!

INTRODUCTION

Problem Statement

Recruiting and retaining drivers is a significant challenge for ride-hailing companies, and Ola is no exception. As competition grows, the high churn rate among drivers has become a major concern for the company. Many drivers choose to leave on short notice, or they switch to competitors like Uber, which further compounds the problem. As Ola expands, this churn could escalate, leading to higher operational costs and diminishing morale among the existing workforce.

Ola's approach to addressing this issue includes targeting a wide pool of potential drivers, including those who don’t own vehicles, as a way to reduce the dependency on existing drivers. However, recruiting new drivers is expensive, and losing experienced drivers to churn only adds to the cost. Retaining drivers is not only crucial to maintaining a stable workforce but also significantly more cost-effective than constantly acquiring new ones.

Structure of the Case Study

This case study is divided into **three distinct parts**, each serving a different purpose in tackling the driver attrition problem at Ola.

Part 1: Insights and Recommendations

Part 1 is the result of the **findings and insights** obtained from the **exploration of the data** through the code. After performing data analysis, we summarize the key patterns, trends, and factors affecting driver churn. This section highlights the most important observations that will help in understanding the problem and guide the subsequent steps in addressing it.

Part 2: Non-Technical Details and Data Understanding

Part 2 is dedicated to providing a **non-technical understanding** of the dataset. Here, we break down the dataset, explain the structure of the data, the meaning of each feature, and discuss potential data quality issues. This section is crucial for anyone who needs to understand the data at a high level before diving into the technical aspects.

Part 3: Technical Code Implementation

Part 3 delves into the **"how"**—it is where we implement the solution using code. In this section, we will show how the findings from Part 1 and the data understanding from Part 2 were **yielded** using various data preprocessing, modeling, and evaluation techniques. It covers the technical steps taken to build the predictive model, including feature engineering, training, and evaluation.

PART 1

1. Income Growth and Churn

Insight:

- Only **1.8% of drivers** experience income growth, indicating a potential issue with stagnating earnings, which is likely contributing to the high churn rate.

Recommendation:

- Implement performance-based incentives or opportunities for income growth. Ensuring that drivers have avenues for increased earnings could significantly reduce churn.

---

2. Rating Growth and Churn

Insight:

- Only **15% of drivers** show a growth in their quarterly ratings, and there is a **negative correlation between churn and latest quarterly ratings**, suggesting that drivers with lower quarterly ratings are more likely to churn.

Recommendation:

- Focus on improving drivers' ratings through regular feedback, training, and performance evaluations. This could help retain drivers, especially those with low ratings.

---

3. Total Business Value and Churn

Insight:

- **Total business value** is **highly left-skewed**, meaning that a small number of drivers generate disproportionately higher business value, while the majority are underperforming.

Recommendation:

- Consider identifying high-performing drivers and applying their strategies to the broader driver pool. Training low-performing drivers or incentivizing them to improve could help boost overall business value and reduce churn.

---

4. Churn by Day and Month

Insight:

- **Churn is most prevalent on the 29th, 28th, and 27th of each month**, with a lower churn on the **3rd, 26th, and 31st**.
- **Churn months** are predominantly **July**, **May**, and **October**, with the least churn observed in **April**, **February**, and **March**.

Recommendation:

- Investigate the operational or environmental factors causing higher churn on these specific days and months. Consider launching retention strategies around these times, such as promotions or personalized communications to reduce churn.

---

5. Churn and Education Level

Insight:

- There is no significant correlation between **education level** and churn. The distribution of drivers across education levels (10+, 12+, Graduate) remains consistent despite churn patterns.

Recommendation:

- Education level should be deprioritized as a feature for churn prediction. Instead, focus on performance indicators such as ratings, income, and business value for more impactful retention strategies.

---

6. Correlation Between Ratings and Income

Insight:

- **First income**, **last income**, and **average income** all show a **perfect positive correlation** (1.0), suggesting that income growth is consistent across the data set.

Recommendation:

- If the income distribution is skewed and stagnant, consider offering additional earning opportunities or bonuses to diversify the income streams for drivers. This could increase driver engagement and reduce churn.

---

7. Churn by Year (2019 vs. 2020)

Insight:

- There were **slightly fewer drivers who churned in 2020** compared to 2019, indicating that there may have been changes in operational strategies, or external factors that impacted churn in 2020.

Recommendation:

- Investigate the operational or external factors that influenced this reduction in churn in 2020. Applying similar strategies to the 2021 dataset could help lower churn further.

---

8. Joining Designation and Grade Distribution

Insight:

- A large percentage of drivers join with **designations 1 and 2** (43% and 34%), with very few receiving higher designations or grades (Grade 5 is given to only 0.88% of drivers).

Recommendation:

- Consider creating clear career progression opportunities and incentive programs for drivers to achieve higher designations and grades. This could improve overall retention and employee satisfaction.

---

9. Quarterly Rating Growth Indicator

Insight:

- Only **15% of drivers** show growth in their quarterly ratings. This low rate suggests that most drivers experience stagnant performance metrics over time, contributing to dissatisfaction.

Recommendation:

- Offer structured programs that help drivers improve their ratings over time. These programs could include performance reviews, rewards for improvement, and personalized coaching to support career progression.

---

10. Gender Distribution and Churn

Insight:

- **Churn rates are very similar between male and female drivers**, with both groups showing **68% churn** and **32% retention**.

Recommendation:

- Gender does not seem to be a differentiator for churn, so retention efforts should focus on other factors such as performance and ratings rather than gender-based segmentation.

These insights prioritize factors most directly related to churn, income, and ratings, which are crucial for formulating a successful retention strategy.

PART 2- NON-TECHNICAL DATA UNDERSTANDING

Dataset

The dataset provided for this task is titled **"ola_driver.csv"** and contains information about Ola's drivers over the course of 2019 and 2020. The dataset includes attributes related to both the personal and professional aspects of the drivers' careers with Ola.

Column Profiling:

- **MMMM-YY:** Reporting Date (Monthly)
- **Driver_ID:** Unique ID assigned to each driver
- **Age:** Age of the driver
- **Gender:** Gender of the driver (0: Male, 1: Female)
- **City:** City Code of the driver
- **Education_Level:** Education level (0: 10+, 1: 12+, 2: Graduate)
- **Income:** Monthly average income of the driver
- **Date Of Joining:** Joining date of the driver
- **LastWorkingDate:** Last date the driver worked for the company
- **Joining Designation:** Designation of the driver at the time of joining
- **Grade:** Grade of the driver at the time of reporting
- **Total Business Value:** Total business value acquired by the driver in a month (negative values indicate cancellations, refunds, or car EMI adjustments)
- **Quarterly Rating:** Quarterly performance rating of the driver (1–5, where 5 is the best)

This data serves as the foundation for building a predictive model that can help Ola predict which drivers are likely to churn, allowing the company to take proactive measures to retain valuable drivers and improve overall performance.

Data Overview

The dataset represents a collection of **driver-level data** for **2381 drivers** spanning across **29 unique cities** in the period of **2019-2020**. It tracks key metrics for each driver, including demographic details, performance ratings, income, and churn status. The data was collected in a **granular format**, meaning that there were **multiple records for each driver**, with each record representing a snapshot of that driver at a particular point in time.

The primary challenge in the analysis was to transform this **granular data** into a more usable format at the **driver level**, where each driver would have a **single aggregated record** that represents their behavior and performance over time. This transformation is crucial for the predictive modeling of churn.

Key features in the data include:

1. **Demographic Information**:
    - **Age** and **Gender**: Basic personal details to profile the drivers.
    - **Education Level**: Categorical feature indicating the education level of the driver (e.g., 10+, 12+, Graduate).
2. **Performance Metrics**:
    - **Income**: Recorded at multiple time intervals, showing the driver's earnings.
    - **Business Value**: The total business value generated by the driver.
    - **Quarterly Ratings**: Performance ratings given to drivers at regular intervals.
3. **Churn Data**:
    - **Last Working Date**: This column indicates the date a driver stopped working, used to create a **churn indicator** (1 for churned, 0 for still active).
4. **Categorical Features**:
    - **Joining Designation**: The designation given to drivers at the time of joining (e.g., 1, 2, 3, etc.).
    - **Grade**: A feature representing the grade assigned to the driver during their employment.

Given the multiple records per driver, the goal was to **aggregate** the data at a **driver-level** for effective churn prediction. This means consolidating the multiple observations per driver into a single representative record that captures key statistics such as income, ratings, churn status, and other performance measures.

In summary, the dataset required preprocessing to handle **missing values**, **data imbalances**, and **data aggregation**, which are all essential steps before moving into further stages of analysis and model building.

Dealing with Null Values

During the data preparation phase, I encountered null values in three key features: **Age**, **Gender**, and **LastWorkingDate**.

- **LastWorkingDate**: This feature had a significant proportion of null values (around 91%). These nulls occurred because the data is **granular**, with multiple records for each driver. If a driver had not churned, their **LastWorkingDate** would remain null in all records. Since the **LastWorkingDate** is a key indicator of churn, I utilized this column later to create a new **churned indicator** (1 for churned, 0 for still active). This means that **LastWorkingDate** did not require imputation but instead was transformed into a churn-related feature post aggregation.

- **Age and Gender**: Both features had a very small amount of missing data—only 102 drivers had missing values for **Age** or **Gender**. Given the **granular nature** of the data, I wanted to explore the pattern of missing data. I investigated whether missing values were distributed across a driver's multiple records or if they were entirely missing for that driver. After analyzing the pattern, I found that

100% of the missing data were **false nulls**—in other words, **Age** and **Gender** were missing for only some records of drivers, not all records.

Given this, I needed to determine the best way to impute the missing values.

Imputation Strategy

For both the columns, I adopted a **logical imputation method** rather than using a **KNN imputer**. The reasoning behind this decision is as follows:

- **Age Imputation**: The assumption was that the **last recorded age** for a driver would be the most representative value for their other records. Since **Age** doesn't typically change drastically over short periods, I felt it was reasonable to use the **last recorded age** for imputation. This approach seemed more appropriate considering the available data size and the nature of the problem.

- **Gender Imputation**: Since **Gender** does not change over time, it was straightforward to fill in missing gender values with the most recent valid value for each driver.

To compare these methods with a more **automated KNN imputation** approach, I implemented both. For **KNN imputation**, I set the number of neighbors (k) to 5, based on the assumption that drivers with similar records would likely share similar **Age** and **Gender** values. Here are the results of this comparison:

- **Age Imputation**:
  - Out of 102 missing age values, **KNN imputation** correctly predicted 59 ages exactly. For the rest, the **range of difference** between my logical imputation and KNN was between **0 to 8.71 years**.
- **Gender Imputation**:
  - For **Gender**, **KNN imputation** performed exceptionally well, correctly predicting **101 out of 102** missing gender values. The only mistake was one misclassified gender, showing the reliability of KNN in this case.

Decision on Imputation Strategy

Given the smaller dataset, I opted to continue with **logical imputation** for **Age** and **Gender** for the following reasons:

1. **Simplicity**: The logical approach is straightforward and computationally cheaper than KNN imputation, which can be more resource-intensive.
2. **Accuracy**: In this case, **logical imputation** performed well and was more efficient, particularly for **Age**, where the difference between KNN and my logic was significant enough to justify the usage of logical imputation over algorithmic based imputation.
3. **Computation**: KNN has to do lot of computation and given a larger dataset in future, it would be computationally very expensive.

Data Aggregation

Aggregation is a crucial step in the data preparation phase, especially when dealing with **granular data**. In this project, I had to transform the data into a more **driver-level** format to ensure meaningful insights and analysis.

Why Aggregation?

Since the dataset contained multiple records for the same driver (due to the granularity of the data), a key task was to **aggregate these records** into a single, summarized entry per driver. The objective was to focus on each driver's overall behavior rather than treating individual records as separate entities. This would allow me to:

1. **Understand each driver's performance** across their entire lifecycle.
2. **Simplify analysis**, making it easier to identify patterns and trends.
3. Ensure that the data is aligned with the business goal, which is to understand and predict **driver churn**.

Resulting Data Structure

The final structure after aggregation was a **driver-level dataset**, where each row represented a unique driver, and the columns summarized the key performance and behavioral metrics across all their individual records. This structure was ideal for further analysis, including **churn prediction** and performance evaluation.

Impact of Aggregation on Analysis

- The aggregation allowed me to **focus on individual driver-level insights**, rather than getting lost in the granular details of individual records.
- It significantly **reduced the complexity** of the dataset, which made it easier to spot patterns in **driver behavior**, such as income fluctuations, churn likelihood, and rating trends.

In summary, the **aggregation** of the dataset was essential for transforming the raw, granular records into a format that could provide actionable insights. It not only streamlined the analysis but also allowed me to focus on high-level trends and patterns that would drive decisions related to **driver retention**, **performance optimization**, and **churn prevention**.

Modelling

- **Imbalanced Data Handling**: Initially, I used the **Balanced Random Forest Classifier**, which was robust to imbalanced data. It provided **76% accuracy** on the test set but had a higher rate of **false positives**.

- **XGBoost Classifier**: After testing multiple models, **XGBoost** performed the best, achieving **81% testing accuracy** without the need for SMOTE. It showed a strong performance but also had a higher **false positive** rate.

- **SMOTE**: I experimented with SMOTE (Synthetic Minority Over-sampling Technique) for upsampling the minority class (churned drivers). However, it led to overfitting and decreased the model's performance.

- **Final Model**: The **XGBoost Classifier** with imbalanced data was selected as the best-performing model, and further fine-tuning is planned to reduce false positives and improve overall predictions.

PART 3 - TECHNICAL CHAMBER

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import datetime
pd.set_option('display.max_columns',None)
warnings.filterwarnings(action='ignore')
%matplotlib inline
from sklearn.impute import KNNImputer
from sklearn.feature_selection import mutual_info_classif
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split, cross_validate, GridSearchCV
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score, roc_cu
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

In [2]:

```
df = pd.read_csv('ola_driver_data.csv')
df.head()
```

Out[2]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWo
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	
2	2	03/01/19	1	28.0	0.0	C23	2	57387	24/12/18	
3	3	11/01/20	2	31.0	0.0	C7	2	67016	11/06/20	
4	4	12/01/20	2	31.0	0.0	C7	2	67016	11/06/20	

In [3]:

```
#the shape of data post immediate import/raw untouched data
rows,col = df.shape
print(f'The original imported data had {rows} records along with {col} features')
```

```
The original imported data had 19104 records along with 14 features
```

In [4]:

```
#dropping irrelevant column
df.drop('Unnamed: 0',axis=1,inplace=True)
```

In [5]:

```
df.dtypes
```

Out[5]:

```
MMM-YY                    object
Driver_ID                  int64
Age                      float64
Gender                   float64
City                      object
Education_Level            int64
Income                     int64
Dateofjoining             object
LastWorkingDate           object
Joining Designation        int64
Grade                      int64
Total Business Value       int64
Quarterly Rating           int64
dtype: object
```

In [6]:

```
# handling date features to format into date
date_features = ['MMM-YY','Dateofjoining','LastWorkingDate']
# used stack and unstack operation, stacking would convert all into a single series and
df[date_features] = pd.to_datetime(df[date_features].stack(),errors='coerce').unstack()

# leaving the conversion of following columns to be taken as int only as for all the col
df['Gender'] = df['Gender'].astype('category')
# category_features = ['Gender','Education_Level','Joining Designation','Grade','Quarter
# df[category_features] = df[category_features].astype('category',errors='ignore')
```

In [7]:

```
#null check
df.isna().sum()
```

Out[7]:

```
MMM-YY                     0
Driver_ID                  0
Age                       61
Gender                    52
City                       0
Education_Level            0
Income                     0
Dateofjoining              0
LastWorkingDate        17488
Joining Designation        0
Grade                      0
Total Business Value       0
Quarterly Rating           0
dtype: int64
```

In [8]:

```
# as the data is such that for each driver there are multiple records which means multip
# in their few readings the age or gender is not recorded. Before imputation i want to c
# the age or gender is not capture. so i will check for that and check the ratio for whi

missing_driver_info_ids = list(set(df[df[['Age', 'Gender']].isna().any(axis=1)]['Driver_
total_missing_driver_info_count = len(missing_driver_info_ids)
print(f'There is missing driver info in Age or Gender feature for {total_missing_driver_

missing_df = df[df['Driver_ID'].isin(missing_driver_info_ids)][['Driver_ID','Age','Gende

missing_df['Age_Null'] = missing_df['Age'].apply(lambda x:1 if pd.isna(x) else 0)
```

```
missing_df['Gender_Null'] = missing_df['Gender'].apply(lambda x:1 if pd.isna(x) else 0)

missing_df = missing_df.groupby('Driver_ID').aggregate(Total_Record_Count = ('Driver_ID'
                                                       Gender=('Gender','first'),Total_A
                                                       Total_Gender_Null = ('Gender_Null

missing_df['False_Null'] = (
    (missing_df['Total_Record_Count'] >= missing_df['Total_Age_Null']) &
    (missing_df['Total_Record_Count'] >= missing_df['Total_Gender_Null'])).astype(int)

missing_df['False_Null'].value_counts()
```

```
There is missing driver info in Age or Gender feature for 102 drivers
```

Out[8]:

```
False_Null
1    102
Name: count, dtype: int64
```

In [9]:

```
missing_df[['Total_Record_Count']].value_counts()
```

Out[9]:

```
Total_Record_Count
24                    17
5                     15
4                      9
9                      8
6                      8
7                      6
10                     5
2                      5
8                      4
11                     4
13                     4
3                      3
14                     3
18                     3
16                     2
12                     2
15                     1
17                     1
20                     1
22                     1
Name: count, dtype: int64
```

In [10]:

```
knn_input_data = df.drop(date_features + ['Driver_ID','City'], axis=1)
scaler = MinMaxScaler()
knn_input_scaled = pd.DataFrame(scaler.fit_transform(knn_input_data),columns=knn_input_d
print('Scaled Input data for KNN Imputation')
knn_input_scaled
```

```
Scaled Input data for KNN Imputation
```

Out[10]:

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating
0	0.189189	0.0	1.0	0.262508	0.00	0.00	0.210856	0.333333
1	0.189189	0.0	1.0	0.262508	0.00	0.00	0.134209	0.333333

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating
2	0.189189	0.0	1.0	0.262508	0.00	0.00	0.150952	0.333333
3	0.270270	0.0	1.0	0.316703	0.25	0.25	0.150952	0.000000
4	0.270270	0.0	1.0	0.316703	0.25	0.25	0.150952	0.000000
...	...	...	...	...	...	...	...	...
19099	0.243243	0.0	1.0	0.334928	0.25	0.25	0.169577	0.666667
19100	0.243243	0.0	1.0	0.334928	0.25	0.25	0.162232	0.666667
19101	0.243243	0.0	1.0	0.334928	0.25	0.25	0.150952	0.333333
19102	0.243243	0.0	1.0	0.334928	0.25	0.25	0.155994	0.333333
19103	0.243243	0.0	1.0	0.334928	0.25	0.25	0.161304	0.333333

19104 rows × 8 columns

In [11]:

```
knn_imputer = KNNImputer(n_neighbors=7)
knn_imputed_scaled = knn_imputer.fit_transform(knn_input_scaled)
knn_imputed_df = pd.DataFrame(scaler.inverse_transform(knn_imputed_scaled),columns=knn_i
knn_imputed_df['Driver_ID'] = df['Driver_ID']
print('Transformed & Inverse Scaled Data output post KNN Imputation')
knn_imputed_df
```

```
Transformed & Inverse Scaled Data output post KNN Imputation
```

Out[11]:

	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value	Quarterly Rating	Driver_ID
0	28.0	0.0	2.0	57387.0	1.0	1.0	2381060.0	2.0	1
1	28.0	0.0	2.0	57387.0	1.0	1.0	-665480.0	2.0	1
2	28.0	0.0	2.0	57387.0	1.0	1.0	0.0	2.0	1
3	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2
4	31.0	0.0	2.0	67016.0	2.0	2.0	0.0	1.0	2
...	...	...	...	...	...	...	...	...	...
19099	30.0	0.0	2.0	70254.0	2.0	2.0	740280.0	3.0	2788
19100	30.0	0.0	2.0	70254.0	2.0	2.0	448370.0	3.0	2788
19101	30.0	0.0	2.0	70254.0	2.0	2.0	0.0	2.0	2788
19102	30.0	0.0	2.0	70254.0	2.0	2.0	200420.0	2.0	2788
19103	30.0	0.0	2.0	70254.0	2.0	2.0	411480.0	2.0	2788

19104 rows × 9 columns

In [12]:

```
# checking for accuracy for which technique performed the best
logic_imputation_results = missing_df[['Total_Record_Count','Age','Gender']].reset_index
logic_imputation_results.columns = ['Driver_ID','Total_Record_Count','Age_logical','Gend

knn_imputation_filtered_df = knn_imputed_df[knn_imputed_df['Driver_ID'].isin(missing_dri
knn_imputation_results = knn_imputation_filtered_df.groupby('Driver_ID').aggregate(Age_k

logic_imputation_results.sort_values(by='Driver_ID',ascending=True,inplace=True)
knn_imputation_results.sort_values(by='Driver_ID',ascending=True,inplace=True)

imputation_comparision_df = pd.concat([logic_imputation_results,knn_imputation_results],
imputation_comparision_df['Age_diff'] = round(abs(imputation_comparision_df['Age_logical
imputation_comparision_df['Gender_knn'] = round(imputation_comparision_df['Gender_knn'])
imputation_comparision_df['Gender_knn'] = imputation_comparision_df['Gender_knn'].astype
imputation_comparision_df['Gender_diff'] = imputation_comparision_df['Gender_logical'] =
```

In [13]:

```
imputation_comparision_df['Age_diff'].value_counts()
```

Out[13]:

```
Age_diff
0.00    59
4.71     3
3.43     2
1.14     2
0.57     2
0.29     2
1.29     2
4.57     2
0.86     2
3.57     1
1.43     1
4.00     1
5.14     1
8.29     1
2.86     1
2.43     1
4.29     1
0.14     1
3.71     1
2.57     1
1.57     1
6.29     1
8.43     1
0.43     1
6.86     1
5.43     1
2.29     1
6.71     1
4.86     1
2.00     1
3.14     1
5.86     1
3.86     1
5.57     1
8.71     1
Name: count, dtype: int64
```

In [14]:

```
imputation_comparision_df['Gender_diff'].value_counts()
```

Out[14]:

```
Gender_diff
True     101
False      1
Name: count, dtype: int64
```

In [15]:

```
age_dict = missing_df['Age'].to_dict()
gender_dict = missing_df['Gender'].to_dict()

def imputer(row):
    if pd.isna(row['Age']):
        row['Age'] = age_dict.get(row['Driver_ID'],row['Age'])
    if pd.isna(row['Gender']):
        row['Gender'] = gender_dict.get(row['Driver_ID'],row['Gender'])
    return row

df = df.apply(imputer,axis=1)
print('Post Imputation null counts - ')
pd.DataFrame(df.isna().sum()).T
```

```
Post Imputation null counts -
```

Out[15]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDate	Des
0	0	0	0	0	0	0	0	0	17488	

In [16]:

```
driver_df = df.groupby(['Driver_ID'],sort={'MMM-YY':'asc'}).aggregate({
    'Age':'last',
    'Gender':'first',
    'City':['first','nunique'],
    'Education_Level':'last',
    'Income':['mean','first','last'],
    'Dateofjoining':'first',
    'LastWorkingDate':'last',
    'Joining Designation':'first',
    'Grade':'first',
    'Total Business Value':'sum',
    'Quarterly Rating':['first','last']})

driver_df.sample(3)
```

Out[16]:

	Age	Gender	City		Education_Level	Income			Dateofjoining	LastWo
	last	first	first	nunique	last	mean	first	last	first	
Driver_ID										
2097	36.0	0.0	C25	1	0	70496.0	70496	70496	2020-07-31	
384	26.0	1.0	C27	1	1	52364.0	52364	52364	2018-10-21	2

	Age	Gender	City		Education_Level	Income			Dateofjoining	LastWo
	last	first	first	nunique	last	mean	first	last	first	
Driver_ID										
1217	38.0	0.0	C5	1	2	48409.0	48409	48409	2019-08-01	2

In [17]:

```
driver_df.columns = driver_df.columns.to_flat_index()
driver_df.columns = ['_'.join(col).strip() for col in driver_df.columns]
driver_df.sample(3)
```

Out[17]:

	Age_last	Gender_first	City_first	City_nunique	Education_Level_last	Income_mean	Income_
Driver_ID							
2672	37.0	1.0	C18	1	2	56621.0	5
1800	32.0	0.0	C3	1	1	35542.0	3
377	35.0	0.0	C5	1	0	121332.0	12

In [18]:

```
# new feature creation
driver_df['churned'] = driver_df['LastWorkingDate_last'].apply(lambda x:0 if pd.isna(x)

def growth_indicator(row):
    if row['Quarterly Rating_last'] > row['Quarterly Rating_first']:
        row['quarterly_rating_growth_indicator'] = 1
    else:
        row['quarterly_rating_growth_indicator'] = 0

    if row['Income_last'] > row['Income_first']:
        row['income_growth_indicator'] = 1
    else:
        row['income_growth_indicator'] = 0

    return row

driver_df = driver_df.apply(growth_indicator,axis=1)
driver_df.columns = ['age','gender','city','unique_cities','education_level','avg_income
driver_df
```

Out[18]:

	age	gender	city	unique_cities	education_level	avg_income	first_income	last_income	da
Driver_ID									
1	28.0	0.0	C23	1	2	57387.0	57387	57387	
2	31.0	0.0	C7	1	2	67016.0	67016	67016	
4	43.0	0.0	C13	1	2	65603.0	65603	65603	
5	29.0	0.0	C9	1	0	46368.0	46368	46368	

Driver_ID	age	gender	city	unique_cities	education_level	avg_income	first_income	last_income	da
6	31.0	1.0	C11	1	1	78728.0	78728	78728	
...	...	...	...	...	...	...	...	...	
2784	34.0	0.0	C24	1	0	82815.0	82815	82815	
2785	34.0	1.0	C9	1	0	12105.0	12105	12105	
2786	45.0	0.0	C19	1	0	35370.0	35370	35370	
2787	28.0	1.0	C20	1	2	69498.0	69498	69498	
2788	30.0	0.0	C27	1	2	70254.0	70254	70254	

2381 rows × 18 columns

In [19]:

```
driver_df['date_ofjoining'] = driver_df['dateofjoining'].dt.day
driver_df['month_ofjoining'] = driver_df['dateofjoining'].dt.month
driver_df['year_ofjoining'] = driver_df['dateofjoining'].dt.year

driver_df['lastworking_date'] = driver_df['lastworkingdate'].dt.day
driver_df['lastworking_month'] = driver_df['lastworkingdate'].dt.month
driver_df['lastworking_year'] = driver_df['lastworkingdate'].dt.year
```

In [20]:

```
rows, cols = driver_df.shape
print(f'🚕 After preparing the data at a granular driver level, we now have information
```

```
🚕 After preparing the data at a granular driver level, we now have information for 2381
drivers.
```

In [21]:

```
print(driver_df['gender'].value_counts(normalize=True)*100)
print('\n🚕It can be seen from that data that ~60% of drivers are Male(0.0) and remainin
```

```
gender
0.0    58.966821
1.0    41.033179
Name: proportion, dtype: float64

🚕It can be seen from that data that ~60% of drivers are Male(0.0) and remaining ~40% ar
e Female(1.0)
```

In [22]:

```
print(driver_df['churned'].value_counts(normalize=True) * 100)
print('🚕Out of all the drivers data we have, around 68% of drivers are churned and otht
print('\n')
print(driver_df[driver_df['gender']==1.0]['churned'].value_counts(normalize=True) * 100)
print('🚕Out of all Female Drivers, 68% drivers have churned')
print('\n')
print(driver_df[driver_df['gender']==0.0]['churned'].value_counts(normalize=True) * 100)
print('🚕Out of all Male Drivers, 68% drivers have churned')
```

```
churned
1    67.870643
0    32.129357
Name: proportion, dtype: float64
```

🚕Out of all the drivers data we have, around 68% of drivers are churned and othter 32% are not churned

```
churned
1    68.372569
0    31.627431
Name: proportion, dtype: float64
```

🚕Out of all Female Drivers, 68% drivers have churned

```
churned
1    67.521368
0    32.478632
Name: proportion, dtype: float64
```

🚕Out of all Male Drivers, 68% drivers have churned

In [23]:

```
print(f'🚕The data covers driver information across {driver_df['city'].nunique()} unique
pd.DataFrame(driver_df['city'].value_counts(normalize=True)*100).T
```

🚕The data covers driver information across 29 unique cities

Out[23]:

city	C20	C15	C29	C26	C8	C27	C10	C16	C22
proportion	6.383872	4.241915	4.031919	3.905922	3.737925	3.737925	3.611928	3.527929	3.443931

In [24]:

```
unique_cities = driver_df['unique_cities'].max()

if unique_cities == 1:
    print('🚕All drivers have consistently operated in only one city throughout the data
    driver_df.drop('unique_cities',axis=1,inplace=True)
```

🚕All drivers have consistently operated in only one city throughout the data collection process, without changing their city of operation.

In [25]:

```
categorical_features = ['education_level','joining_designation','grade','first_quarter_r
for col in categorical_features:
    print(f"Value counts for {col}:\n")
    print(round(driver_df[col].value_counts(normalize=True)*100,2))
    print("\n" + "-" * 50 + "\n")
```

```
Value counts for education_level:

education_level
2    33.68
1    33.39
0    32.93
Name: proportion, dtype: float64

--------------------------------------------------

Value counts for joining_designation:

joining_designation
1    43.09
```

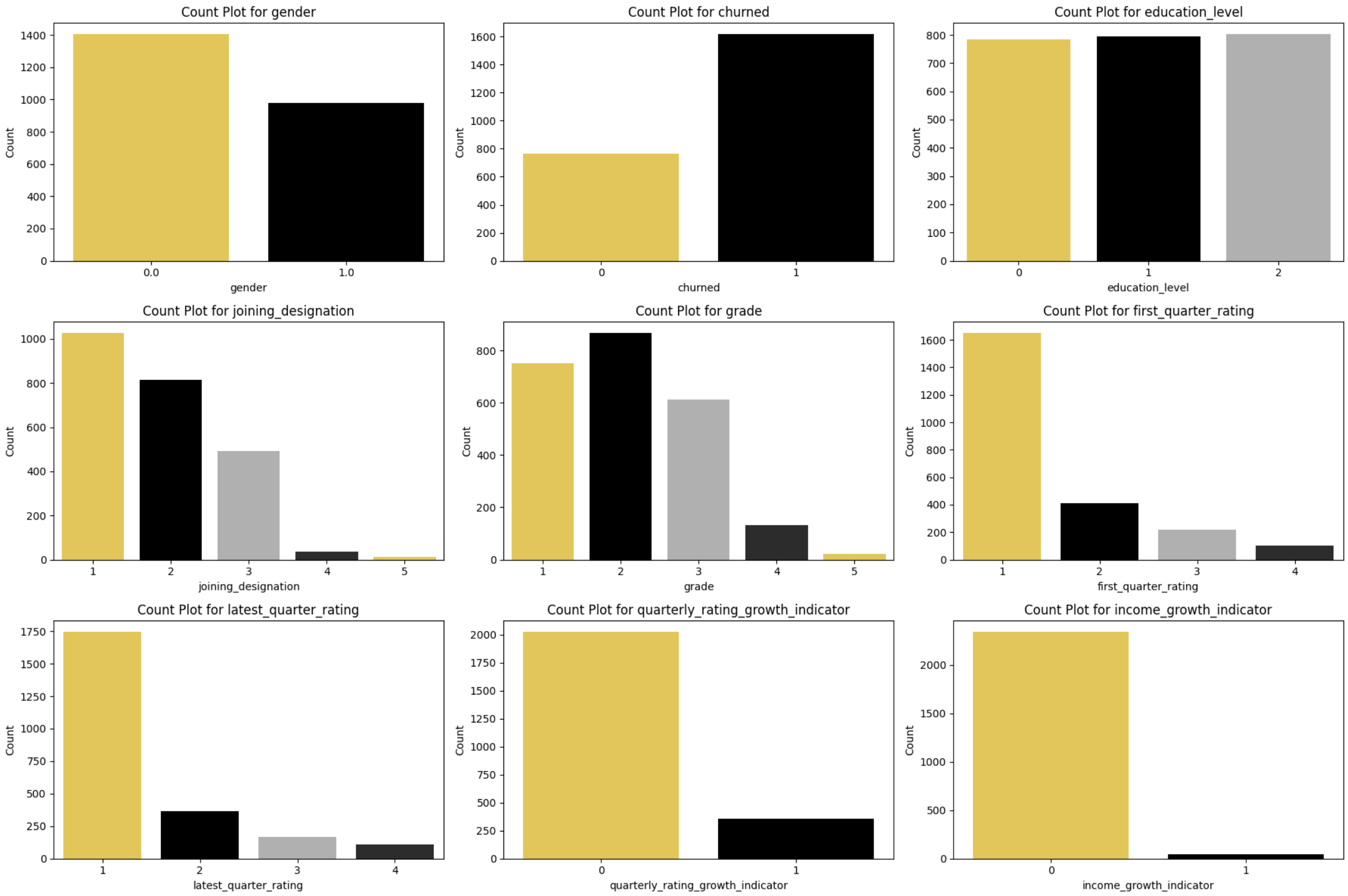
```
2    34.23
3    20.71
4     1.51
5     0.46
Name: proportion, dtype: float64

--------------------------------------------------

Value counts for grade:

grade
2    36.37
1    31.54
3    25.66
4     5.54
5     0.88
Name: proportion, dtype: float64

--------------------------------------------------

Value counts for first_quarter_rating:

first_quarter_rating
1    69.26
2    17.26
3     9.07
4     4.41
Name: proportion, dtype: float64

--------------------------------------------------

Value counts for latest_quarter_rating:

latest_quarter_rating
1    73.25
2    15.20
3     7.06
4     4.49
Name: proportion, dtype: float64

--------------------------------------------------

Value counts for quarterly_rating_growth_indicator:

quarterly_rating_growth_indicator
0    84.96
1    15.04
Name: proportion, dtype: float64

--------------------------------------------------

Value counts for income_growth_indicator:

income_growth_indicator
0    98.19
1     1.81
Name: proportion, dtype: float64
```

In [26]:

```python
ola_palette = sns.color_palette(["#F9D342", "#000000", "#B0B0B0", "#2C2C2C"])
```

In [27]:

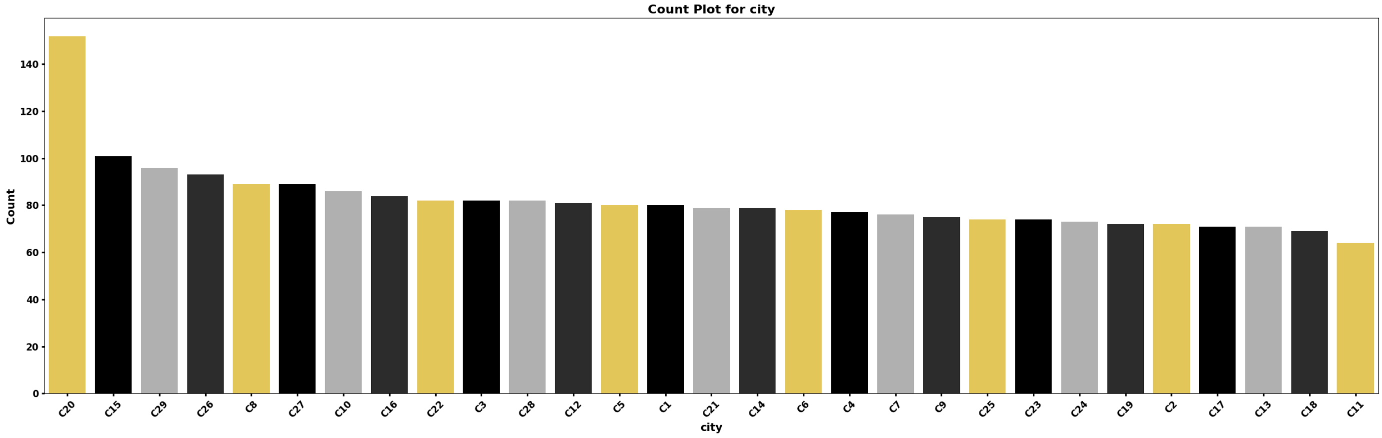
```python
categorical_features = [
    'gender', 'churned', 'education_level', 'joining_designation',
    'grade', 'first_quarter_rating', 'latest_quarter_rating',
    'quarterly_rating_growth_indicator', 'income_growth_indicator'
]

fig, axes = plt.subplots(3, 3, figsize=(18, 12))
axes = axes.flatten()

for i, col in enumerate(categorical_features):
    sns.countplot(data=driver_df, x=col, ax=axes[i], palette=ola_palette)
    axes[i].set_title(f"Count Plot for {col}", fontsize=12)
    axes[i].set_xlabel(col, fontsize=10)
    axes[i].set_ylabel("Count", fontsize=10)
    # axes[i].tick_params(axis='x', rotation=45)

for j in range(len(categorical_features), 9):
    fig.delaxes(axes[j])

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



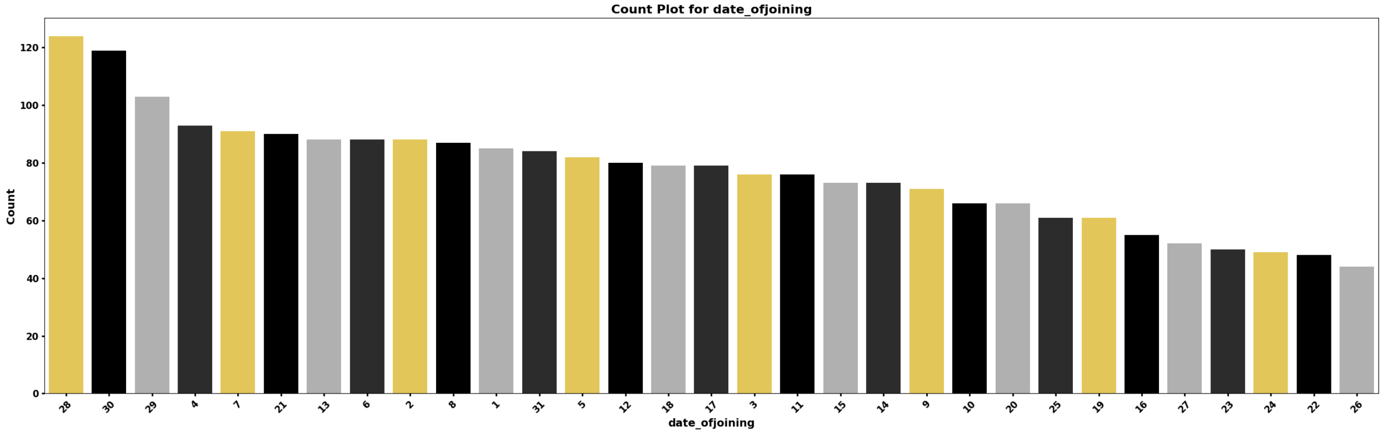
In [28]:

```
features_list = [
    'city',
    'date_ofjoining', 'lastworking_date',
    'month_ofjoining', 'lastworking_month',
    'year_ofjoining', 'lastworking_year']

fig, axes = plt.subplots(len(features_list), 1, figsize=(25, 8 * len(features_list)))
axes = axes.flatten()

for i, col in enumerate(features_list):
    sns.countplot(
        data=driver_df,
        x=col,
        ax=axes[i],
        order=driver_df[col].value_counts().index,
        palette=ola_palette
    )

    axes[i].set_title(f"Count Plot for {col}", fontsize=16, fontweight='bold')
    axes[i].set_xlabel(col, fontsize=14, fontweight='bold')
    axes[i].set_ylabel("Count", fontsize=14, fontweight='bold')

    axes[i].tick_params(axis='x', labelsize=12, labelrotation=45, width=2)  # Rotate x-a
    axes[i].tick_params(axis='y', labelsize=12, width=2)

    for tick in axes[i].get_xticklabels():
        tick.set_fontweight('bold')
    for tick in axes[i].get_yticklabels():
        tick.set_fontweight('bold')

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

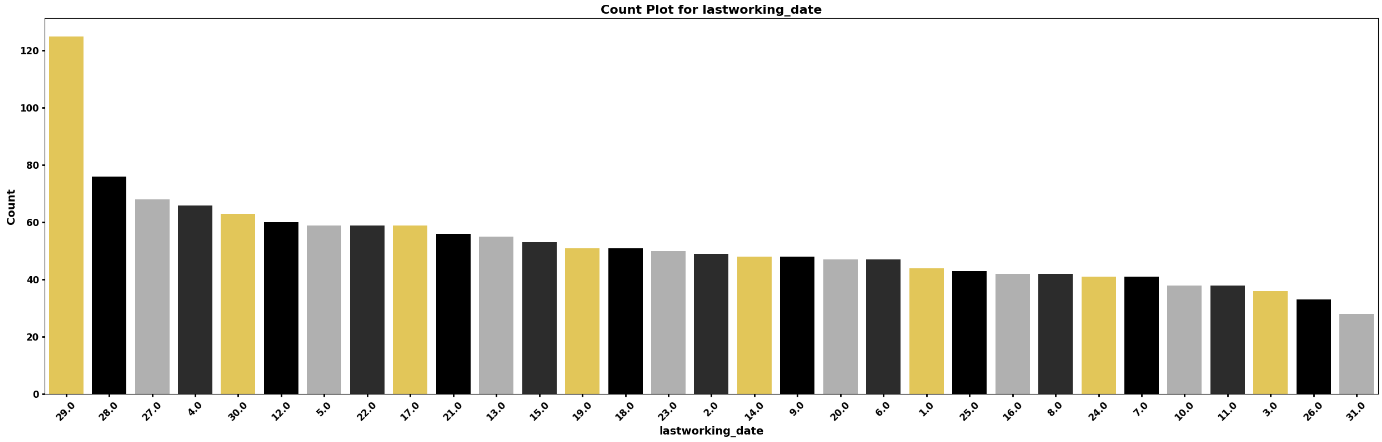
Count Plot for city



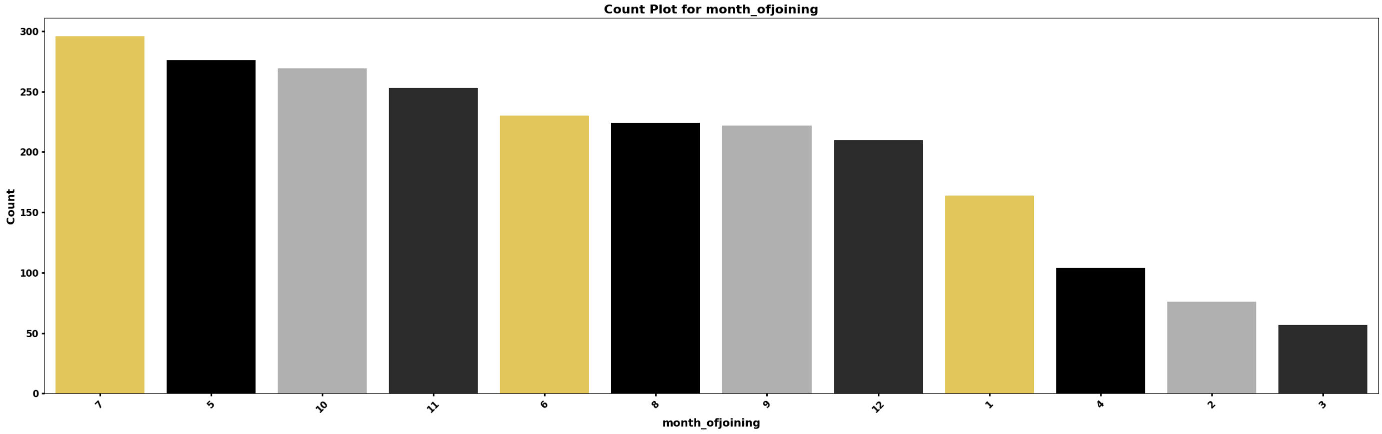
Count Plot for date_ofjoining



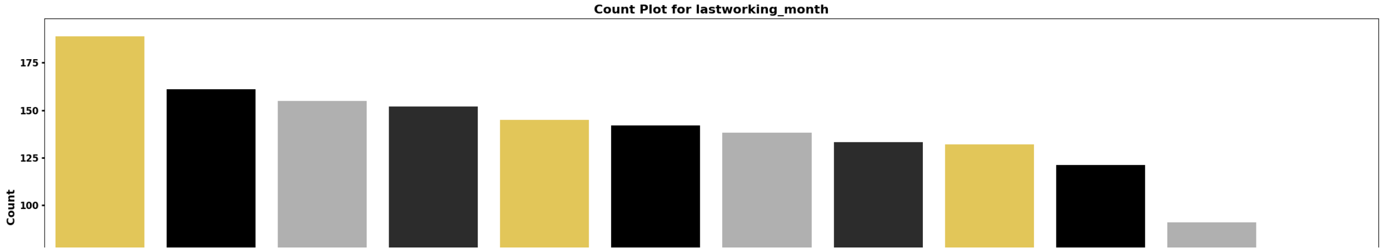
Count Plot for lastworking_date

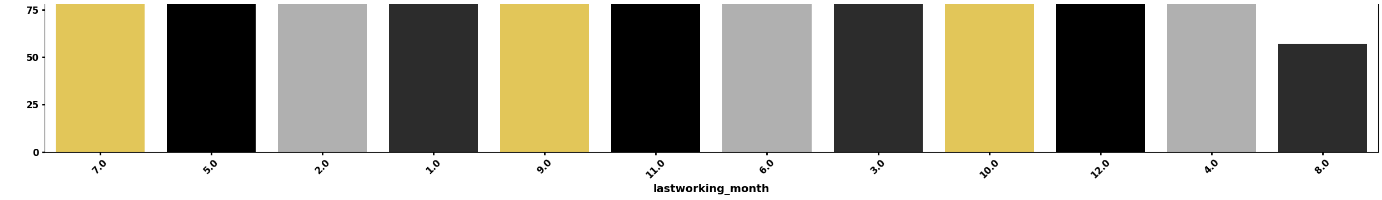


Count Plot for month_ofjoining

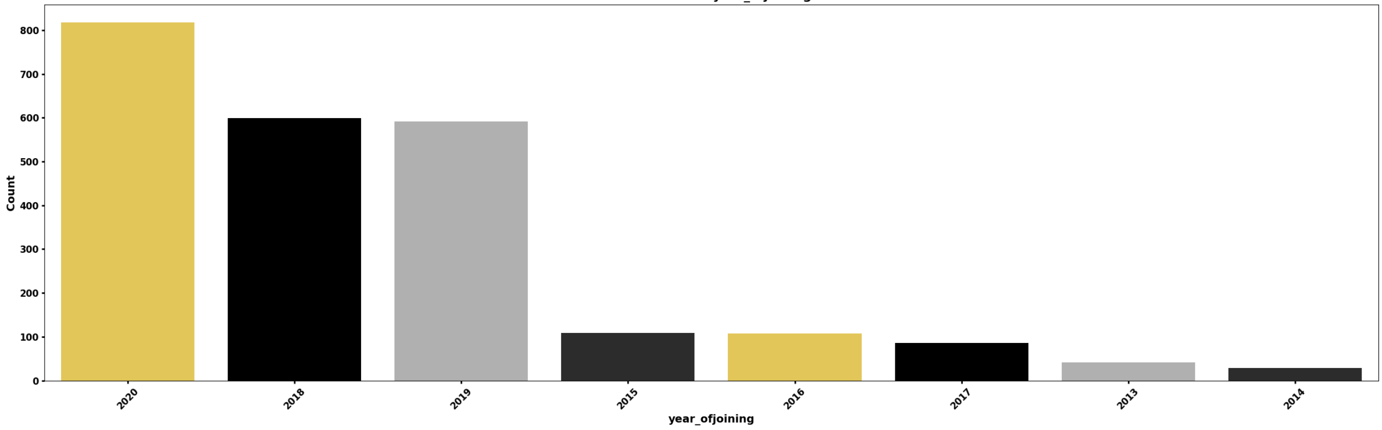


Count Plot for lastworking_month

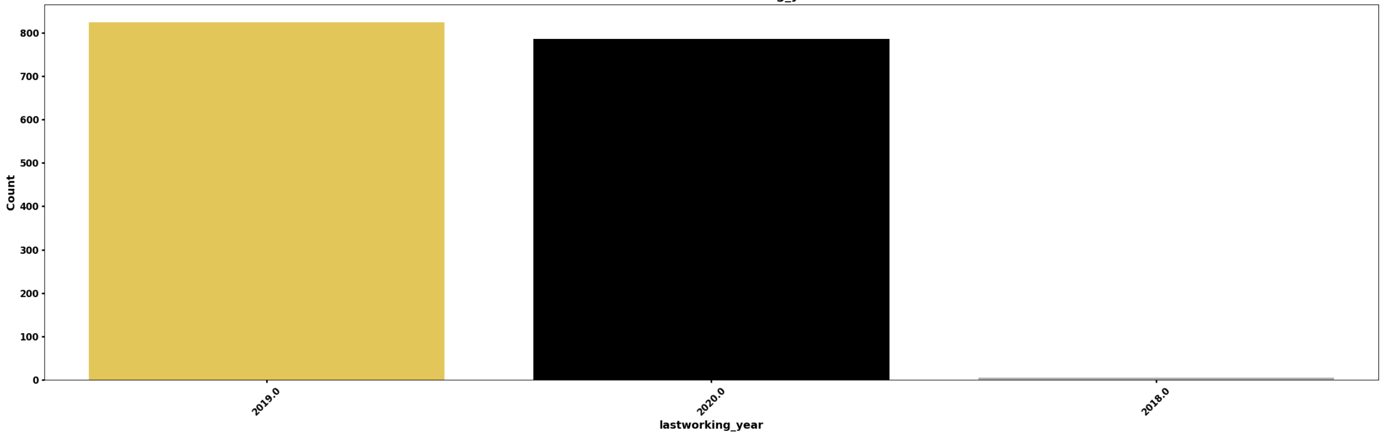




Count Plot for year_ofjoining



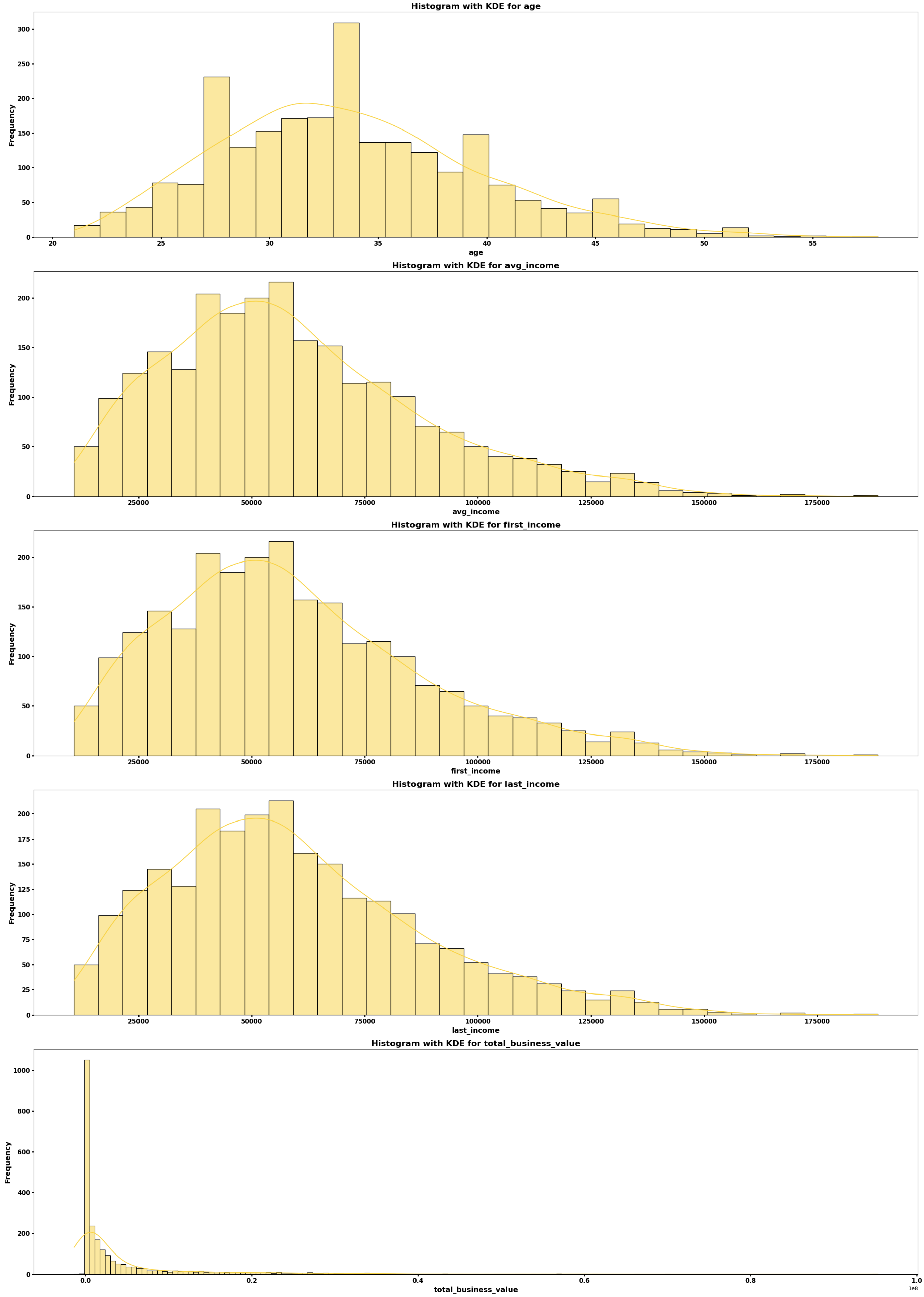
Count Plot for lastworking_year



In [29]:

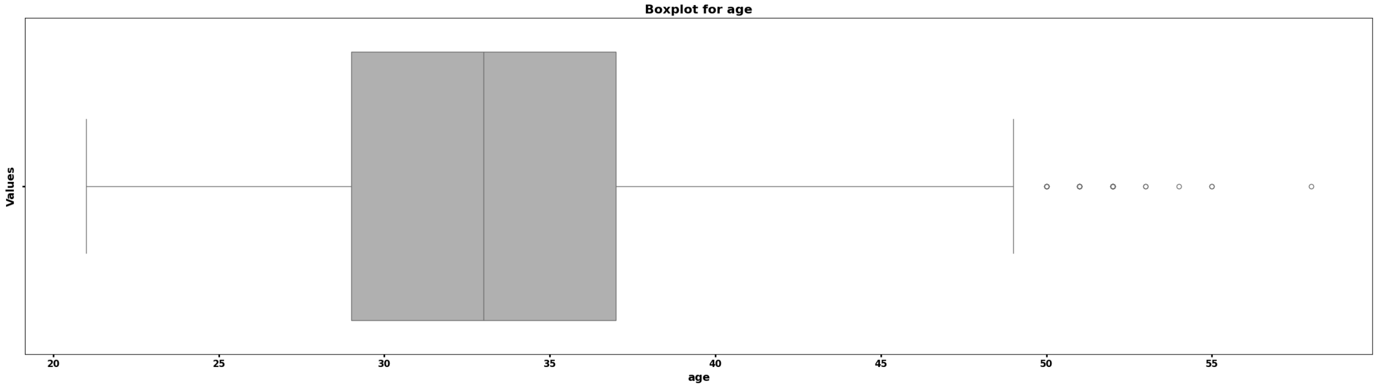
```
continuous_features = ['age','avg_income','first_income','last_income','total_business_v

fig, axes = plt.subplots(len(continuous_features), 1, figsize=(25, 7 * len(continuous_fe
axes = axes.flatten()

for i, col in enumerate(continuous_features):
    sns.histplot(
        data=driver_df,
        x=col,
        kde=True,
        ax=axes[i],
        color="#F9D342"
    )

    axes[i].set_title(f"Histogram with KDE for {col}", fontsize=16, fontweight='bold')
    axes[i].set_xlabel(col, fontsize=14, fontweight='bold')
    axes[i].set_ylabel("Frequency", fontsize=14, fontweight='bold')

    axes[i].tick_params(axis='x', labelsize=12, width=2)
    axes[i].tick_params(axis='y', labelsize=12, width=2)

    for tick in axes[i].get_xticklabels():
        tick.set_fontweight('bold')
    for tick in axes[i].get_yticklabels():
        tick.set_fontweight('bold')
```

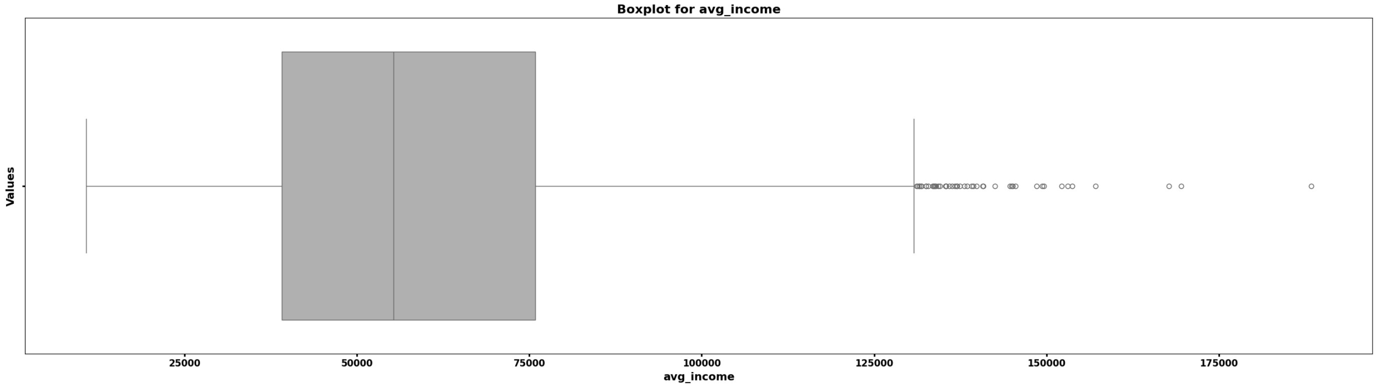
```
plt.tight_layout()
plt.show()
```

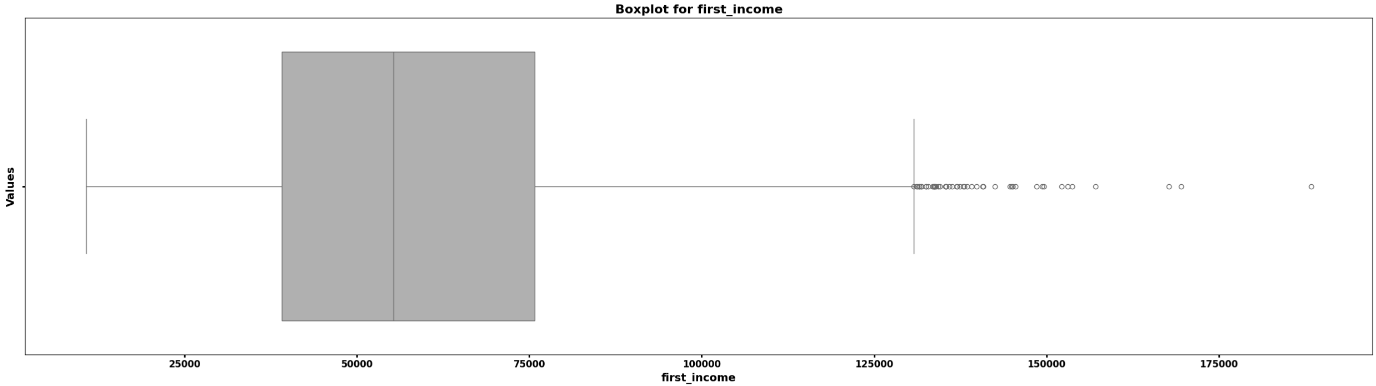

Histogram with KDE for age
Frequency
0
50
100
150
200
250
300
20
25
30
35
40
45
50
55
age
Histogram with KDE for avg_income
Frequency
0
50
100
150
200
25000
50000
75000
100000
125000
150000
175000
avg_income
Histogram with KDE for first_income
Frequency
0
50
100
150
200
25000
50000
75000
100000
125000
150000
175000
first_income
Histogram with KDE for last_income
Frequency
0
25
50
75
100
125
150
175
200
25000
50000
75000
100000
125000
150000
175000
last_income
Histogram with KDE for total_business_value
Frequency
0
200
400
600
800
1000
0.0
0.2
0.4
0.6
0.8
1.0
1e8
total_business_value

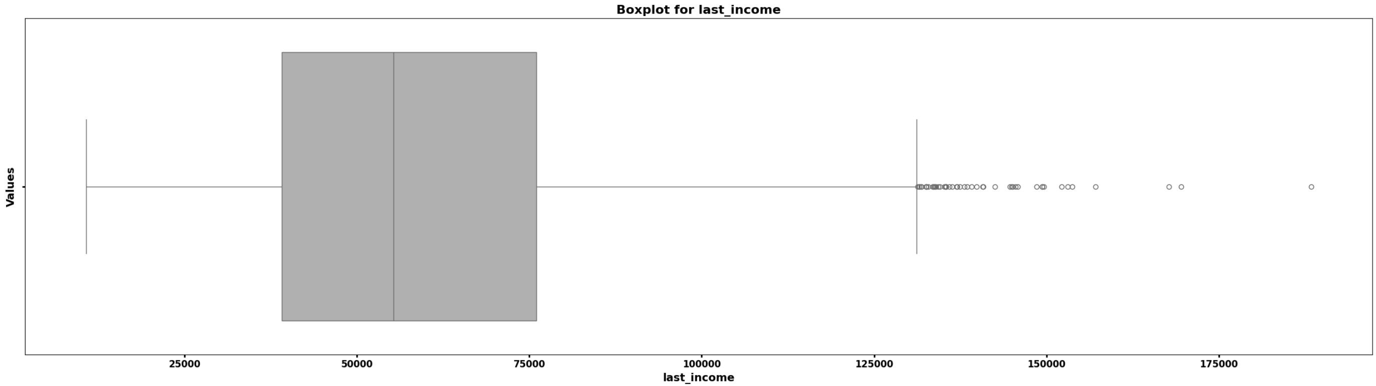
In [30]:

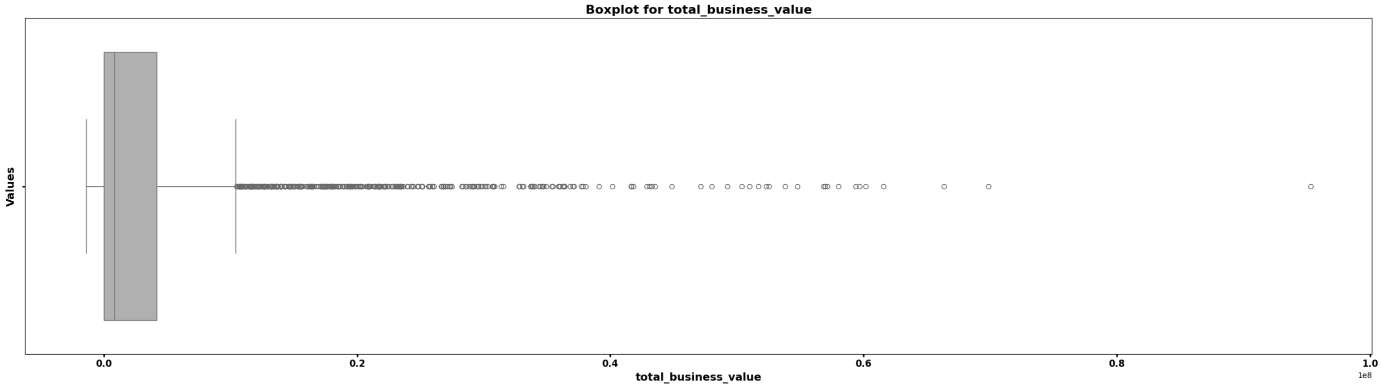
```
fig, axes = plt.subplots(len(continuous_features), 1, figsize=(25, 7 * len(continuous_fe
axes = axes.flatten()

for i, col in enumerate(continuous_features):
    sns.boxplot(
        data=driver_df,
        x=col,
        ax=axes[i],
        color=ola_palette[2]
    )

    axes[i].set_title(f"Boxplot for {col}", fontsize=16, fontweight='bold')
    axes[i].set_xlabel(col, fontsize=14, fontweight='bold')
    axes[i].set_ylabel("Values", fontsize=14, fontweight='bold')

    axes[i].tick_params(axis='x', labelsize=12, width=2)
    axes[i].tick_params(axis='y', labelsize=12, width=2)

    for tick in axes[i].get_xticklabels():
        tick.set_fontweight('bold')
    for tick in axes[i].get_yticklabels():
        tick.set_fontweight('bold')

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


Boxplot for age
Values
20
25
30
35
40
45
50
55
age


Boxplot for avg_income
Values
25000
50000
75000
100000
125000
150000
175000
avg_income

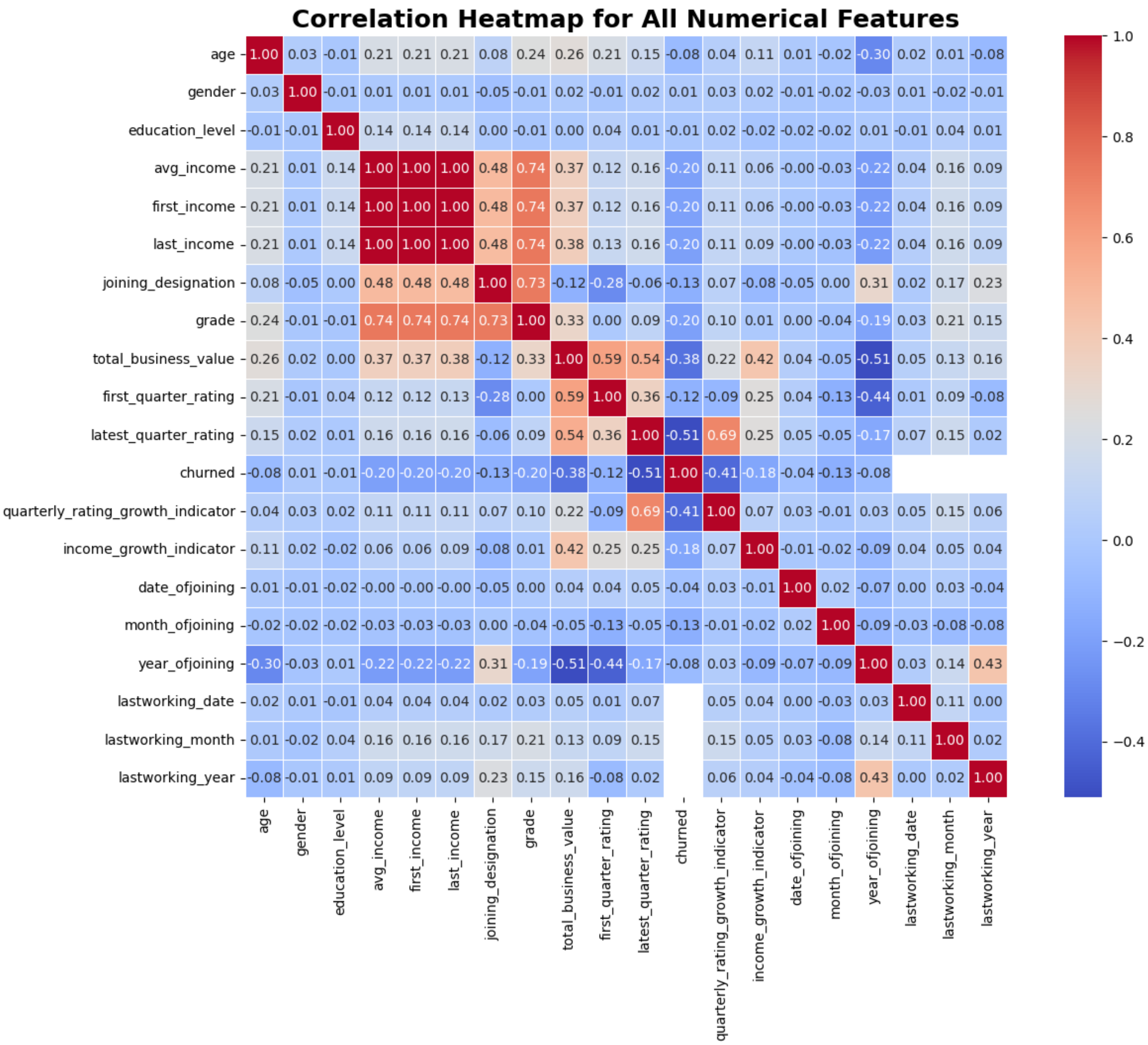

Boxplot for first_income
Values
25000
50000
75000
100000
125000
150000
175000
first_income


Boxplot for last_income
Values
25000
50000
75000
100000
125000
150000
175000
last_income


Boxplot for total_business_value
Values
0.0
0.2
0.4
0.6
0.8
1.0
1e8
total_business_value

In [31]:

```python
numerical_columns = driver_df.select_dtypes(include=['float', 'int']).columns
corr_matrix = driver_df[numerical_columns].corr()

plt.figure(figsize=(15, 10))
sns.heatmap(
    corr_matrix,
    annot=True,           # Show correlation values on the heatmap
    fmt=".2f",            # Format the annotation to two decimal places
    cmap="coolwarm",      # Use a diverging color map
    linewidths=0.5,       # Add lines between cells
    cbar=True,            # Show the color bar
    square=True           # Make cells square-shaped
)

plt.title("Correlation Heatmap for All Numerical Features", fontsize=18, fontweight='bol

plt.show()
```



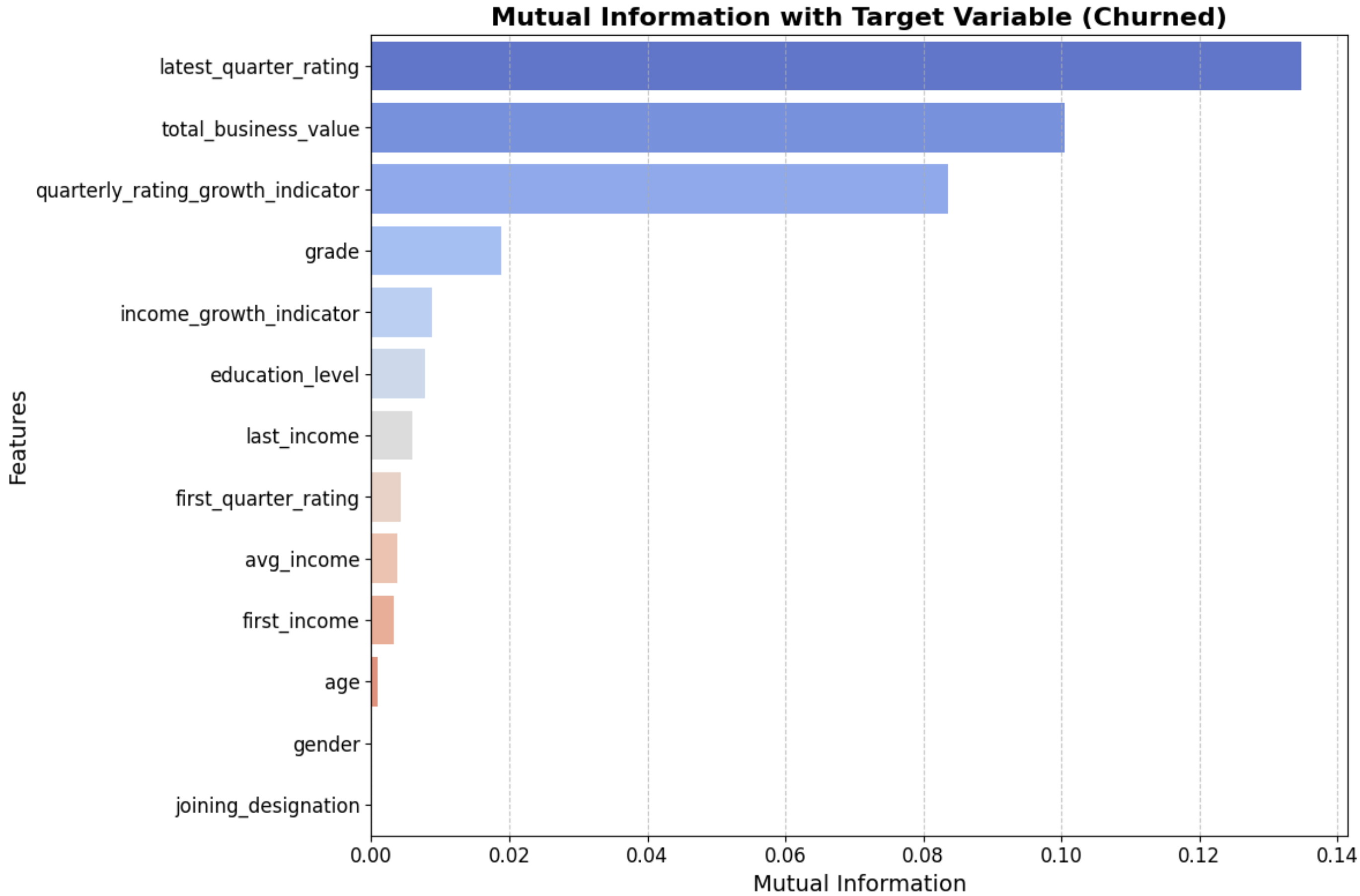
In [32]:

```python
# dropping all the date related features as time-series anaylsis is out of scope for thi
driver_df.drop(['dateofjoining','lastworkingdate','date_ofjoining','month_ofjoining','ye
```

```
                'lastworking_date','lastworking_month','lastworking_year'],axis=1,inplac
```

In [33]:

```
X = driver_df.drop(columns=['churned','city'])
y = driver_df['churned']

mutual_info = mutual_info_classif(X, y, discrete_features='auto', random_state=42)

mi_df = pd.DataFrame({
    'Feature': X.columns,
    'Mutual Information': mutual_info
}).sort_values(by='Mutual Information', ascending=False)

plt.figure(figsize=(12, 8))
sns.barplot(
    x='Mutual Information',
    y='Feature',
    data=mi_df,
    palette='coolwarm'
)
plt.title('Mutual Information with Target Variable (Churned)', fontsize=16, fontweight='
plt.xlabel('Mutual Information', fontsize=14)
plt.ylabel('Features', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.7)

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



In [34]:

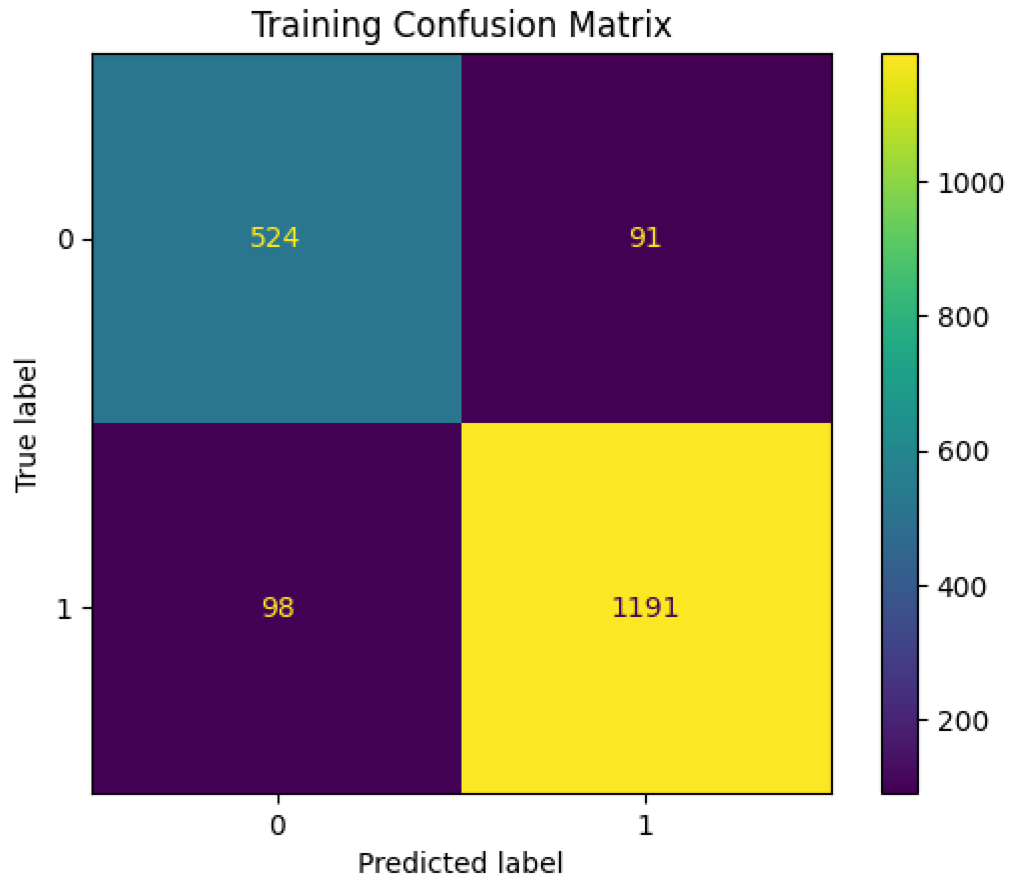
```
X = driver_df.drop(['churned'],axis=1)
y = driver_df['churned']

city_counts = X['city'].value_counts()
X['city_encoded'] = X['city'].map(city_counts)
X.drop('city',axis=1,inplace=True)

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

In [35]:

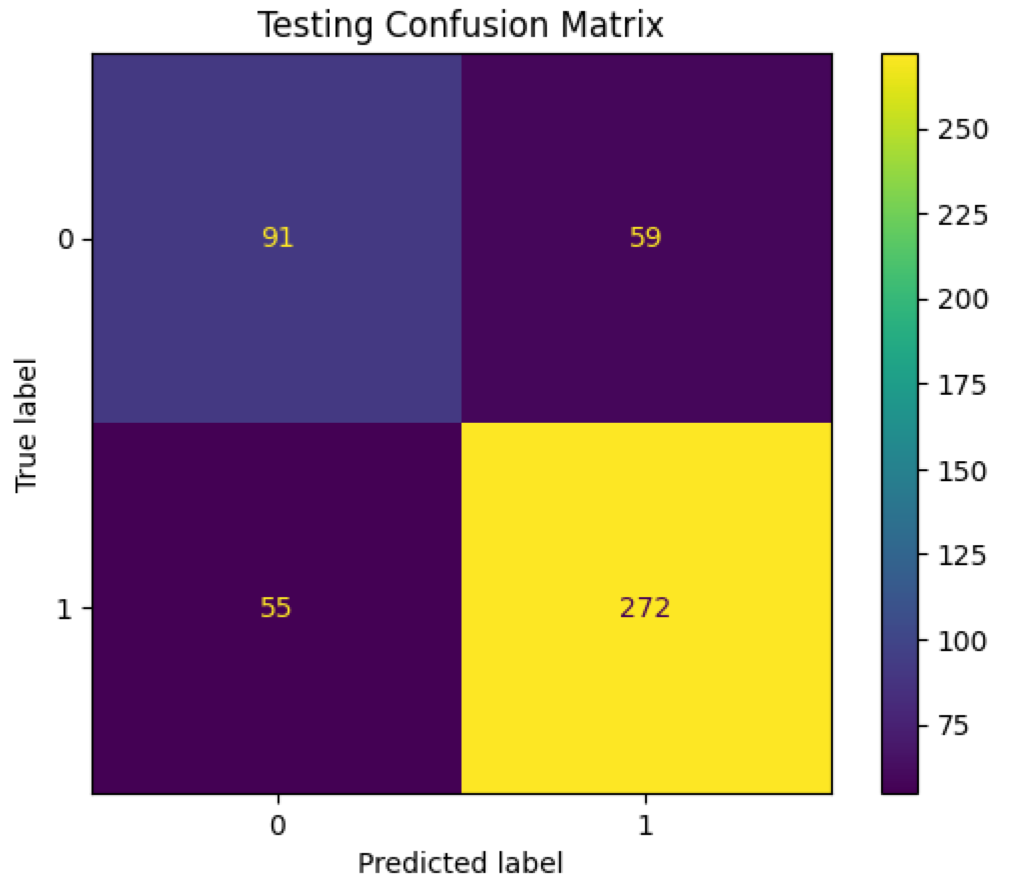
```
print('TRAINING WITH BalancedRandomForestClassifier')

brf = BalancedRandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [50, 150, 200],
    'max_depth': [None, 10, 20],
    'max_features': ['sqrt', 'log2']
}
grid_search_brf_raw = GridSearchCV(estimator=brf, param_grid=param_grid, cv=5, scoring='
start = datetime.datetime.now()
grid_search_brf_raw.fit(X_train, y_train)
end = datetime.datetime.now()
duration = end - start
print(f"Time taken to train: {duration}")
print(f"Best parameters found: {grid_search_brf_raw.best_params_}")

# Training accuracy and confusion matrix
ypred_train = grid_search_brf_raw.predict(X_train)
train_accuracy = accuracy_score(y_train, ypred_train)
print(f"\nTraining Accuracy: {train_accuracy:.4f}")

train_cm = confusion_matrix(y_train, ypred_train, labels=grid_search_brf_raw.classes_)
print("\nTraining Confusion Matrix:")
print(train_cm)

disp_train = ConfusionMatrixDisplay(confusion_matrix=train_cm,
                                    display_labels=grid_search_brf_raw.classes_)
disp_train.plot()
plt.title("Training Confusion Matrix")
plt.show()

# Testing accuracy and confusion matrix
ypred_test = grid_search_brf_raw.predict(X_test)
test_accuracy = accuracy_score(y_test, ypred_test)
print(f"\nTesting Accuracy: {test_accuracy:.4f}")

print("\nTesting Confusion Matrix:")
cm = confusion_matrix(y_test, ypred_test, labels=grid_search_brf_raw.classes_)
print(cm)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=grid_search_brf_raw.classes_)
disp.plot()
plt.title("Testing Confusion Matrix")
plt.show()

# Classification Report
```

```
print("\nClassification Report for Testing Data:")
print(classification_report(y_test, ypred_test))

# Calculate ROC AUC score
roc_auc = roc_auc_score(y_test, grid_search_brf_raw.predict_proba(X_test)[:, 1])
print(f"\nROC AUC Score: {roc_auc:.2f}")
fpr, tpr, thresholds = roc_curve(y_test, grid_search_brf_raw.predict_proba(X_test)[:, 1]
roc_auc = auc(fpr, tpr)

# Plot the ROC Curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--', label="Random Guess")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate", fontsize=12)
plt.ylabel("True Positive Rate", fontsize=12)
plt.title("Receiver Operating Characteristic (ROC) Curve", fontsize=14)
plt.legend(loc="lower right")
plt.grid(alpha=0.3)
plt.show()

# Feature importance
best_brf = grid_search_brf_raw.best_estimator_
feature_importances = best_brf.feature_importances_
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
})
feature_importance_df.sort_values(by='Importance',ascending=False,inplace=True)
print("\nFeature Importances:")
print(feature_importance_df)
```

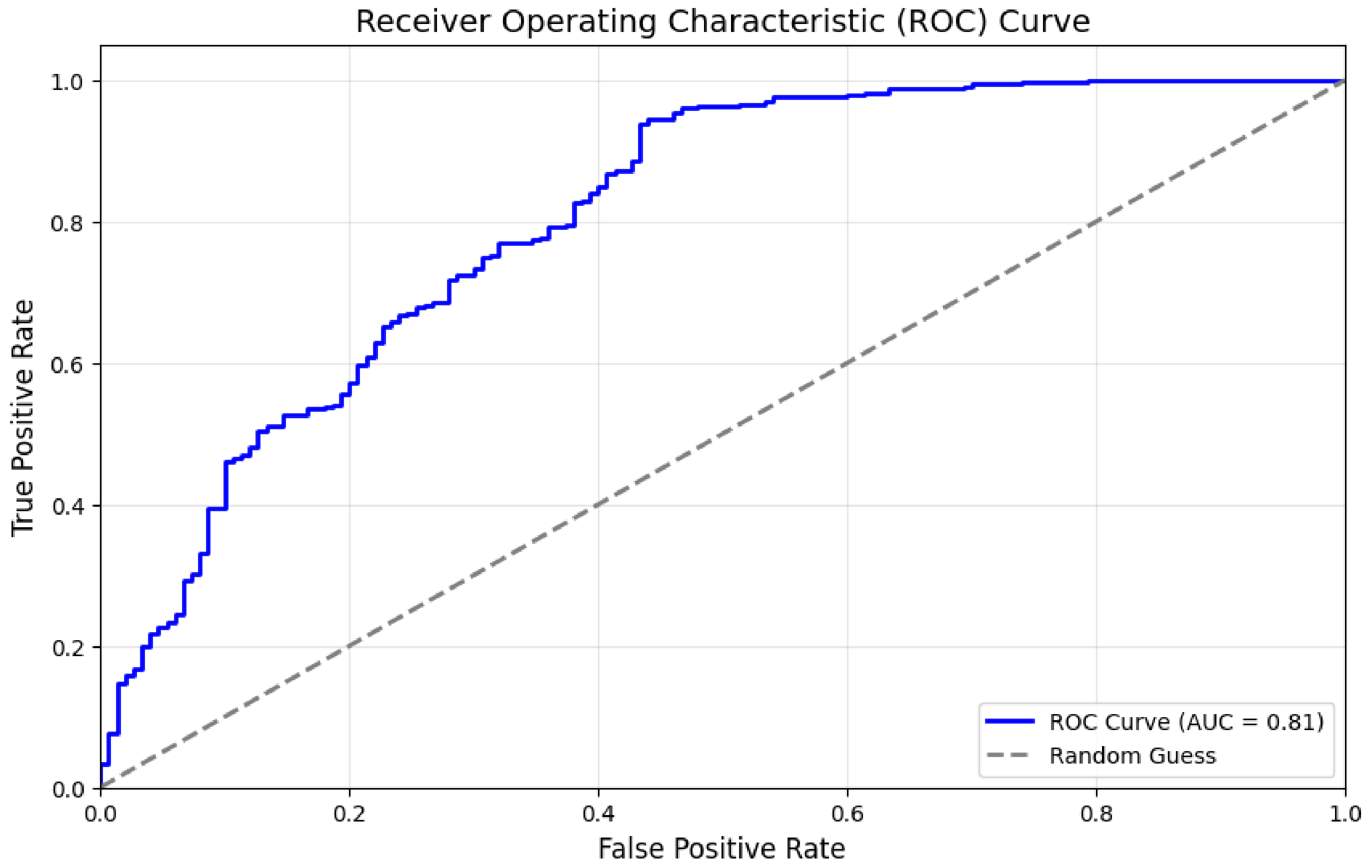
```
TRAINING WITH BalancedRandomForestClassifier
Time taken to train: 0:02:20.550200
Best parameters found: {'max_depth': 10, 'max_features': 'sqrt', 'n_estimators': 200}

Training Accuracy: 0.9007

Training Confusion Matrix:
[[ 524   91]
 [  98 1191]]
```




```
Testing Accuracy: 0.7610

Testing Confusion Matrix:
[[ 91  59]
 [ 55 272]]
```



```
Classification Report for Testing Data:
              precision    recall  f1-score   support

           0       0.62      0.61      0.61       150
           1       0.82      0.83      0.83       327

    accuracy                           0.76       477
   macro avg       0.72      0.72      0.72       477
weighted avg       0.76      0.76      0.76       477


ROC AUC Score: 0.81
```




```
Feature Importances:
                             Feature  Importance
0                                age    0.068970
1                             gender    0.013151
2                    education_level    0.019459
3                         avg_income    0.081580
4                       first_income    0.086042
5                        last_income    0.090758
6                 joining_designation    0.045990
7                              grade    0.032777
8                 total_business_value    0.181670
9                first_quarter_rating    0.028741
10              latest_quarter_rating    0.200009
11   quarterly_rating_growth_indicator    0.089041
12            income_growth_indicator    0.005948
13                       city_encoded    0.055866
```

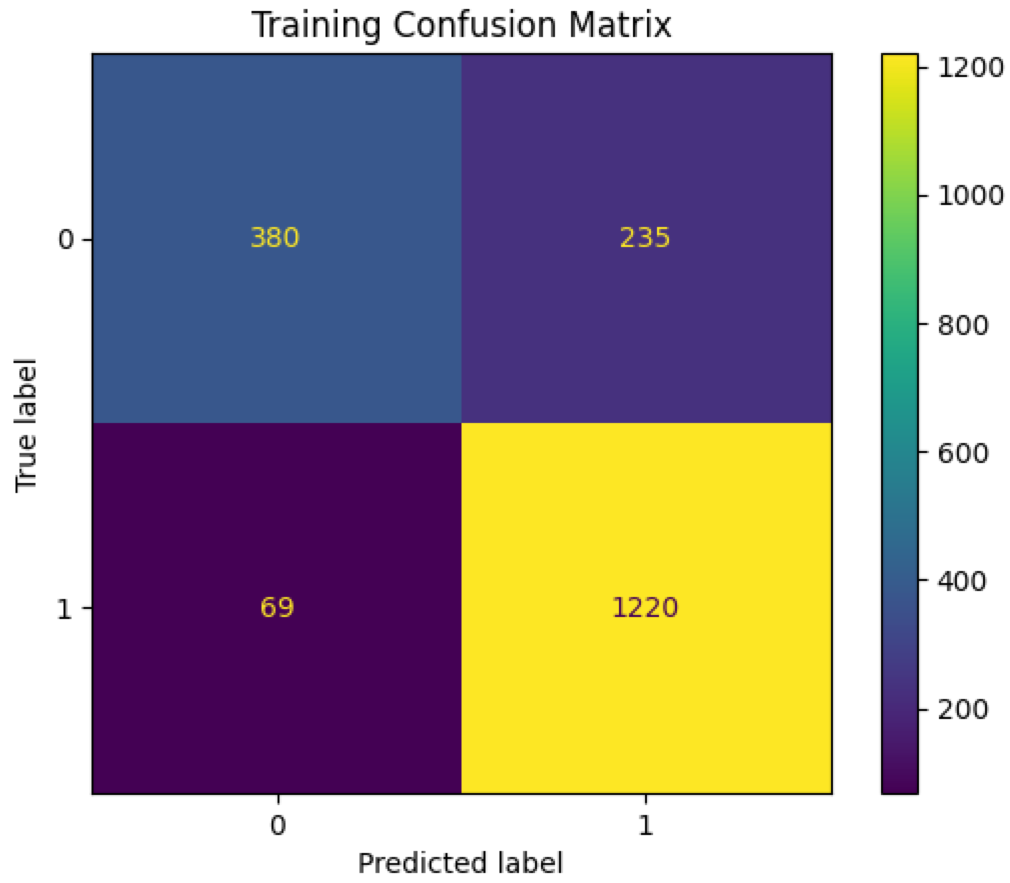
In [36]:

```
print('Training with XGB Classifier using original training data (no SMOTE)')

# Initialize XGBoost Classifier
```

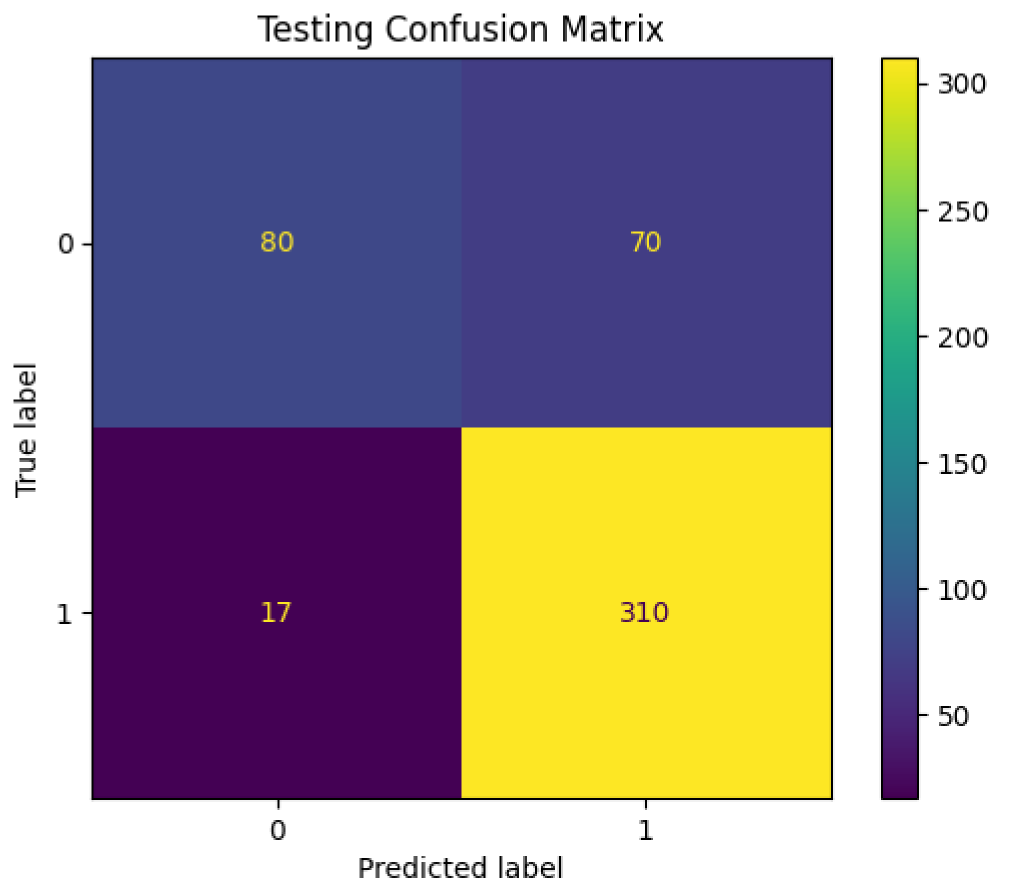
```
xgb_model = XGBClassifier(random_state=42, eval_metric="logloss", use_label_encoder=Fals

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# Perform Grid Search
grid_search_xgb = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, scoring

# Fit the model with original training data
start = datetime.datetime.now()
grid_search_xgb.fit(X_train, y_train)
end = datetime.datetime.now()
duration = end - start
print(f"Time taken for GridSearchCV: {duration}")

# Best parameters and best score
print(f"\nBest parameters found: {grid_search_xgb.best_params_}")
print(f"Best cross-validated accuracy: {grid_search_xgb.best_score_:.4f}")

# Get the best model
best_xgb_model = grid_search_xgb.best_estimator_

# Training predictions and accuracy
ypred_train = best_xgb_model.predict(X_train)
train_accuracy = accuracy_score(y_train, ypred_train)
print(f"\nTraining Accuracy: {train_accuracy:.4f}")

train_cm = confusion_matrix(y_train, ypred_train)
print("\nTraining Confusion Matrix:")
print(train_cm)

# Display training confusion matrix
disp_train = ConfusionMatrixDisplay(confusion_matrix=train_cm, display_labels=np.unique(
disp_train.plot()
plt.title("Training Confusion Matrix")
plt.show()

# Testing predictions and accuracy
ypred_test = best_xgb_model.predict(X_test)
test_accuracy = accuracy_score(y_test, ypred_test)
print(f"\nTesting Accuracy: {test_accuracy:.4f}")

test_cm = confusion_matrix(y_test, ypred_test)
print("\nTesting Confusion Matrix:")
print(test_cm)

# Display testing confusion matrix
disp_test = ConfusionMatrixDisplay(confusion_matrix=test_cm, display_labels=np.unique(y_
disp_test.plot()
plt.title("Testing Confusion Matrix")
plt.show()

# Classification report for testing data
```

```python
print("\nClassification Report for Testing Data:")
print(classification_report(y_test, ypred_test))

# Calculate and plot ROC AUC score
roc_auc = roc_auc_score(y_test, best_xgb_model.predict_proba(X_test)[:, 1])
print(f"\nROC AUC Score: {roc_auc:.2f}")

fpr, tpr, thresholds = roc_curve(y_test, best_xgb_model.predict_proba(X_test)[:, 1])
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--', label="Random Guess")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate", fontsize=12)
plt.ylabel("True Positive Rate", fontsize=12)
plt.title("Receiver Operating Characteristic (ROC) Curve", fontsize=14)
plt.legend(loc="lower right")
plt.grid(alpha=0.3)
plt.show()

# Feature importances
feature_importances = best_xgb_model.feature_importances_

# Create a DataFrame for feature importances
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,  # Assuming X_train has the correct feature column names
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False,inplace=True)

print("\nFeature Importances:")
print(feature_importance_df)
```

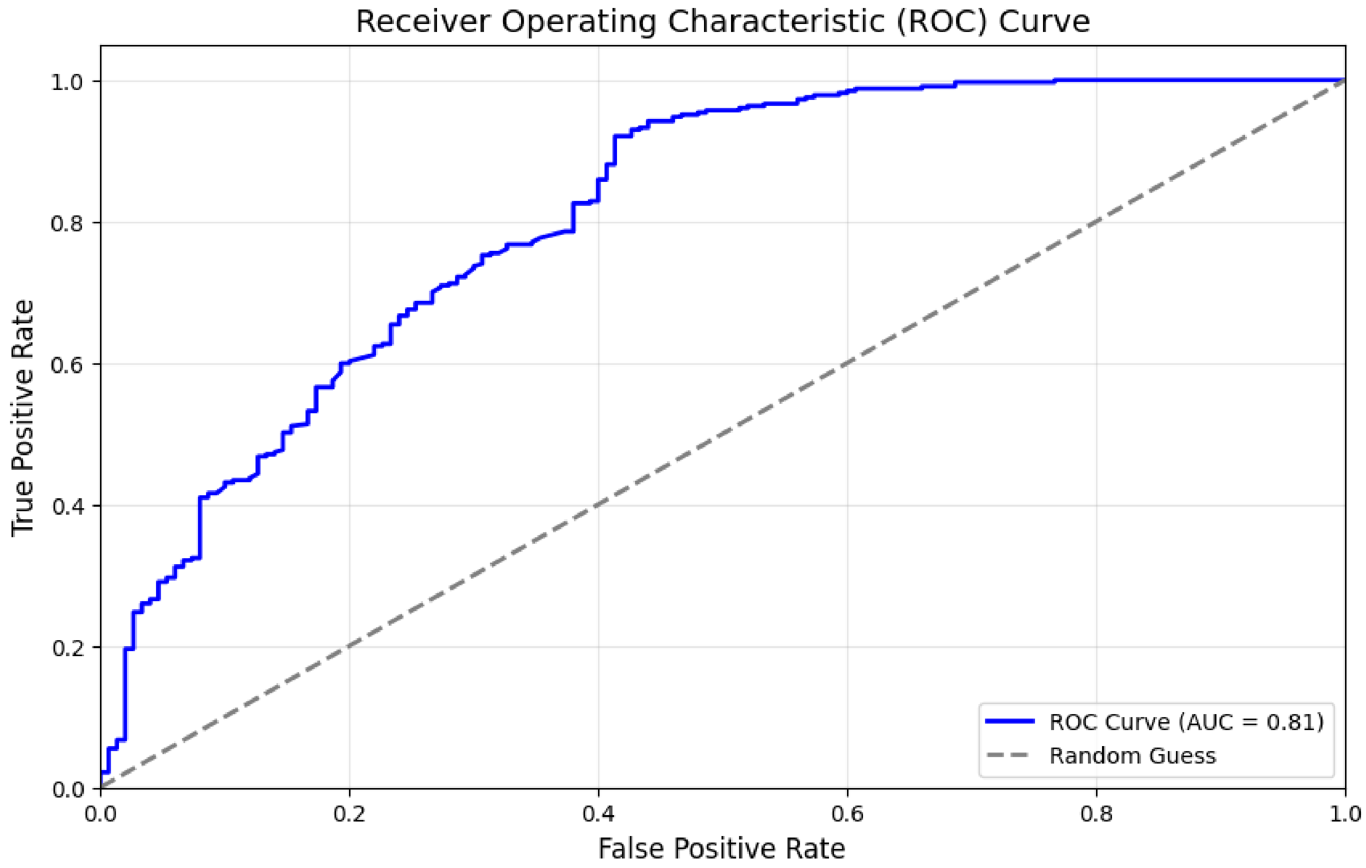
```
Training with XGB Classifier using original training data (no SMOTE)
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Time taken for GridSearchCV: 0:00:59.816034

Best parameters found: {'colsample_bytree': 0.8, 'learning_rate': 0.2, 'max_depth': 3,
'n_estimators': 50, 'subsample': 1.0}
Best cross-validated accuracy: 0.8120

Training Accuracy: 0.8403

Training Confusion Matrix:
[[ 380  235]
 [  69 1220]]
```




```
Testing Accuracy: 0.8176

Testing Confusion Matrix:
[[ 80  70]
 [ 17 310]]
```



```
Classification Report for Testing Data:
              precision    recall  f1-score   support

           0       0.82      0.53      0.65       150
           1       0.82      0.95      0.88       327

    accuracy                           0.82       477
   macro avg       0.82      0.74      0.76       477
weighted avg       0.82      0.82      0.80       477


ROC AUC Score: 0.81
```




```
Feature Importances:
                              Feature  Importance
10              latest_quarter_rating    0.402550
11  quarterly_rating_growth_indicator    0.114494
6                  joining_designation    0.103659
8                 total_business_value    0.068640
9                 first_quarter_rating    0.053180
7                                grade    0.046589
12             income_growth_indicator    0.043374
4                         first_income    0.034824
5                          last_income    0.030650
0                                  age    0.026486
3                           avg_income    0.023247
13                        city_encoded    0.021782
1                               gender    0.018369
2                      education_level    0.012155
```

In [37]:

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

```
print(y_train.value_counts())
print('\n')
print('After SMOTE')
print(y_train_smote.value_counts())
```

```
Before SMOTE
churned
1    1289
0     615
Name: count, dtype: int64


After SMOTE
churned
0    1289
1    1289
Name: count, dtype: int64
```

In [38]:

```
rf_model = RandomForestClassifier(random_state=42)

# Parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'max_features': ['sqrt', 'log2'],
    'min_samples_split': [2, 5, 10]
}

# GridSearchCV
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, scoring='
grid_search_rf.fit(X_train_smote, y_train_smote)

# Best model
best_rf_model = grid_search_rf.best_estimator_

# Predictions
ypred_test = best_rf_model.predict(X_test)
ypred_train = best_rf_model.predict(X_train_smote)

# Evaluate model
print(f"Training Accuracy: {accuracy_score(y_train_smote, ypred_train):.4f}")
print(f"Testing Accuracy: {accuracy_score(y_test, ypred_test):.4f}")

print("\nConfusion Matrix (Test):")
print(confusion_matrix(y_test, ypred_test))

print("\nClassification Report:")
print(classification_report(y_test, ypred_test))
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits
Training Accuracy: 0.9503
Testing Accuracy: 0.7820

Confusion Matrix (Test):
[[ 93  57]
 [ 47 280]]

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

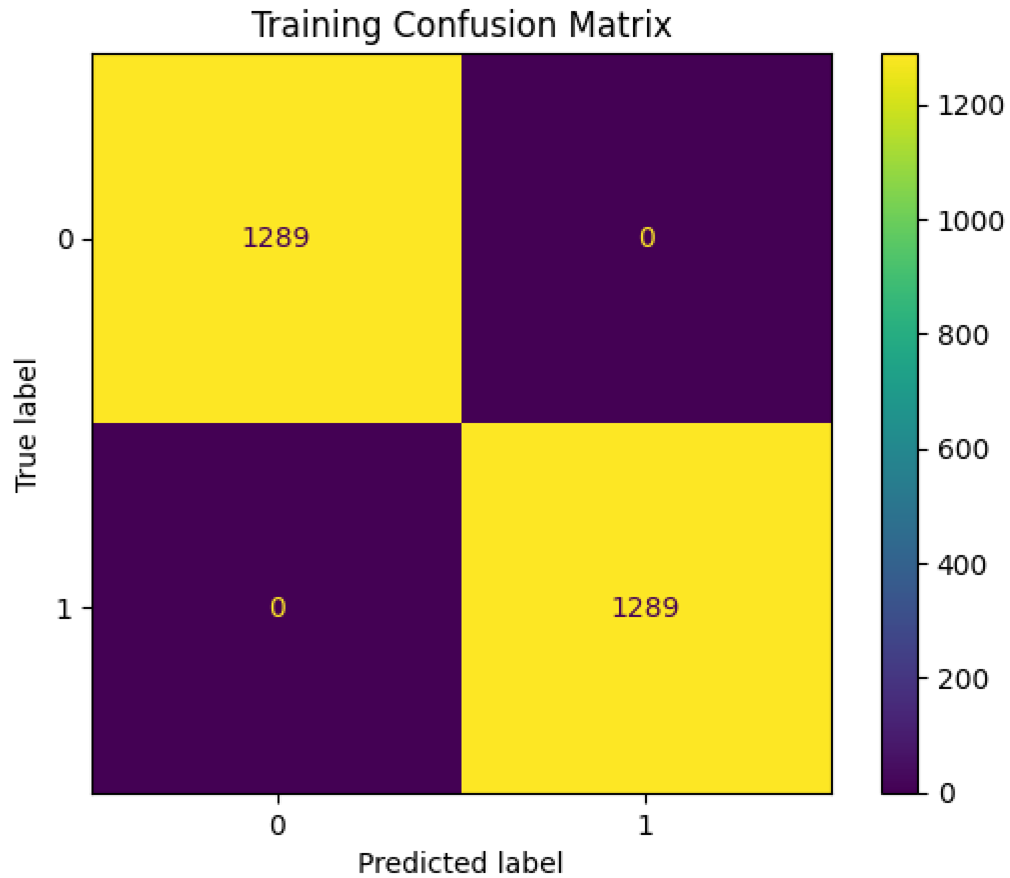
```
           0       0.66      0.62      0.64       150
           1       0.83      0.86      0.84       327

    accuracy                           0.78       477
   macro avg       0.75      0.74      0.74       477
weighted avg       0.78      0.78      0.78       477
```

In [39]:

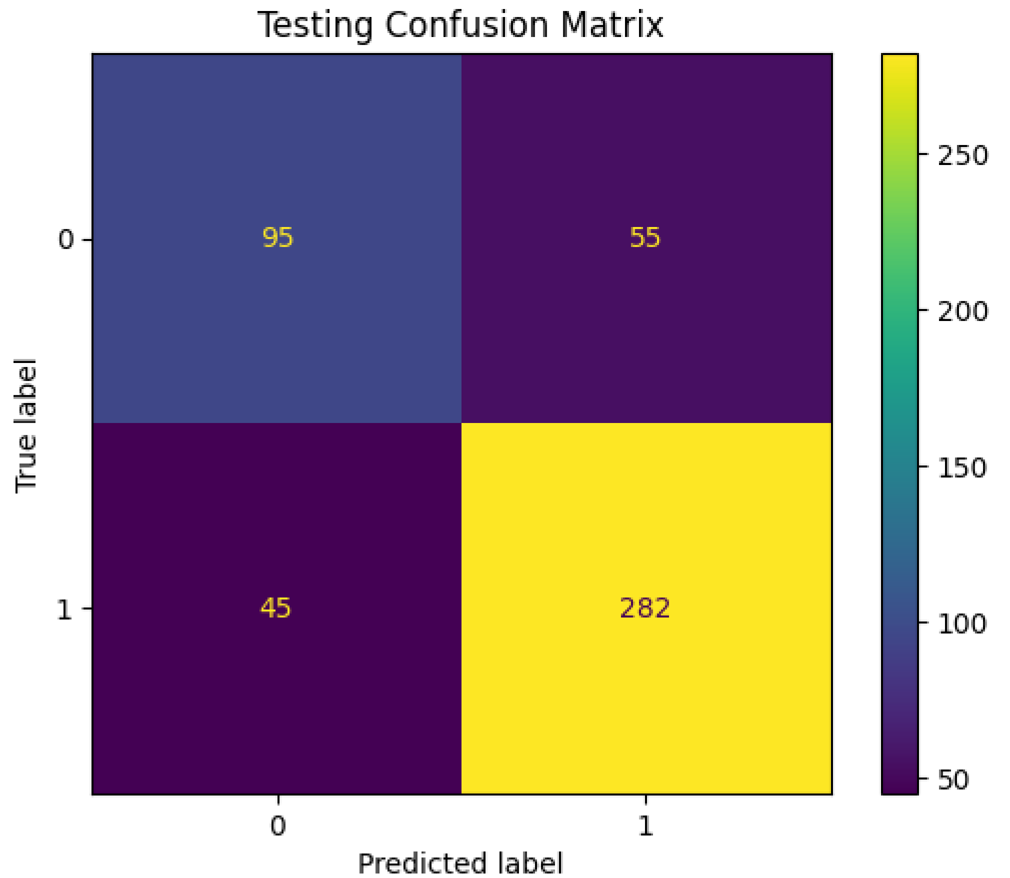
```
print('TRAINING WITH BalancedRandomForestClassifier post SMOTE')

brf_smote = BalancedRandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [50, 150, 200],
    'max_depth': [None, 10, 20],
    'max_features': ['sqrt', 'log2']
}

grid_search_brf_smote = GridSearchCV(estimator=brf_smote, param_grid=param_grid, cv=5, s
start = datetime.datetime.now()
grid_search_brf_smote.fit(X_train_smote, y_train_smote)
end = datetime.datetime.now()
duration = end - start
print(f"Time taken to train: {duration}")
print(f"Best parameters found: {grid_search_brf_smote.best_params_}")

# Training accuracy and confusion matrix
ypred_train = grid_search_brf_smote.predict(X_train_smote)
train_accuracy = accuracy_score(y_train_smote, ypred_train)
print(f"\nTraining Accuracy: {train_accuracy:.4f}")

train_cm = confusion_matrix(y_train_smote, ypred_train, labels=grid_search_brf_smote.cla
print("\nTraining Confusion Matrix:")
print(train_cm)

disp_train = ConfusionMatrixDisplay(confusion_matrix=train_cm,
                                    display_labels=grid_search_brf_smote.classes_)
disp_train.plot()
plt.title("Training Confusion Matrix")
plt.show()

# Testing accuracy and confusion matrix
ypred_test = grid_search_brf_smote.predict(X_test)
test_accuracy = accuracy_score(y_test, ypred_test)
print(f"\nTesting Accuracy: {test_accuracy:.4f}")

print("\nTesting Confusion Matrix:")
cm = confusion_matrix(y_test, ypred_test, labels=grid_search_brf_smote.classes_)
print(cm)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=grid_search_brf_smote.classes_)
disp.plot()
plt.title("Testing Confusion Matrix")
plt.show()

# Classification Report
```

```python
print("\nClassification Report for Testing Data:")
print(classification_report(y_test, ypred_test))

# Calculate ROC AUC score
roc_auc = roc_auc_score(y_test, grid_search_brf_smote.predict_proba(X_test)[:, 1])
print(f"\nROC AUC Score: {roc_auc:.2f}")
fpr, tpr, thresholds = roc_curve(y_test, grid_search_brf_smote.predict_proba(X_test)[:,
roc_auc = auc(fpr, tpr)

# Plot the ROC Curve
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--', label="Random Guess")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate", fontsize=12)
plt.ylabel("True Positive Rate", fontsize=12)
plt.title("Receiver Operating Characteristic (ROC) Curve", fontsize=14)
plt.legend(loc="lower right")
plt.grid(alpha=0.3)
plt.show()

# Feature importance
best_brf = grid_search_brf_smote.best_estimator_
feature_importances = best_brf.feature_importances_
feature_importance_df = pd.DataFrame({
    'Feature': X_train_smote.columns,
    'Importance': feature_importances
})
feature_importance_df.sort_values(by='Importance',ascending=False,inplace=True)
print("\nFeature Importances:")
print(feature_importance_df)
```

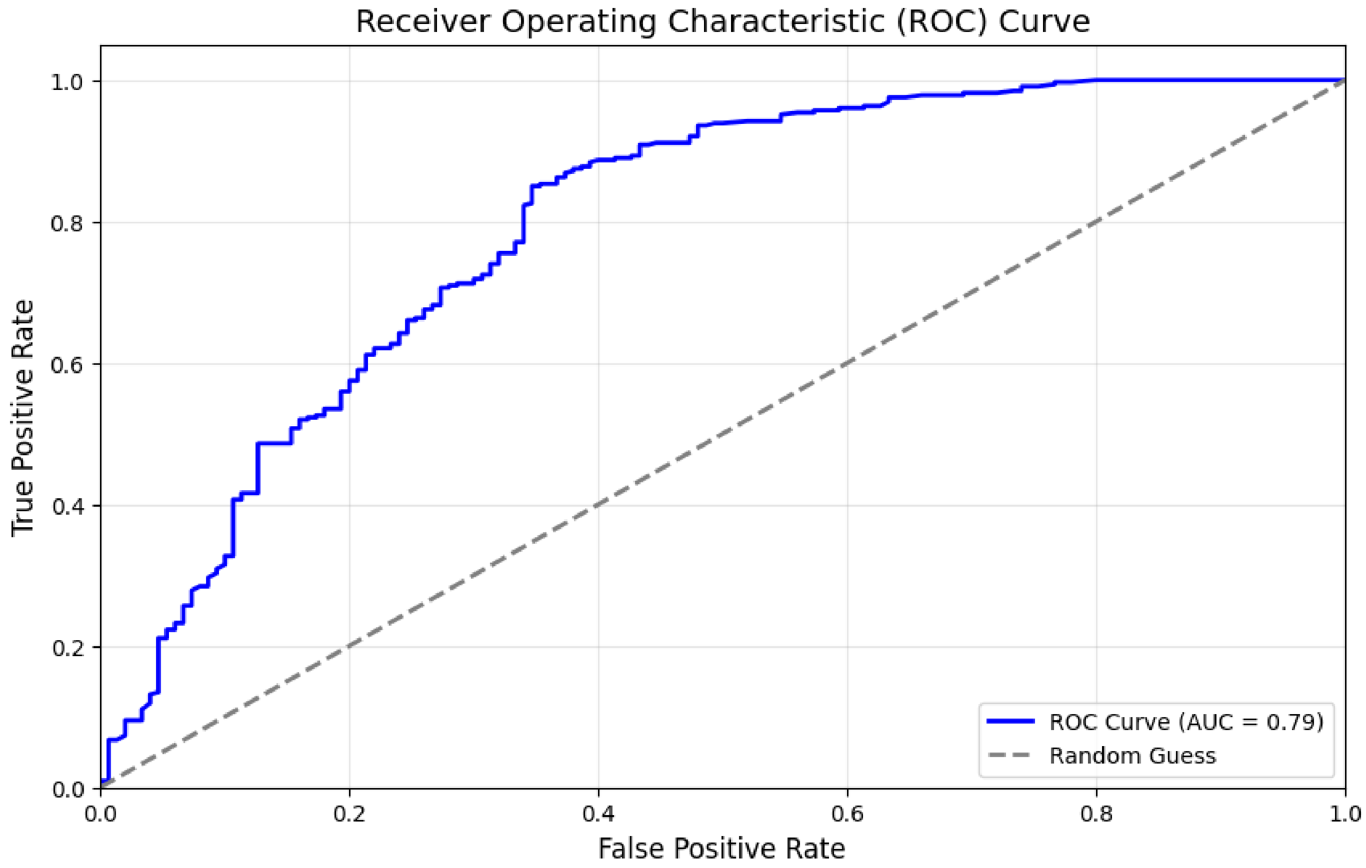
```
TRAINING WITH BalancedRandomForestClassifier post SMOTE
Time taken to train: 0:03:20.209801
Best parameters found: {'max_depth': 20, 'max_features': 'sqrt', 'n_estimators': 200}

Training Accuracy: 1.0000

Training Confusion Matrix:
[[1289    0]
 [   0 1289]]
```




```
Testing Accuracy: 0.7904

Testing Confusion Matrix:
[[ 95  55]
 [ 45 282]]
```



```
Classification Report for Testing Data:
              precision    recall  f1-score   support

           0       0.68      0.63      0.66       150
           1       0.84      0.86      0.85       327

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


ROC AUC Score: 0.79
```




```
Feature Importances:
                             Feature  Importance
0                                age    0.098950
1                             gender    0.069989
2                    education_level    0.031373
3                         avg_income    0.096310
4                       first_income    0.099826
5                        last_income    0.104006
6                joining_designation    0.031061
7                              grade    0.023333
8               total_business_value    0.161542
9               first_quarter_rating    0.030148
10             latest_quarter_rating    0.136828
11  quarterly_rating_growth_indicator    0.038340
12           income_growth_indicator    0.001615
13                      city_encoded    0.076680
```

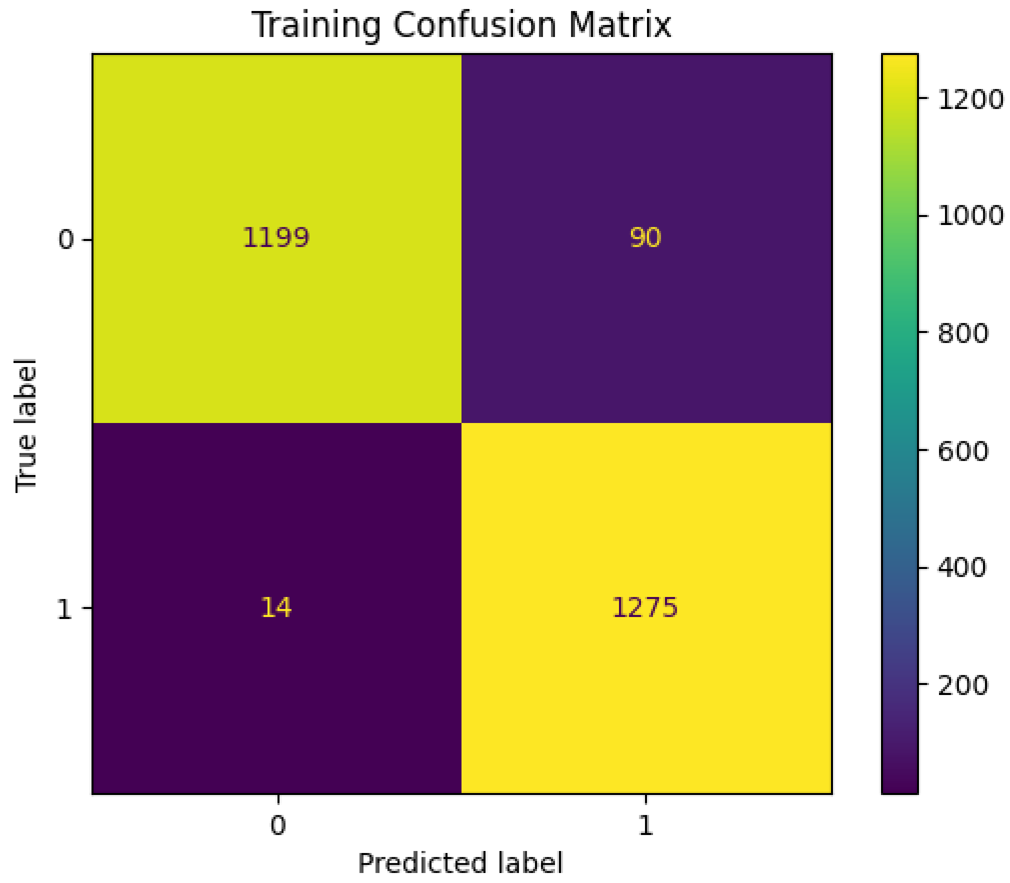
In [40]:

```
print('Training with XGB Classifier with SMOTE upsampled data')

# Initialize XGBoost Classifier
```

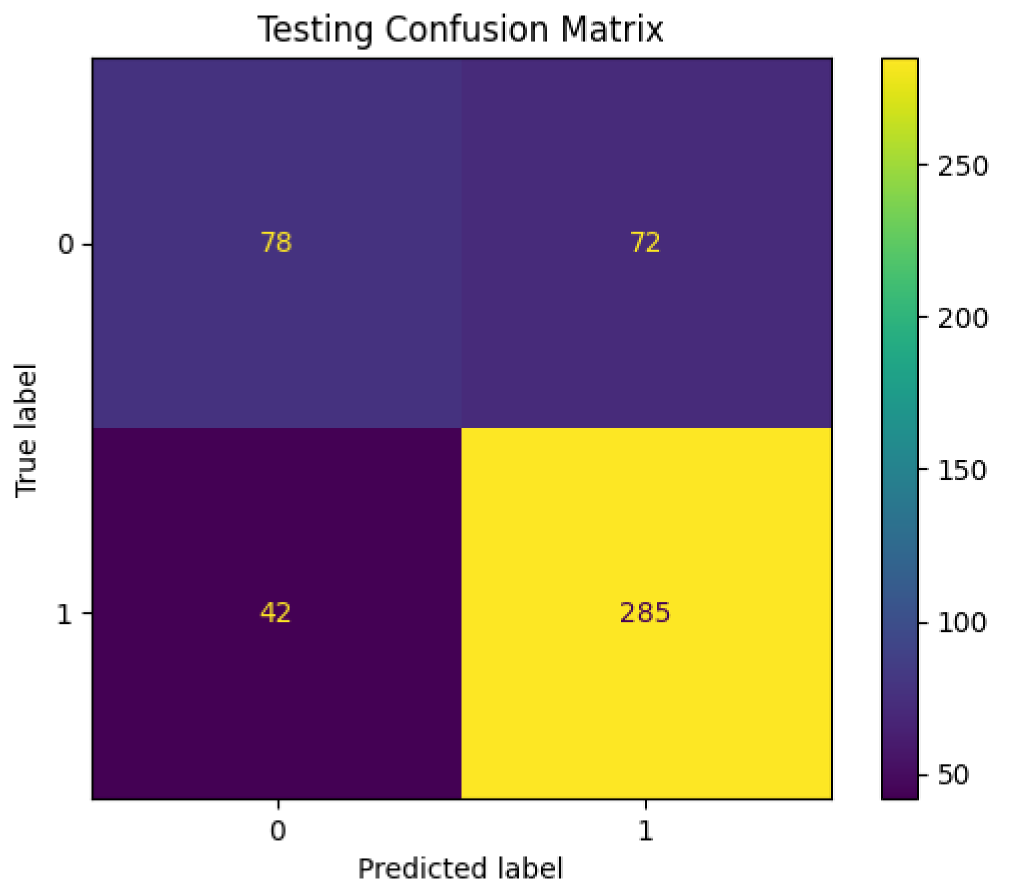
```
xgb_model = XGBClassifier(random_state=42, eval_metric="logloss", use_label_encoder=Fals

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

# Perform Grid Search
grid_search_xgb = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, scoring

# Fit the model with SMOTE-augmented training data
start = datetime.datetime.now()
grid_search_xgb.fit(X_train_smote, y_train_smote)
end = datetime.datetime.now()
duration = end - start
print(f"Time taken for GridSearchCV: {duration}")

# Best parameters and best score
print(f"\nBest parameters found: {grid_search_xgb.best_params_}")
print(f"Best cross-validated accuracy: {grid_search_xgb.best_score_:.4f}")

# Get the best model
best_xgb_model = grid_search_xgb.best_estimator_

# Training predictions and accuracy
ypred_train = best_xgb_model.predict(X_train_smote)
train_accuracy = accuracy_score(y_train_smote, ypred_train)
print(f"\nTraining Accuracy: {train_accuracy:.4f}")

train_cm = confusion_matrix(y_train_smote, ypred_train)
print("\nTraining Confusion Matrix:")
print(train_cm)

# Display training confusion matrix
disp_train = ConfusionMatrixDisplay(confusion_matrix=train_cm, display_labels=np.unique(
disp_train.plot()
plt.title("Training Confusion Matrix")
plt.show()

# Testing predictions and accuracy
ypred_test = best_xgb_model.predict(X_test)
test_accuracy = accuracy_score(y_test, ypred_test)
print(f"\nTesting Accuracy: {test_accuracy:.4f}")

test_cm = confusion_matrix(y_test, ypred_test)
print("\nTesting Confusion Matrix:")
print(test_cm)

# Display testing confusion matrix
disp_test = ConfusionMatrixDisplay(confusion_matrix=test_cm, display_labels=np.unique(y_
disp_test.plot()
plt.title("Testing Confusion Matrix")
plt.show()

# Classification report for testing data
```

```python
print("\nClassification Report for Testing Data:")
print(classification_report(y_test, ypred_test))

# Calculate and plot ROC AUC score
roc_auc = roc_auc_score(y_test, best_xgb_model.predict_proba(X_test)[:, 1])
print(f"\nROC AUC Score: {roc_auc:.2f}")

fpr, tpr, thresholds = roc_curve(y_test, best_xgb_model.predict_proba(X_test)[:, 1])
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--', label="Random Guess")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate", fontsize=12)
plt.ylabel("True Positive Rate", fontsize=12)
plt.title("Receiver Operating Characteristic (ROC) Curve", fontsize=14)
plt.legend(loc="lower right")
plt.grid(alpha=0.3)
plt.show()

# Feature importances
feature_importances = best_xgb_model.feature_importances_

# Create a DataFrame for feature importances
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,  # Assuming X_train and X_train_smote have the same colu
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

print("\nFeature Importances:")
print(feature_importance_df)
```

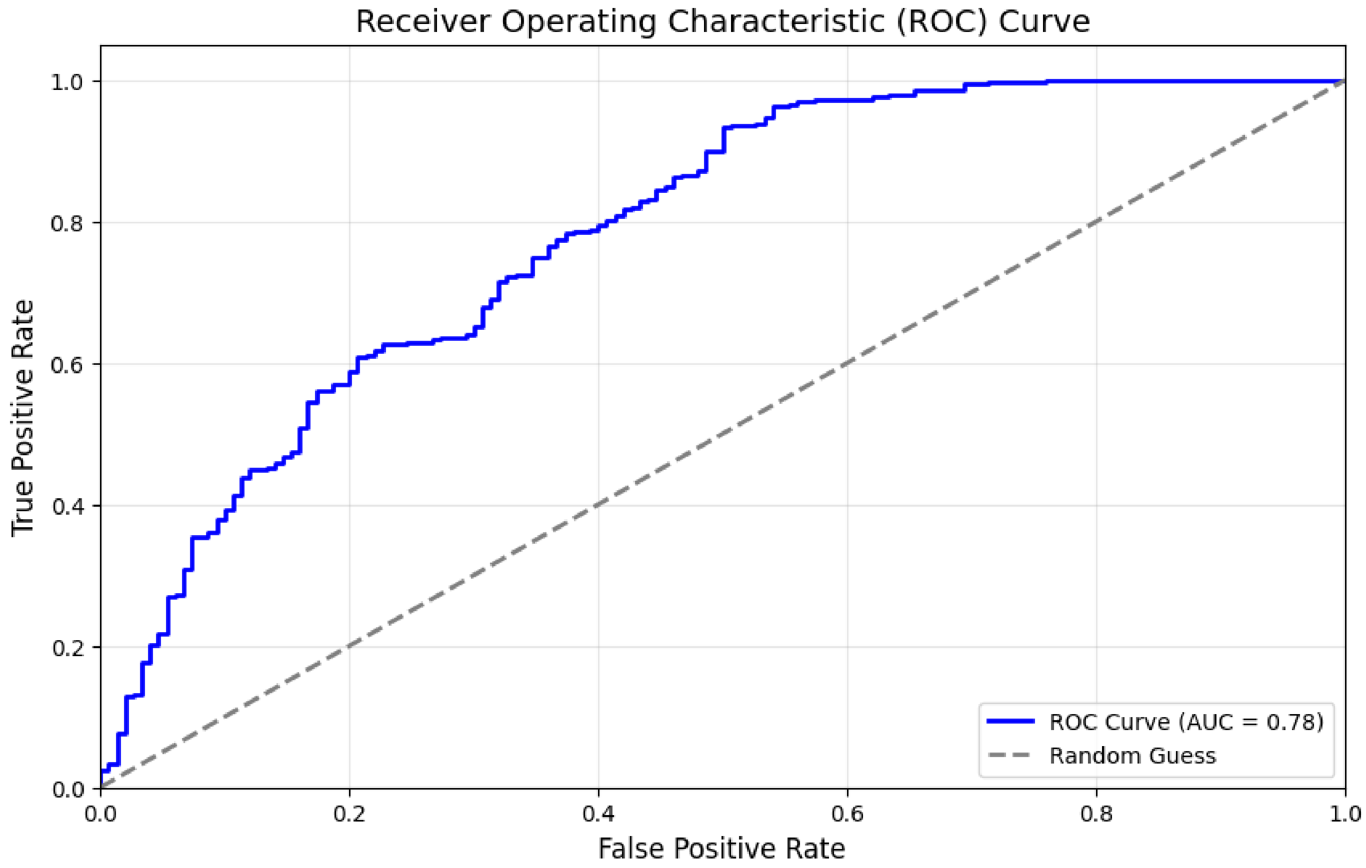
```
Training with XGB Classifier with SMOTE upsampled data
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Time taken for GridSearchCV: 0:01:19.040176

Best parameters found: {'colsample_bytree': 0.8, 'learning_rate': 0.2, 'max_depth': 7,
'n_estimators': 50, 'subsample': 1.0}
Best cross-validated accuracy: 0.8317

Training Accuracy: 0.9597

Training Confusion Matrix:
[[1199   90]
 [  14 1275]]
```




```
Testing Accuracy: 0.7610

Testing Confusion Matrix:
[[ 78  72]
 [ 42 285]]
```



```
Classification Report for Testing Data:
              precision    recall  f1-score   support

           0       0.65      0.52      0.58       150
           1       0.80      0.87      0.83       327

    accuracy                           0.76       477
   macro avg       0.72      0.70      0.71       477
weighted avg       0.75      0.76      0.75       477


ROC AUC Score: 0.78
```




```
Feature Importances:
                              Feature  Importance
10               latest_quarter_rating    0.393703
1                               gender    0.114595
11    quarterly_rating_growth_indicator    0.103521
6                  joining_designation    0.075072
9                 first_quarter_rating    0.073001
8                 total_business_value    0.041259
7                                grade    0.039201
0                                  age    0.037312
4                         first_income    0.030216
3                           avg_income    0.028246
2                      education_level    0.023045
13                        city_encoded    0.022522
5                          last_income    0.018306
12             income_growth_indicator    0.000000
```

In [43]:

```
print(f'Last edited on {datetime.datetime.now()}')
```

```
Last edited on 2025-01-28 17:16:53.411406
```