Problem Definition

Driver retention is a significant operational challenge for Ola, primarily due to fare competition and more attractive incentives offered by competitors like Uber. High attrition rates result in elevated acquisition costs, disrupt business continuity, and negatively affect driver morale. As Ola continues to scale, reducing driver churn is crucial for sustaining a stable, cost-effective workforce and ensuring long-term operational efficiency.

Project Goals

- **Exploratory Data Analysis (EDA)**: Analyze driver demographics, income patterns, tenure, and churn behavior. Detect missing values and outliers.
- **Feature Engineering**: Create derived features such as income and rating trends; define the churn target variable.
- **Data Preparation**: Handle missing values using KNN imputation, apply one-hot encoding to categorical variables, and scale numerical features.
- **Class Imbalance Handling**: Address churn class imbalance using techniques like SMOTE or undersampling.
- **Model Building**: Train and tune ensemble classifiers including Random Forest, XGBoost, LightGBM, and CatBoost.
- **Model Evaluation**: Evaluate model performance using classification reports, ROC-AUC scores, and feature importance analysis.
- **Business Insights**: Identify key churn drivers and recommend actionable retention strategies.

Dataset Description

The dataset contains monthly records of a segment of Ola drivers from 2019 and 2020. It includes attributes related to driver demographics, tenure, performance, and income, which are useful for predicting attrition.

Column Name	Description
`MMMM-YY`	Reporting date (monthly period for the data record)
`Driver_ID`	Unique identifier for each driver
`Age`	Age of the driver
`Gender`	Gender of the driver (`0` : Male, `1` : Female)
`City`	City code indicating the driver's operating location
`Education_Level`	Education level (`0` : 10th grade+, `1` : 12th grade+, `2` : Graduate)
`Income`	Driver's average monthly income
`Date Of Joining`	Date the driver joined Ola
`LastWorkingDate`	Date the driver last worked for Ola (if applicable)
`Joining Designation`	The designation or role of the driver at the time of joining
`Grade`	Driver's grade at the time of reporting
`Total Business Value`	Business value generated by the driver that month (negative values may indicate cancellations or EMI adjustments)
`Quarterly Rating`	Driver's quarterly performance rating (scale of 1 to 5; higher is better)

Data Understanding

Imports

```
import warnings
warnings.filterwarnings('ignore')

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

from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, cross_val_score
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import (RandomForestClassifier,AdaBoostClassifier, StackingClassifier)
from sklearn.inspection import permutation_importance
from sklearn.metrics import precision_recall_curve, average_precision_score

from imblearn.over_sampling import SMOTE

from xgboost import XGBClassifier
```

```
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
```

Initial Summary Stats

```
df = pd.read_csv('ola_driver.csv')
df.head(10)
```

	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	
5	5	12/01/19	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	
6	6	01/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	
7	7	02/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	
8	8	03/01/20	4	43.0	0.0	C13	2	65603	12/07/19	NaN	2	2	3

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
print("\nDescriptive statistics (numeric columns):")
print(df.describe())

print("\nDataFrame info:")
df.info()

missing_cols = df.columns[df.isnull().any()].tolist()
print("\nColumns with missing values:")
print(missing_cols)

print("\nNumber of unique values per column:")
print(df.nunique())
```

```
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

DataFrame info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
```

```
Columns with missing values:
['Age', 'Gender', 'LastWorkingDate']

Number of unique values per column:
Unnamed: 0             19104
MMM-YY                    24
Driver_ID               2381
Age                       36
Gender                     2
City                      29
Education_Level            3
Income                  2383
Dateofjoining            869
LastWorkingDate          493
Joining Designation        5
Grade                      5
Total Business Value   10181
Quarterly Rating           4
dtype: int64
```

Initial Interpretation

- **Drop Redundant Columns**
  A redundant column exists in the dataset and should be removed.
- **Data Type Conversion**
  - A date column in "MMM-YY" format should be renamed and converted for consistency.
  - Joining and exit dates should be converted to proper datetime format.
  - Age and gender fields need to be cast to appropriate integer types.
- **Handle Missing Values**
  Some key fields contain missing values and must be imputed before proceeding with analysis or modeling.
- **Aggregate Records at Individual Level**
  The dataset includes 2381 unique individuals, with multiple records for some.
  Consolidating data at the individual level is necessary to ensure accurate summaries and reduce duplication.
- **Categorical Encoding**
  One or more categorical fields need to be encoded for compatibility with machine learning algorithms.
- **Create Target Variable**
  The target variable is not explicitly present.
  It should be derived by checking whether an exit date is present (indicating attrition) or not.

Data Cleaning & Preprocessing

```
# Dropping redundant column
df.drop(df.columns[0], inplace=True, axis=1, errors='ignore')

# Renaming Date columns
df.rename(columns={
    'MMM-YY': 'Reporting_Date',
    'Dateofjoining': 'Joining_Date',
    'LastWorkingDate': 'Last_Working_Date'
}, inplace=True)

# Converting Date columns to Date datatype
for col in ['Reporting_Date', 'Joining_Date', 'Last_Working_Date']:
    if col in df.columns:
        df[col] = pd.to_datetime(df[col], errors='coerce')

# Performing KNN imputation to handle missing data in Age
age_imputer = KNNImputer(n_neighbors=5, weights='distance')
df[['Age']] = age_imputer.fit_transform(df[['Age']])

# Performing KNN imputation to handle missing data in Gender
gender_imputer = KNNImputer(n_neighbors=5, weights='uniform')
df[['Gender']] = gender_imputer.fit_transform(df[['Gender']])

# Converting Age and Gender Columns to int Datatype
df[['Age', 'Gender']] = df[['Age', 'Gender']].astype(int)

# Aggregate Data to Driver Level
agg_df = df.groupby('Driver_ID').agg({
    'Reporting_Date': 'max',
    'Age': 'max',
    'Gender': lambda x: x.mode().iloc[0] if not x.mode().empty else x.iloc[0],
```

```
    'City': 'first',
    'Education_Level': 'max',
    'Income': ['max', 'sum'],
    'Joining_Date': 'first',
    'Last_Working_Date': 'max',
    'Joining Designation': 'first',
    'Grade': 'max',
    'Total Business Value': ['mean', 'sum'],
    'Quarterly Rating': 'max'
}).reset_index()

agg_df.columns = [
    f"{col[0]}_{col[1]}" if col[0] in ['Total Business Value', 'Income'] else col[0]
    for col in agg_df.columns.values
]

agg_df.rename(columns={
    'Total Business Value_mean': 'Avg_Business_Value',
    'Total Business Value_sum': 'Total_Business_Value',
    'Income_sum': 'Total_Income',
    'Income_max': 'Income'
}, inplace=True)

# Target Variable - Attrition
agg_df = agg_df.merge(
    df.groupby('Driver_ID')['Last_Working_Date']
      .apply(lambda x: int(x.notnull().any()))
      .reset_index(name='Attrition'),
    on='Driver_ID', how='left'
)
```

Dropping Redundant Columns

- Removed the first unnamed column that appeared as Unnamed: 0 or Unknown:0 in the dataset.

Date Conversion

- Renamed MMM-YY to Reporting_Date and converted it to datetime format for accurate date-based calculations.
- Converted Dateofjoining and LastWorkingDate to datetime format to enable precise tenure and attrition tracking.

Missing Value Imputation (KNN)

KNN imputer applied with variable-specific weighting:

Variable	Weights Used	Rationale
Age	Distance	Age is continuous; closer points yield better similarity for imputation.
Gender	Uniform	Gender is binary; uniform weights prevent bias from numeric distance.

Aggregation to Driver Level

Multiple monthly records per driver were aggregated to a single row using appropriate summary statistics:

Feature	Aggregation Method	Rationale
Reporting_Date	max	To capture the most recent information.
Age	max	Age increases or remains constant over time.
Gender	mode	Mode reflects the most frequent gender value for the driver.
City	first	Assumed static; retained the first record.
Education_Level	max	To capture the latest or highest education level achieved.
Income	max	Highest monthly income during driver tenure.
Total_Income	sum	Cumulative income over all months.
Joining_Date	first	Earliest date assumed as joining date.
Last_Working_Date	max	Latest date indicates exit if present.
Joining Designation	first	Initial designation assumed unchanged.
Grade	max	Highest grade attained.
Avg_Business_Value	mean	Average monthly performance.
Total_Business_Value	sum	Total business value, including negative adjustments.
Quarterly Rating	max	Retain the best quarterly rating.
Attrition	Derived post-aggregation	1 if `Last_Working_Date` exists (driver left), else 0.

Target Variable Creation

- Created binary Attrition flag post-aggregation: 1 indicates driver has left, 0 otherwise.

Categorical Encoding

- City encoding will be done later after EDA before creating models.

```
agg_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Driver_ID             2381 non-null   int64
 1   Reporting_Date        2381 non-null   datetime64[ns]
 2   Age                   2381 non-null   int64
 3   Gender                2381 non-null   int64
 4   City                  2381 non-null   object
 5   Education_Level       2381 non-null   int64
 6   Income                2381 non-null   int64
 7   Total_Income          2381 non-null   int64
 8   Joining_Date          2381 non-null   datetime64[ns]
 9   Last_Working_Date     1616 non-null   datetime64[ns]
 10  Joining Designation   2381 non-null   int64
 11  Grade                 2381 non-null   int64
 12  Avg_Business_Value    2381 non-null   float64
 13  Total_Business_Value  2381 non-null   int64
 14  Quarterly Rating      2381 non-null   int64
 15  Attrition             2381 non-null   int64
dtypes: datetime64[ns](3), float64(1), int64(11), object(1)
memory usage: 297.8+ KB
```

```
agg_df.head(5)
```

	Driver_ID	Reporting_Date	Age	Gender	City	Education_Level	Income	Total_Income	Joining_Date	Last_Working_Date	Joining Designation
0	1	2019-03-01	28	0	C23	2	57387	172161	2018-12-24	2019-03-11	1
1	2	2020-12-01	31	0	C7	2	67016	134032	2020-11-06	NaT	2
2	4	2020-04-01	43	0	C13	2	65603	328015	2019-12-07	2020-04-27	2
3	5	2019-03-01	29	0	C9	0	46368	139104	2019-01-09	2019-03-07	1
4	6	2020-12-01	31	1	C11	1	78728	393640	2020-07-31	NaT	3

Next steps: Generate code with agg_df View recommended plots New interactive sheet

```
agg_df[['Reporting_Date', 'Joining_Date', 'Last_Working_Date']].agg(['min', 'max'])
```

	Reporting_Date	Joining_Date	Last_Working_Date
min	2019-01-01	2013-04-01	2018-12-31
max	2020-12-01	2020-12-28	2020-12-28

Post-Processing Data Summary

The post-processed dataset contains 2,381 driver-level records with 16 columns: 3 datetime, 11 integer, 1 float, and 1 object type. Most columns have complete data, except for Last_Working_Date, which has some missing values.

Reporting dates range from January 2019 to December 2020, covering the entire analysis period.

The earliest recorded attrition date is December 2018, while the latest last working date is December 2020, indicating attrition started prior to the analysis window.

The last joining date and last working date recorded is December 28, 2020. We assume that drivers active until December 28, 2020, remain with the company.

Data Preparation

Feature Engineering

```
data_end_date = pd.to_datetime('2020-12-29')
agg_df['Tenure'] = (agg_df['Last_Working_Date'].fillna(data_end_date) - agg_df['Joining_Date']).dt.days

agg_df['Income_per_Tenure'] = agg_df['Income'] / (agg_df['Tenure'] + 1)
agg_df['Grade_Income_Interaction'] = agg_df['Grade'] * agg_df['Income']
agg_df['BusinessValue_per_Grade'] = agg_df['Total_Business_Value'] / (agg_df['Grade'] + 1)

agg_df['Reporting_Quarter'] = agg_df['Reporting_Date'].dt.quarter
```

```
agg_df['Joining_Month'] = agg_df['Joining_Date'].dt.month
agg_df['Joining_Quarter'] = agg_df['Joining_Date'].dt.quarter

agg_df['Had_Business'] = (agg_df['Total_Business_Value'] > 0).astype(int)

agg_increase = (
    df.sort_values(['Driver_ID', 'Reporting_Date'])
      .groupby('Driver_ID')
      .agg(
          QuarterlyRating_Increased=('Quarterly Rating', lambda x: int(x.iloc[-1] > x.iloc[0])),
          Income_Increased=('Income', lambda x: int(x.iloc[-1] > x.iloc[0]))
      )
      .reset_index()
)

agg_df = agg_df.merge(agg_increase, on='Driver_ID', how='left')
```

The table below outlines the key engineered features designed to improve model accuracy and interpretability. These features capture driver tenure, income efficiency, interaction effects, temporal patterns, and performance trends.

Feature	Description & Rationale
`Tenure`	Number of days between joining date and last working date (or 2020-12-29), representing actual experience duration.
`Income_per_Tenure`	Income divided by tenure (plus 1 to avoid division by zero) to measure income efficiency over time.
`Grade_Income_Interaction`	Product of grade and income capturing combined influence on driver performance or business value.
`BusinessValue_per_Grade`	Business value normalized by grade (plus 1 to prevent division by zero), allowing comparison across grades.
`Reporting_Quarter`	Quarter extracted from reporting date to incorporate seasonality in analysis.
`Joining_Month`	Month extracted from joining date to analyze cohort and seasonal effects.
`Joining_Quarter`	Quarter extracted from joining date to assess cohort grouping and trends.
`Had_Business`	Binary flag indicating whether the driver generated any business value, aiding classification tasks.
`QuarterlyRating_Increased`	Driver-level flag indicating if quarterly rating improved at any point, useful for performance trend analysis.
`Income_Increased`	Driver-level flag indicating if income increased at any point, helping identify positive earnings trends.

```
agg_df.head()
agg_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 26 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Driver_ID                  2381 non-null   int64
 1   Reporting_Date             2381 non-null   datetime64[ns]
 2   Age                        2381 non-null   int64
 3   Gender                     2381 non-null   int64
 4   City                       2381 non-null   object
 5   Education_Level            2381 non-null   int64
 6   Income                     2381 non-null   int64
 7   Total_Income               2381 non-null   int64
 8   Joining_Date               2381 non-null   datetime64[ns]
 9   Last_Working_Date          1616 non-null   datetime64[ns]
 10  Joining Designation        2381 non-null   int64
 11  Grade                      2381 non-null   int64
 12  Avg_Business_Value         2381 non-null   float64
 13  Total_Business_Value       2381 non-null   int64
 14  Quarterly Rating           2381 non-null   int64
 15  Attrition                  2381 non-null   int64
 16  Tenure                     2381 non-null   int64
 17  Income_per_Tenure          2381 non-null   float64
 18  Grade_Income_Interaction   2381 non-null   int64
 19  BusinessValue_per_Grade    2381 non-null   float64
 20  Reporting_Quarter          2381 non-null   int32
 21  Joining_Month              2381 non-null   int32
 22  Joining_Quarter            2381 non-null   int32
 23  Had_Business               2381 non-null   int64
 24  QuarterlyRating_Increased  2381 non-null   int64
 25  Income_Increased           2381 non-null   int64
dtypes: datetime64[ns](3), float64(3), int32(3), int64(16), object(1)
memory usage: 455.9+ KB
```

- After adding interaction terms, the dataset contains **2,381 rows** and **26 columns**.
- All columns are **non-null** except Last_Working_Date, which has missing values.
- Missing values in Last_Working_Date correspond to drivers who **have not left the organization** yet.
- Data types are properly corrected:
    - Reporting_Date, Joining_Date, and Last_Working_Date Inline code as **datetime** types
    - Other columns as **integers** or **floats** as appropriate
- The dataset is clean and ready for further analysis.

```
agg_df.describe().T
```

	count	mean	min	25%	50%	75%	max	st
Driver_ID	2381.0	1397.559009	1.0	695.0	1400.0	2100.0	2788.0	806.16162
Reporting_Date	2381	2020-03-31 15:04:09.475010560	2019-01-01 00:00:00	2019-09-01 00:00:00	2020-06-01 00:00:00	2020-12-01 00:00:00	2020-12-01 00:00:00	Nal
Age	2381.0	33.790004	21.0	30.0	33.0	37.0	58.0	5.907
Gender	2381.0	0.409492	0.0	0.0	0.0	1.0	1.0	0.49184
Education_Level	2381.0	1.00756	0.0	0.0	1.0	2.0	2.0	0.8162
Income	2381.0	59336.159597	10747.0	39104.0	55315.0	75986.0	188418.0	28383.01214
Total_Income	2381.0	526760.305754	10883.0	139895.0	292980.0	651456.0	4522032.0	623163.27837
Joining_Date	2381	2019-02-08 07:14:50.550189056	2013-04-01 00:00:00	2018-06-29 00:00:00	2019-07-21 00:00:00	2020-05-02 00:00:00	2020-12-28 00:00:00	Nal
Last_Working_Date	1616	2019-12-21 20:59:06.534653440	2018-12-31 00:00:00	2019-06-06 00:00:00	2019-12-20 12:00:00	2020-07-03 00:00:00	2020-12-28 00:00:00	Nal
Joining Designation	2381.0	1.820244	1.0	1.0	2.0	2.0	5.0	0.84143
Grade	2381.0	2.097018	1.0	1.0	2.0	3.0	5.0	0.94170
Avg_Business_Value	2381.0	312085.359327	-197932.857143	0.0	150624.444444	429498.75	3972127.5	449570.50671
Total_Business_Value	2381.0	4586741.822764	-1385530.0	0.0	817680.0	4173650.0	95331060.0	9127115.31344
Quarterly Rating	2381.0	1.929861	1.0	1.0	1.0	3.0	4.0	1.10485
Attrition	2381.0	0.678706	0.0	0.0	1.0	1.0	1.0	0.46707
Tenure	2381.0	436.455271	0.0	99.0	192.0	480.0	2829.0	567.46067
Income_per_Tenure	2381.0	581.241469	9.745464	94.247059	242.777202	554.337349	56498.0	2088.33887
Grade_Income_Interaction	2381.0	144232.056699	10883.0	47594.0	109942.0	197262.0	942090.0	127113.86968
BusinessValue_per_Grade	2381.0	1334369.179056	-692765.0	0.0	273380.0	1391216.666667	15888510.0	2297608.75394
Reporting_Quarter	2381.0	2.946661	1.0	2.0	3.0	4.0	4.0	1.17577
Joining_Month	2381.0	7.357413	1.0	5.0	7.0	10.0	12.0	3.14314
Joining_Quarter	2381.0	2.801764	1.0	2.0	3.0	4.0	4.0	1.01178
Had_Business	2381.0	0.693826	0.0	0.0	1.0	1.0	1.0	0.46

Interpretation

Basic Stats

- Total records: 2,381 drivers
- Churn Rate (Attrition): Approximately 68% churned, indicating high imbalance that may require handling

Driver Demographics

- Age: Average around 33.8 years, range from 21 to 58
- Gender: Approximately 41% Female and 59% Male
- Education: Mostly encoded as 1 (12+ or Graduate); very few postgraduates

Employment and Ratings

- Joining Designation: Predominantly level 1 or 2, indicating junior-level hires
- Grade: Average grade is 2.1, skewed toward lower grades
- Tenure: Average tenure is 1.2 years (436 days), with some up to 7.7 years
- Minimum tenure of zero indicates some drivers joined and left on the same day, representing immediate churn
- Quarterly Rating: Mean rating is 1.9, with most rated at 1. Maxumum quaterly rating received by any driver is not greater than 4
- Rating Increased: Only about 15% showed improvement
- Income Increased: Only 1.8% of drivers had an increase in salary

Income and Business Metrics

- Monthly Income: Average is ₹59,000 with a large range (₹15,000 to ₹1.88 lakh)

- Total Income: Mean total income is around ₹5.26 lakh; includes some very high earners
- Total Business Value: Mean value around ₹45 lakh; includes negative values
- The total business value generated by a driver in a month can be negative, reflecting cancellations, refunds, or car EMI adjustments
- Average Business Value: High variability indicating possible outliers or top performers
- Business Assigned: Around 69% of drivers made positive business

Dates and Quarters

- Joining Dates: Range from 2013 to 2020
- Last Working Dates: Available for about 68% of drivers (those who churned)
- Most drivers joined in Q2–Q3, with reporting concentrated in Q4 2020

```
agg_df[agg_df['Tenure']==0].T
```

	220	1026	1344	2041
Driver_ID	264	1207	1581	2397
Reporting_Date	2020-12-01 00:00:00	2020-04-01 00:00:00	2019-07-01 00:00:00	2020-05-01 00:00:00
Age	25	28	29	38
Gender	0	0	0	1
City	C11	C24	C15	C8
Education_Level	2	0	0	0
Income	49439	56498	25873	47818
Total_Income	49439	56498	25873	47818
Joining_Date	2020-12-18 00:00:00	2020-04-12 00:00:00	2019-06-30 00:00:00	2020-05-15 00:00:00
Last_Working_Date	2020-12-18 00:00:00	2020-04-12 00:00:00	2019-06-30 00:00:00	2020-05-15 00:00:00
Joining Designation	1	2	1	2
Grade	1	2	1	2
Avg_Business_Value	0.0	0.0	0.0	0.0
Total_Business_Value	0	0	0	0
Quarterly Rating	1	1	1	1
Attrition	1	1	1	1
Tenure	0	0	0	0
Income_per_Tenure	49439.0	56498.0	25873.0	47818.0
Grade_Income_Interaction	49439	112996	25873	95636
BusinessValue_per_Grade	0.0	0.0	0.0	0.0
Reporting_Quarter	4	2	3	2
Joining_Month	12	4	6	5
Joining_Quarter	4	2	2	2
Had_Business	0	0	0	0
QuarterlyRating_Increased	0	0	0	0
Income_Increased	0	0	0	0

There are 4 drivers who exited on their very first day of joining, indicating immediate churn. This points to potential issues with onboarding, job fit, or the initial driver experience. Identifying these cases is crucial, as they can disproportionately affect tenure analyses and highlight key areas for operational improvement.

Exploratory Data Analysis

```
attrition_palette = {'0': 'green', '1': 'red'}
attrition_palette1 = {0: 'green', 1: 'red'}
```

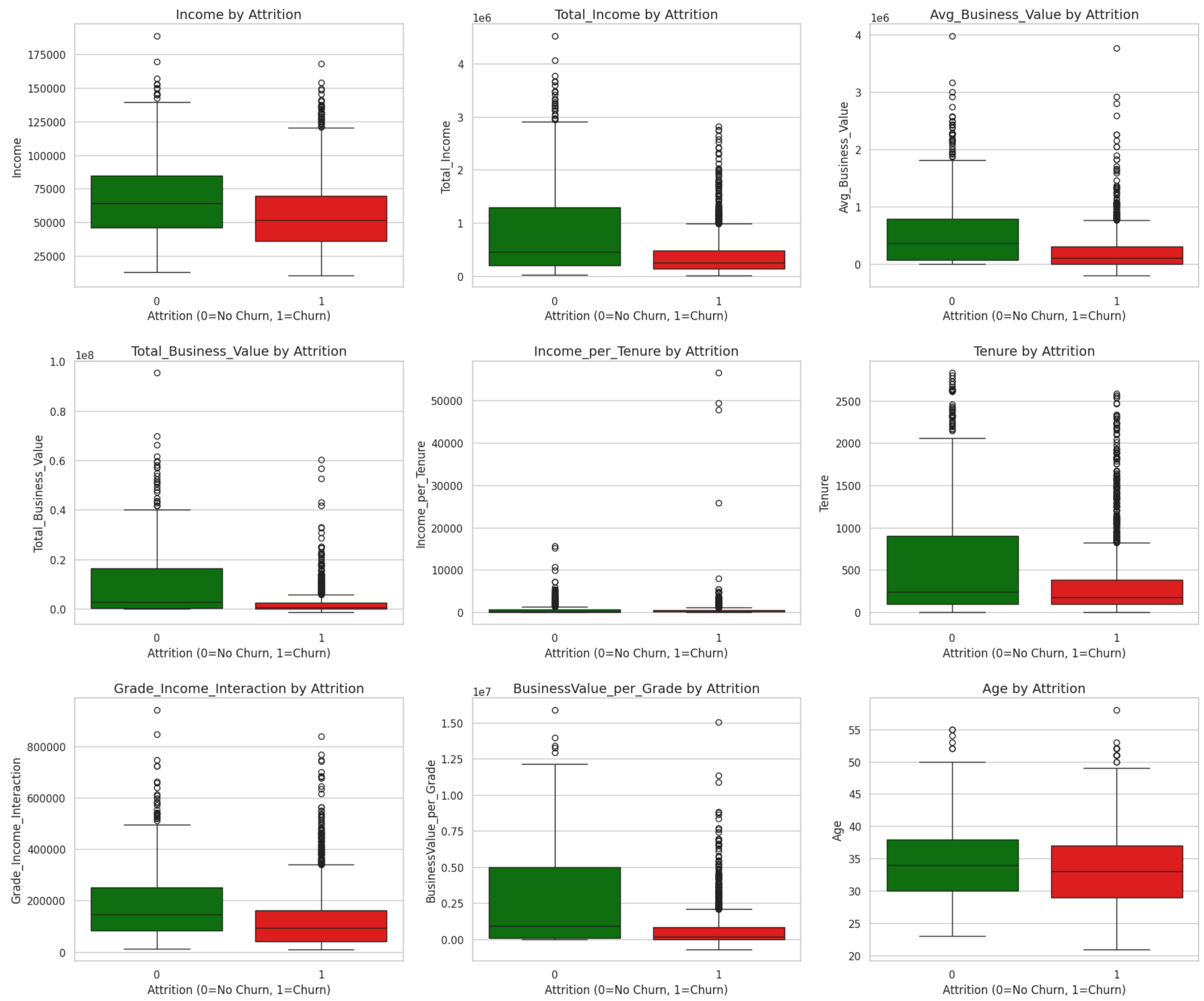
```
prop = agg_df['Attrition'].value_counts(normalize=True).sort_index()

fig, ax = plt.subplots(figsize=(6, 2))
ax.barh(0, prop[0], color=attrition_palette1[0], edgecolor='black')
ax.barh(0, prop[1], left=prop[0], color=attrition_palette1[1], edgecolor='black')
```

```
ax.text(prop[0] / 2, 0, f'{prop[0]*100:.1f}%', va='center', ha='center', color='white', fontweight='bold', fontsize=14)
ax.text(prop[0] + prop[1] / 2, 0, f'{prop[1]*100:.1f}%', va='center', ha='center', color='white', fontweight='bold', fontsize=14)

ax.set(yticks=[], xticks=[])
for spine in ax.spines.values():
    spine.set_visible(False)

plt.show()
```

The dataset is imbalanced, with approximately 68% of drivers having churned and 32% remaining active. We will use the color palette red to represent churn and green to represent non-churn drivers throughout the EDA.

Outlier Detection

```
cols_to_plot = [
    'Income', 'Total_Income', 'Avg_Business_Value', 'Total_Business_Value',
    'Income_per_Tenure', 'Tenure', 'Grade_Income_Interaction', 'BusinessValue_per_Grade', 'Age'
]

sns.set(style='whitegrid')

fig, axes = plt.subplots(3, 3, figsize=(18, 15))
axes = axes.flatten()

for i, col in enumerate(cols_to_plot):
    sns.boxplot(x='Attrition', y=col, data=agg_df, palette=attrition_palette, ax=axes[i])
    axes[i].set_title(f'{col} by Attrition', fontsize=14)
    axes[i].set_xlabel('Attrition (0=No Churn, 1=Churn)')
    axes[i].set_ylabel(col)

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



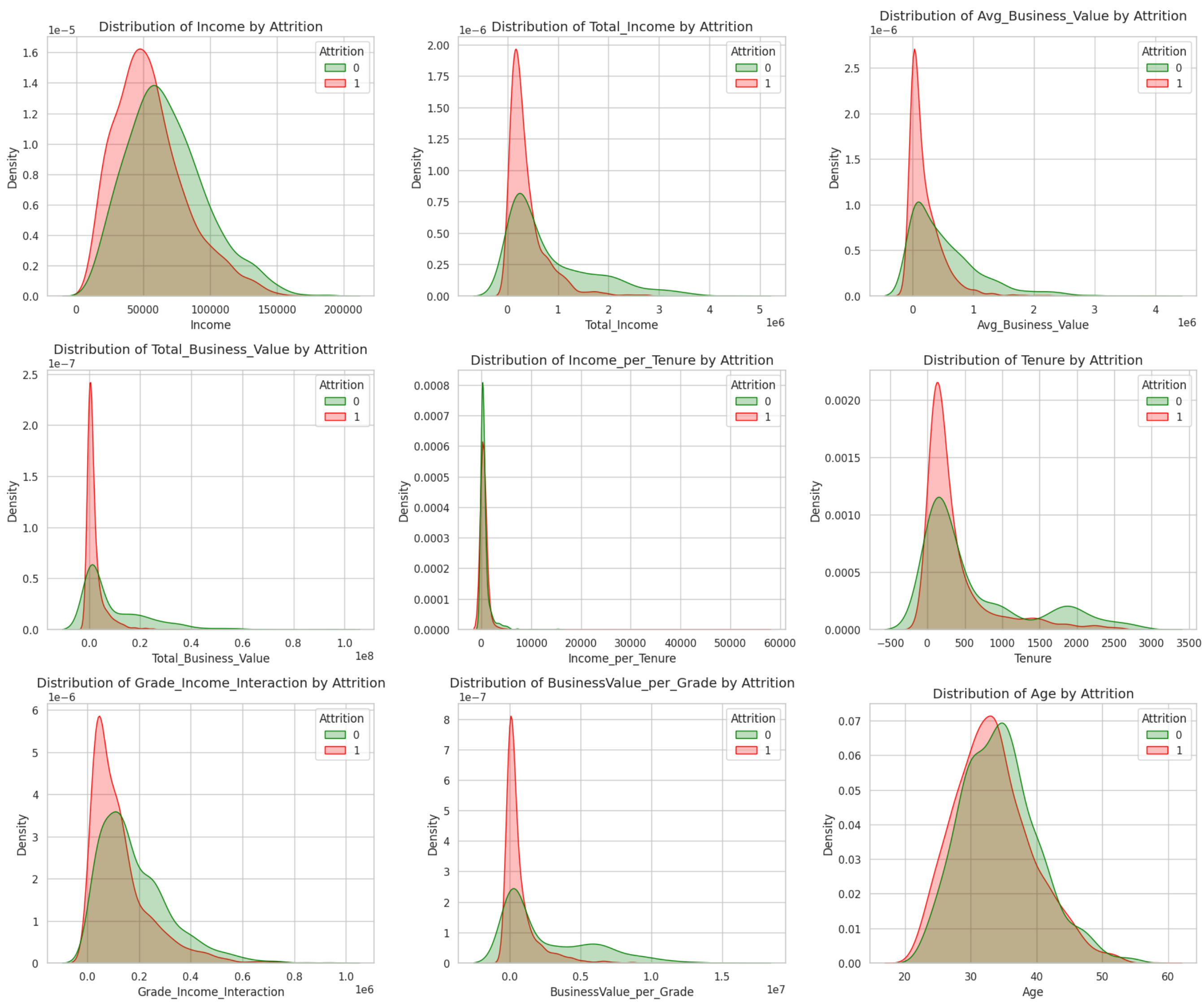
Observation

- Income per Tenure boxplots are highly skewed for both churned and non-churned drivers, indicating long-tailed distributions.
- Most features show the presence of outliers.
- Churned drivers tend to exhibit more outliers overall compared to non-churned drivers.
- Tenure-related features show significant outliers among attrited drivers.
- Features such as Total Business Value, Average Business Value, Total Income, Tenure, and Grade-Income Interaction have a wider spread for churned drivers.
- Age and Income fields show a similar spread for both churned and non-churned groups.
- Outliers are minimal in:
  - Age
  - Income per Tenure

Univariate Numerical Analysis

```
sns.set(style='whitegrid')
```

```
fig, axes = plt.subplots(3, 3, figsize=(18, 15))
axes = axes.flatten()

for i, col in enumerate(cols_to_plot):
    sns.kdeplot(
        data=agg_df,
        x=col,
        hue='Attrition',
        palette=attrition_palette1,
        fill=True,
        common_norm=False,
        ax=axes[i]
    )
    axes[i].set_title(f'Distribution of {col} by Attrition', fontsize=14)
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Density')

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

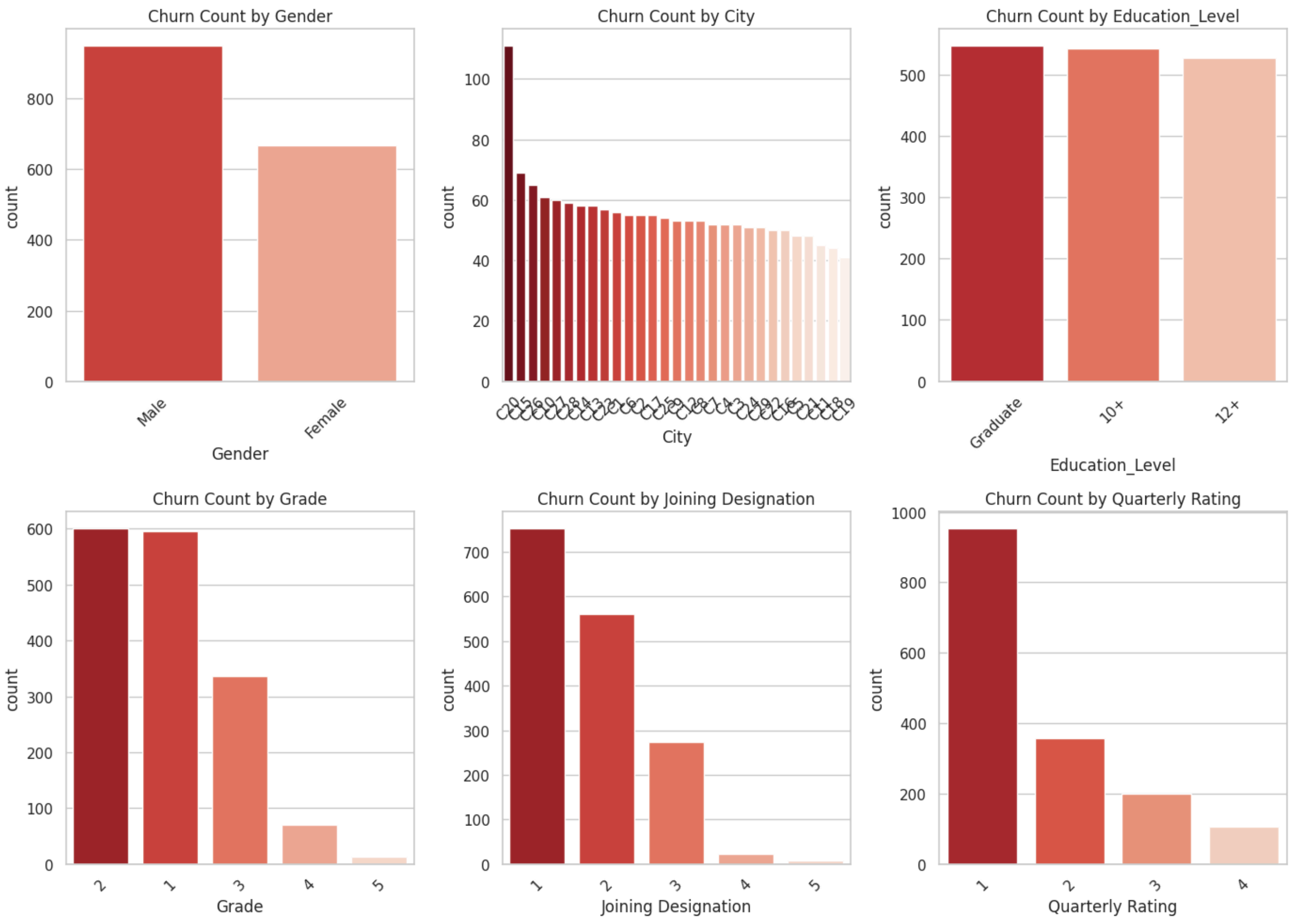


Univariate Observations (Boxplots and KDE)

- **Income & Total Income:** Churned drivers earn lower and more similar incomes, with narrower distributions compared to retained drivers.
- **Business Value:** Churned drivers show higher peaks at lower business values, especially when normalized by grade.
- **Tenure:** Churned drivers have shorter tenure; retained drivers have longer, right-skewed tenure distributions.
- **Age:** Little difference between churned and retained groups.
- **Grade:** Minor variation between the two groups.
- **Overall:** Lower income and business contributions correlate with higher churn risk.

Categorical Variable Analysis

```
churned_df = agg_df[agg_df['Attrition'] == 1].copy()

gender_labels = {'0': 'Male', '1': 'Female'}
education_labels = {'0': '10+', '1': '12+', '2': 'Graduate'}

cat_cols = ['Gender', 'City', 'Education_Level', 'Grade', 'Joining Designation', 'Quarterly Rating']

churned_df[cat_cols] = churned_df[cat_cols].astype(str).fillna('Unknown')

fig, axes = plt.subplots(2, 3, figsize=(14, 10))
axes = axes.flatten()

for i, col in enumerate(cat_cols):
    counts = churned_df[col].value_counts()
    order = counts.index.tolist()
    base_palette = sns.color_palette("Reds", n_colors=len(order))
    palette = dict(zip(order, base_palette[::-1]))

    ax = axes[i]
    sns.countplot(data=churned_df, x=col, order=order, palette=palette, ax=ax)

    if col == 'Gender':
        ax.set_xticklabels([gender_labels.get(label, label) for label in order], rotation=45)
    elif col == 'Education_Level':
        ax.set_xticklabels([education_labels.get(label, label) for label in order], rotation=45)
    else:
        ax.tick_params(axis='x', rotation=45)

    ax.set_title(f'Churn Count by {col}')

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



Observations

To uncover attrition patterns, we focus on **churned drivers** to identify **at-risk segments** based on demographics, performance, and location. This targeted approach enables precise retention strategies and improved churn prediction.

- **Gender:** More male drivers have churned, likely reflecting Ola's predominantly male driver base.
- **City:** Drivers from **City C20** show the highest churn rates, indicating possible regional or operational challenges.
- **Education Level:** Churn rates are similar across education levels, with only marginally higher churn among graduates.
- **Grade:** Drivers with **lower grades** churn more frequently, suggesting links to performance or experience.
- **Joining Designation:** Higher churn among drivers who joined at **lower designations**, possibly due to limited growth or unmet expectations.
- **Quarterly Rating:** Lower ratings correlate strongly with higher churn.

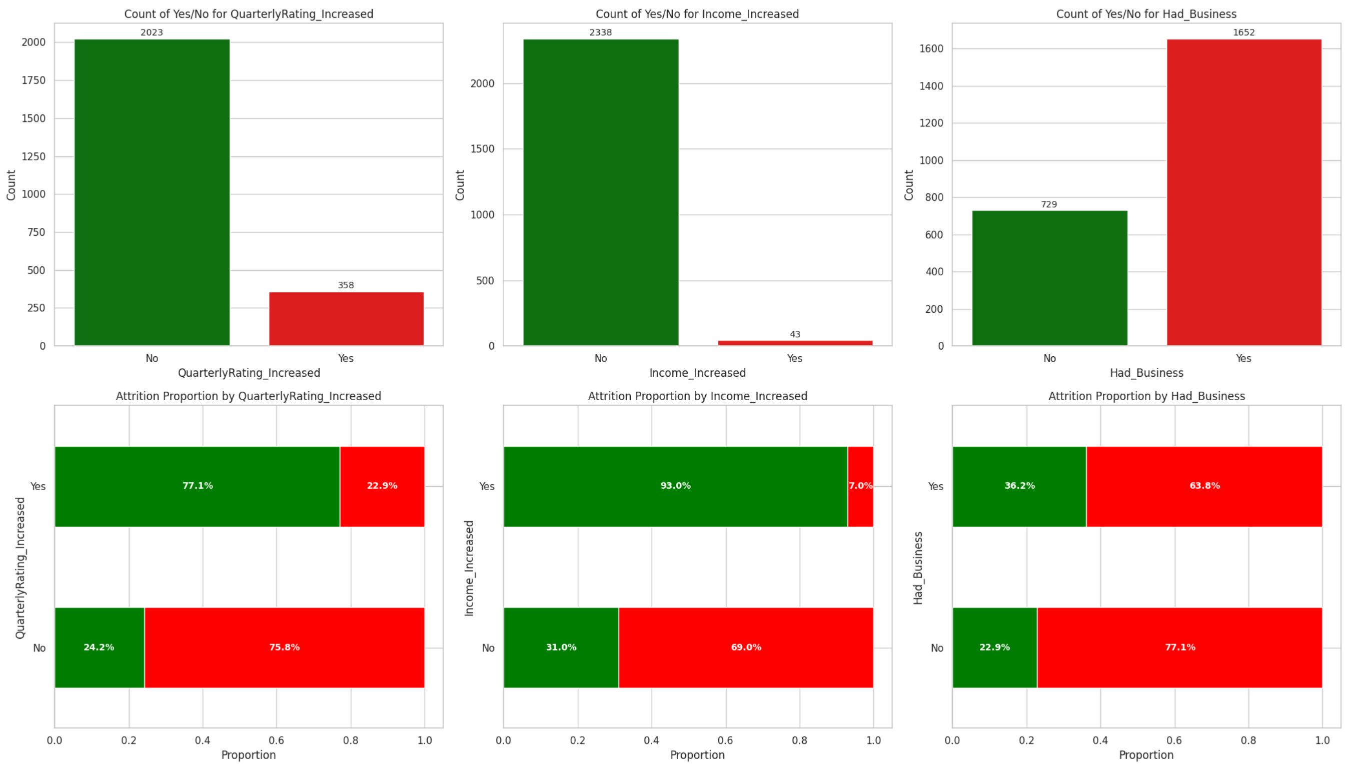
Overall Insight

Churn is concentrated among drivers with **lower performance indicators** (grades, designations, ratings), with additional influence from **regional factors** (notably City C20) and a **male-skewed workforce**. These findings support the development of targeted retention efforts for vulnerable driver groups.

Binary Flag Analysis

```
sns.set(style="whitegrid")

plot_df = agg_df.copy()
for col in ['QuarterlyRating_Increased', 'Income_Increased', 'Had_Business', 'Attrition']:
```

```
        plot_df[col] = plot_df[col].map({0: 'No', 1: 'Yes'})

    palette = {'No': 'green', 'Yes': 'red'}

    fig, axes = plt.subplots(2, 3, figsize=(21, 12))
    axes = axes.flatten()

    def plot_counts(ax, feature):
        counts = plot_df[feature].value_counts().reindex(['No', 'Yes'])
        sns.barplot(
            x=counts.index,
            y=counts.values,
            palette=[palette[x] for x in counts.index],
            ax=ax
        )
        ax.set_ylabel('Count')
        ax.set_title(f'Count of Yes/No for {feature}')
        for i, v in enumerate(counts.values):
            ax.text(i, v + max(counts.values)*0.01, str(v), ha='center', fontsize=10)

    def plot_prop(ax, feature):
        prop_df = plot_df.groupby([feature, 'Attrition']).size().unstack(fill_value=0)
        prop_df = prop_df.div(prop_df.sum(axis=1), axis=0)
        prop_df.plot(
            kind='barh',
            stacked=True,
            color=[palette[col] for col in prop_df.columns],
            ax=ax,
            legend=False
        )
        ax.set_xlabel('Proportion')
        ax.set_ylabel(feature)
        ax.set_title(f'Attrition Proportion by {feature}')
        for i, row in enumerate(prop_df.itertuples(index=False)):
            cum = 0
            for val, col in zip(row, prop_df.columns):
                if val > 0.02:
                    ax.text(
                        cum + val / 2, i,
                        f'{val*100:.1f}%',
                        ha='center', va='center',
                        color='white',
                        fontsize=10,
                        fontweight='bold'
                    )
                cum += val

    features = ['QuarterlyRating_Increased', 'Income_Increased', 'Had_Business']

    for i, feat in enumerate(features):
        plot_counts(axes[i], feat)
        plot_prop(axes[i + 3], feat)

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



Observations

1. **Quarterly Rating Increase**
   - 358 drivers experienced an increase; 2,338 did not.
   - Among those with increased ratings, 77% remain and 23% have churned.
   - Among those without increase, 24% remain and 76% have churned.
2. **Income Increase**
   - 43 drivers had income growth; 2,338 did not.
   - Of those with income increase, 93% remain and 7% churned.
   - Among those without income growth, 31% remain and 69% churned.
3. **Business Contribution**
   - 1,652 drivers generated positive business value; 729 had zero or negative.
   - Churn rates: 63.8% for positive contributors, 77% for non-contributors.

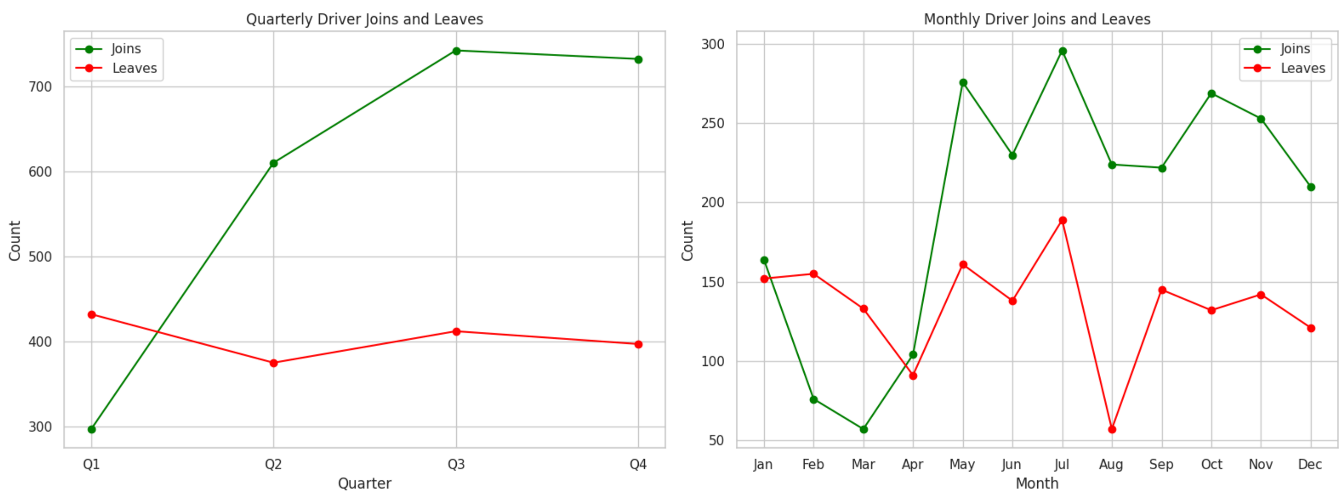
These findings indicate that increases in rating and income are linked to higher retention, and contributing business value correlates with lower churn.

Drivers showing positive trends in performance and income, along with business contribution, are significantly more likely to stay. Focusing retention efforts on those with stagnant or declining performance and incentivizing business contributions can help reduce churn and boost engagement.

Seasonality Analysis

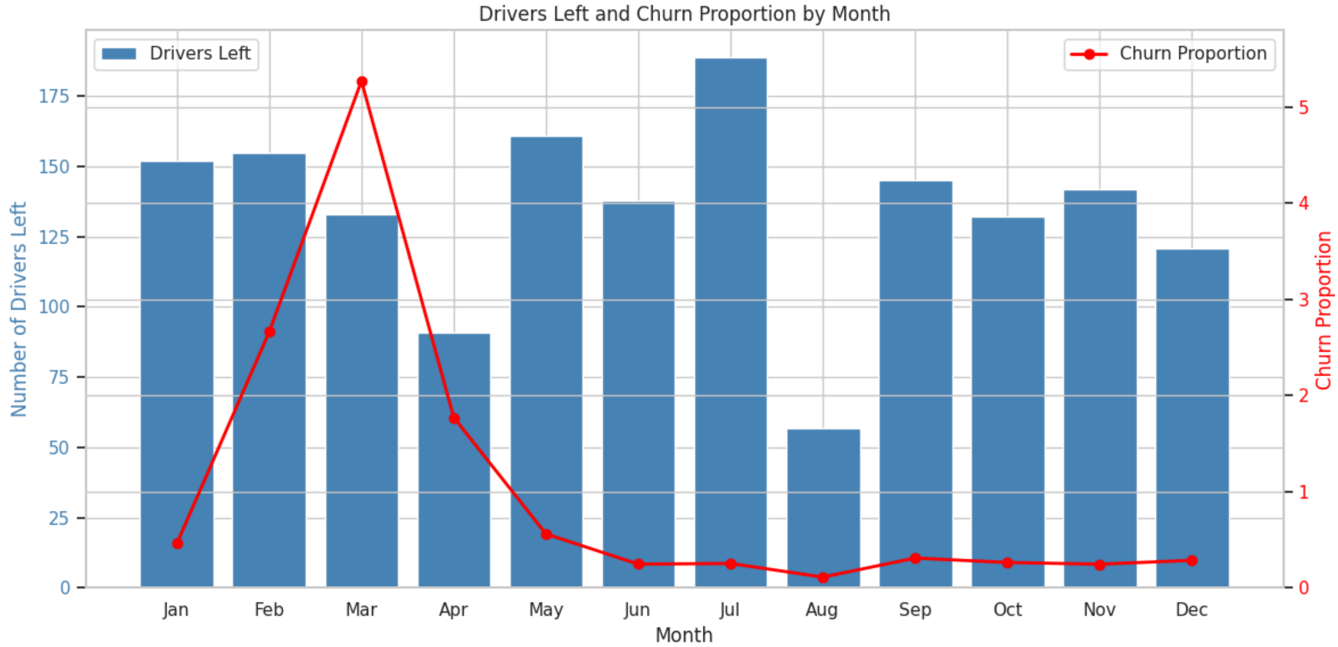
```
agg_df['Leaving_Month'] = agg_df['Last_Working_Date'].dt.month

monthly_joins = agg_df.groupby('Joining_Month').size().sort_index()
monthly_leaves = agg_df[agg_df['Attrition'] == 1].groupby('Leaving_Month').size().sort_index()

quarterly_joins = agg_df.groupby('Joining_Quarter').size().sort_index()
quarterly_leaves = agg_df[agg_df['Attrition'] == 1].groupby('Reporting_Quarter').size().sort_index()

monthly_data = (
```

```
    monthly_joins.to_frame(name='Joins')
    .join(monthly_leaves.to_frame(name='Leaves'), how='outer')
    .fillna(0)
)

quarterly_data = (
    quarterly_joins.to_frame(name='Joins')
    .join(quarterly_leaves.to_frame(name='Leaves'), how='outer')
    .fillna(0)
)

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
          'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
quarters = ['Q1', 'Q2', 'Q3', 'Q4']

fig, axs = plt.subplots(1, 2, figsize=(16, 6))

axs[0].plot(quarterly_data.index, quarterly_data['Joins'], marker='o', label='Joins', color=attrition_palette1[0])
axs[0].plot(quarterly_data.index, quarterly_data['Leaves'], marker='o', label='Leaves', color=attrition_palette1[1])
axs[0].set_xticks(range(1, 5))
axs[0].set_xticklabels(quarters)
axs[0].set_xlabel('Quarter')
axs[0].set_ylabel('Count')
axs[0].set_title('Quarterly Driver Joins and Leaves')
axs[0].legend()

axs[1].plot(monthly_data.index, monthly_data['Joins'], marker='o', label='Joins', color=attrition_palette1[0])
axs[1].plot(monthly_data.index, monthly_data['Leaves'], marker='o', label='Leaves', color=attrition_palette1[1])
axs[1].set_xticks(range(1, 13))
axs[1].set_xticklabels(months)
axs[1].set_xlabel('Month')
axs[1].set_ylabel('Count')
axs[1].set_title('Monthly Driver Joins and Leaves')
axs[1].legend()

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



- Q1 experiences more drivers leaving than joining, indicating a net workforce loss early in the year. This may result from reduced demand post-holiday season or drivers transitioning jobs.
- Q2, Q3, and Q4 see significantly higher driver join counts, with Q3 recording the highest new joins, likely due to rising demand during festive seasons and strategic recruitment efforts.
- February and March have more drivers leaving than joining, highlighting these months as critical for attrition. Possible reasons include contract renewals, academic year endings, or seasonal job changes.
- August shows the lowest number of drivers leaving, suggesting better retention or workforce stability, potentially due to festivals that encourage drivers to stay longer for increased earnings.
- July records the highest number of driver joins, marking it as the peak hiring month, likely driven by pre-festival demand buildup and mid-year recruitment campaigns.

```
left_drivers = agg_df[agg_df['Attrition'] == 1].copy()
left_drivers['Leaving_Year'] = left_drivers['Last_Working_Date'].dt.year
left_drivers['Leaving_Month'] = left_drivers['Last_Working_Date'].dt.month

agg_df['Joining_Year'] = agg_df['Joining_Date'].dt.year
agg_df['Joining_Month'] = agg_df['Joining_Date'].dt.month

left_counts_month = left_drivers.groupby('Leaving_Month').size()

joined_per_month_year = agg_df.groupby(['Joining_Year', 'Joining_Month']).size()
left_counts_month_year = left_drivers.groupby(['Leaving_Year', 'Leaving_Month']).size()
left_counts_month_year.index.names = ['Joining_Year', 'Joining_Month']

churn_prop = (left_counts_month_year / joined_per_month_year).fillna(0)
churn_prop_by_month = churn_prop.groupby('Joining_Month').mean()

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

fig, ax1 = plt.subplots(figsize=(12, 6))

ax1.bar(left_counts_month.index, left_counts_month.values, color='steelblue', label='Drivers Left')
ax1.set_xlabel('Month')
ax1.set_ylabel('Number of Drivers Left', color='steelblue')
ax1.set_xticks(range(1, 13))
ax1.set_xticklabels(months)
ax1.tick_params(axis='y', labelcolor='steelblue')

ax2 = ax1.twinx()
ax2.plot(churn_prop_by_month.index, churn_prop_by_month.values, color='red', marker='o', linewidth=2, label='Churn Proportion')
ax2.set_ylabel('Churn Proportion', color='red')
ax2.tick_params(axis='y', labelcolor='red')
ax2.set_ylim(0, churn_prop_by_month.max() * 1.1)

plt.title('Drivers Left and Churn Proportion by Month')
fig.tight_layout()

ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

plt.show()
```



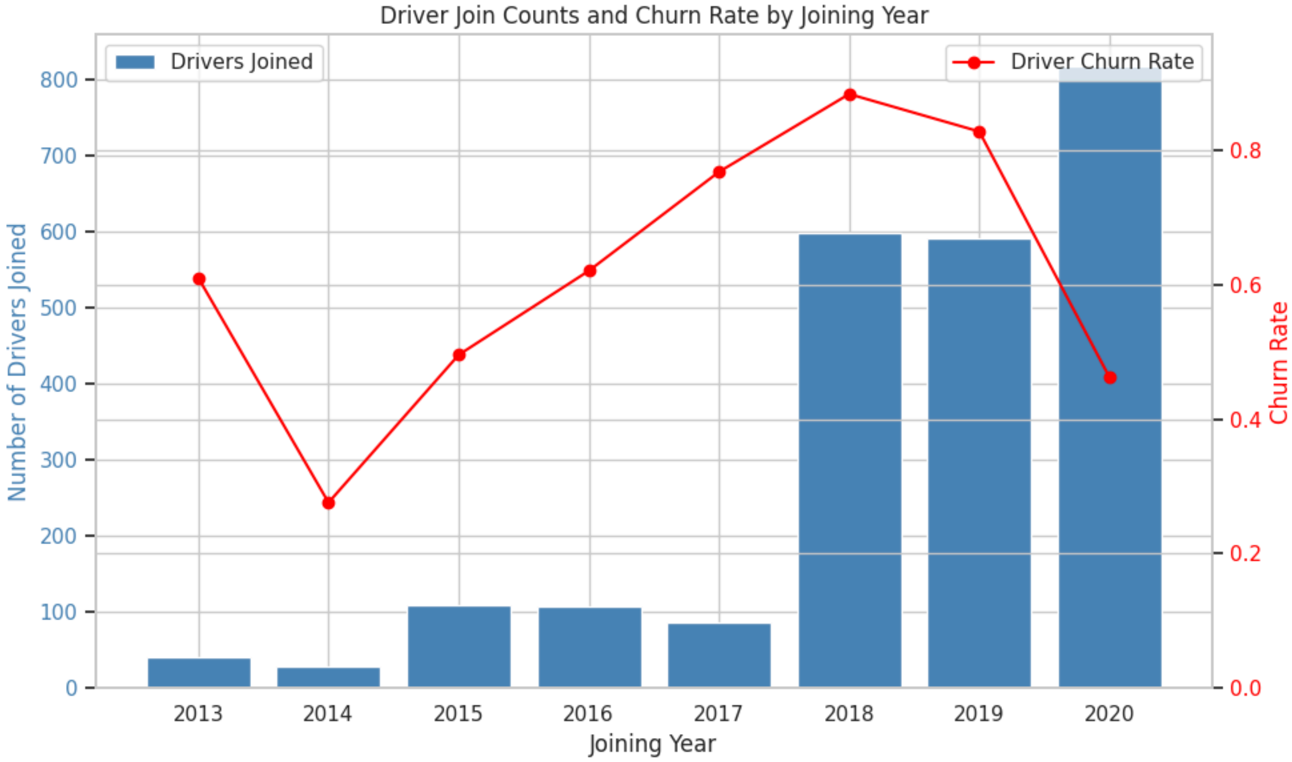
Q: When do drivers leave during the year?

- **July has the highest number of drivers leaving**, indicating a bulk exit likely related to seasonal trends or contract cycles. This significantly impacts operational capacity and requires proactive workforce planning.
- **March records the highest churn rate**, suggesting drivers active during this period are more likely to leave, highlighting retention challenges that need targeted interventions.

Business action:

Focus retention initiatives on drivers around March to lower churn risk, and prepare for operational fluctuations in July through hiring drives or incentive programs. Understanding these trends enables better driver engagement and service quality maintenance.

```
agg_df['Joining_Year'] = agg_df['Joining_Date'].dt.year

joined_per_year = agg_df.groupby('Joining_Year').size()
churn_rate = agg_df.groupby('Joining_Year')['Attrition'].mean()

fig, ax1 = plt.subplots(figsize=(10, 6))

ax1.bar(joined_per_year.index, joined_per_year.values, color='steelblue', label='Drivers Joined')
ax1.set_xlabel('Joining Year')
ax1.set_ylabel('Number of Drivers Joined', color='steelblue')
ax1.tick_params(axis='y', labelcolor='steelblue')

ax2 = ax1.twinx()
ax2.plot(churn_rate.index, churn_rate.values, color='red', marker='o', label='Driver Churn Rate')
ax2.set_ylabel('Churn Rate', color='red')
ax2.tick_params(axis='y', labelcolor='red')
ax2.set_ylim(0, churn_rate.max() * 1.1)

plt.title('Driver Join Counts and Churn Rate by Joining Year')
fig.tight_layout()

ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

plt.show()
```



Q: How does retention vary depending on the year drivers joined?

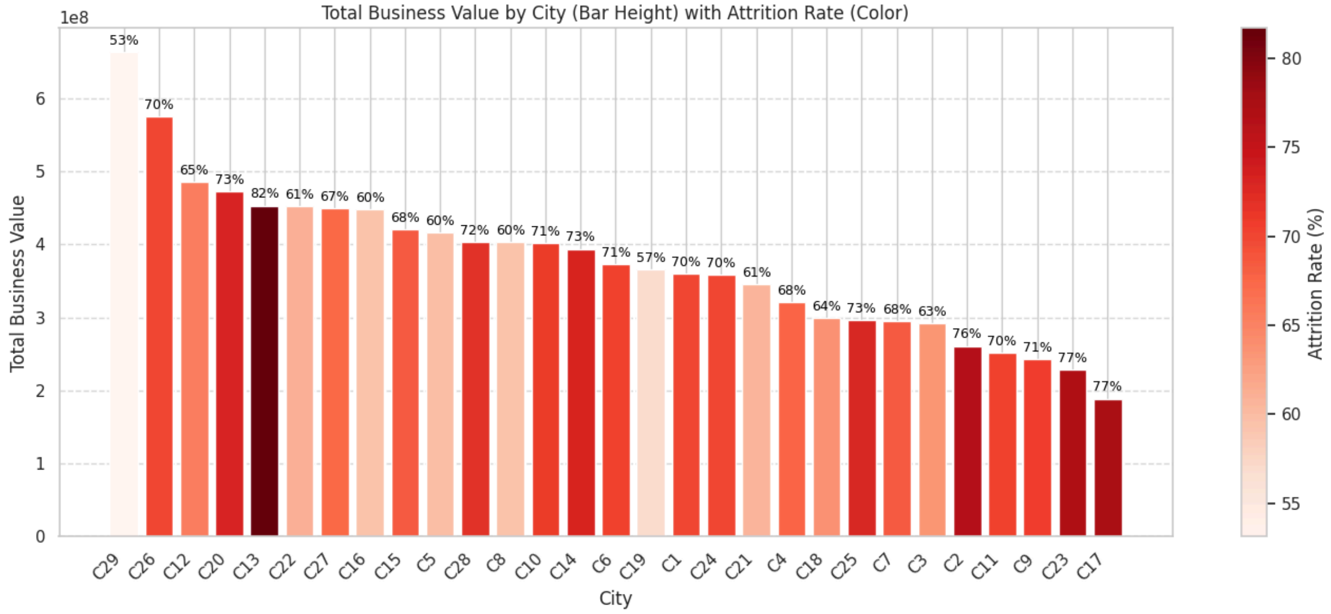
- **Rapid Driver Growth (2018–2020):** Driver joins surged sharply from 2018, peaking near 800 in 2020, reflecting major expansion.
- **Churn Spike in 2018:** The churn rate reached its highest (~85%) in 2018, coinciding with the rapid influx of new drivers.
- **Improved Retention in 2020:** Despite record driver joins, the 2020 cohort's churn dropped significantly to around 40%, indicating better retention.
- **Early Years Fluctuation (2013–2017):** Join counts and churn rates were unstable, with a notable churn dip in 2014, followed by rising churn alongside modest driver growth through 2017.

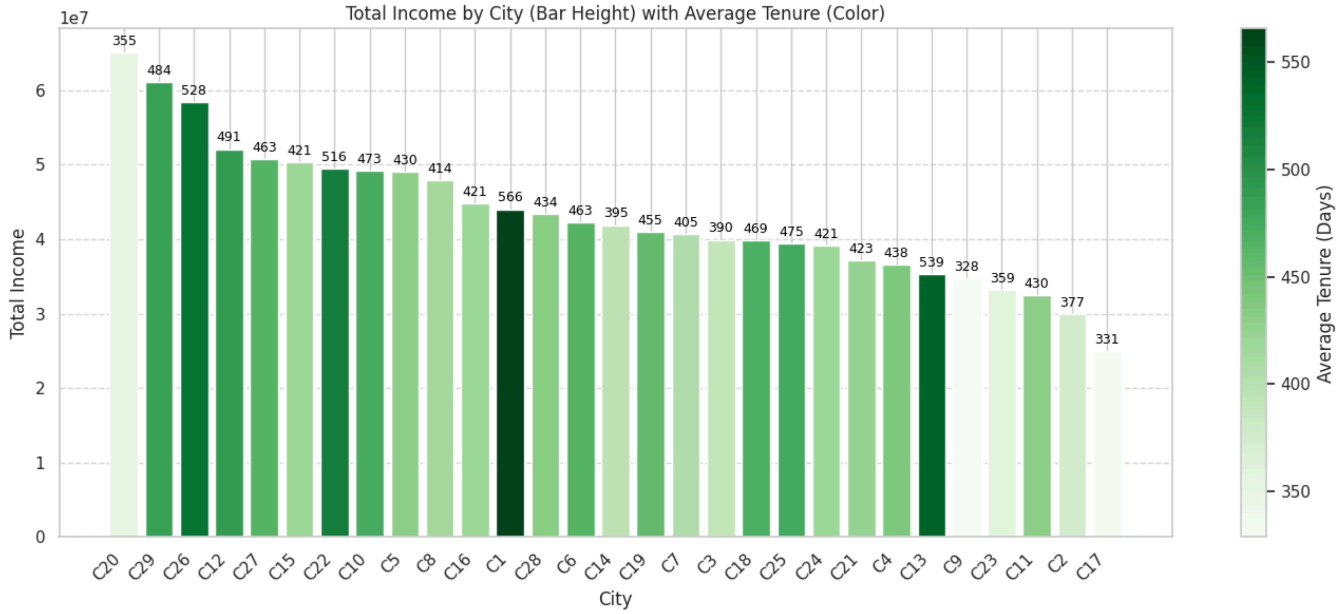
Overall:

After early volatility, the driver base grew explosively starting in 2018, initially accompanied by high churn. However, retention improved substantially by 2020, suggesting effective driver engagement and retention strategies.

City-Level Analysis

```
import matplotlib.cm as cm
import matplotlib.colors as mcolors

fig, axes = plt.subplots(2, 1, figsize=(14, 12))

city_joins = agg_df.groupby('City')['Joining_Date'].count()
city_leaves = agg_df[agg_df['Last_Working_Date'].notna()].groupby('City')['Last_Working_Date'].count()
city_business = agg_df.groupby('City')['Total_Business_Value'].sum()

city_attrition = pd.concat([city_joins, city_leaves, city_business], axis=1)
city_attrition.columns = ['Total_Joins', 'Total_Leaves', 'Total_Business_Value']
city_attrition['Attrition_Rate (%)'] = (city_attrition['Total_Leaves'] / city_attrition['Total_Joins']) * 100
city_attrition = city_attrition.dropna().sort_values('Total_Business_Value', ascending=False)

norm1 = mcolors.Normalize(vmin=city_attrition['Attrition_Rate (%)'].min(), vmax=city_attrition['Attrition_Rate (%)'].max())
cmap1 = cm.get_cmap('Reds')
colors1 = [cmap1(norm1(val)) for val in city_attrition['Attrition_Rate (%)']]

bars1 = axes[0].bar(city_attrition.index, city_attrition['Total_Business_Value'], color=colors1)
for bar, attr_rate in zip(bars1, city_attrition['Attrition_Rate (%)']):
    axes[0].annotate(
        f'{int(round(attr_rate))}%',
        xy=(bar.get_x() + bar.get_width() / 2, bar.get_height()),
        xytext=(0, 3),
        textcoords='offset points',
        ha='center',
        va='bottom',
        fontsize=9,
        color='black'
    )
sm1 = cm.ScalarMappable(cmap=cmap1, norm=norm1)
sm1.set_array([])
cbar1 = fig.colorbar(sm1, ax=axes[0])
cbar1.set_label('Attrition Rate (%)')

axes[0].set_title('Total Business Value by City (Bar Height) with Attrition Rate (Color)')
axes[0].set_ylabel('Total Business Value')
axes[0].set_xlabel('City')
axes[0].set_xticklabels(city_attrition.index, rotation=45, ha='right')
axes[0].grid(axis='y', linestyle='--', alpha=0.7)

city_income = agg_df.groupby('City')['Total_Income'].sum().sort_values(ascending=False)
city_tenure = agg_df.groupby('City')['Tenure'].mean().loc[city_income.index]

norm2 = mcolors.Normalize(vmin=city_tenure.min(), vmax=city_tenure.max())
cmap2 = cm.get_cmap('Greens')
colors2 = [cmap2(norm2(val)) for val in city_tenure]

bars2 = axes[1].bar(city_income.index, city_income.values, color=colors2)
for bar, tenure in zip(bars2, city_tenure):
    axes[1].annotate(
        f'{int(round(tenure))}',
        xy=(bar.get_x() + bar.get_width() / 2, bar.get_height()),
        xytext=(0, 3),
        textcoords='offset points',
        ha='center',
        va='bottom',
        fontsize=9,
        color='black'
    )
sm2 = cm.ScalarMappable(cmap=cmap2, norm=norm2)
sm2.set_array([])
cbar2 = fig.colorbar(sm2, ax=axes[1])
cbar2.set_label('Average Tenure (Days)')

axes[1].set_title('Total Income by City (Bar Height) with Average Tenure (Color)')
axes[1].set_ylabel('Total Income')
axes[1].set_xlabel('City')
axes[1].set_xticklabels(city_income.index, rotation=45, ha='right')
axes[1].grid(axis='y', linestyle='--', alpha=0.7)

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





These two city-wise graphs provide complementary perspectives—one highlights churn risk relative to business value, while the other reveals tenure patterns linked to income stability. Together, they offer actionable insights for prioritizing retention and growth strategies.

Graph 1: Total Business Value vs Attrition Rate by City

Q: Which cities generate high business value but also suffer from high driver attrition?

- **Bar Height:** Total Business Value
- **Color Intensity:** Attrition Rate (darker red = higher attrition)

Key Observations:

- **City C29** leads with the highest total business value and one of the lowest attrition rates, indicating a strong and stable market.
- **City C13** ranks 4th in business value but has a high attrition rate of 82%, making it a critical area needing focused retention efforts to boost business value.

- **City C20** ranks 3rd in business value with a high attrition rate of 73%, similarly requiring intervention.
- **City C26** is 3rd in business value with an average attrition rate and should be prioritized for retention strategies.
- **Cities C23 and C17** have the lowest business values combined with very high attrition rates, suggesting a need to reevaluate strategies, possibly reallocating resources or considering divestment.
- **City C19** and **City C8** exhibit average business value but low attrition, presenting opportunities for growth through driver or service expansion.
- **City C16** has above-average business value with low attrition, highlighting it as another stable region worth investment.

Business Insights

- **Prioritize retention efforts in cities with high business value but elevated attrition**, such as C13 and C20, to maximize ROI by reducing churn among valuable drivers.
- **Focus on strengthening stable markets like C29, C16, C19, and C8**, where attrition is low and business value is moderate to high, to sustain and grow their performance further.
- **Reevaluate strategies in cities with low business value and high attrition, like C23 and C17**, which may require operational changes, resource reallocation, or divestment to improve efficiency.
- **Tailor region-specific retention programs** addressing unique challenges—such as targeted incentives in high-churn cities and growth investments in stable cities—to optimize overall driver retention and business growth.

Graph 2: Total Income vs Average Tenure by City

Q: How does driver tenure influence total income generation across cities?

- **Bar Height**: Total Income (descending order)
- **Bar Color**: Average Tenure (dark green = longer tenure)

Key Observations:

-C20 is highest earning city but drivers stay for small tenure less than an year -C29 and C12 have high income and average tunure. tenure can be imploved

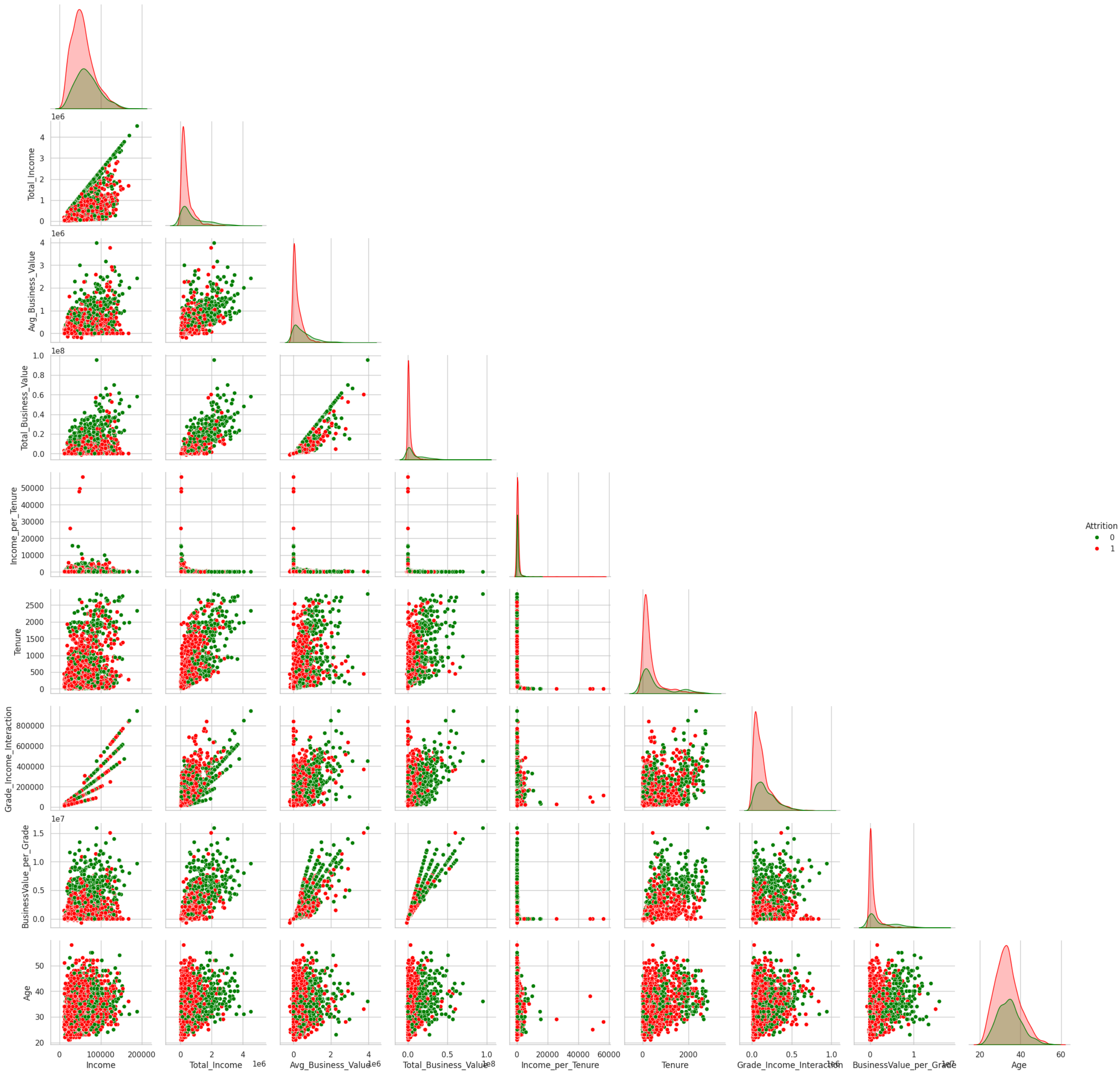
- C26 stands out with high income and long driver tenure, reinforcing it as a **model city**.
- C1 has average income but strongest tenure
- C13 has very strong tenure but oncome on lower side
- C2 and C17 show low income and short tenure, potentially signaling inefficiencies or market saturation.

Business Insights

- **High-priority retention zones**: C13, C26, C20 — reduce churn to protect high revenue.
- **Evaluate for strategic action**: C23, C17 — consider restructuring, incentives, or reallocation.
- **Best practices cities**: C29 (high value, low attrition, long tenure) and C19 (low attrition, loyal base).
- Replicate success in similar markets using C29 and C19 strategies for tenure, onboarding, and support.

MultiColinearity Detection

```
sns.pairplot(agg_df[cols_to_plot + ['Attrition']],
             hue='Attrition',
             diag_kind='kde',
             corner=True,
             palette=attrition_palette1)
plt.show()
```



Observations from Pairplot

Based on the pairplot, we observe the following:

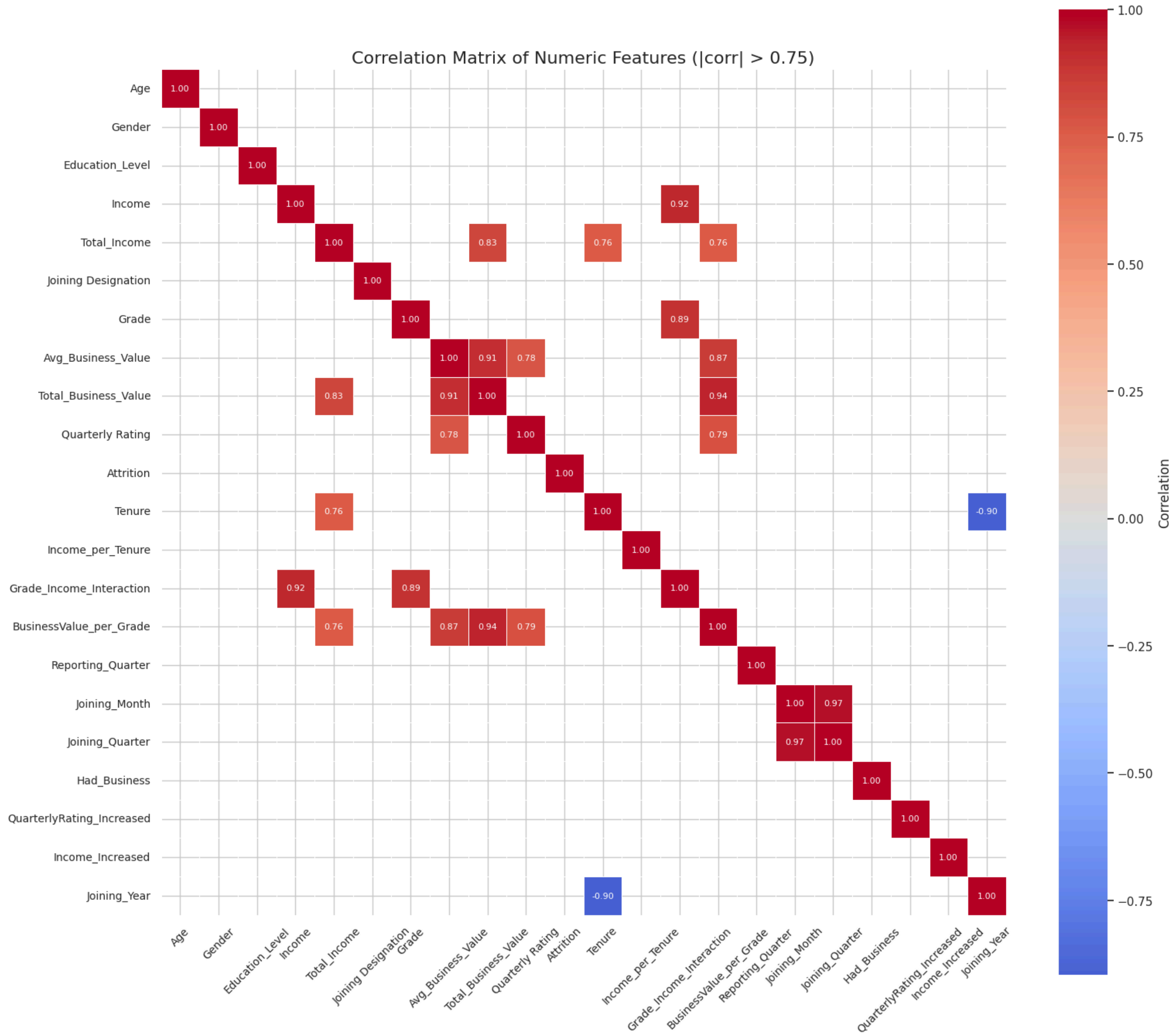
- **Grade Income Interaction** shows a strong linear relationship with both **Income** (latest income) and **Total Income**, indicating potential redundancy.
- **Avg Business Value** is strongly related to both **Total Business Value** and **BusinessValue per Grade**, suggesting these features form a multicollinear group.
- **Income per Tenure** has a highly skewed and sparse distribution, making it a weak candidate for predictive modeling.

These visual cues point to potential multicollinearity or redundancy among several features. We will validate these findings using a correlation heatmap.

While the pairplot suggests that **Grade Income Interaction** is largely explained by its components (**Grade** and **Income**) and that **Avg Business Value** overlaps heavily with **Total Business Value**, final feature selection will be based on correlation analysis. However, we are confident in dropping **Income per Tenure** due to its poor distribution.

Next, we will include temporal features such as **Joining Month** and **Joining Quarter** to evaluate correlations across time-based variables and determine which to retain.

```
num_cols = agg_df.select_dtypes(include=['int64', 'float64', 'int32']).columns.tolist()
for col_to_remove in ['Driver_ID', 'Leaving_Month']:
    if col_to_remove in num_cols:
        num_cols.remove(col_to_remove)

corr_matrix = agg_df[num_cols].corr()

threshold = 0.75
mask = corr_matrix.abs() <= threshold

masked_corr = corr_matrix.mask(mask)

plt.figure(figsize=(16, 14))
sns.heatmap(
    masked_corr,
    annot=True,
    fmt=".2f",
    cmap='coolwarm',
    center=0,
    annot_kws={"size": 8},
    cbar_kws={'label': 'Correlation'},
    linewidths=0.5,
    square=True,
    mask=mask
)
plt.title('Correlation Matrix of Numeric Features (|corr| > 0.75)', fontsize=16)
plt.xticks(rotation=45, fontsize=10)
plt.yticks(rotation=0, fontsize=10)
plt.tight_layout()
plt.show()

corr_pairs = corr_matrix.unstack().reset_index()
corr_pairs.columns = ['Feature_1', 'Feature_2', 'Correlation']

corr_pairs = corr_pairs[corr_pairs['Feature_1'] < corr_pairs['Feature_2']]

corr_pairs['Abs_Correlation'] = corr_pairs['Correlation'].abs()

high_corr_pairs = corr_pairs[corr_pairs['Abs_Correlation'] > threshold]

high_corr_pairs = high_corr_pairs.sort_values(by='Abs_Correlation', ascending=False)

high_corr_pairs = high_corr_pairs.drop(columns='Abs_Correlation').reset_index(drop=True)

print("\nTop Feature Pairs with High Correlation (|r| > 0.75):\n")
print(high_corr_pairs.to_string(index=False))
```




```
Top Feature Pairs with High Correlation (|r| > 0.75):

               Feature_1                 Feature_2  Correlation
           Joining_Month           Joining_Quarter     0.966554
 BusinessValue_per_Grade      Total_Business_Value     0.936520
Grade_Income_Interaction                    Income     0.920715
      Avg_Business_Value      Total_Business_Value     0.909796
            Joining_Year                    Tenure    -0.896183
                   Grade  Grade_Income_Interaction     0.888911
      Avg_Business_Value   BusinessValue_per_Grade     0.865222
    Total_Business_Value              Total_Income     0.827558
 BusinessValue_per_Grade          Quarterly Rating     0.787851
      Avg_Business_Value          Quarterly Rating     0.780214
 BusinessValue_per_Grade              Total_Income     0.761086
                  Tenure              Total_Income     0.756735
```

Observation

The correlation matrix reveals several strong relationships among features, indicating potential multicollinearity and redundancy:

- Joining_Month and Joining_Quarter have a correlation of 0.97 and capture similar seasonal signals; only one is necessary.
- BusinessValue_per_Grade and Total_Business_Value have a correlation of 0.94; Avg_Business_Value and Total_Business_Value correlate at 0.91. These are highly overlapping business contribution metrics.
- Grade_Income_Interaction correlates with Income at 0.92 and with Grade at 0.89. The interaction term adds little beyond the base variables.
- Joining_Year and Tenure have a correlation of -0.90; older joiners have higher tenure, making the year redundant when tenure is available.
- Avg_Business_Value and BusinessValue_per_Grade correlate at 0.87 and are largely interchangeable derived metrics.
- Total_Business_Value and Total_Income correlate at 0.83, reflecting overlap between compensation and business contribution.
- BusinessValue_per_Grade and Quarterly Rating correlate at 0.79; Avg_Business_Value and Quarterly Rating at 0.78, showing moderate alignment of performance and contribution.
- BusinessValue_per_Grade and Total_Income correlate at 0.76.
- Tenure and Total_Income correlate at 0.76, indicating tenure influences compensation and performance.

Business Implications

- High correlation among business value metrics suggests consolidation to avoid inflated model weights.
- Redundant time features (month vs. quarter) dilute seasonal insights and complicate interpretation.
- Interaction terms highly correlated with base variables reduce clarity without adding signal.
- Tenure is a valuable predictor with a strong inverse relationship to attrition.
- Selecting fewer representative features improves model stability, interpretability, and generalization.

Feature Removal Decision

Feature	Reason
Joining_Month	Strongly correlated with Joining_Quarter; dropped for consistent granularity.
Grade_Income_Interaction	Redundant with Grade and Income; removed to maintain interpretability.
Avg_Business_Value	Overlaps with Total_Business_Value and BusinessValue_per_Grade.
BusinessValue_per_Grade	Redundant with Total_Business_Value and Avg_Business_Value.
Total_Income	Correlates with Business Value and Tenure; maximum income captured elsewhere.

This focused feature pruning ensures the model remains interpretable and statistically robust while preserving key business signals like Tenure and Quarterly Rating. It balances predictive strength with clarity, enabling better business insights and more reliable predictions.

Other than the features discussed above, we will remove the following from our model:

- **Leaving_Month**: Missing for active employees; risks target leakage.
- **Income_per_Tenure**: Skewed distribution and redundant with Income and Tenure.
- **Reporting_Date**, **Joining_Date**, **Last_Working_Date**: Raw dates replaced by quarterly features to reduce noise.
- **Driver_ID**: Unique identifier; no predictive value and may cause overfitting.

```
cols_to_drop = [
    'Leaving_Month',
    'Joining_Month',
    'Grade_Income_Interaction',
    'Avg_Business_Value',
    'Income_per_Tenure',
    'Total_Income',
    'BusinessValue_per_Grade',
    'Reporting_Date',
    'Joining_Date',
    'Last_Working_Date',
    'Driver_ID'
]

final_df = agg_df.drop(columns=cols_to_drop, errors='ignore')

final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2381 entries, 0 to 2380
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Age                     2381 non-null   int64
 1   Gender                  2381 non-null   int64
 2   City                    2381 non-null   object
 3   Education_Level         2381 non-null   int64
 4   Income                  2381 non-null   int64
 5   Joining Designation     2381 non-null   int64
 6   Grade                   2381 non-null   int64
 7   Total_Business_Value    2381 non-null   int64
 8   Quarterly Rating        2381 non-null   int64
```

```
 9   Attrition                  2381 non-null   int64
 10  Tenure                     2381 non-null   int64
 11  Reporting_Quarter          2381 non-null   int32
 12  Joining_Quarter            2381 non-null   int32
 13  Had_Business               2381 non-null   int64
 14  QuarterlyRating_Increased  2381 non-null   int64
 15  Income_Increased           2381 non-null   int64
 16  Joining_Year               2381 non-null   int32
dtypes: int32(3), int64(13), object(1)
memory usage: 288.5+ KB
```

Model Development

Preprocessing

```
X = final_df.drop(columns=['Attrition'])
y = final_df['Attrition']

for col in X.select_dtypes(include='object').columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])

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

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

- Dropped `Attrition` from features and stored it as target variable `y`.
- Label encoded the `City` column.
- Performed train-test split with an 80-20 ratio using stratified sampling to preserve class balance.
- Applied SMOTE to the training set to handle class imbalance by generating synthetic samples.
- Did not apply feature scaling since ensemble models (e.g., Random Forest, XGBoost) are tree-based and use feature thresholds rather than distances, making scaling unnecessary.

Creating a common function to:

- Print the confusion matrix
- Display the classification report
- Plot ROC and Precision-Recall (PR) curves
- Show the top 5 features based on model feature importance

```
def train_evaluate_model(model, X_train, y_train, X_test, y_test, model_name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test)[:, 1]
    elif hasattr(model, "decision_function"):
        y_proba = model.decision_function(X_test)
    else:
        y_proba = None

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'{model_name} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    print(f"\n{model_name} Classification Report:")
    print(classification_report(y_test, y_pred))

    if y_proba is not None:
        auc_score = roc_auc_score(y_test, y_proba)
        print(f"{model_name} AUC: {auc_score:.4f}")

        fpr, tpr, _ = roc_curve(y_test, y_proba)
        plt.figure(figsize=(6, 5))
        plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc_score:.4f})')
        plt.plot([0, 1], [0, 1], 'k--')
        plt.title(f'{model_name} ROC Curve')
```

```
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.legend(loc='lower right')
        plt.show()

        precision, recall, _ = precision_recall_curve(y_test, y_proba)
        avg_precision = average_precision_score(y_test, y_proba)
        plt.figure(figsize=(6, 5))
        plt.plot(recall, precision, label=f'PR curve (AP = {avg_precision:.4f})')
        plt.title(f'{model_name} Precision-Recall Curve')
        plt.xlabel('Recall')
        plt.ylabel('Precision')
        plt.legend(loc='lower left')
        plt.show()
    else:
        print(f"AUC and PR curve not available for {model_name} (no predict_proba or decision_function)")

    if hasattr(model, 'feature_importances_'):
        importances = model.feature_importances_
    elif model_name.lower() == 'catboost':
        importances = model.get_feature_importance()
    else:
        print("Feature importances not available for this model.")
        return

    feat_imp = sorted(zip(X_train.columns, importances), key=lambda x: x[1], reverse=True)[:5]
    features, importances_vals = zip(*feat_imp)

    importances_df = pd.DataFrame({
        'Feature': features,
        'Importance': importances_vals
    })
    print(f"\nTop 5 Important Features for {model_name}:")
    print(importances_df.to_string(index=False))

    plt.figure(figsize=(7, 4))
    sns.barplot(x=importances_vals, y=features)
    plt.title(f'{model_name} Top 5 Feature Importances')
    plt.xlabel('Importance')
    plt.show()
```

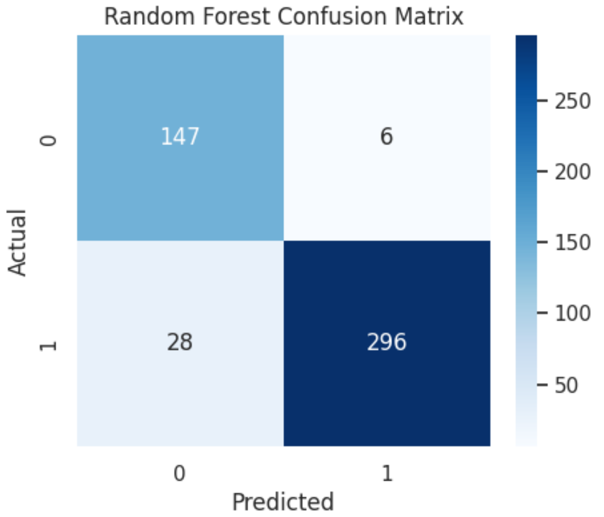
Building Classification Models

We will train the following ensemble classifiers for driver churn prediction:

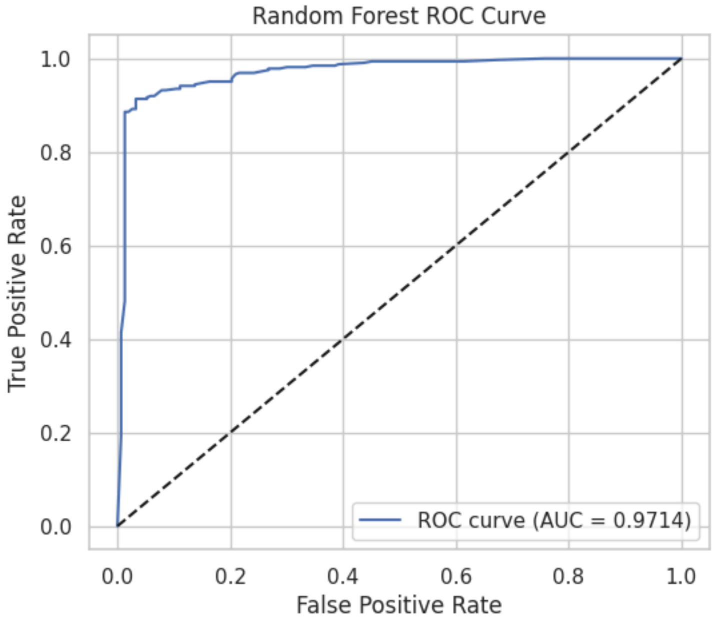
- **Random Forest**
- **XGBoost**
- **LightGBM**
- **CatBoost**
- **AdaBoost**

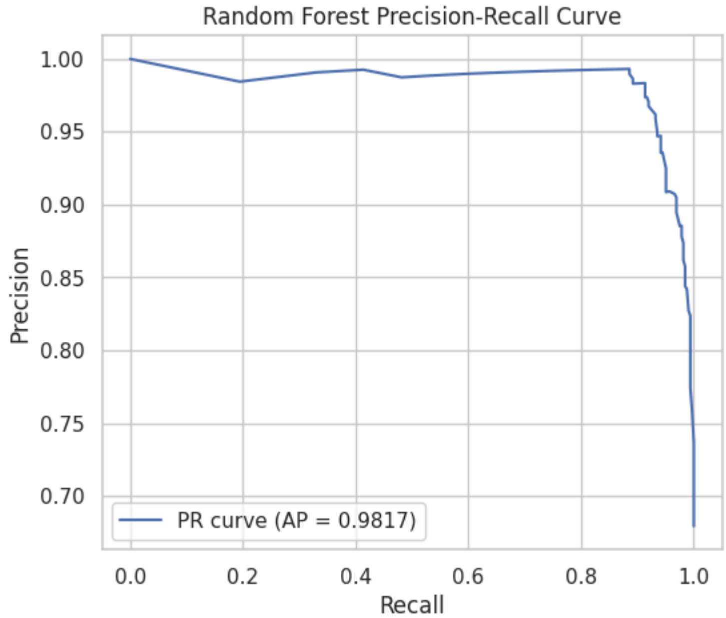
Each model will be trained using the same training data, followed by hyperparameter tuning and evaluation using metrics such as ROC-AUC and classification reports. This approach helps identify the best-performing algorithm for the task.

```
# Random Forest
rf = RandomForestClassifier(random_state=42)
train_evaluate_model(rf, X_train_smote, y_train_smote, X_test, y_test, 'Random Forest')

# XGBoost
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
train_evaluate_model(xgb, X_train_smote, y_train_smote, X_test, y_test, 'XGBoost')

# LightGBM
lgbm = LGBMClassifier(random_state=42)
train_evaluate_model(lgbm, X_train_smote, y_train_smote, X_test, y_test, 'LightGBM')

# CatBoost
cat = CatBoostClassifier(verbose=0, random_seed=42)
train_evaluate_model(cat, X_train_smote, y_train_smote, X_test, y_test, 'CatBoost')

# AdaBoost
ada = AdaBoostClassifier(random_state=42)
train_evaluate_model(ada, X_train_smote, y_train_smote, X_test, y_test, 'AdaBoost')
```

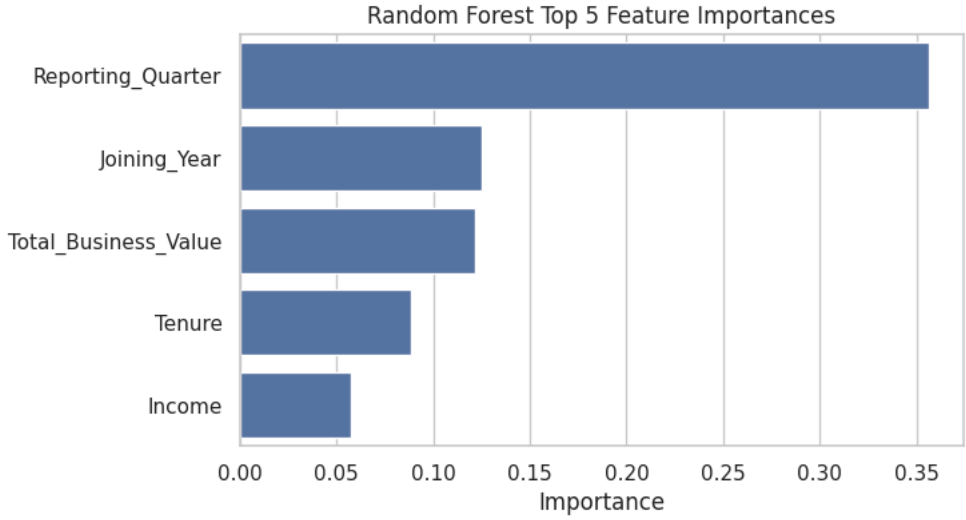


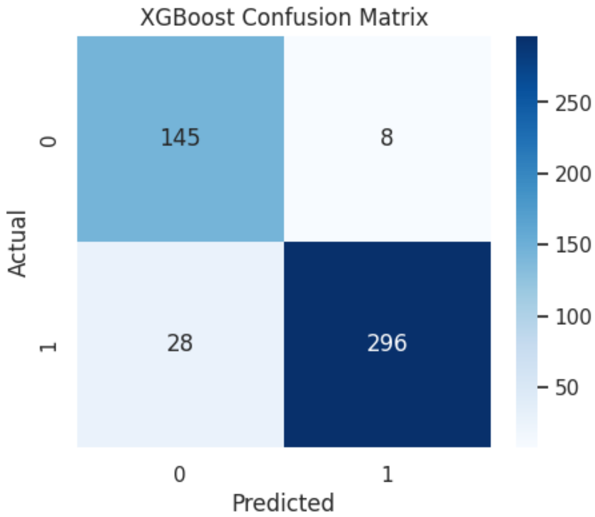

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.96      0.90       153
           1       0.98      0.91      0.95       324

    accuracy                           0.93       477
   macro avg       0.91      0.94      0.92       477
weighted avg       0.94      0.93      0.93       477

Random Forest AUC: 0.9714
```



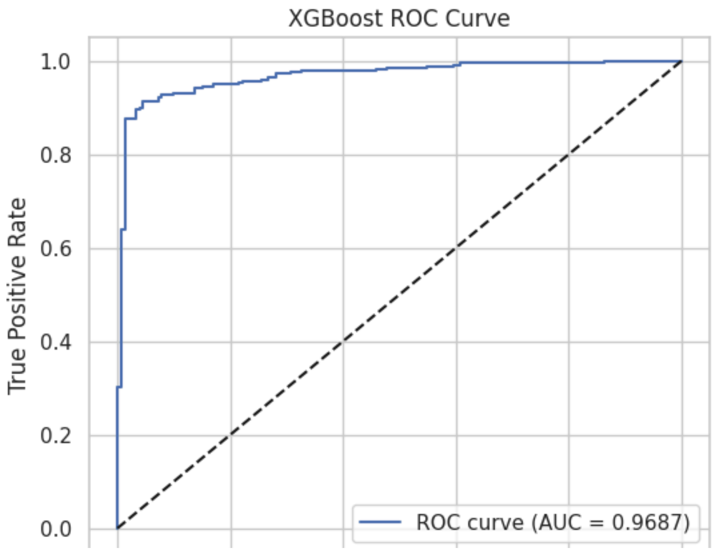


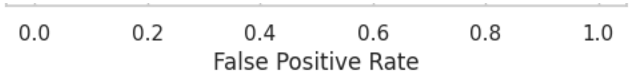
```
Top 5 Important Features for Random Forest:
             Feature  Importance
   Reporting_Quarter    0.356679
        Joining_Year    0.124838
Total_Business_Value    0.121819
              Tenure    0.088760
              Income    0.057560
```

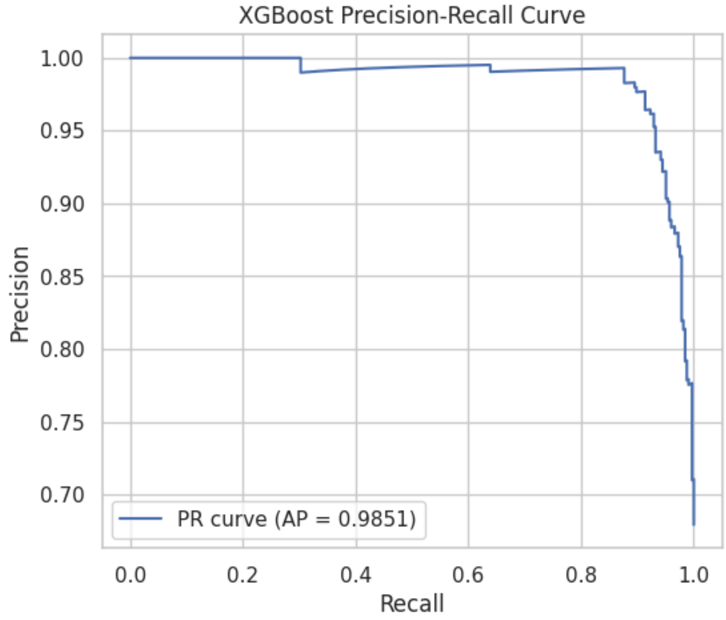




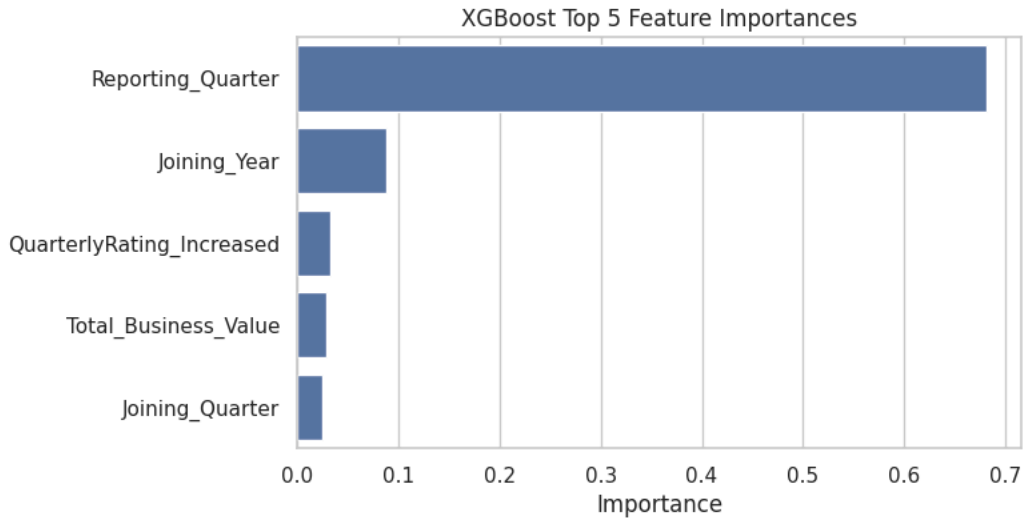

```
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.95      0.89       153
           1       0.97      0.91      0.94       324

    accuracy                           0.92       477
   macro avg       0.91      0.93      0.92       477
weighted avg       0.93      0.92      0.93       477

XGBoost AUC: 0.9687
```



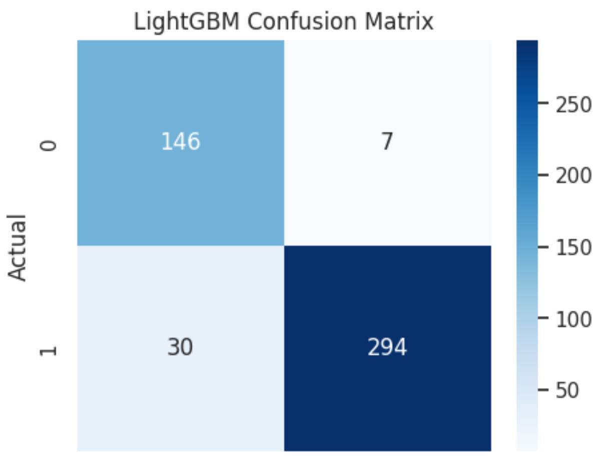


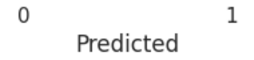



```
Top 5 Important Features for XGBoost:
                  Feature  Importance
        Reporting_Quarter    0.681639
             Joining_Year    0.087732
QuarterlyRating_Increased    0.032690
     Total_Business_Value    0.028637
          Joining_Quarter    0.025184
```

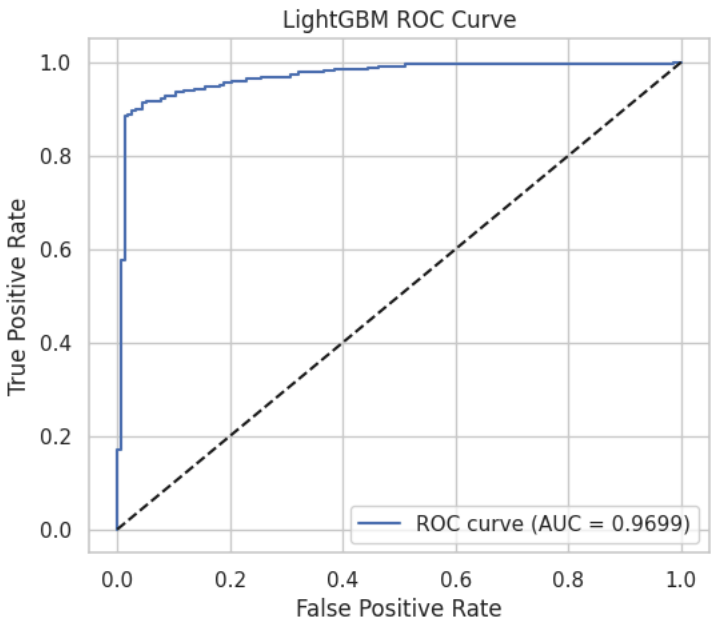


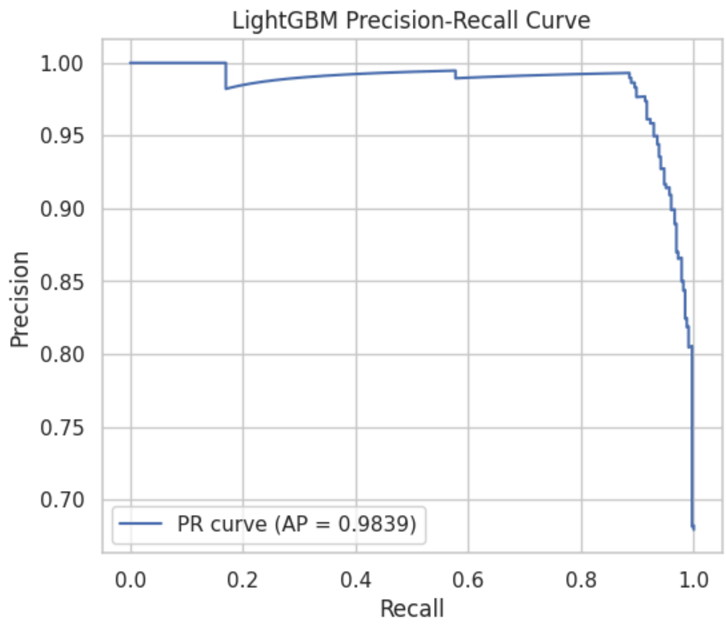

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 1292, number of negative: 1292
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000293 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 875
[LightGBM] [Info] Number of data points in the train set: 2584, number of used features: 16
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
```



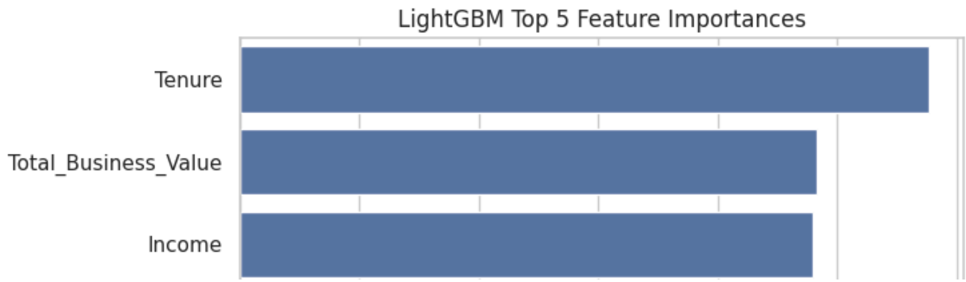


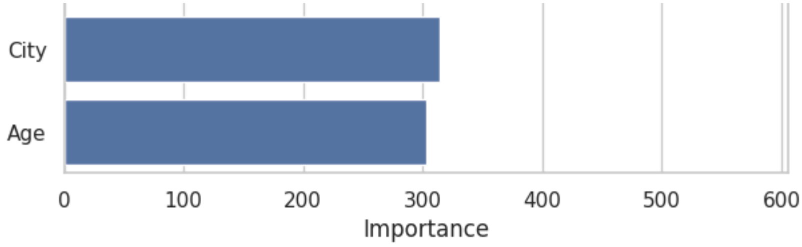

```
LightGBM Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.95      0.89       153
           1       0.98      0.91      0.94       324

    accuracy                           0.92       477
   macro avg       0.90      0.93      0.91       477
weighted avg       0.93      0.92      0.92       477

LightGBM AUC: 0.9699
```

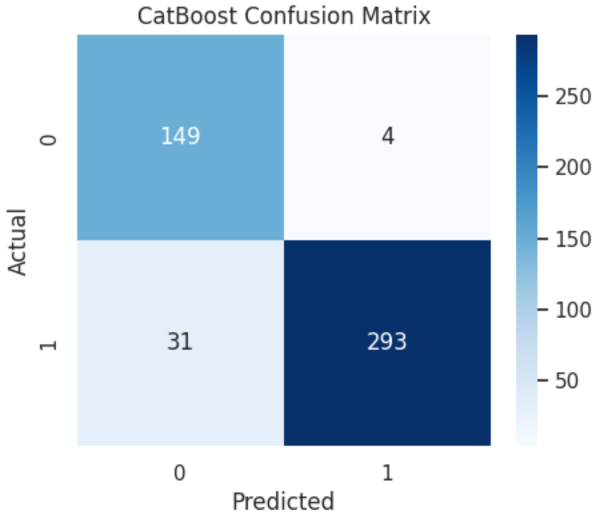




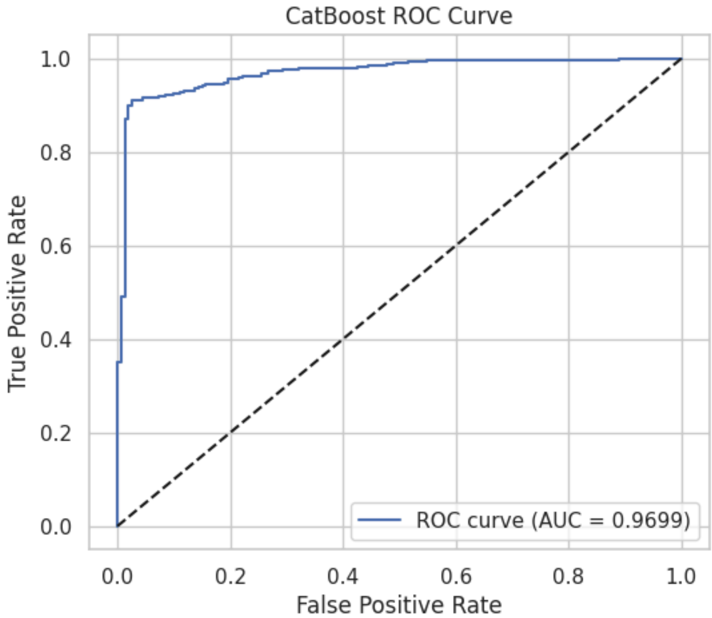

```
Top 5 Important Features for LightGBM:
           Feature  Importance
            Tenure         577
Total_Business_Value       483
            Income         479
              City         314
               Age         303
```

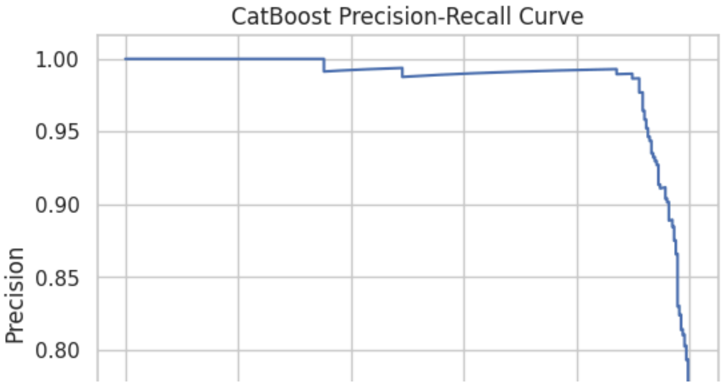


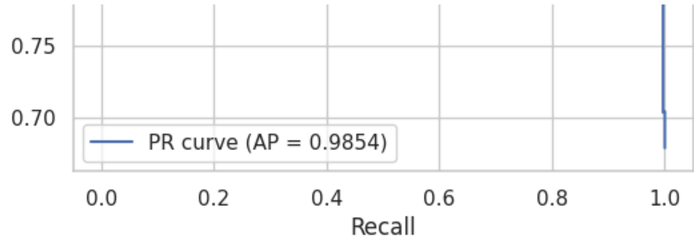




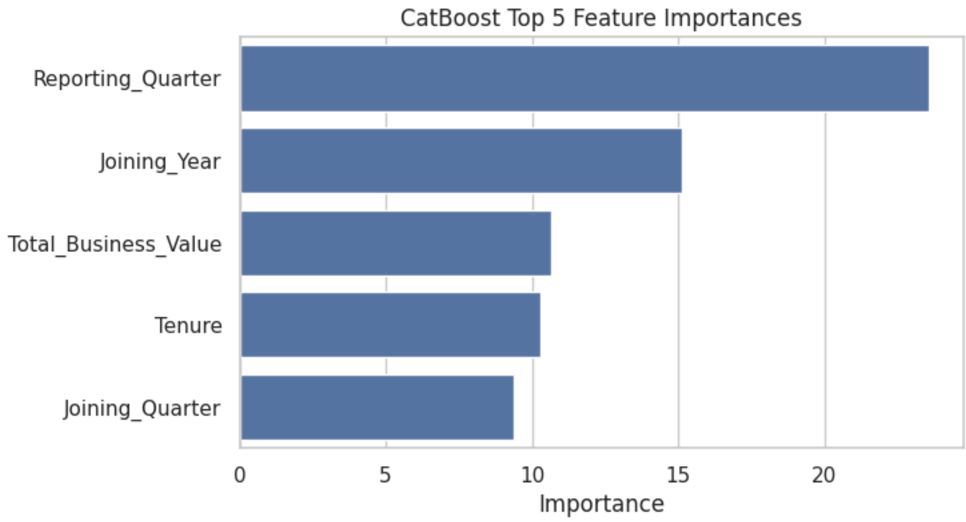

```
CatBoost Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.97      0.89       153
           1       0.99      0.90      0.94       324

    accuracy                           0.93       477
   macro avg       0.91      0.94      0.92       477
weighted avg       0.94      0.93      0.93       477

CatBoost AUC: 0.9699
```

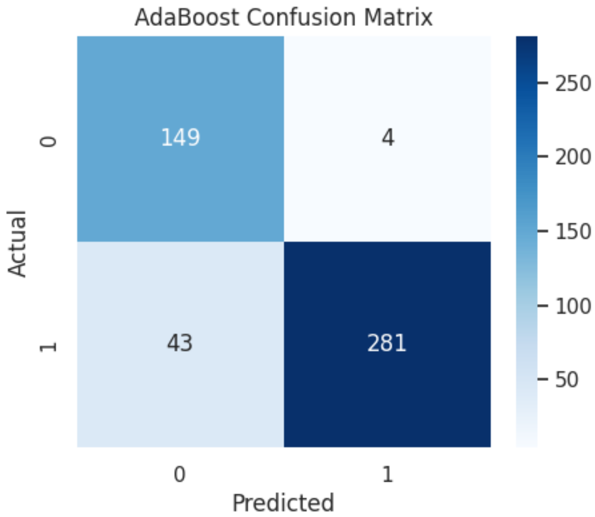




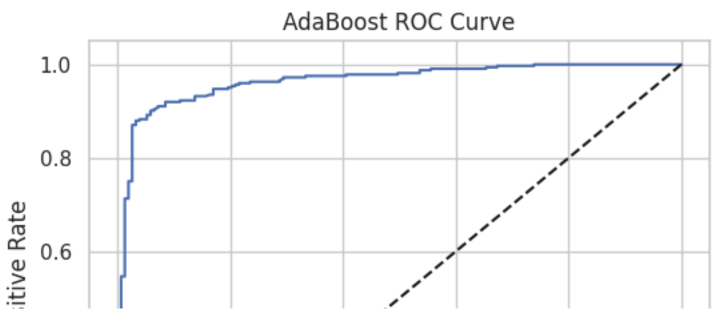


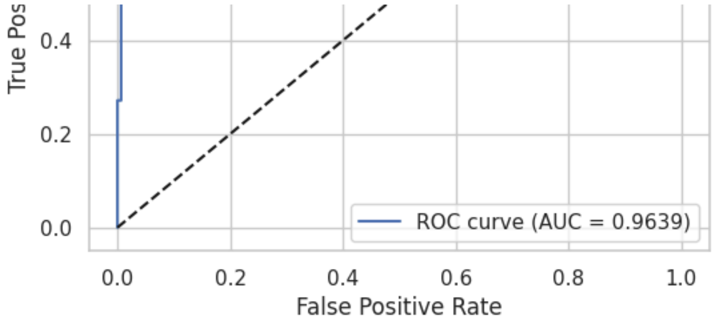

```
Top 5 Important Features for CatBoost:
             Feature  Importance
   Reporting_Quarter   23.598415
        Joining_Year   15.126110
Total_Business_Value   10.667372
              Tenure   10.298806
     Joining_Quarter    9.385868
```

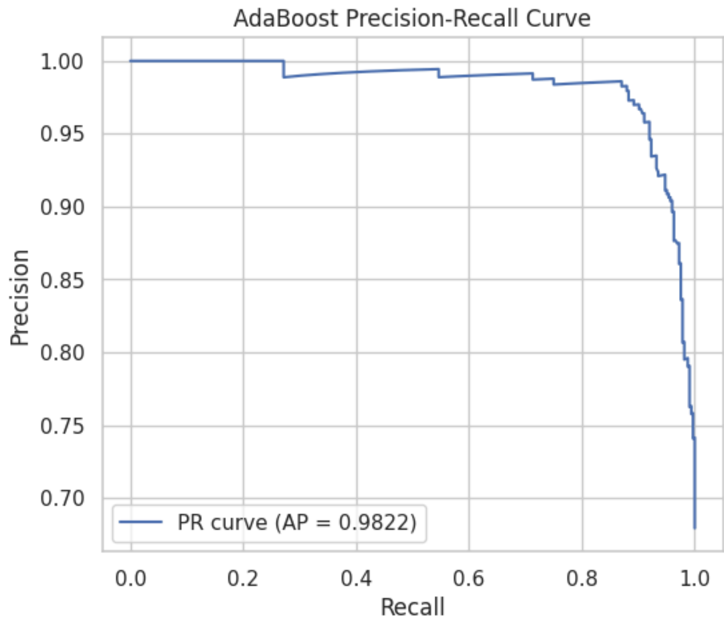




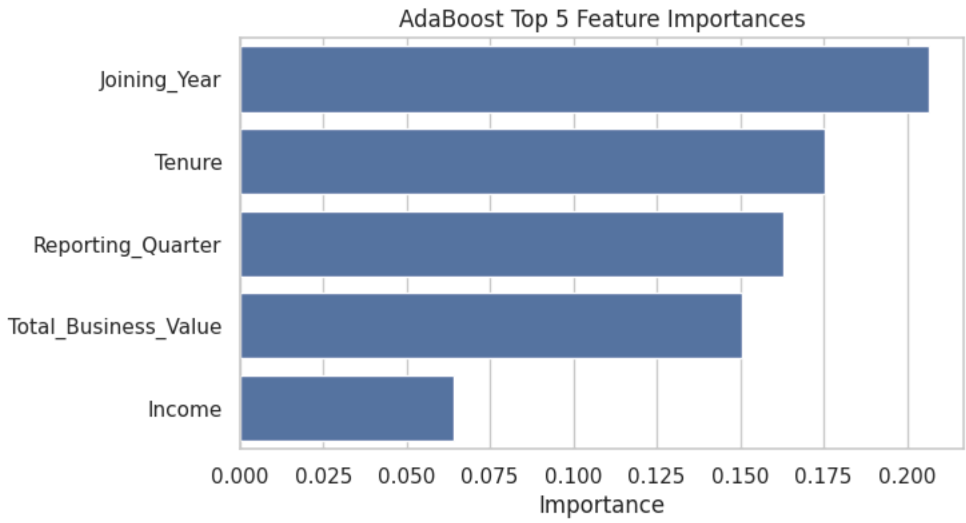

```
AdaBoost Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.97      0.86       153
           1       0.99      0.87      0.92       324

    accuracy                           0.90       477
   macro avg       0.88      0.92      0.89       477
weighted avg       0.92      0.90      0.90       477

AdaBoost AUC: 0.9639
```








```
Top 5 Important Features for AdaBoost:
            Feature  Importance
       Joining_Year    0.206675
             Tenure    0.175414
  Reporting_Quarter    0.162839
Total_Business_Value   0.150352
             Income    0.064260
```



Base Model Performance Comparison

Model	Accuracy	Precision (0)	Recall (0)	F1-score (0)	Precision (1)	Recall (1)	F1-score (1)	AUC
Random Forest	0.93	0.84	0.96	0.90	0.98	0.91	0.95	0.9714
XGBoost	0.92	0.84	0.95	0.89	0.97	0.91	0.94	0.9687
LightGBM	0.92	0.83	0.95	0.89	0.98	0.91	0.94	0.9699
CatBoost	0.93	0.83	0.97	0.89	0.99	0.90	0.94	0.9699
AdaBoost	0.90	0.78	0.97	0.86	0.99	0.87	0.92	0.9639

Top 5 Features by Model

Model	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
Random Forest	Reporting_Quarter	Joining_Year	Total_Business_Value	Tenure	Income
XGBoost	Reporting_Quarter	Joining_Year	QuarterlyRating_Increased	Total_Business_Value	Joining_Quarter
LightGBM	Tenure	Total_Business_Value	Income	City	Age
CatBoost	Reporting_Quarter	Joining_Year	Total_Business_Value	Tenure	Joining_Quarter
AdaBoost	Joining_Year	Tenure	Reporting_Quarter	Total_Business_Value	Income

Insights

- Random Forest and CatBoost show the highest accuracy (0.93) and strong balance between precision and recall for both classes.
- All models have high AUC scores above 0.96, indicating good discrimination ability.
- AdaBoost performs comparatively lower across most metrics, especially accuracy and precision for class 0.
- Key predictive features across models include `Reporting_Quarter`, `Joining_Year`, `Total_Business_Value`, and `Tenure`.
- LightGBM uniquely highlights `City` and `Age` among its top features.

Next Steps

- We will drop AdaBoost due to its comparatively weaker performance.
- Proceed with hyperparameter tuning on Random Forest, XGBoost, LightGBM, and CatBoost to optimize their predictive power.

```
rf = RandomForestClassifier(random_state=42)

param_grid_rf = {
    'n_estimators': [100, 200, 300, 400],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

rf_random = RandomizedSearchCV(
    estimator=rf, param_distributions=param_grid_rf,
    n_iter=20, scoring='f1', cv=3, verbose=2, random_state=42, n_jobs=-1
)

rf_random.fit(X_train_smote, y_train_smote)
print("Best RF params:", rf_random.best_params_)

best_rf = rf_random.best_estimator_
train_evaluate_model(best_rf, X_train_smote, y_train_smote, X_test, y_test, 'Random Forest (Tuned)')
```