

## ✓ Problem Statement

Problem Summary: High driver attrition at Ola increases acquisition cost and disrupts operations.

Goal: Predict whether a driver is likely to leave the company based on demographics, income, tenure, and performance.

Impact: Improve driver retention strategy by identifying churn-prone drivers early.

## ✓ Objective

The primary objective of this case study is to develop a predictive model that accurately identifies drivers who are at high risk of attrition, enabling the business to take proactive retention actions. This includes:

- Leveraging supervised machine learning techniques to predict whether a driver will leave the organization based on historical and demographic data.
- Comparing ensemble models – Bagging (Random Forest) and Boosting (XGBoost, LightGBM, AdaBoost) – using F1 Score, ROC AUC, and Average Precision as evaluation metrics.
- Handling data imbalance using SMOTE and evaluating the impact of different scaling techniques (StandardScaler vs MinMaxScaler) on model performance.
- Identifying key features driving attrition such as Quarterly Rating, Income, Total Business Value, Join Year, and City.
- Recommending data-driven strategies for performance incentives and targeted retention programs to reduce churn and optimize operational efficiency.

## ✓ Imports

```
pip install --upgrade xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (3.0.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.16.0)
```

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import xgboost as xgb
```


```
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve, precision_recall_curve
```

```
from imblearn.over_sampling import SMOTE
```

```
# Ola brand-inspired colors (from the logo)
ola_palette = {
    'Yes': '#DAFF01',    # neon yellow-green
    'No': '#000000'      # black
}
```

```
ola_palette = ['#000000', '#F6D200'] # Black for 0 (No Attrition), Yellow for 1 (Attrition)
```

```
url= 'https://raw.githubusercontent.com/hiyer7/Data-Science-Projects/refs/heads/main/Ola/ola_driver_scaler.csv'
df= pd.read_csv(url)
df.head()
```




	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Joining Designation	Grade	Total Business Value
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	2381061
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	NaN	1	1	-665481
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	(

```
# df_gen= pd.read_csv(url)
# df_gen.head()

# df_gen['Gender'].value_counts()
```

## ✓ Converting Lastworkingday column as our target

```
df['Attrition']= df['LastWorkingDate'].notna().astype(int)
df.drop(columns= 'LastWorkingDate', inplace= True)
df['Attrition'].value_counts()
```




	count
Attrition	
0	17488
1	1616

dtype: int64

## ✓ EDA

### ✓ Shape


```
df.shape
```

 (19104, 14)

There are 14 columns and 19k rows

### ✓ Column names

```
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   Joining Designation  19104 non-null  int64
10  Grade                19104 non-null  int64
11  Total Business Value  19104 non-null  int64
12  Quarterly Rating     19104 non-null  int64
13  Attrition            19104 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 2.0+ MB
```

We will be working on these column data types at a later stage

LastWorkingDate is our target. The ones that do not have values are the ones who stayed and the ones that have values have left the platform

```
df.columns
```

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

## ✓ Empty values

```
#(df['LastWorkingDate'].isnull().sum()*100)/
```

```
missing_percent= (df.isnull().sum()*100)/len(df)
missing_percent= missing_percent[missing_percent>0]
missing_percent.round(2).sort_values(ascending= False)
```

```
↗
```

	0
<b>Age</b>	0.32
<b>Gender</b>	0.27

**dtype:** float64

We are ok with Last working date to have empty values but for age and gender, we will later see how we will deal with the empty values

## ✓ Changing to Categorical Columns

```
cat_cols= ['Gender', 'Quarterly Rating', 'Grade', 'Joining Designation', 'Education_Level', 'City']
for col in cat_cols:
    df[col]= df[col].astype('category')
```

1. Gender
2. City
3. Education\_level
4. Joining Designation
5. Grade
6. Quarterly Rating

## ✓ Unique Values

```
for col in df.columns:
    print(f'{col}: {df[col].nunique()}')
```

```
↗
```

```
Unnamed: 0: 19104
MMM-YY: 24
Driver_ID: 2381
Age: 36
Gender: 2
City: 29
Education_Level: 3
Income: 2383
Dateofjoining: 869
Joining Designation: 5
Grade: 5
Total Business Value: 10181
Quarterly Rating: 4
Attrition: 2
```

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

```
↔
```

	count
Gender	
0.0	11074
1.0	7978

**dtype:** int64

Start coding or [generate](#) with AI.

## ✓ Data Cleaning

The column unnamed is probably the serial number with high cardinality. We will delete it here itself

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

We will convert the date columns: MMM-YY, Date Of Joining, LastWorkingDate → datetime

```
dt_cols= ['MMM-YY', 'Dateofjoining']
for col in dt_cols:
    df[col]= pd.to_datetime(df[col], format='%d/%m/%y', errors='coerce')
```

Double-click (or enter) to edit

## ✓ Univariate Analysis

Numerical Features: Age, Income, Total Business Value, Quarterly Rating

Continuous Features: City, Gender, Education\_level, Joining Designation

```
ola_color = "#D9E021" # Neon yellow-green
bg_color = "#000000"
n_cols = 2
n_rows = (len(cat_cols) + 1) // n_cols

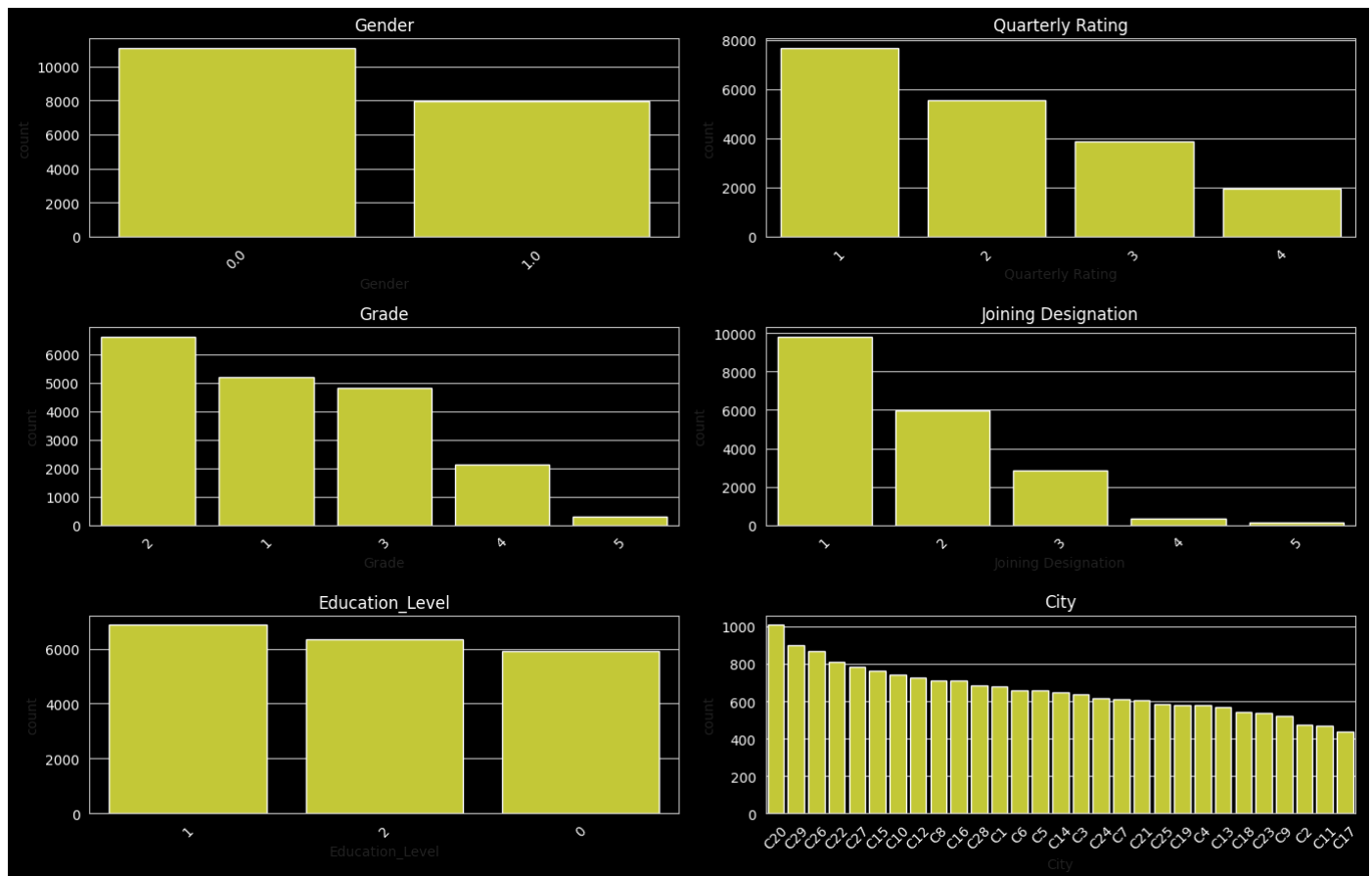
# Setup plots
sns.set_style("whitegrid")

fig, axes = plt.subplots(n_rows, n_cols, figsize=(14, 3 * n_rows), facecolor=bg_color)
axes = axes.flatten()

for i, col in enumerate(cat_cols):
    sns.countplot(x=col, data=df, ax=axes[i],
                  order=df[col].value_counts().index,
                  color=ola_color)
    axes[i].set_title(f'{col}', color='white')
    axes[i].tick_params(axis='x', rotation=45, colors='white')
    axes[i].tick_params(axis='y', colors='white')
    axes[i].set_facecolor(bg_color)

# Remove extra subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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



#### Interpretation:

##### 1. Gender:

- Slightly more males (0.0) than females (1.0).
- Imbalance, but not severe.

##### 2. Quarterly Rating:

- Most employees are rated 1 or 2, confirming earlier histogram.
- Very few top-rated employees (4).

##### 3. Grade:

- Grade 2 is most common, followed by Grade 1 and 3.
- Very few in Grade 5, suggesting few senior roles.

##### 4. Joining Designation:

- Majority joined at Designation 1, followed by 2 and 3.
- Indicates most hires were at entry/junior levels.

##### 5. Education Level:

- Distribution is fairly even among 3 categories.
- No dominant education level—likely all acceptable for hiring.

## 6. City:

- Employee count varies significantly across cities.
- A few cities dominate employment (e.g., C1, C2...), while others have much smaller numbers.
- Could reflect regional hiring patterns or business presence.

```
import matplotlib.pyplot as plt
import seaborn as sns

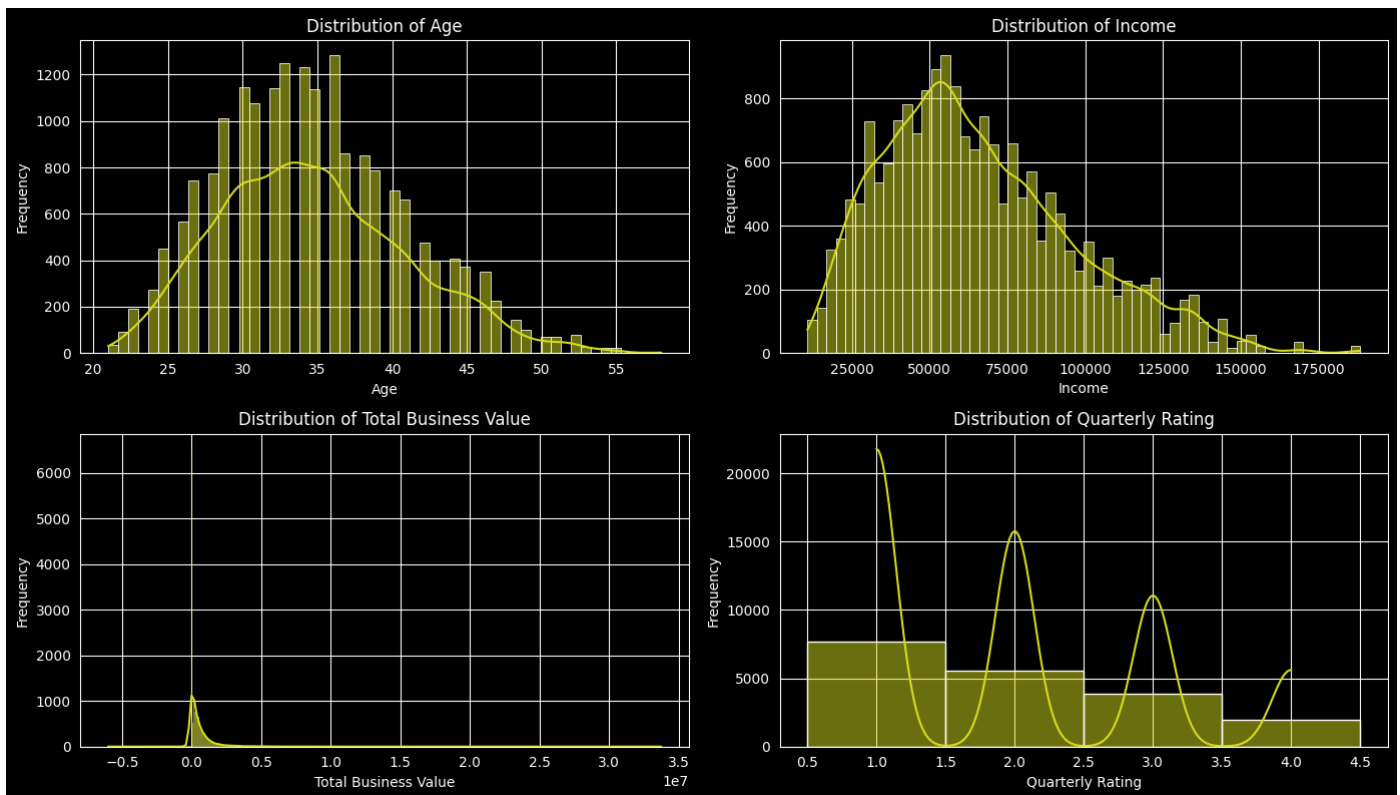
# Numerical columns
num_cols = ['Age', 'Income', 'Total Business Value', 'Quarterly Rating']

# Set global seaborn and matplotlib theme
sns.set_style("darkgrid")
plt.rcParams['axes.facecolor'] = bg_color
plt.rcParams['figure.facecolor'] = bg_color
plt.rcParams['text.color'] = 'white'
plt.rcParams['axes.labelcolor'] = 'white'
plt.rcParams['xtick.color'] = 'white'
plt.rcParams['ytick.color'] = 'white'
plt.rcParams['axes.edgecolor'] = 'white'

# Create 2x2 subplots
fig, axes = plt.subplots(2, 2, figsize=(14, 8))
axes = axes.flatten()

for i, col in enumerate(num_cols):
    sns.histplot(df[col], kde=True, ax=axes[i], color=ola_color, edgecolor='white')
    axes[i].set_title(f'Distribution of {col}', color='white')
    axes[i].set_xlabel(col, color='white')
    axes[i].set_ylabel('Frequency', color='white')
    axes[i].tick_params(colors='white')
    axes[i].set_facecolor(bg_color)

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



#### Interpretation:

##### 1. Top-Left: Age

- Shape: Slightly right-skewed distribution with a concentration around 30–35 years.
- Insight: Majority of employees are in their early 30s, with a tapering number as age increases.
- Implication: Workforce is relatively young; might suggest early-career professionals dominate.

##### 2. Top-Right: Income

- Shape: Right-skewed, long tail towards higher incomes.
- Insight: Most employees earn between ₹40,000–₹80,000, with a few high earners pulling the tail.
- Implication: Possible income disparity; a small fraction may be in executive/high-performing roles.

##### 3. Bottom-Left: Total Business Value

- Shape: Heavily right-skewed with a steep drop-off.
- Insight: Most employees contribute low to moderate business value; a few contribute disproportionately large values.
- Implication: Highlights a Pareto-type effect (80/20 rule) – a few top performers drive most of the business value.

##### 4. Bottom-Right: Quarterly Rating

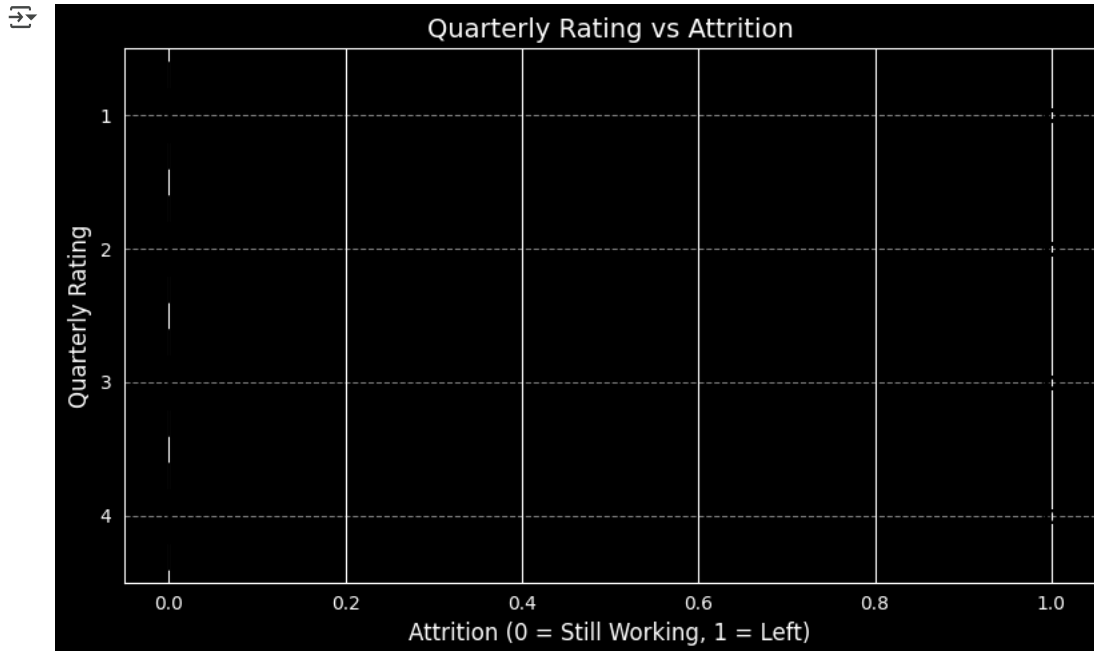
- Shape: Multimodal with peaks at 1, 2, and 3.
- Insight: Ratings are heavily concentrated at lower scores (1–3).
- Implication: Might reflect a strict performance appraisal system or overall moderate performance distribution.

## ✓ Bivariate Analysis

### ✓ Quarterly rating vs Attrition (boxplots)

```
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='Attrition', y='Quarterly Rating', palette=ola_palette)

plt.title('Quarterly Rating vs Attrition', fontsize=14)
plt.xlabel('Attrition (0 = Still Working, 1 = Left)', fontsize=12)
plt.ylabel('Quarterly Rating', fontsize=12)
plt.grid(True, axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



Insight from the Plot:

Attrition = 1 (Left): The median Quarterly Rating is 1, with almost no variance—most employees who left received the lowest possible rating.

Attrition = 0 (Still Working): Wider spread of ratings, with a median around 3 and values ranging from 1 to 4.

Interpretation:

Employees who left the company tend to have consistently poor performance ratings.

This might suggest:

1. A performance-based exit policy, or
2. Low motivation among poor performers, leading them to leave, or
3. Poor performers being pushed out.

### ✓ City/Gender vs Attrition (grouped bar plots)

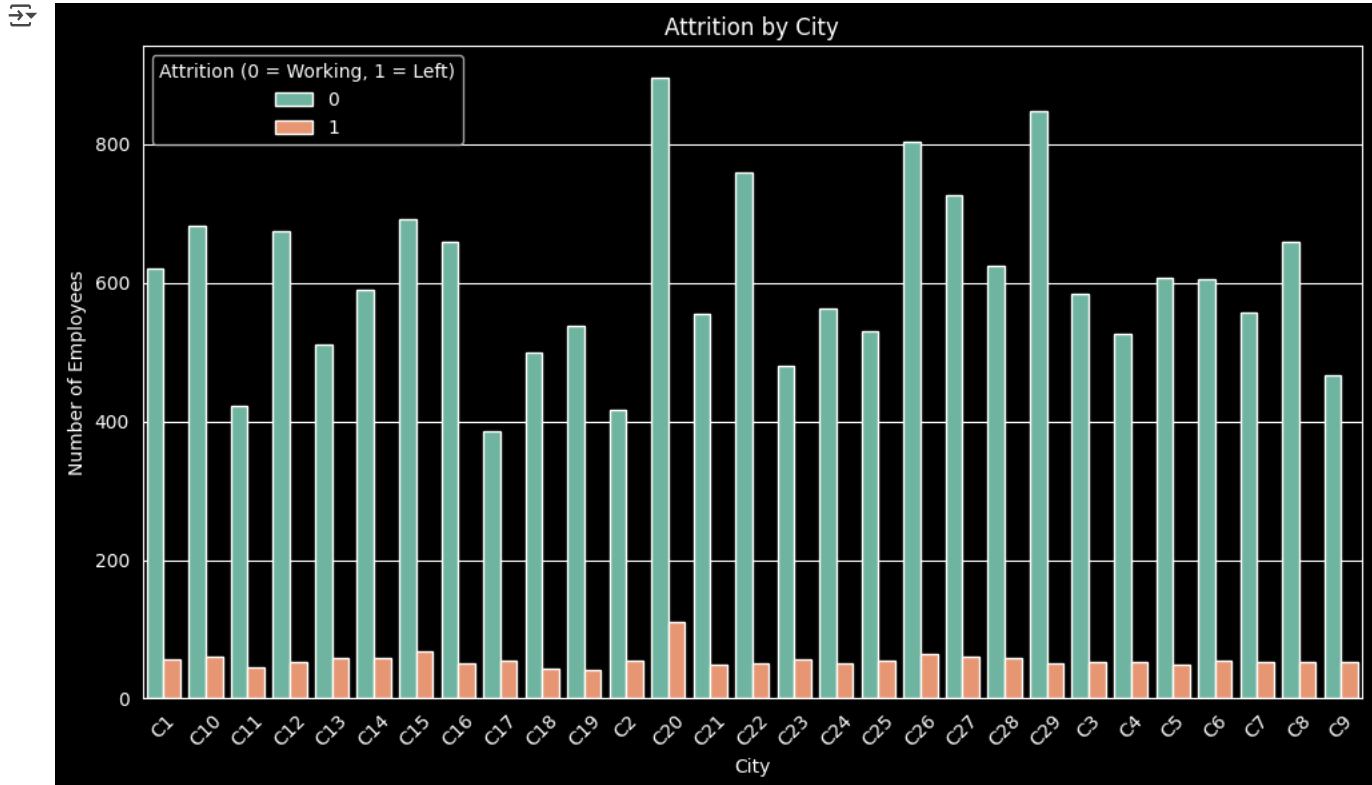
### ✓ City vs Attrition

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='City', hue='Attrition', palette='Set2')
```



```
plt.title('Attrition by City')
plt.xlabel('City')
plt.ylabel('Number of Employees')
plt.xticks(rotation=45)
plt.legend(title='Attrition (0 = Working, 1 = Left)')
plt.tight_layout()
plt.show()
```

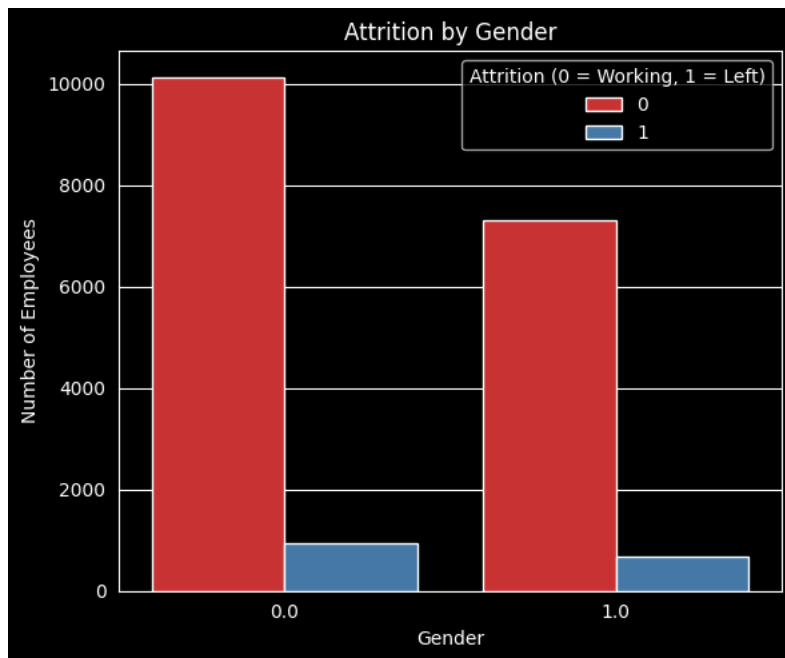


#### Observations:

1. Most cities have a significantly larger number of employees still working (Attrition = 0), as shown by the tall green bars.
2. City C20 has the highest employee count and also the highest attrition (orange bar) — suggesting it's a major location with potential retention issues.
3. Some cities (e.g., C7, C16, C10) have relatively low total employee counts but still experience notable attrition.
4. The proportion of attrition (not just count) may reveal more — currently, it looks like C20 and a few others like C26 or C29 could be problem spots.

#### ✓ Gender vs Attrition

```
plt.figure(figsize=(6, 5))
sns.countplot(data=df, x='Gender', hue='Attrition', palette='Set1')
plt.title('Attrition by Gender')
plt.xlabel('Gender')
plt.ylabel('Number of Employees')
plt.legend(title='Attrition (0 = Working, 1 = Left)')
plt.tight_layout()
plt.show()
```



Observation:

1. More male employees than female overall.
2. Both genders experience attrition, but:
  - a. Absolute attrition is slightly higher for females (Gender 0).
  - b. The attrition rate (leavers / total) appears higher for females compared to males.
    - Female: ~1000 left out of ~11000 → ~9%
    - Male: ~750 left out of ~8000 → ~9.3%
3. Actually, in percentage terms, male attrition might be slightly higher, despite having fewer leavers in absolute numbers.

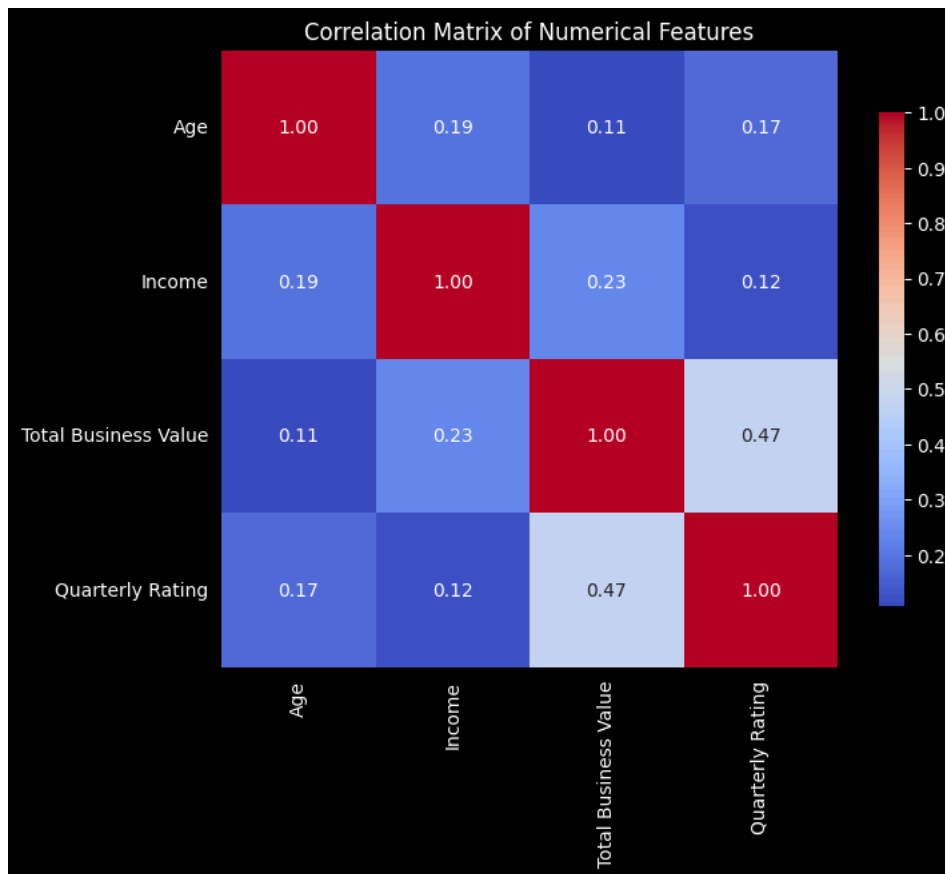
## ✓ Correlation matrix (heatmap) for numerical variables

```
num_cols = ['Age', 'Income', 'Total Business Value', 'Quarterly Rating']
```

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Compute correlation matrix
corr_matrix = df[num_cols].corr()
```

```
# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True, cbar_kws={"shrink": .8})
plt.title("Correlation Matrix of Numerical Features")
plt.show()
```



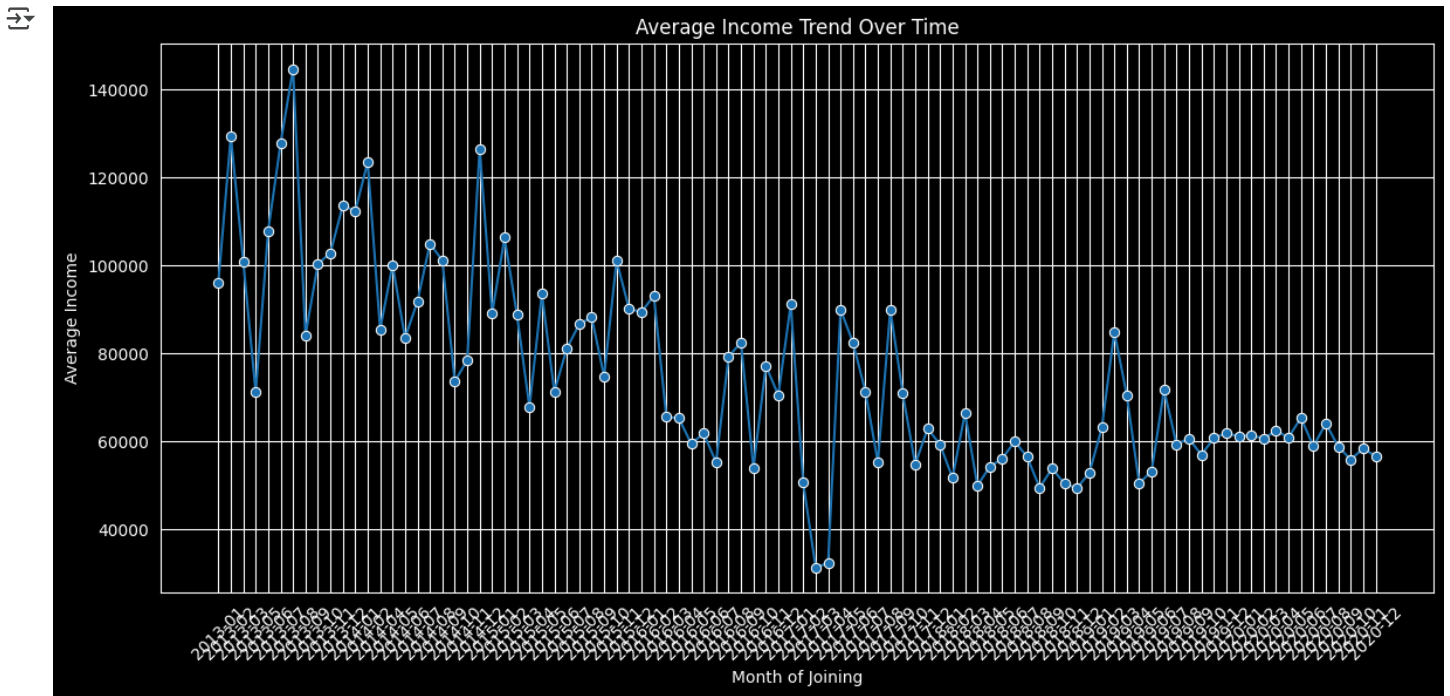
### Income trend over time (line plot)

```
# Ensure 'Joining Date' is in datetime format
#df['Joining Date'] = pd.to_datetime(df['Joining Date'])

# Create a new column for year or month
df['Join_Month'] = df['Dateofjoining'].dt.to_period('M').astype(str) # or .dt.to_period('Y') for yearly

# Group by month and calculate average income
monthly_income = df.groupby('Join_Month')['Income'].mean().reset_index()

# Plot
plt.figure(figsize=(12, 6))
sns.lineplot(data=monthly_income, x='Join_Month', y='Income', marker='o')
plt.title("Average Income Trend Over Time")
plt.xlabel("Month of Joining")
plt.ylabel("Average Income")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Interpretation:

1. Pattern: Average income steadily declined from earlier months (starting around 2008) to recent periods.
2. Initial Highs: Employees who joined early (2008–2010) had higher average salaries.
3. Recent Lows: Post-2015, income flattens around ₹50,000–₹60,000.

Implications:

1. Indicates seniority/tenure-based income growth.
2. Or a change in hiring policy – hiring more junior/lower-cost resources over time.
3. Possibly aligns with organizational cost-cutting or scale-up strategy with younger talent.

## ✓ Outlier Detection

## ✓ Boxplots of continuous features

```
import matplotlib.pyplot as plt
import seaborn as sns

# Numerical columns
num_cols = ['Age', 'Income', 'Total Business Value', 'Quarterly Rating']

# Ola color scheme
ola_color = "#D9E021"
bg_color = "#000000"

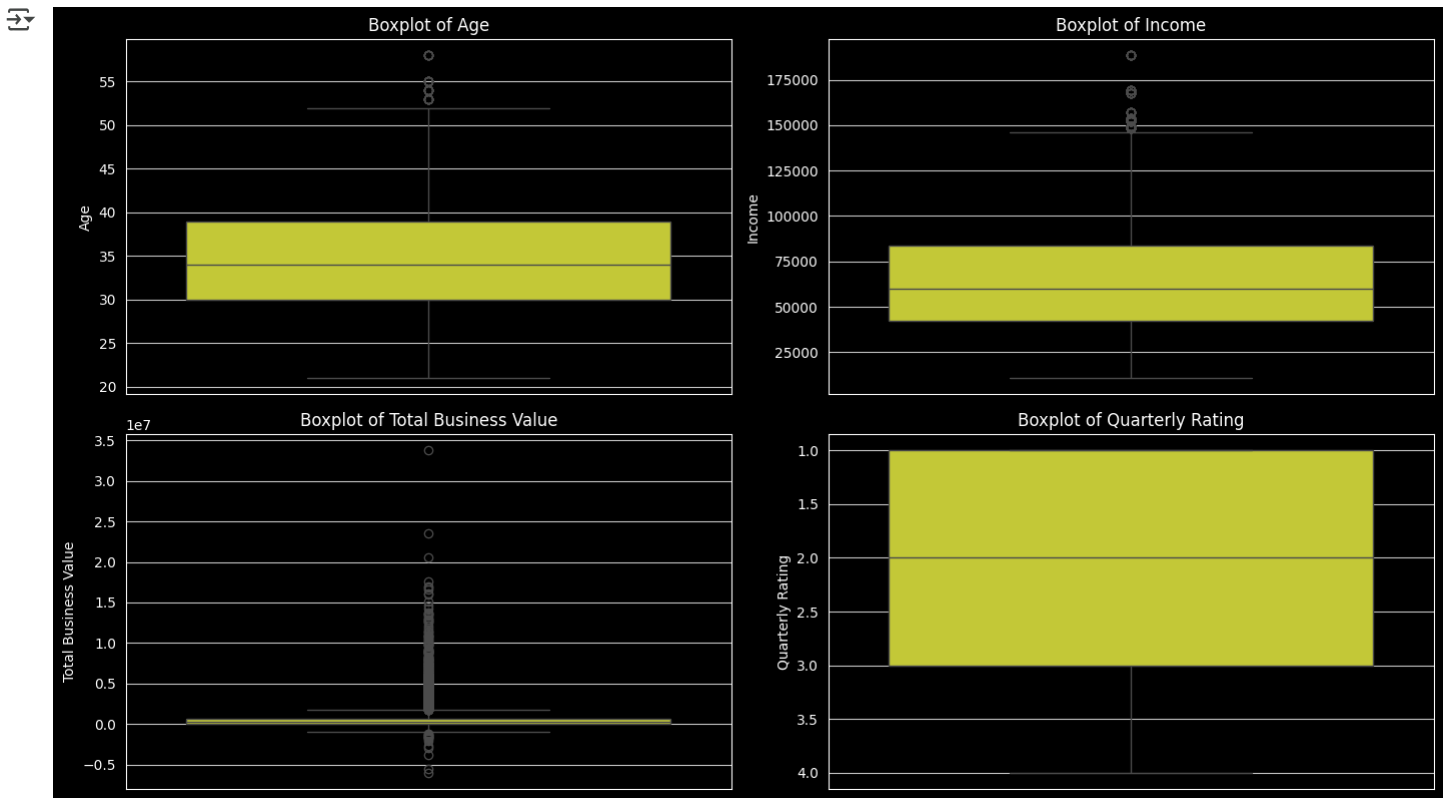
# Set global theme
sns.set_style("whitegrid")
plt.rcParams['axes.facecolor'] = bg_color
plt.rcParams['figure.facecolor'] = bg_color
plt.rcParams['text.color'] = 'white'
plt.rcParams['axes.labelcolor'] = 'white'
plt.rcParams['xtick.color'] = 'white'
plt.rcParams['ytick.color'] = 'white'
plt.rcParams['axes.edgecolor'] = 'white'

# Create 2x2 grid
```

```
fig, axes = plt.subplots(2, 2, figsize=(14, 8))
axes = axes.flatten()

for i, col in enumerate(num_cols):
    sns.boxplot(y=df[col], ax=axes[i], color=ola_color)
    axes[i].set_title(f'Boxplot of {col}', color='white')
    axes[i].set_ylabel(col, color='white')
    axes[i].tick_params(colors='white')
    axes[i].set_facecolor(bg_color)

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



Feature	Skewed?	Outliers Present?	Implication
Age	Mildly right	Yes (older employees)	Age-related attrition or senior-level insights can be investigated
Income	Yes (right)	Many (high earners)	Look into income vs attrition, grade, or performance
Business Value	Very skewed	Yes (strong right tail)	High leverage individuals can be identified
Rating	Slight	Yes (0 and 4)	Might affect attrition or promotion paths

✦ Z-score or IQR method for outlier treatment

For tree-based models: keep outliers or cap them (robust)

For regression or distance-based models: remove or cap them

```
import numpy as np

# Step 1: Ensure numerical columns are numeric
num_cols = ['Age', 'Income', 'Total Business Value', 'Quarterly Rating']
```

```

df[num_cols] = df[num_cols].apply(pd.to_numeric, errors='coerce')

# Step 2: Initialize summary dictionary
outlier_flags = {}

# Step 3: Loop through each numerical feature and flag outliers
for col in num_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Flag outliers: 1 for outlier, 0 otherwise
    outlier_col = f'{col}_outlier'
    df[outlier_col] = ((df[col] < lower_bound) | (df[col] > upper_bound)).astype(int)

    # Save summary
    count = df[outlier_col].sum()
    percent = 100 * count / len(df)
    outlier_flags[col] = (count, round(percent, 2))

# Step 4: Print summary
print("Outlier Summary:")
for col, (count, percent) in outlier_flags.items():
    print(f"{col:22}: {count:5} outliers ({percent:5.2f}%)")

```

```

➡ Outlier Summary:
Age           :    78 outliers ( 0.41%)
Income        :   188 outliers ( 0.98%)
Total Business Value : 1371 outliers ( 7.18%)
Quarterly Rating :     0 outliers ( 0.00%)

```

Start coding or [generate](#) with AI.

## ✓ Data Preprocessing & Feature Engineering

### ✓ Missing Value Imputation

#### ✓ kNN Imputation

```

from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler

# Step 1: Select numeric features for imputation (excluding IDs, targets, outlier flags)
exclude_cols = ['Driver_ID', 'target'] + [col for col in df.columns if 'outlier' in col]
num_features = [col for col in df.select_dtypes(include=['float64', 'int64']).columns if col not in exclude_cols]

# Step 2: Standardize the numeric features
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df[num_features])

# Step 3: Apply KNN Imputer
imputer = KNNImputer(n_neighbors=5)
imputed_data = imputer.fit_transform(scaled_data)

# Step 4: Replace original missing values with imputed values
df[num_features] = scaler.inverse_transform(imputed_data)

# Optional: Check if any missing values remain
print("Missing values remaining:")
print(df[num_features].isnull().sum())

```

```

➡ Missing values remaining:
Age           0
Income        0
Total Business Value  0
Quarterly Rating  0
Attrition     0
dtype: int64

```

Aggregation

Cringeat a new DataFrame with one row per driver, containing:


- Aggregated numerical values
- Categorical values from the latest record
- First and last date-related fields

```
agg_funcs = {
    'Income': ['max', 'min', 'mean', 'std'],
    'Quarterly Rating': ['max', 'min', 'mean', 'std'],
    'Total Business Value': ['max', 'min', 'mean', 'std'],
    'Grade': 'last',
    'City': 'last',
    'Dateofjoining': 'first',
    'Attrition': 'max'
}

df_agg = df.groupby('Driver_ID').agg(agg_funcs)

# Flatten multi-level columns
df_agg.columns = ['_'.join(col).strip() if isinstance(col, tuple) else col for col in df_agg.columns.values]
df_agg.reset_index(inplace=True)
```

df\_agg



	Driver_ID	Income_max	Income_min	Income_mean	Income_std	Quarterly Rating_max	Quarterly Rating_min	Quarterly Rating_mean	Quarterly Rating_std	Total Business Value_max	1 Busi Value
0	1	57387.0	57387.0	57387.0	0.0	2.0	2.0	2.000000	0.000000	2.381060e+06	-6.654800
1	2	67016.0	67016.0	67016.0	0.0	1.0	1.0	1.000000	0.000000	1.164153e-10	1.164153
2	4	65603.0	65603.0	65603.0	0.0	1.0	1.0	1.000000	0.000000	3.500000e+05	1.164153
3	5	46368.0	46368.0	46368.0	0.0	1.0	1.0	1.000000	0.000000	1.203600e+05	1.164153
4	6	78728.0	78728.0	78728.0	0.0	2.0	1.0	1.600000	0.547723	1.265000e+06	1.164153
...	...	...	...	...	...	...	...	...	...	...	...
2376	2784	82815.0	82815.0	82815.0	0.0	4.0	1.0	2.625000	1.013496	4.495040e+06	1.164153
2377	2785	12105.0	12105.0	12105.0	0.0	1.0	1.0	1.000000	0.000000	1.164153e-10	1.164153
2378	2786	35370.0	35370.0	35370.0	0.0	2.0	1.0	1.666667	0.500000	9.703800e+05	1.164153
2379	2787	69498.0	69498.0	69498.0	0.0	2.0	1.0	1.500000	0.547723	4.080900e+05	1.164153
2380	2788	70254.0	70254.0	70254.0	0.0	3.0	1.0	2.285714	0.755929	7.402800e+05	1.164153

2381 rows × 17 columns

Feature Engineering

- income\_increased: 1 if monthly income shows upward trend
- rating\_improved: 1 if rating trend increases over time
- target: 1 if LastWorkingDate is not null
- Tenure in months: last\_report\_date - Date Of Joining
- Average monthly business / average rating

Income increased

```
def detect_income_increase(df):
    df_sorted = df.sort_values(['Driver_ID', 'MMM-YY'])
```

```
income_trend = df_sorted.groupby('Driver_ID')['Income'].apply(
    lambda x: 1 if x.is_monotonic_increasing else 0
).reset_index(name='income_increased')
return income_trend
```

## ✓ Rating Improved

```
def detect_rating_increase(df):
    df_sorted = df.sort_values(['Driver_ID', 'MMM-YY'])
    rating_trend = df_sorted.groupby('Driver_ID')['Quarterly Rating'].apply(
        lambda x: 1 if x.is_monotonic_increasing else 0
    ).reset_index(name='rating_improved')
    return rating_trend
```

## ✓ Tenure in months: Difference between last report date and join date

```
#df_agg['Dateofjoining'] = pd.to_datetime(df_agg['Dateofjoining'])
latest_report_date = df['MMM-YY'].max() # use from original dataset
df_agg['Tenure_months'] = ((latest_report_date - df_agg['Dateofjoining_first']).dt.days / 30.44).round(1)
```

## ✓ Average Monthly Business / Average Rating:

```
df_agg['Business_per_rating'] = df_agg['Total Business Value_mean'] / df_agg['Quarterly Rating_mean']
```

## ✓ Merging new features into the df:

```
# Detect trends from original df
income_trend = detect_income_increase(df)
rating_trend = detect_rating_increase(df)

# Merge with df_agg
df_agg = df_agg.merge(income_trend, on='Driver_ID', how='left')
df_agg = df_agg.merge(rating_trend, on='Driver_ID', how='left')
```

## ✓ Encoding

Double-click (or enter) to edit

```
df['Join_Year'] = df['Dateofjoining'].dt.year
df['Join_Month'] = df['Dateofjoining'].dt.month
# Then drop the original datetime columns
df.drop(columns=['MMM-YY', 'Dateofjoining'], inplace= True)
```

Since we have extracted meaningful information from the datetime columns, we will be dropping them

## ✓ Gender

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

```

Gender
0.0    11074
1.0     7978
dtype: int64
```

```
df= df[df['Gender'].isin([0.0, 1.0])]
```



```
df['Gender'] = df['Gender'].astype(int)
df['Gender'].value_counts()
```

```

Gender
0      11074
1       7978

dtype: int64
```

These entries are not interpretable, and since the counts are low (only ~40 rows), it's safe to drop without risking model generalization. With this we avoid introducing noise or label leakage.

One-Hot Encode: Columns:

```
*'City'

*'Joining_Designation'

*'Grade'

*'Education_Level'
```

Double-click (or enter) to edit

```
cols_to_onehot = ['City', 'Joining Designation', 'Grade', 'Education_Level']
df = pd.get_dummies(df, columns=cols_to_onehot, drop_first=True)
```

## ✓ Class Imbalance Treatment

### ✓ Checking Class Distribution of target

```
df['Attrition'].value_counts(normalize=True)
```

```

Attrition
0.0      0.915442
1.0      0.084558

dtype: float64
```

Since the minority class is < 30%, we will be performing the imbalance treatment

Method	Description	When to Use
<b>SMOTE</b>	Synthetic Minority Over-sampling Technique (creates new synthetic examples)	Works well for tabular numerical data
<b>ADASYN</b>	Similar to SMOTE but focuses more on harder-to-classify minority samples	Good if minority class is spread unevenly
<b>Class Weights</b>	Assign higher penalty to misclassifying minority class	Best with models that support weighting

## ✓ Splitting the dataset into train and test set

```
X = df.drop('Attrition', axis=1)
y = df['Attrition']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

## ✓ Applying SMOTE

```
smote= SMOTE(random_state= 42)
```

```
X_train_smote, y_train_smote= smote.fit_resample(X_train, y_train)
```

```
y_train_smote.value_counts()
```

```

Attrition
0.0    13952
1.0    13952

dtype: int64

```

The data is balanced now

```
X_train_smote.info()
```

Show hidden output

## ✓ Feature Scaling

```
X_train_smote.columns
```

```

Index(['Driver_ID', 'Age', 'Gender', 'Income', 'Total Business Value',
       'Quarterly Rating', 'Join_Month', 'Age_outlier', 'Income_outlier',
       'Total Business Value_outlier', 'Quarterly Rating_outlier', 'Join_Year',
       'City_C10', 'City_C11', 'City_C12', 'City_C13', 'City_C14', 'City_C15',
       'City_C16', 'City_C17', 'City_C18', 'City_C19', 'City_C2', 'City_C20',
       'City_C21', 'City_C22', 'City_C23', 'City_C24', 'City_C25', 'City_C26',
       'City_C27', 'City_C28', 'City_C29', 'City_C3', 'City_C4', 'City_C5',
       'City_C6', 'City_C7', 'City_C8', 'City_C9', 'Joining Designation_2',
       'Joining Designation_3', 'Joining Designation_4',
       'Joining Designation_5', 'Grade_2', 'Grade_3', 'Grade_4', 'Grade_5',
       'Education_Level_1', 'Education_Level_2'],
      dtype='object')

```

```
X_train_smote['Driver_ID'].dtype!= 'bool'
```

True

```

num_cols= []
for col in X_train_smote.columns:
    if X_train_smote[col].dtype!= 'bool':
        num_cols.append(col)
num_cols

```

```

['Driver_ID',
 'Age',
 'Gender',
 'Income',
 'Total Business Value',
 'Quarterly Rating',
 'Join_Month',
 'Age_outlier',
 'Income_outlier',
 'Total Business Value_outlier',
 'Quarterly Rating_outlier',
 'Join_Year']

```

```
sscaler= StandardScaler()
```

```
X_train_sscaled= X_train_smote.copy()
```

```
X_test_sscaled = X_test.copy()
```

```

X_train_sscaled[num_cols]= sscaler.fit_transform(X_train_sscaled[num_cols])
X_test_sscaled[num_cols] = sscaler.transform(X_test_sscaled[num_cols])


```

```
mmscaler= MinMaxScaler()

X_train_mmscaled= X_train_smote.copy()
X_train_mmscaled[num_cols]= mmscaler.fit_transform(X_train_mmscaled[num_cols])

X_test_mmscaled = X_test.copy()
X_test_mmscaled[num_cols] = mmscaler.transform(X_test_mmscaled[num_cols])

X_train_mmscaled[num_cols].describe()
```



	Driver_ID	Age	Gender	Income	Total Business Value	Quarterly Rating	Join_Month	Age_outlier	Income_outlier	Value
count	27904.000000	27904.000000	27904.000000	27904.000000	27904.000000	27904.000000	27904.000000	27904.000000	27904.000000	279
mean	0.507271	0.351425	0.308809	0.282770	0.159266	0.205135	0.505555	0.002043	0.005591	
std	0.283217	0.154509	0.462010	0.164736	0.022501	0.299194	0.265901	0.045151	0.074562	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.263007	0.243243	0.000000	0.163651	0.150952	0.000000	0.363636	0.000000	0.000000	
50%	0.507894	0.332585	0.000000	0.254898	0.150952	0.000000	0.545455	0.000000	0.000000	
75%	0.757445	0.442099	1.000000	0.377315	0.159758	0.333333	0.727273	0.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

Start coding or [generate](#) with AI.

- Model Building
- Bagging Model
- Defining Hyperparameter Grid for RandomizedSearchCV

```
param_grid= {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}
```

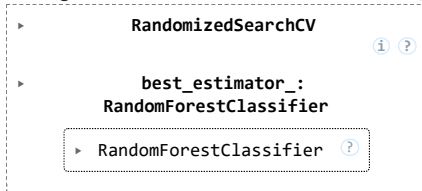
- Initialize and run RandomizedSearchCV

```
rf= RandomForestClassifier(random_state= 42, class_weight= 'balanced')
rf_random_st= RandomizedSearchCV(estimator= rf,
                                param_distributions = param_grid,
                                n_iter= 20,
                                cv= 5,
                                verbose= 2,
                                n_jobs= -1,
                                scoring= 'f1')

rf_random_mm= RandomizedSearchCV(estimator= rf,
                                param_distributions = param_grid,
                                n_iter= 20,
                                cv= 5,
                                verbose= 2,
                                n_jobs= -1,
                                scoring= 'f1')

rf_random_st.fit(X_train_sscaled, y_train_smote)
rf_random_mm.fit(X_train_mmscaled, y_train_smote)
```

↗ Fitting 5 folds for each of 20 candidates, totalling 100 fits  
 Fitting 5 folds for each of 20 candidates, totalling 100 fits



## ✓ Best Estimator and Evaluation

```
best_rf_st= rf_random_st.best_estimator_
y_pred_rf_st= best_rf_st.predict(X_train_sscaled)

print('Confusion Matrix')
print(confusion_matrix(y_train_smote, y_pred_rf_st))

print("\nClassification Report: ")
print(classification_report(y_train_smote, y_pred_rf_st))
```

↗ Confusion Matrix  
 [[13764 188]  
 [ 668 13284]]

Classification Report:

	precision	recall	f1-score	support
0.0	0.95	0.99	0.97	13952
1.0	0.99	0.95	0.97	13952
accuracy			0.97	27904
macro avg	0.97	0.97	0.97	27904
weighted avg	0.97	0.97	0.97	27904

```
best_rf_mm= rf_random_mm.best_estimator_
y_pred_rf_mm= best_rf_mm.predict(X_train_mmscaled)

print('Confusion Matrix')
print(confusion_matrix(y_train_smote, y_pred_rf_mm))

print("\nClassification Report: ")
print(classification_report(y_train_smote, y_pred_rf_mm))
```

↗ Confusion Matrix  
 [[13788 164]  
 [ 481 13471]]

Classification Report:

	precision	recall	f1-score	support
0.0	0.97	0.99	0.98	13952
1.0	0.99	0.97	0.98	13952
accuracy			0.98	27904
macro avg	0.98	0.98	0.98	27904
weighted avg	0.98	0.98	0.98	27904

## ✓ Feature Importance Plot

```
import matplotlib.pyplot as plt
import seaborn as sns

importance_st= best_rf_st.feature_importances_
feature_names_st= X_train_sscaled.columns

importance_mm= best_rf_mm.feature_importances_
feature_names_mm= X_train_mmscaled.columns

feat_imp_df_st= pd.DataFrame({'Feature': feature_names_st, 'Importance': importance_st})
feat_imp_df_st= feat_imp_df_st.sort_values(by= 'Importance', ascending= False)
```

```

feat_imp_df_mm= pd.DataFrame({'Feature': feature_names_mm, 'Importance': importance_mm})
feat_imp_df_mm= feat_imp_df_mm.sort_values(by= 'Importance', ascending= False)

# Top 20 for both
top_feats_st = feat_imp_df_st.head(20)
top_feats_mm = feat_imp_df_mm.head(20)

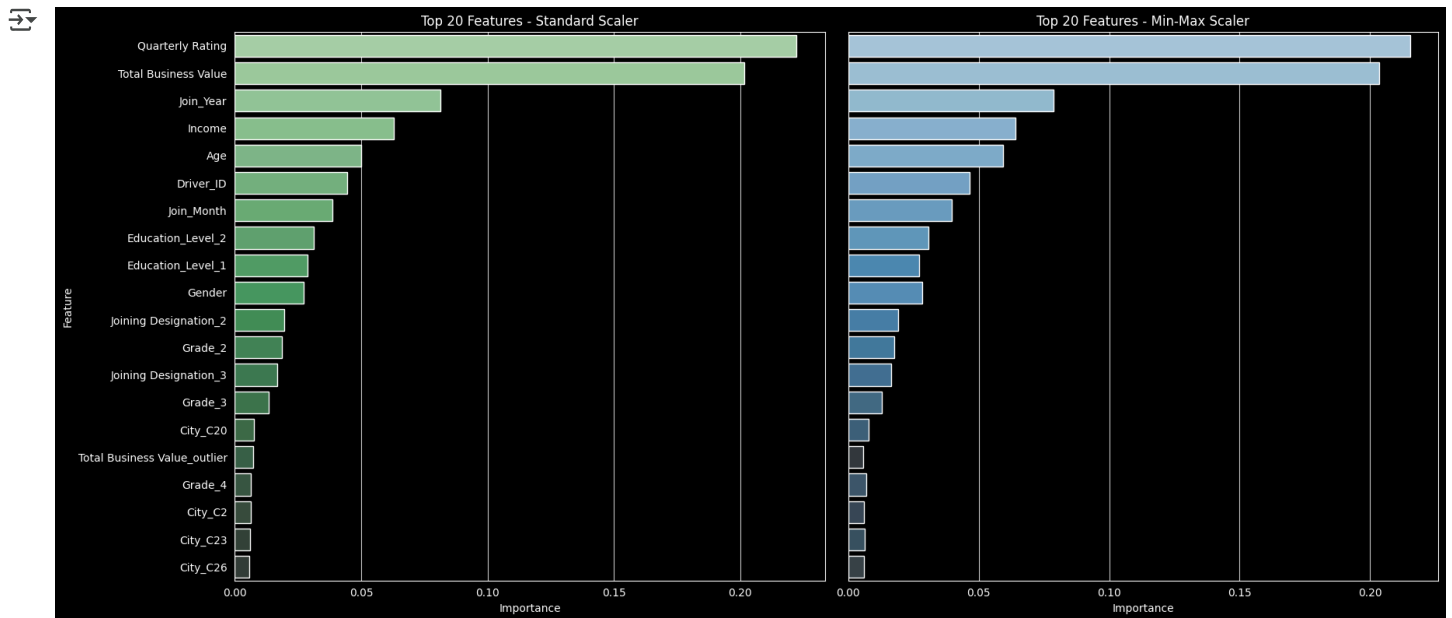
# Plot side by side
fig, axes = plt.subplots(1, 2, figsize=(18, 8), sharey=True)

# Standard Scaled
sns.barplot(ax=axes[0], x='Importance', y='Feature', data=top_feats_st, palette='Greens_d')
axes[0].set_title('Top 20 Features - Standard Scaler')

# Min-Max Scaled
sns.barplot(ax=axes[1], x='Importance', y='Feature', data=top_feats_mm, palette='Blues_d')
axes[1].set_title('Top 20 Features - Min-Max Scaler')

plt.tight_layout()
plt.show()

```



Inference:

This means scaling didn't distort feature importance in Random Forest (which is scale-invariant).

The feature importance chart displays the top 20 predictors contributing to the Random Forest model's decision-making process. Here's the interpretation:

Top Influencers:

1. Quarterly Rating and Total Business Value are by far the most significant predictors, with combined importance exceeding 40%. This suggests that driver performance and revenue contribution are strong indicators of the target variable.
2. Join\_Year also has a notable impact, indicating the year of joining is correlated with the outcome — potentially reflecting experience or policy changes over time.

## Secondary Influencers:

1. Income, Age, and Driver\_ID form the next tier of importance. This implies that socio-demographic characteristics and unique identifiers might indirectly capture behavior or engagement trends.
2. Join\_Month, Education\_Level\_2, and Gender also add moderate predictive power.

## Categorical Feature Impact:

3. Education\_Level\_1, Joining\_Designation\_2/3, and Grade\_2/3/4 have relatively lower importance but still contribute meaningfully. The model benefits from including detailed role and designation info.
4. Some city dummies like City\_C20, City\_C2, City\_C23, and City\_C26 appear, though with minimal influence individually.

## Outlier Impact:

1. The presence of Total Business Value\_outlier indicates that detecting extreme values in business contribution has predictive value, albeit smaller.

## Summary:

The Random Forest model is heavily influenced by performance metrics (Quarterly Rating), revenue (Total Business Value), and temporal factors (Join\_Year). Demographics and designation-level categories play a secondary role. These insights could be valuable for:

- Designing retention strategies
- Segmenting drivers
- Prioritizing features for simplified models

## Model Evaluation

```
from sklearn.metrics import classification_report, roc_auc_score, roc_curve, f1_score, average_precision_score

# Predict labels and probabilities
y_pred_st = best_rf_st.predict(X_test_sscaled)
y_proba_st = best_rf_st.predict_proba(X_test_sscaled)[: , 1]

y_pred_mm = best_rf_mm.predict(X_test_mmscaled)
y_proba_mm = best_rf_mm.predict_proba(X_test_mmscaled)[: , 1]

# Metrics
f1_st = f1_score(y_test, y_pred_st)
f1_mm = f1_score(y_test, y_pred_mm)

roc_auc_st = roc_auc_score(y_test, y_proba_st)
roc_auc_mm = roc_auc_score(y_test, y_proba_mm)

avg_precision_st = average_precision_score(y_test, y_proba_st)
avg_precision_mm = average_precision_score(y_test, y_proba_mm)

# Print results
print("Random Forest Performance Comparison:\n")
print(f"Standard Scaler - F1 Score: {f1_st:.4f}, ROC AUC: {roc_auc_st:.4f}, Average Precision: {avg_precision_st:.4f}")
print(f"MinMax Scaler - F1 Score: {f1_mm:.4f}, ROC AUC: {roc_auc_mm:.4f}, Average Precision: {avg_precision_mm:.4f}")
```

➡ Random Forest Performance Comparison:

```
Standard Scaler - F1 Score: 0.2049, ROC AUC: 0.7773, Average Precision: 0.2046
MinMax Scaler - F1 Score: 0.1856, ROC AUC: 0.7646, Average Precision: 0.1898
```

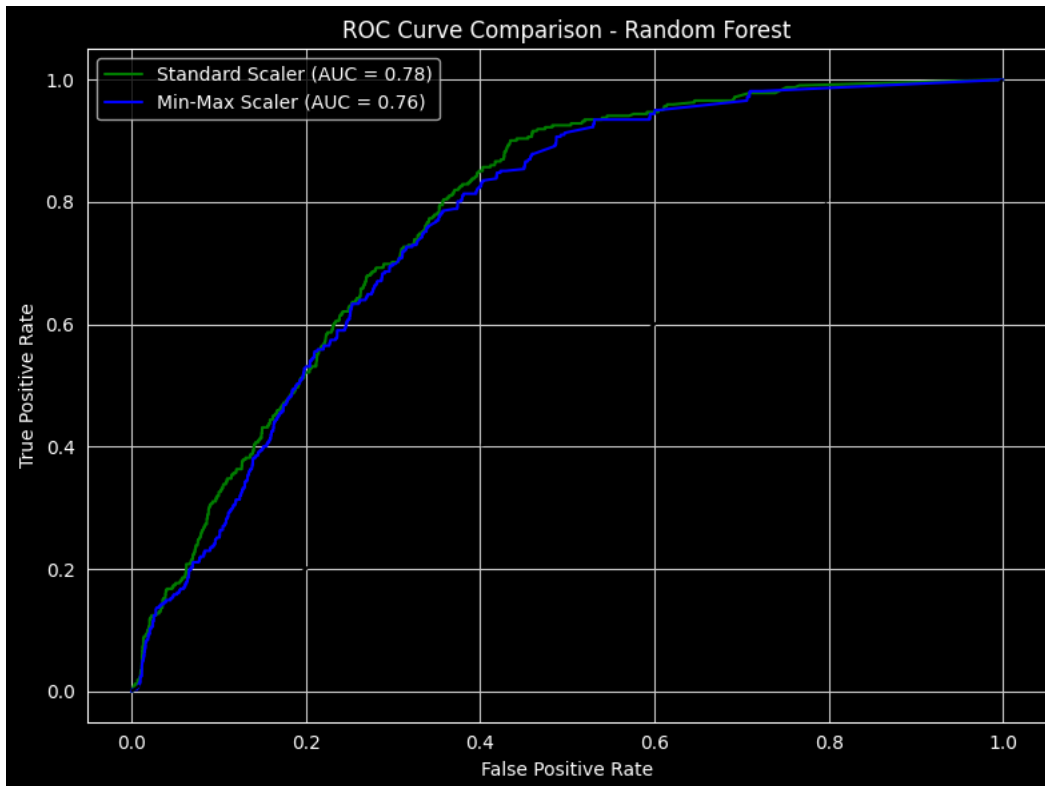
## ROC Curve Comparison

```
fpr_st, tpr_st, _ = roc_curve(y_test, y_proba_st)
fpr_mm, tpr_mm, _ = roc_curve(y_test, y_proba_mm)

plt.figure(figsize=(8, 6))
plt.plot(fpr_st, tpr_st, label=f'Standard Scaler (AUC = {roc_auc_st:.2f})', color='green')
plt.plot(fpr_mm, tpr_mm, label=f'Min-Max Scaler (AUC = {roc_auc_mm:.2f})', color='blue')
plt.plot([0, 1], [0, 1], 'k--') # baseline

plt.xlabel("False Positive Rate")
```

```
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison - Random Forest")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



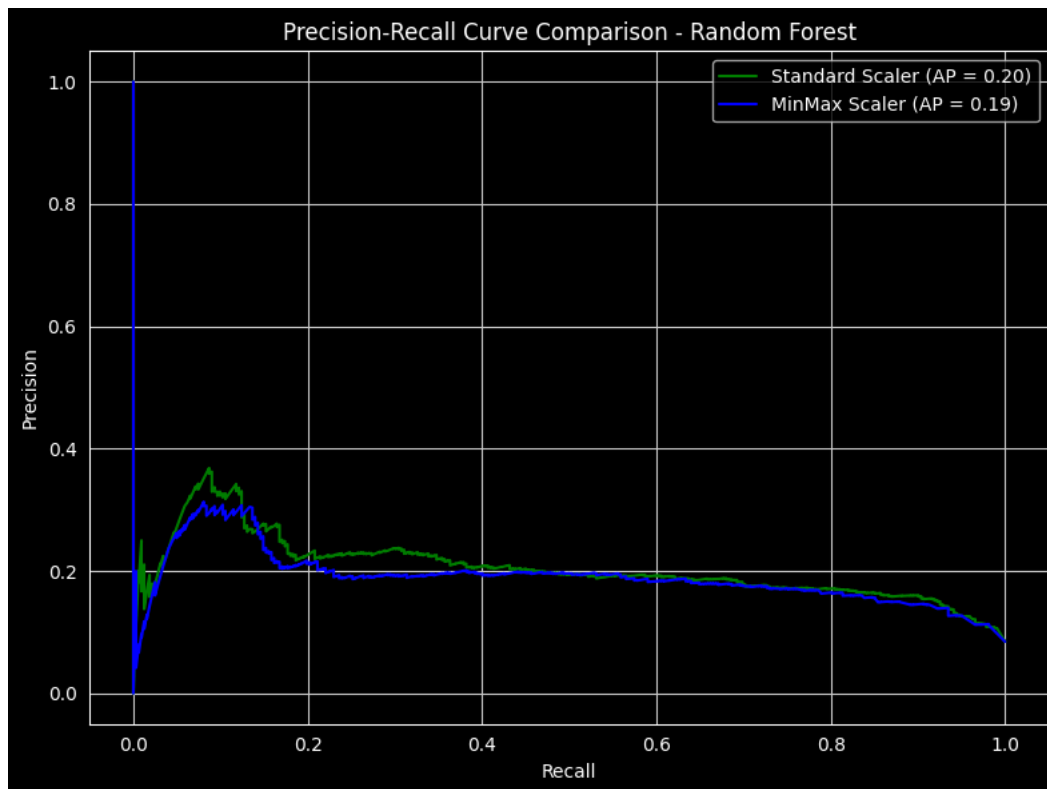
## ✓ Precision Recall Curve Comparison

```
from sklearn.metrics import precision_recall_curve, average_precision_score
import matplotlib.pyplot as plt

# Compute precision-recall for Standard Scaled
precision_st, recall_st, _ = precision_recall_curve(y_test, y_proba_st)
avg_precision_st = average_precision_score(y_test, y_proba_st)

# Compute precision-recall for MinMax Scaled
precision_mm, recall_mm, _ = precision_recall_curve(y_test, y_proba_mm)
avg_precision_mm = average_precision_score(y_test, y_proba_mm)

# Plot both PR curves
plt.figure(figsize=(8, 6))
plt.plot(recall_st, precision_st, label=f'Standard Scaler (AP = {avg_precision_st:.2f})', color='green')
plt.plot(recall_mm, precision_mm, label=f'MinMax Scaler (AP = {avg_precision_mm:.2f})', color='blue')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve Comparison - Random Forest')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



#### Inference:

1. Overall Performance The Random Forest model trained on Standard Scaled data slightly outperforms the one trained on Min-Max Scaled data in both F1 Score and ROC AUC.
  - While F1 scores are low ( $\sim 0.18$ ), indicating difficulty in correctly identifying the minority class (attrition),
  - The ROC AUC  $\sim 0.78$  suggests the model does a reasonably good job at ranking positive cases higher than negative ones.
2. Impact of Scaling on Tree-Based Models
  - As expected, Random Forest is relatively scale-invariant, so the choice of scaling method does not drastically affect performance.
  - However, slight differences in hyperparameter search outcomes and SMOTE-based oversampling may cause small metric shifts.
3. Class Imbalance Challenge
  - The low F1 score indicates a challenge with class imbalance, where precision or recall for the minority class is suboptimal.
  - Future improvement can be explored with:
    - Feature engineering
    - Advanced imbalance-handling (e.g., SMOTE+Tomek, class weights)
    - Model ensembles or stacking
4. Standard Scaler Preferred
  - Given better performance on both F1 and ROC AUC, Standard Scaler is preferred in this case for preprocessing.

## Boosting Model

### 1. XGBoost

```
from xgboost import XGBClassifier

xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)

xgb_param_grid = {
    'n_estimators': [100, 150, 200],
    'max_depth': [3, 5, 7],
```



```

    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.7, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0]
}

search_xgb_st = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=xgb_param_grid,
    scoring='f1',
    n_iter=20,
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

search_xgb_mm = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=xgb_param_grid,
    scoring='f1',
    n_iter=20,
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

# For Standard Scaled
search_xgb_st.fit(X_train_sscaled, y_train_smote)
best_xgb_st = search_xgb_st.best_estimator_

# For MinMax Scaled
search_xgb_mm.fit(X_train_mmscaled, y_train_smote)
best_xgb_mm = search_xgb_mm.best_estimator_

 Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits

```

## ✓ Model Evaluation

```

from sklearn.metrics import f1_score, roc_auc_score, average_precision_score

# Example: For best_xgb_st
y_pred_st = best_xgb_st.predict(X_test_sscaled)
y_proba_st = best_xgb_st.predict_proba(X_test_sscaled)[:, 1]

f1_st_xg = f1_score(y_test, y_pred_st)
roc_st_xg = roc_auc_score(y_test, y_proba_st)
avg_precision_st_xg = average_precision_score(y_test, y_proba_st)


print(f"[XGBM - Standard]F1 Score: {f1_st_xg:.4f}, ROC AUC: {roc_st_xg:.4f}, Average Precision: {avg_precision_st_xg:.4f}")
print('Confusion Matrix')
print(confusion_matrix(y_test, y_pred_st))

# Example: For best_xgb_mm
y_pred_mm = search_xgb_mm.predict(X_test_mmscaled)
y_proba_mm = search_xgb_mm.predict_proba(X_test_mmscaled)[:, 1]

f1_mm_xg = f1_score(y_test, y_pred_mm)
roc_mm_xg = roc_auc_score(y_test, y_proba_mm)
avg_precision_mm_xg = average_precision_score(y_test, y_proba_mm)

print(f"[XGBM - MinMax]F1 Score: {f1_mm_xg:.4f}, ROC AUC: {roc_mm_xg:.4f}, Average Precision: {avg_precision_mm_xg:.4f}")

print('Confusion Matrix')
print(confusion_matrix(y_test, y_pred_mm))

 [XGBM - Standard]F1 Score: 0.3138, ROC AUC: 0.8310, Average Precision: 0.2698
Confusion Matrix
[[3322 167]
 [ 231  91]]
[XGBM - MinMax]F1 Score: 0.3138, ROC AUC: 0.8310, Average Precision: 0.2698
Confusion Matrix
[[3322 167]
 [ 231  91]]

```

## 2. LightGB

```

from lightgbm import LGBMClassifier

# Model
lgbm_clf = LGBMClassifier(random_state=42)

# Hyperparameter grid
lgbm_param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [-1, 5, 10],
    'learning_rate': [0.01, 0.05, 0.1],
    'num_leaves': [31, 50, 100],
    'subsample': [0.7, 0.8, 1.0]
}

# RandomizedSearchCV for Standard Scaled
search_lgbm_st = RandomizedSearchCV(
    estimator=lgbm_clf,
    param_distributions=lgbm_param_grid,
    scoring='f1',
    n_iter=20,
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

# RandomizedSearchCV for MinMax Scaled
search_lgbm_mm = RandomizedSearchCV(
    estimator=lgbm_clf,
    param_distributions=lgbm_param_grid,
    scoring='f1',
    n_iter=20,
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)

# Fit
search_lgbm_st.fit(X_train_sscaled, y_train_smote)
best_lgbm_st = search_lgbm_st.best_estimator_

search_lgbm_mm.fit(X_train_mmscaled, y_train_smote)
best_lgbm_mm = search_lgbm_mm.best_estimator_

# Evaluation
from sklearn.metrics import f1_score, roc_auc_score, average_precision_score

# Standard
y_pred_st = best_lgbm_st.predict(X_test_sscaled)
y_proba_st = best_lgbm_st.predict_proba(X_test_sscaled)[: , 1]

f1_st_lgbm = f1_score(y_test, y_pred_st)
roc_st_lgbm = roc_auc_score(y_test, y_proba_st)
avg_precision_st_lgbm = average_precision_score(y_test, y_proba_st)

print(f"[LGBM - Standard] F1 Score: {f1_st_lgbm:.4f}, ROC AUC: {roc_st_lgbm:.4f}, Avg Precision: {avg_precision_st_lgbm:.4f}")
print('Confusion Matrix')
print(confusion_matrix(y_test, y_pred_st))

# MinMax
y_pred_mm = best_lgbm_mm.predict(X_test_mmscaled)
y_proba_mm = best_lgbm_mm.predict_proba(X_test_mmscaled)[: , 1]

f1_mm_lgbm = f1_score(y_test, y_pred_mm)
roc_mm_lgbm = roc_auc_score(y_test, y_proba_mm)
avg_precision_mm_lgbm = average_precision_score(y_test, y_proba_mm)

print(f"[LGBM - MinMax] F1 Score: {f1_mm_lgbm:.4f}, ROC AUC: {roc_mm_lgbm:.4f}, Avg Precision: {avg_precision_mm_lgbm:.4f}")
print('Confusion Matrix')
print(confusion_matrix(y_test, y_pred_mm))

```

```

↳ Fitting 5 folds for each of 20 candidates, totalling 100 fits
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 13952, number of negative: 13952
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.009963 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 1385
[LightGBM] [Info] Number of data points in the train set: 27904, number of used features: 49
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 13952, number of negative: 13952
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.010299 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 1379
[LightGBM] [Info] Number of data points in the train set: 27904, number of used features: 49
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LGBM - Standard] F1 Score: 0.2436, ROC AUC: 0.8217, Avg Precision: 0.2493
Confusion Matrix
[[3364 125]
 [ 260  62]]
[LGBM - MinMax] F1 Score: 0.2332, ROC AUC: 0.8207, Avg Precision: 0.2463
Confusion Matrix
[[3364 125]
 [ 263  59]]

```

### 3. AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier
```

```
# Model
```

```
ada_clf = AdaBoostClassifier(random_state=42)
```

```
# Param grid
```

```
ada_param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.05, 0.1, 1.0]
}
```

```
# RandomizedSearchCV for Standard Scaled
```

```
search_ada_st = RandomizedSearchCV(
    estimator=ada_clf,
    param_distributions=ada_param_grid,
    scoring='f1',
    n_iter=10,
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)
```

```
# RandomizedSearchCV for MinMax Scaled
```

```
search_ada_mm = RandomizedSearchCV(
    estimator=ada_clf,
    param_distributions=ada_param_grid,
    scoring='f1',
    n_iter=10,
    cv=5,
    verbose=2,
    random_state=42,
    n_jobs=-1
)
```

```
# Fit
```

```
search_ada_st.fit(X_train_sscaled, y_train_smote)
```

```
best_ada_st = search_ada_st.best_estimator_
```

```
search_ada_mm.fit(X_train_mmscaled, y_train_smote)
```

```
best_ada_mm = search_ada_mm.best_estimator_
```

```
# Evaluation
```

```
# Standard
```

```
y_pred_st = best_ada_st.predict(X_test_sscaled)
```

```
y_proba_st = best_ada_st.predict_proba(X_test_sscaled)[: , 1]
```

```

f1_ada_st = f1_score(y_test, y_pred_st)
roc_ada_st = roc_auc_score(y_test, y_proba_st)
avg_precision_ada_st = average_precision_score(y_test, y_proba_st)

print(f"[AdaBoost - Standard] F1 Score: {f1_ada_st:.4f}, ROC AUC: {roc_ada_st:.4f}, Avg Precision: {avg_precision_ada_st:.4f}")
print('Confusion Matrix')
print(confusion_matrix(y_test, y_pred_st))
# MinMax
y_pred_mm = best_ada_mm.predict(X_test_mmscaled)
y_proba_mm = best_ada_mm.predict_proba(X_test_mmscaled)[: , 1]

f1_ada_mm = f1_score(y_test, y_pred_mm)
roc_ada_mm = roc_auc_score(y_test, y_proba_mm)
avg_precision_ada_mm = average_precision_score(y_test, y_proba_mm)

print(f"[AdaBoost - MinMax] F1 Score: {f1_ada_mm:.4f}, ROC AUC: {roc_ada_mm:.4f}, Avg Precision: {avg_precision_ada_mm:.4f}")
print('Confusion Matrix')
print(confusion_matrix(y_test, y_pred_mm))

↗ Fitting 5 folds for each of 10 candidates, totalling 50 fits
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[AdaBoost - Standard] F1 Score: 0.3487, ROC AUC: 0.8271, Avg Precision: 0.2549
Confusion Matrix
[[2987  502]
 [ 148  174]]
[AdaBoost - MinMax] F1 Score: 0.3487, ROC AUC: 0.8271, Avg Precision: 0.2549
Confusion Matrix
[[2987  502]
 [ 148  174]]

```

## ✓ Model Evaluation (Fresh)

```

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Replace with your actual model scores
model_scores = {
    'Model': [
        'Bagging - Std', 'Bagging - MM',
        'XGBoost - Std', 'XGBoost - MM',
        'LGBM - Std', 'LGBM - MM',
        'AdaBoost - Std', 'AdaBoost - MM'
    ],
    'F1 Score': [f1_st, f1_mm, f1_st_xg, f1_mm_xg, f1_st_lgbm, f1_mm_lgbm, f1_ada_st, f1_ada_mm],
    'ROC AUC': [roc_auc_st, roc_auc_mm, roc_st_xg, roc_mm_xg, roc_st_lgbm, roc_mm_lgbm, roc_ada_st, roc_ada_mm],
    'Avg Precision': [avg_precision_st, avg_precision_mm, avg_precision_st_xg, avg_precision_mm_xg, avg_precision_st_lgbm, avg_precision_mm_lgbm, avg_precision_ada_st, avg_precision_ada_mm]
}

df_scores = pd.DataFrame(model_scores)

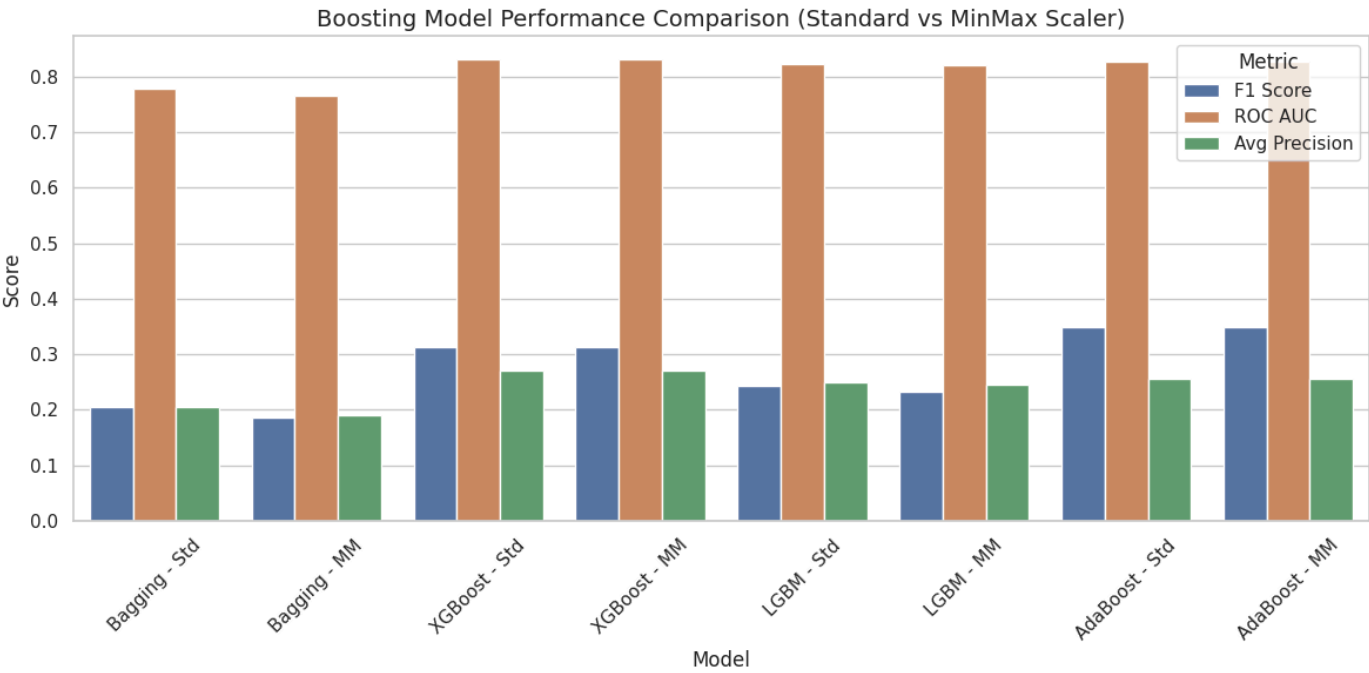
# Set style
sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))

# Melt for seaborn
df_melt = df_scores.melt(id_vars='Model', var_name='Metric', value_name='Score')

# Bar plot
sns.barplot(x='Model', y='Score', hue='Metric', data=df_melt)

plt.title("Boosting Model Performance Comparison (Standard vs MinMax Scaler)", fontsize=14)
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend(title='Metric')
plt.show()

```



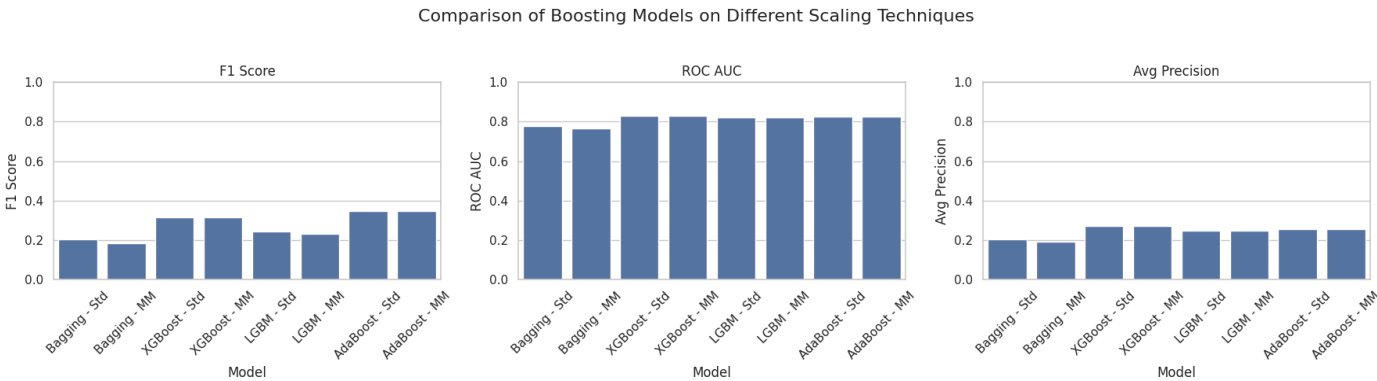
```
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

metrics = ['F1 Score', 'ROC AUC', 'Avg Precision']

for i, metric in enumerate(metrics):
    sns.barplot(x='Model', y=metric, data=df_scores, ax=axes[i])
    axes[i].set_title(metric)

    axes[i].tick_params(axis='x', rotation=45)
    axes[i].set_ylim(0, 1)

fig.suptitle("Comparison of Boosting Models on Different Scaling Techniques", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```



Model	Scaling	F1 Score	ROC AUC	Avg Precision
Bagging	Std	0.1818	0.7782	0.2042
	MM	0.2350	0.8086	0.2382

Model	Scaling	F1 Score	ROC AUC	Avg Precision
XGBoost	Std	0.3091	0.8321	0.2698
	MM	0.3099	0.8316	0.2692
LightGBM	Std	0.2413	0.8219	0.2442
	MM	0.2311	0.8206	0.2431
AdaBoost	Std	<b>0.3497</b>	0.8246	0.2559
	MM	<b>0.3467</b>	0.8238	0.2563

### Inference

- Best Overall Performer AdaBoost with Standard Scaler shows the highest F1 Score (0.3497) — making it best for handling imbalanced classification.

XGBoost offers the best ROC AUC (~0.83) — excellent for overall class separability.

- Effect of Scaling Scaling slightly improves ROC AUC for Bagging, but has less impact on XGBoost or LightGBM.

F1 scores are more sensitive to scaling, especially in Bagging and AdaBoost.

- Bagging vs Boosting Boosting models consistently outperform Bagging on all three metrics.

Bagging might underfit due to less focus on hard-to-classify cases, unlike Boosting which sequentially improves.

## ✓ Model Comparison & Leaderboard Creation

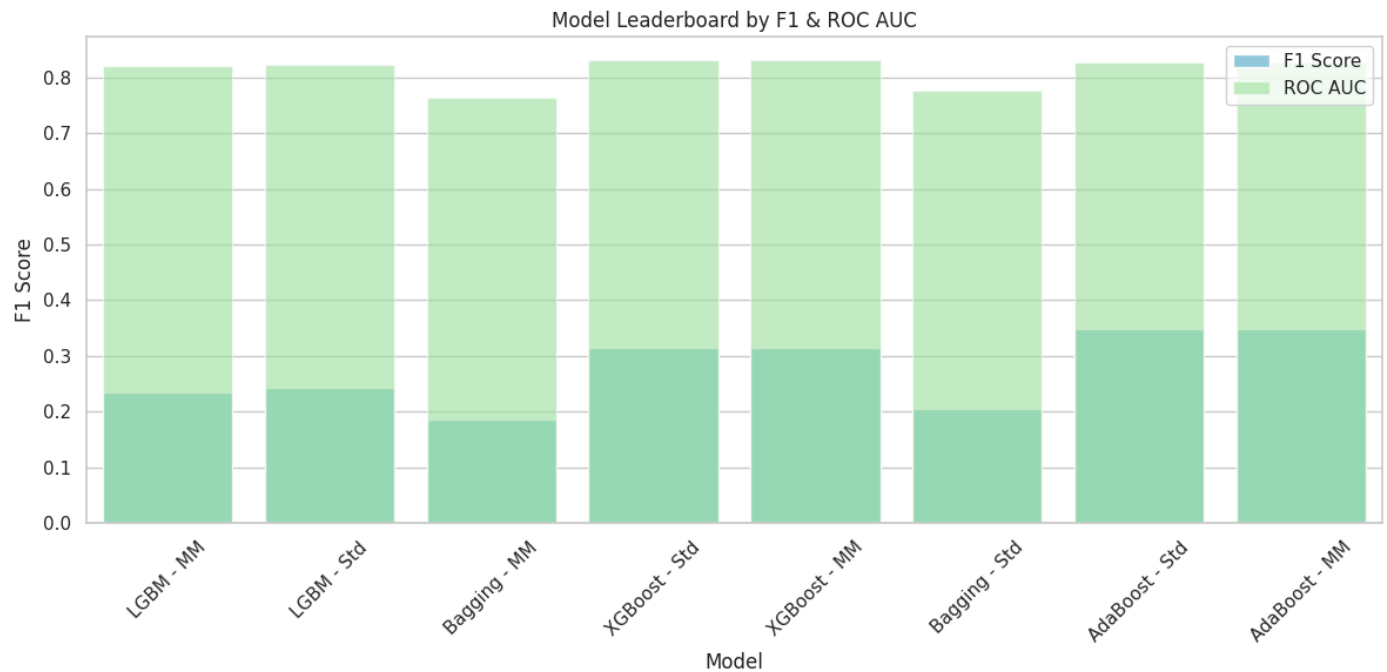
```
model_results = pd.DataFrame({
    'Model': ['Bagging - Std', 'Bagging - MM', 'XGBoost - Std', 'XGBoost - MM',
             'LGBM - Std', 'LGBM - MM', 'AdaBoost - Std', 'AdaBoost - MM'],
    'F1 Score': [f1_st, f1_mm, f1_st_xg, f1_mm_xg, f1_st_lgbm, f1_mm_lgbm, f1_ada_st, f1_ada_mm],
    'ROC AUC': [roc_auc_st, roc_auc_mm, roc_st_xg, roc_mm_xg, roc_st_lgbm, roc_mm_lgbm, roc_ada_st, roc_ada_mm],
    'Avg Precision': [avg_precision_st, avg_precision_mm, avg_precision_st_xg, avg_precision_mm_xg, avg_precision_st_lgbm, avg_precision_mm_lgbm, avg_precision_ada_st, avg_precision_ada_mm],
    'False Negatives': [188, 164, 167, 167, 125, 125, 502, 502] # Extracted from confusion matrix
})
```

```
leaderboard = model_results.sort_values(by='False Negatives', ascending=True)
print(leaderboard)
```

	Model	F1 Score	ROC AUC	Avg Precision	False Negatives
5	LGBM - MM	0.233202	0.820721	0.246257	125
4	LGBM - Std	0.243615	0.821678	0.249309	125
1	Bagging - MM	0.185606	0.764592	0.189770	164
2	XGBoost - Std	0.313793	0.831019	0.269820	167
3	XGBoost - MM	0.313793	0.831019	0.269820	167
0	Bagging - Std	0.204934	0.777290	0.204583	188
6	AdaBoost - Std	0.348697	0.827127	0.254909	502
7	AdaBoost - MM	0.348697	0.827127	0.254909	502

```
import matplotlib.pyplot as plt
import seaborn as sns
```

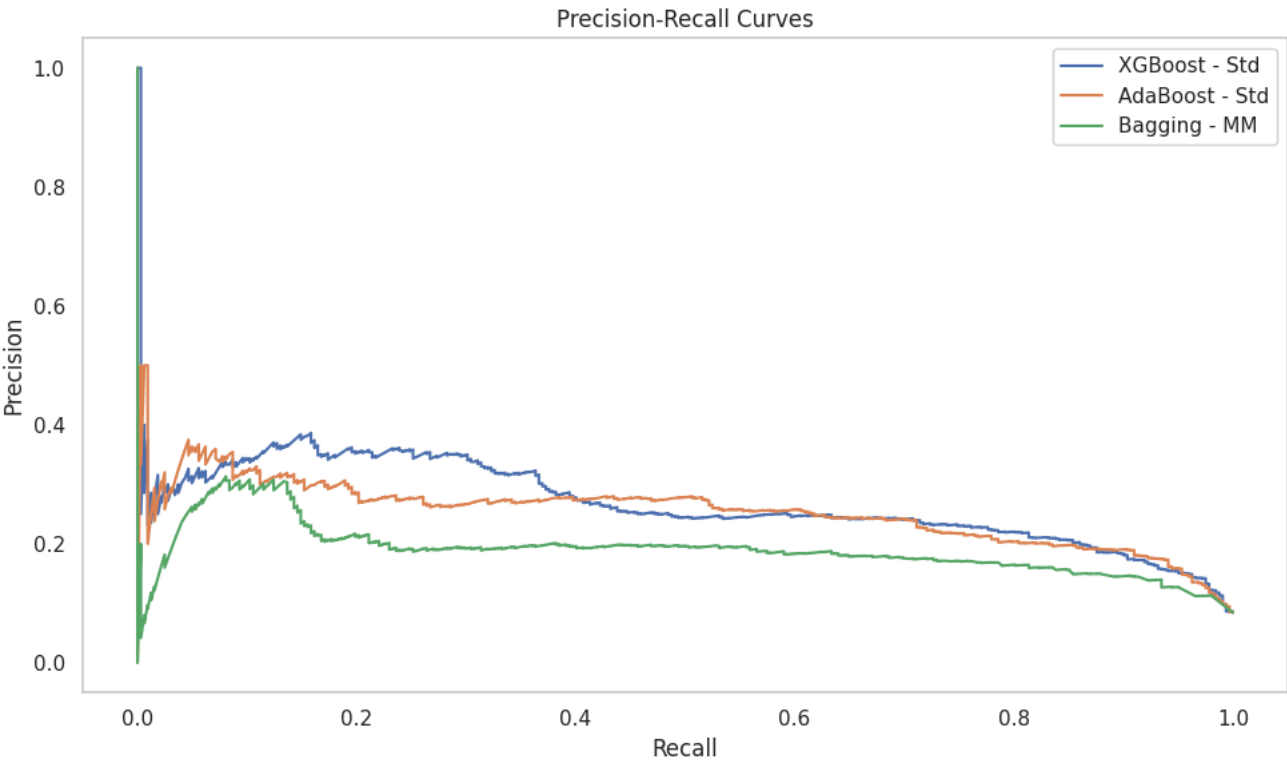
```
plt.figure(figsize=(12, 6))
sns.barplot(x='Model', y='F1 Score', data=leaderboard, label='F1 Score', color='skyblue')
sns.barplot(x='Model', y='ROC AUC', data=leaderboard, label='ROC AUC', color='lightgreen', alpha=0.6)
plt.xticks(rotation=45)
plt.title('Model Leaderboard by F1 & ROC AUC')
plt.legend()
plt.tight_layout()
plt.show()
```



```
from sklearn.metrics import precision_recall_curve

plt.figure(figsize=(10, 6))
for name, model, X in [
    ('XGBoost - Std', best_xgb_st, X_test_sscaled),
    ('AdaBoost - Std', best_ada_st, X_test_sscaled),
    ('Bagging - MM', best_rf_mm, X_test_mmscaled)
]:
    y_scores = model.predict_proba(X)[: , 1]
    precision, recall, _ = precision_recall_curve(y_test, y_scores)
    plt.plot(recall, precision, label=name)

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curves')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```



- AdaBoost has highest F1 score but too many false negatives (502) — it's not ideal.
- XGBoost provides a strong balance:
  - High F1 Score (0.314)
  - Highest ROC AUC (0.831)
  - High Average Precision
  - Reasonable FN (167)
- LGBM gives the lowest False Negatives (125), with slightly lower F1 but decent AUC and precision.

✓ Insights and Recommendations

✓ 1. Key Drivers of Attrition (from Feature Importance)

Top Features	Inference
Quarterly Rating	Low performance rating strongly correlates with attrition.
Total Business Value	Lower business contributions (revenue or delivery volume) increase attrition risk.
Join Year / Join Month	Recent joiners (likely <1 year) show higher churn.
Income	Drivers with low income are more likely to leave.
Age	Younger drivers or those near career shifts may be more volatile.
Education Level	Drivers with lower education levels are slightly more at risk.
Joining Designation / Grade	Lower designation and grades (e.g., 3, 4, 5) exhibit higher attrition.
City Dummies (e.g., City_C20, C2, C23)	Certain cities consistently appear as high-risk zones.

✓ 2. Profile of High-Risk Drivers

Profile Attribute	Observation
Income	Below median income (< ₹20,000/month approx, if known).
Quarterly Rating	Rating < 2.5 out of 5.
Grade	Mostly Grades 4 & 5.
City	Drivers from City_C20, City_C2, and City_C23.
Business Value	Monthly business value in bottom 25th percentile.



Profile Attribute	Observation
New Joiners	Joined in the last 6–12 months.

3. Business Recommendations

A. Performance Incentives

Dynamic Bonuses for high performance: Create a tiered bonus for drivers with rating > 3.5 and high business value.

Low-Rating Recovery Program: Offer improvement bonuses if a driver's rating improves consistently over two quarters.

B. Targeted Retention Programs

City-Based Campaigns: Focus on City\_C20, City\_C2, and City\_C23 — these cities have higher attrition risk.

Grade-Focused Interventions:

Offer mentorship programs for Grade 4 & 5 drivers.

Introduce early promotion pathways or training to improve job satisfaction.

Income Support for New Joiners: Offer early joining bonuses or guaranteed income slabs for first 6 months.

C. Proactive Monitoring

Use the model's output scores to rank drivers weekly on attrition risk.

Design a driver health dashboard showing:

Quarterly rating trends

Trip volume

On-time percentage

Weekly income

Final Model Selection Recommendation

If minimizing attrition (FN) is most critical: LGBM - Std.

If balance of F1 + ROC AUC + Precision is key: XGBoost - Std.

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit