0	2018-07-21	2019-06-20	1	1	977830
2380	2788	29.857143	0.0	C27	2	70254.0	2020-06-08	NaT	2	2	2298240

2381 rows × 13 columns

```
aggregated_data['Target'] = aggregated_data['Last_Working_Date'].notna().astype(int)
```

```
aggregated_data
```

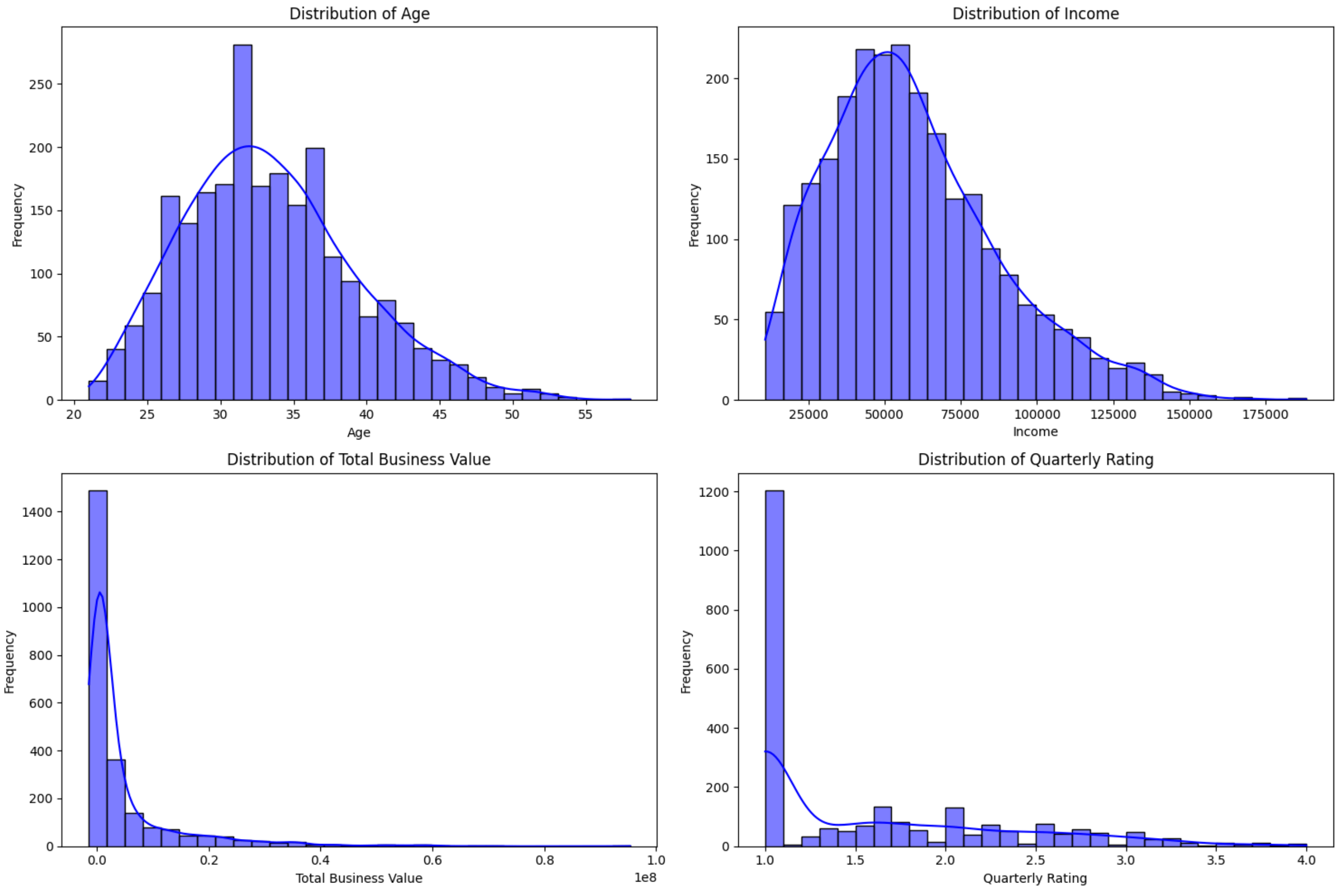
	Driver_ID	Age	Gender	City	Education_Level	Income	Date_Of_Joining	Last_Working_Date	Joining Designation	Grade	Total Business Value
0	1	28.000000	0.0	C23	2	57387.0	2018-12-24	2019-03-11	1	1	1715580
1	2	31.000000	0.0	C7	2	67016.0	2020-11-06	NaT	2	2	0
2	4	43.000000	0.0	C13	2	65603.0	2019-12-07	2020-04-27	2	2	350000
3	5	29.000000	0.0	C9	0	46368.0	2019-01-09	2019-03-07	1	1	120360
4	6	31.000000	1.0	C11	1	78728.0	2020-07-31	NaT	3	3	1265000
...	...	...	...	...	...	...	...	...	...	...	...
2376	2784	33.500000	0.0	C24	0	82815.0	2015-10-15	NaT	2	3	21748820
2377	2785	34.000000	1.0	C9	0	12105.0	2020-08-28	2020-10-28	1	1	0
2378	2786	44.888889	0.0	C19	0	35370.0	2018-07-31	2019-09-22	2	2	2815090
2379	2787	28.000000	1.0	C20	2	69498.0	2018-07-21	2019-06-20	1	1	977830
2380	2788	29.857143	0.0	C27	2	70254.0	2020-06-08	NaT	2	2	2298240

2381 rows × 14 columns

```
aggregated_data.describe()
```

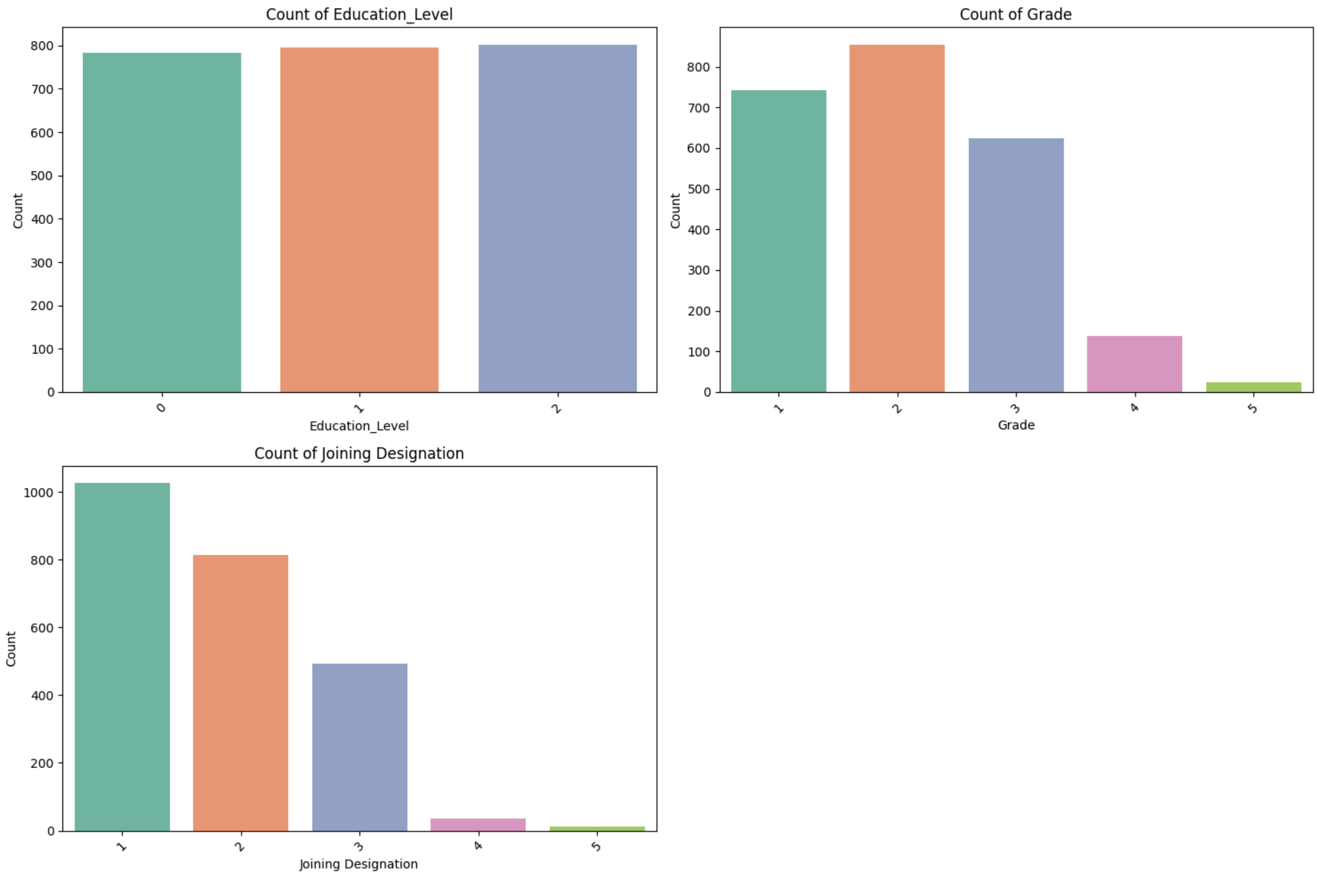
	Driver_ID	Age	Gender	Education_Level	Income	Date_Of_Joining	Last_Working_Date	Joining Designation	
count	2381.000000	2381.000000	2381.000000	2381.00000	2381.000000	2381	1616	2381.000000	2381.(
mean	1397.559009	33.376784	0.410477	1.00756	59232.460484	2019-02-08 07:14:50.550189056	2019-12-21 20:59:06.534653440	1.820244	2.(
min	1.000000	21.000000	0.000000	0.00000	10747.000000	2013-04-01 00:00:00	2018-12-31 00:00:00	1.000000	1.(
25%	695.000000	29.000000	0.000000	0.00000	39104.000000	2018-06-29 00:00:00	2019-06-06 00:00:00	1.000000	1.(
50%	1400.000000	33.000000	0.000000	1.00000	55285.000000	2019-07-21 00:00:00	2019-12-20 12:00:00	2.000000	2.(
75%	2100.000000	37.000000	1.000000	2.00000	75835.000000	2020-05-02 00:00:00	2020-07-03 00:00:00	2.000000	3.(
max	2788.000000	58.000000	1.000000	2.00000	188418.000000	2020-12-28 00:00:00	2020-12-28 00:00:00	5.000000	5.(
std	806.161628	5.878336	0.491918	0.81629	28298.214012	NaN	NaN	0.841433	0.!

```
# 1. Distribution plots for all continuous variables
continuous_columns = ['Age', 'Income', 'Total Business Value', 'Quarterly Rating']  # Add any continuous variables

# Create a figure with subplots
plt.figure(figsize=(15, 10))

for i, col in enumerate(continuous_columns, 1):
    plt.subplot(2, 2, i)  # 2 rows, 2 columns for subplots
    sns.histplot(aggregated_data[col], kde=True, color='blue', bins=30)  # kde for density plot
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')

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




```
categorical_columns = ['Education_Level', 'Grade', 'Joining Designation']  # Add categorical variables here

plt.figure(figsize=(15, 10))

for i, col in enumerate(categorical_columns, 1):
    plt.subplot(2, 2, i)  # 2 rows, 2 columns for subplots
    sns.countplot(x=aggregated_data[col], palette='Set2')
    plt.title(f'Count of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45)  # Rotate x labels for better visibility if needed

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

```
<ipython-input-171-ce2ab8926171>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(x=aggregated_data[col], palette='Set2')
<ipython-input-171-ce2ab8926171>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(x=aggregated_data[col], palette='Set2')
<ipython-input-171-ce2ab8926171>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(x=aggregated_data[col], palette='Set2')
```

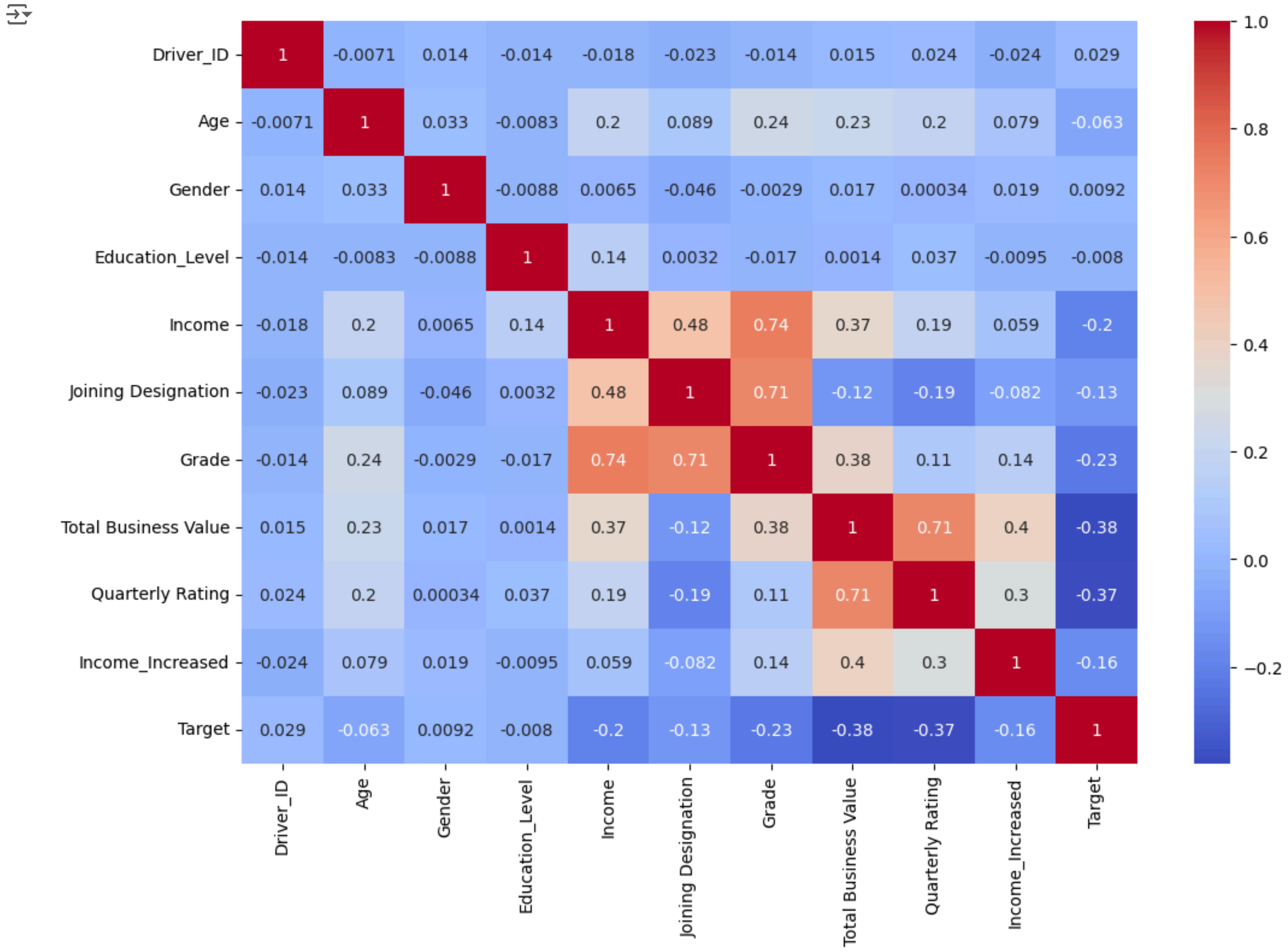



```
# Select only numeric columns for correlation calculation
numeric_columns = aggregated_data.select_dtypes(include=['number']).columns

# Calculate the correlation matrix using only numeric columns
correlation_matrix = aggregated_data[numeric_columns].corr()

# Display the correlation matrix
print(correlation_matrix)
```

```
                      Driver_ID       Age    Gender  Education_Level  \
Driver_ID              1.000000 -0.007068  0.014142        -0.014343
Age                   -0.007068  1.000000  0.032779        -0.008329
Gender                 0.014142  0.032779  1.000000        -0.008766
Education_Level       -0.014343 -0.008329 -0.008766         1.000000
Income                -0.017553  0.195907  0.006518         0.140779
Joining Designation   -0.023126  0.089230 -0.046351         0.003203
Grade                 -0.014345  0.242799 -0.002871        -0.016806
Total Business Value   0.015133  0.227084  0.017458         0.001392
Quarterly Rating       0.023867  0.195670  0.000338         0.037169
Income_Increased      -0.023591  0.079226  0.019417        -0.009485
Target                 0.029269 -0.063012  0.009171        -0.007953
```

```
                      Income  Joining Designation     Grade  \
Driver_ID          -0.017553            -0.023126 -0.014345
Age                 0.195907             0.089230  0.242799
Gender              0.006518            -0.046351 -0.002871
Education_Level     0.140779             0.003203 -0.016806
Income              1.000000             0.484116  0.738869
Joining Designation  0.484116            1.000000  0.712419
Grade               0.738869             0.712419  1.000000
Total Business Value  0.368632          -0.121368  0.383076
Quarterly Rating    0.187621            -0.193807  0.109546
Income_Increased    0.059381            -0.082136  0.138531
Target             -0.197988            -0.127773 -0.226190

                    Total Business Value  Quarterly Rating  \
Driver_ID                       0.015133          0.023867
Age                             0.227084          0.195670
Gender                          0.017458          0.000338
Education_Level                 0.001392          0.037169
Income                          0.368632          0.187621
Joining Designation            -0.121368         -0.193807
Grade                           0.383076          0.109546
Total Business Value            1.000000          0.712487
Quarterly Rating                0.712487          1.000000
Income_Increased                0.398861          0.302464
Target                         -0.379552         -0.373683

                    Income_Increased    Target
Driver_ID                  -0.023591  0.029269
Age                         0.079226 -0.063012
Gender                      0.019417  0.009171
Education_Level            -0.009485 -0.007953
Income                      0.059381 -0.197988
Joining Designation        -0.082136 -0.127773
Grade                       0.138531 -0.226190
Total Business Value        0.398861 -0.379552
Quarterly Rating            0.302464 -0.373683
Income_Increased            1.000000 -0.160790
Target                     -0.160790  1.000000
```

```
import seaborn as sns
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```




```
from sklearn.preprocessing import OneHotEncoder

# Initialize the label encoder
label_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore') # sparse=False for a dense array

# Reshape the 'City' column into a 2D array
city_data = aggregated_data[['City']] # Use double brackets to select as a DataFrame

# Apply label encoding to the 'City_Code' column
encoded_city = label_encoder.fit_transform(city_data)

# Create a DataFrame from the encoded data
encoded_city_df = pd.DataFrame(encoded_city, columns=label_encoder.get_feature_names_out(['City']))

# Concatenate the encoded data with the original DataFrame
aggregated_data = pd.concat([aggregated_data, encoded_city_df], axis=1)

aggregated_data.head()
```

	Driver_ID	Age	Gender	City	Education_Level	Income	Date_Of_Joining	Last_Working_Date	Joining Designation	Grade	...	City_C27	Cit
0	1	28.0	0.0	C23	2	57387.0	2018-12-24	2019-03-11	1	1	...	0.0	
1	2	31.0	0.0	C7	2	67016.0	2020-11-06	NaT	2	2	...	0.0	
2	4	43.0	0.0	C13	2	65603.0	2019-12-07	2020-04-27	2	2	...	0.0	
3	5	29.0	0.0	C9	0	46368.0	2019-01-09	2019-03-07	1	1	...	0.0	
4	6	31.0	1.0	C11	1	78728.0	2020-07-31	NaT	3	3	...	0.0	

5 rows × 44 columns

```
aggregated_data.drop(columns=['City'],inplace=True)
```

```
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
```

```
# Convert datetime columns to numeric (e.g., Unix timestamp)
for col in ['Date_Of_Joining', 'Last_Working_Date']:
    aggregated_data[col] = aggregated_data[col].astype('int64') // 10**9  # Convert to Unix timestamp

# Apply SMOTE to handle class imbalance
smote = SMOTE(random_state=42)

X = aggregated_data.drop(columns=['Target'])
y = aggregated_data['Target']

X_resampled, y_resampled = smote.fit_resample(X, y)

# Step 4: Check the class distribution before and after applying SMOTE
print("Before SMOTE:\n", y.value_counts())
print("\nAfter SMOTE:\n", y_resampled.value_counts())

# Step 5: Standardize the resampled features (X_resampled)
scaler = StandardScaler()
X_resampled_scaled = scaler.fit_transform(X_resampled)

# Check the result (scaled features for resampled data)
print("\nScaled Resampled Training Data (First 5 rows):\n", X_resampled_scaled[:5])
```

```
Before SMOTE:
 Target
1    1616
0     765
Name: count, dtype: int64

After SMOTE:
 Target
1    1616
0    1616
Name: count, dtype: int64

Scaled Resampled Training Data (First 5 rows):
 [[-1.81210602 -0.98375085 -0.87410555  1.3490513  -0.1418648   0.
   1.         -0.96625626 -1.20967523 -0.40912303  0.42461999 -0.11611998
   0.11382795 -0.18785206 -0.20224581 -0.17309401 -0.19791481 -0.16604662
  -0.18840765 -0.22037344 -0.20358346 -0.17956063 -0.18435409 -0.19539535
  -0.17649055 -0.2674719  -0.20498838 -0.19691202  5.91020391 -0.17901023
  -0.18150078 -0.20973228 -0.20634542 -0.20085749 -0.23250496 -0.19264925
  -0.20539167 -0.20066427 -0.18463455 -0.18824631 -0.2167729  -0.18726035]
 [-1.81080801 -0.44887171 -0.87410555  1.3490513   0.20015466  0.
  -1.          0.25986868 -0.12032713 -0.5723096  -0.88327034 -0.11611998
   1.50976151 -0.18785206 -0.20224581 -0.17309401 -0.19791481 -0.16604662
  -0.18840765 -0.22037344 -0.20358346 -0.17956063 -0.18435409 -0.19539535
  -0.17649055 -0.2674719  -0.20498838 -0.19691202 -0.18235598 -0.17901023
  -0.18150078 -0.20973228 -0.20634542 -0.20085749 -0.23250496 -0.19264925
  -0.20539167 -0.20066427 -0.18463455  5.78301496 -0.2167729  -0.18726035]
 [-1.80821198  1.69064484 -0.87410555  1.3490513   0.14996528  0.
   1.          0.25986868 -0.12032713 -0.53901748 -0.88327034 -0.11611998
  -1.28210561 -0.18785206 -0.20224581 -0.17309401 -0.19791481  6.24282567
  -0.18840765 -0.22037344 -0.20358346 -0.17956063 -0.18435409 -0.19539535
  -0.17649055 -0.2674719  -0.20498838 -0.19691202 -0.18235598 -0.17901023
  -0.18150078 -0.20973228 -0.20634542 -0.20085749 -0.23250496 -0.19264925
  -0.20539167 -0.20066427 -0.18463455 -0.18824631 -0.2167729  -0.18726035]
 [-1.80691397 -0.8054578  -0.87410555 -1.16980127 -0.53325668  0.
   1.         -0.96625626 -1.20967523 -0.56086091 -0.88327034 -0.11611998
   1.76356761 -0.18785206 -0.20224581 -0.17309401 -0.19791481 -0.16604662
  -0.18840765 -0.22037344 -0.20358346 -0.17956063 -0.18435409 -0.19539535
  -0.17649055 -0.2674719  -0.20498838 -0.19691202 -0.18235598 -0.17901023
  -0.18150078 -0.20973228 -0.20634542 -0.20085749 -0.23250496 -0.19264925
  -0.20539167 -0.20066427 -0.18463455 -0.18824631 -0.2167729   5.85006285]
 [-1.80561595 -0.44887171  1.25249351  0.08962501  0.61616171  0.
  -1.          1.48599363  0.96902098 -0.45198236 -0.09853614 -0.11611998
  -1.53591171 -0.18785206 -0.20224581  6.31046624 -0.19791481 -0.16604662
  -0.18840765 -0.22037344 -0.20358346 -0.17956063 -0.18435409 -0.19539535
  -0.17649055 -0.2674719  -0.20498838 -0.19691202 -0.18235598 -0.17901023
  -0.18150078 -0.20973228 -0.20634542 -0.20085749 -0.23250496 -0.19264925
  -0.20539167 -0.20066427 -0.18463455 -0.18824631 -0.2167729  -0.18726035]]
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import roc_curve,classification_report,confusion_matrix,auc
from sklearn.model_selection import cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X_resampled_scaled, y_resampled, test_size=0.3, random_state=42)

rf = RandomForestClassifier(random_state=42)

# Hyperparameter tuning with GridSearchCV
param_grid_rf = {
    'n_estimators': [50, 100, 200],
```

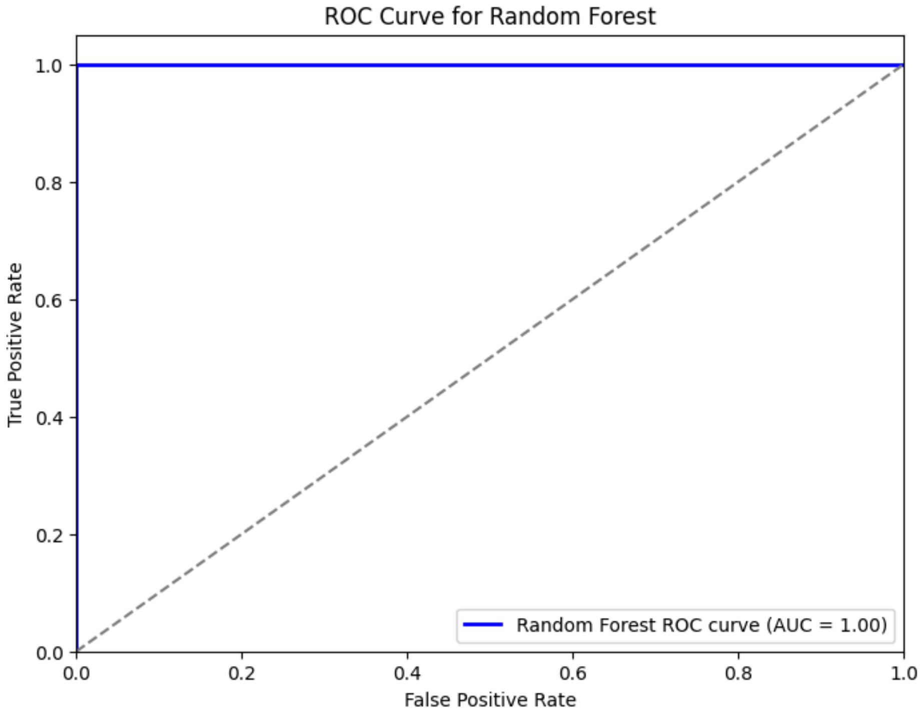
```
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Perform GridSearchCV
grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf, cv=5, n_jobs=-1, verbose=2)
grid_search_rf.fit(X_train, y_train)

# Best hyperparameters for Random Forest
print("Best Hyperparameters for Random Forest:", grid_search_rf.best_params_)

# Predict using the best model
y_pred_rf = grid_search_rf.best_estimator_.predict(X_test)
y_prob_rf = grid_search_rf.best_estimator_.predict_proba(X_test)[:, 1]
```

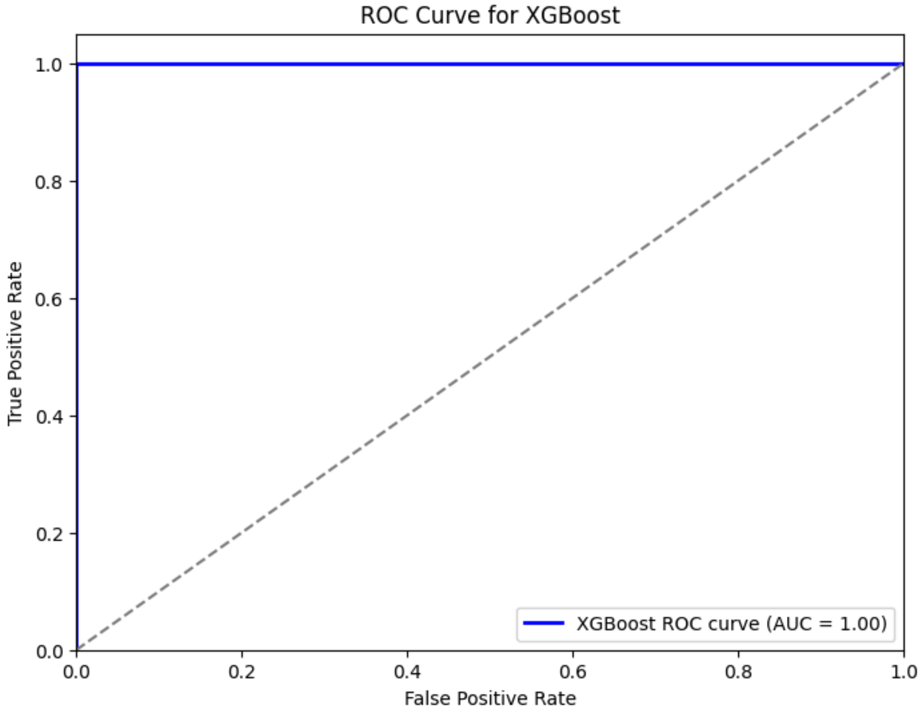
```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters for Random Forest: {'bootstrap': True, 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estir
```

```
import xgboost as xgb

# Initialize XGBoost model
xgb_model = xgb.XGBClassifier(random_state=42)

# Hyperparameter tuning with GridSearchCV for XGBoost
param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

# Perform GridSearchCV
grid_search_xgb = GridSearchCV(estimator=xgb_model, param_grid=param_grid_xgb, cv=5, n_jobs=-1, verbose=2)
grid_search_xgb.fit(X_train, y_train)

# Best hyperparameters for XGBoost
print("Best Hyperparameters for XGBoost:", grid_search_xgb.best_params_)

# Predict using the best model
y_pred_xgb = grid_search_xgb.best_estimator_.predict(X_test)
y_prob_xgb = grid_search_xgb.best_estimator_.predict_proba(X_test)[:, 1]
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
Best Hyperparameters for XGBoost: {'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 50, 'subsample':
```

```
# Function to plot ROC curve
def plot_roc_curve(y_test, y_prob, model_name):
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'{model_name} ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line (random classifier)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {model_name}')
    plt.legend(loc='lower right')
    plt.show()

# --- Model 1: Random Forest (Bagging) ---
# Predict probabilities using the best Random Forest model
y_prob_rf = grid_search_rf.best_estimator_.predict_proba(X_test)[:, 1]  # Get probability for class 1 (positive class)
plot_roc_curve(y_test, y_prob_rf, "Random Forest")

# --- Model 2: XGBoost (Boosting) ---
# Predict probabilities using the best XGBoost model
y_prob_xgb = grid_search_xgb.best_estimator_.predict_proba(X_test)[:, 1]  # Get probability for class 1 (positive class)
plot_roc_curve(y_test, y_prob_xgb, "XGBoost")
```






```
# --- Evaluation: Classification Report ---
print("Random Forest Classification Report:\n", classification_report(y_test, y_pred_rf))
print("\nXGBoost Classification Report:\n", classification_report(y_test, y_pred_xgb))

# --- Evaluation: Confusion Matrix ---
# Function to plot confusion matrix
def plot_confusion_matrix(y_test, y_pred, model_name):
    cm = confusion_matrix(y_test, y_pred)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['True 0', 'True 1'])
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

# Plot Confusion Matrix for Random Forest
plot_confusion_matrix(y_test, y_pred_rf, "Random Forest")

# Plot Confusion Matrix for XGBoost
plot_confusion_matrix(y_test, y_pred_xgb, "XGBoost")
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       498
           1       1.00      1.00      1.00       472

    accuracy                           1.00       970
   macro avg       1.00      1.00      1.00       970
weighted avg       1.00      1.00      1.00       970
```