About OLA 🚗

Ola is India’s largest mobility platform and one of the world’s largest ride-hailing companies, serving 250+ cities across India, Australia, New Zealand, and the UK. The Ola app offers mobility solutions by connecting customers to drivers and a wide range of vehicles across bikes, auto-rickshaws, metered taxis, and cabs, enabling convenience and transparency for hundreds of millions of consumers and over 1.5 million driver-partners.

Business Problem 💡

Recruiting and retaining drivers is seen by industry watchers as a tough battle for Ola. Churn among drivers is high and it’s very easy for drivers to stop working for the service on the fly or jump to Uber depending on the rates.

As the companies get bigger, the high churn could become a bigger problem. To find new drivers, Ola is casting a wide net, including people who don’t have cars for jobs. But this acquisition is really costly. Losing drivers frequently impacts the morale of the organization and acquiring new drivers is more expensive than retaining existing ones.

We are provided with the monthly information for a segment of drivers for 2019 and 2020 and tasked to predict whether a driver will be leaving the company or not based on their attributes like:

- Demographics (city, age, gender etc.)
- Tenure information (joining date, Last Date)
- Historical data regarding the performance of the driver (Quarterly rating, Monthly business acquired, grade, Income)

Dataset 📊

Column Name	Description
MMMM-YY	Reporting Date (Monthly)
Driver_ID	Unique id for drivers
Age	Age of the driver

Column Name	Description
Gender	Gender of the driver – Male : 0, Female: 1
City	City Code of the driver
Education_Level	Education level – 0 for 10+, 1 for 12+, 2 for graduate
Income	Monthly average Income of the driver
Date Of Joining	Joining date for the driver
LastWorkingDate	Last date of working for the driver
Joining Designation	Designation of the driver at the time of joining
Grade	Grade of the driver at the time of reporting
Total Business Value	The total business value acquired by the driver in a month (negative business indicates cancellation/refund or car EMI adjustments)
Quarterly Rating	Quarterly rating of the driver: 1, 2, 3, 4, 5 (higher is better)

Importing Required Libraries 🤝

In [1]:
```
import pandas as pd
import numpy as np

from matplotlib import pyplot as plt
import seaborn as sns
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, 
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.impute import KNNImputer

from imblearn.over_sampling import SMOTE
from collections import Counter

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

In [2]:
```
palette = ['#858585', '#D6DF22', '#EFEFEF', '#000000']
```

Read Dataset 🔍

In [3]:
```
df = pd.read_csv(r'../data/ola_driver_scaler.csv', index_col=0)
df.head()
```

Out[3]:

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

In [4]:
```
print("Shape of the data: ", df.shape)
print("The Given Dataset has {} rows and {} columns".format(df.shape[0], df.shape[1]))
print("Columns: ", df.columns.to_list())
```

```
Shape of the data:  (19104, 13)
The Given Dataset has 19104 rows and 13 columns
Columns:  ['MMM-YY', 'Driver_ID', 'Age', 'Gender', 'City', 'Education_Level', 'Income', 'Dateofjo
ining', 'LastWorkingDate', 'Joining Designation', 'Grade', 'Total Business Value', 'Quarterly Rat
ing']
```

Shape 🕵️

- The dataset comprises 19104 rows and 13 columns, representing a volume of data.
- Each row corresponds to monthly information for a segment of drivers.

Data Structure 📰

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 19104 entries, 0 to 19103
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   MMM-YY                19104 non-null  object 
 1   Driver_ID             19104 non-null  int64  
 2   Age                   19043 non-null  float64
 3   Gender                19052 non-null  float64
 4   City                  19104 non-null  object 
 5   Education_Level       19104 non-null  int64  
 6   Income                19104 non-null  int64  
 7   Dateofjoining         19104 non-null  object 
 8   LastWorkingDate       1616 non-null   object 
 9   Joining Designation   19104 non-null  int64  
 10  Grade                 19104 non-null  int64  
 11  Total Business Value  19104 non-null  int64  
 12  Quarterly Rating      19104 non-null  int64  
dtypes: float64(2), int64(7), object(4)
memory usage: 2.0+ MB
```

In [6]:
```
df.isnull().sum()
```

```
Out[6]: 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
```

🕵️ Dataset Information:

- **Data Consistency**: Age and Gender columns has values in the dataset. LastWorkingDate column has 17488 rows indicating there are still working for the organisation.
- **Data Types**: Columns are classified into integer, float and object types. DateTime columns are stored as Objects types.

```
In [7]: # # Missing Value Imputation - KNN Imputer
        # imputer = KNNImputer(n_neighbors=3)
        # df[['Age', 'Gender']] = imputer.fit_transform(df[['Age', 'Gender']])
        # df.isnull().sum()
```

```
In [8]: df['Age'] = df.groupby(['Driver_ID'])['Age'].transform(lambda x: x.fillna(x.mean()))
        df['Gender'] = df.groupby(['Driver_ID'])['Gender'].transform(lambda x: x.fillna(x.mean()))
```

```
In [9]: df['MMM-YY'] = pd.to_datetime(df['MMM-YY'])
        df['Dateofjoining'] = pd.to_datetime(df['Dateofjoining'])
        df['LastWorkingDate'] = pd.to_datetime(df['LastWorkingDate'])

        df['Age'] = df['Age'].astype('int64')
        df['Gender'] = df['Gender'].astype('int64').astype('category')
        df['Education_Level'] = df['Education_Level'].astype('category')
        df['Joining Designation'] = df['Joining Designation'].astype('category')
        df['City'] = df['City'].astype('category')

        df['is_churn'] = df['LastWorkingDate'].apply(lambda x: 1 if pd.isnull(x) else 0)
        # df['is_churn'].value_counts()
        df['is_churn'] = df['is_churn'].astype('int64')


        df.dtypes
```

Out[9]:
```
MMM-YY                    datetime64[ns]
Driver_ID                          int64
Age                                int64
Gender                          category
City                            category
Education_Level                 category
Income                             int64
Dateofjoining             datetime64[ns]
LastWorkingDate           datetime64[ns]
Joining Designation             category
Grade                              int64
Total Business Value               int64
Quarterly Rating                   int64
is_churn                           int64
dtype: object
```

🕵️ Missing Column Imputations:

- **KNN Imputation**: Age and Gender are imputed with the KNN.
- **Data Types**: Convert date-like features to their respective data type.

In [10]:
```python
df.describe().T
```

Out[10]:

	count	mean	min	25%	50%	75%	max	
MMM-YY	19104	2019-12-11 02:09:29.849246464	2019-01-01 00:00:00	2019-06-01 00:00:00	2019-12-01 00:00:00	2020-07-01 00:00:00	2020-12-01 00:00:00	
Driver_ID	19104.0	1415.591133	1.0	710.0	1417.0	2137.0	2788.0	8
Age	19104.0	34.64955	21.0	30.0	34.0	39.0	58.0	
Income	19104.0	65652.025126	10747.0	42383.0	60087.0	83969.0	188418.0	309
Dateofjoining	19104	2018-04-28 20:52:54.874371840	2013-04-01 00:00:00	2016-11-29 12:00:00	2018-09-12 00:00:00	2019-11-05 00:00:00	2020-12-28 00:00:00	
LastWorkingDate	1616	2019-12-21 20:59:06.534653696	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	
Grade	19104.0	2.25267	1.0	1.0	2.0	3.0	5.0	
Total Business Value	19104.0	571662.074958	-6000000.0	0.0	250000.0	699700.0	33747720.0	11283
Quarterly Rating	19104.0	2.008899	1.0	1.0	2.0	3.0	4.0	
is_churn	19104.0	0.91541	0.0	1.0	1.0	1.0	1.0	

In [11]:
```python
df.describe(include='category').T
```

Out[11]:

	count	unique	top	freq
Gender	19104	2	0	11103
City	19104	29	C20	1008
Education_Level	19104	3	1	6864
Joining Designation	19104	5	1	9831

🕵️ Statistical Information:

- **Count**: All columns have the same count, indicating no missing values in the dataset.
- **datetime column**: The data provided are within the dates 2019-01-01 to 2020-12-01
- **Age column**: The driver age ranges from 21 years to 58 years with the mean temparature of 35
- **Income column**: The income ranges from 10747 to 188418 with the mean of 65652.

Preprocessing ⚙️

In [12]:
```
df['LastWorkingDate'].fillna(pd.to_datetime('2020-12-31'), inplace=True)
df['tenture'] = (df['LastWorkingDate'] - df['Dateofjoining']).dt.days // 30
df['tenture'] = df['tenture'].astype('int64')

df.drop(['LastWorkingDate'], axis=1, inplace=True)
```

In [13]:
```
# Create a column which tells whether the monthly income has increased for that driver - for thos

df['monthly_income_increase'] = df.groupby('Driver_ID')['Income'].diff().apply(lambda x: 1 if x 
df.head()
```

Out[13]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	Joining Designation	Grade	B
0	2019-01-01	1	28	0	C23	2	57387	2018-12-24	1	1	2
1	2019-02-01	1	28	0	C23	2	57387	2018-12-24	1	1	-
2	2019-03-01	1	28	0	C23	2	57387	2018-12-24	1	1	
3	2020-11-01	2	31	0	C7	2	67016	2020-11-06	2	2	
4	2020-12-01	2	31	0	C7	2	67016	2020-11-06	2	2	

In [14]:
```
# Create a column which tells whether the quarterly rating has increased for that driver - for th
df['Quarter'] = df['MMM-YY'].dt.year.astype(str) + "-" + df['MMM-YY'].dt.quarter.astype(str)
df['quarterly_rating_increase'] = df.groupby(['Driver_ID', 'Quarter'])['Quarterly Rating'].diff(

# # Update the quarterly_rating_increase field for all rows in that Quarter
```

```
# df['quarterly_rating_increase'] = df.groupby(['Driver_ID', 'Quarter'])['quarterly_rating_incre

df.head()
```

Out[14]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	Joining Designation	Grade	B
0	2019-01-01	1	28	0	C23	2	57387	2018-12-24	1	1	2
1	2019-02-01	1	28	0	C23	2	57387	2018-12-24	1	1	-
2	2019-03-01	1	28	0	C23	2	57387	2018-12-24	1	1	
3	2020-11-01	2	31	0	C7	2	67016	2020-11-06	2	2	
4	2020-12-01	2	31	0	C7	2	67016	2020-11-06	2	2	

In [52]:

```
# driver_with_rating_increase = df[df['quarterly_rating_increase']==1]['Driver_ID'].values[0]
# df[df['Driver_ID']==driver_with_rating_increase]
```

Out[52]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	Joining Designation	Grade
138	2019-01-01	26	41	0	C14	2	121529	2018-05-07	1	3
139	2019-02-01	26	41	0	C14	2	121529	2018-05-07	1	3
140	2019-03-01	26	41	0	C14	2	121529	2018-05-07	1	3
141	2019-04-01	26	41	0	C14	2	121529	2018-05-07	1	3
142	2019-05-01	26	41	0	C14	2	121529	2018-05-07	1	3
143	2019-06-01	26	41	0	C14	2	121529	2018-05-07	1	3
144	2019-07-01	26	41	0	C14	2	121529	2018-05-07	1	3
145	2019-08-01	26	41	0	C14	2	121529	2018-05-07	1	3
146	2019-09-01	26	42	0	C14	2	121529	2018-05-07	1	3
147	2019-10-01	26	42	0	C14	2	121529	2018-05-07	1	3
148	2019-11-01	26	42	0	C14	2	121529	2018-05-07	1	3
149	2019-12-01	26	42	0	C14	2	121529	2018-05-07	1	3
150	2020-01-01	26	42	0	C14	2	121529	2018-05-07	1	3
151	2020-02-01	26	42	0	C14	2	121529	2018-05-07	1	3
152	2020-03-01	26	42	0	C14	2	132577	2018-05-07	1	4
153	2020-04-01	26	42	0	C14	2	132577	2018-05-07	1	4
154	2020-05-01	26	42	0	C14	2	132577	2018-05-07	1	4
155	2020-06-01	26	42	0	C14	2	132577	2018-05-07	1	4
156	2020-07-01	26	42	0	C14	2	132577	2018-05-07	1	4
157	2020-08-01	26	42	0	C14	2	132577	2018-05-07	1	4
158	2020-09-01	26	43	0	C14	2	132577	2018-05-07	1	4

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	Joining Designation	Grade
159	2020-10-01	26	43	0	C14	2	132577	2018-05-07	1	4
160	2020-11-01	26	43	0	C14	2	132577	2018-05-07	1	4
161	2020-12-01	26	43	0	C14	2	132577	2018-05-07	1	4

In [16]:
```
df.sample(5)
```

Out[16]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	Joining Designation	Grad
14283	2019-06-01	2134	38	0	C29	0	116006	2013-10-30	2	
9869	2020-10-01	1468	31	0	C20	1	66779	2020-09-01	3	
18931	2020-07-01	2761	28	0	C20	2	131805	2020-07-17	3	
18707	2020-11-01	2729	35	0	C10	0	107274	2019-08-04	4	
10930	2019-01-01	1635	29	0	C20	0	13417	2018-12-31	1	

In [51]:
```
# # df[df['Age'].isna()]
# df[df['Driver_ID']==20]
```

Out[51]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	Joining Designation	Grade
68	2019-10-01	20	26	1	C19	0	40342	2019-10-25	3	3
69	2019-11-01	20	26	1	C19	0	40342	2019-10-25	3	3
70	2019-12-01	20	26	1	C19	0	40342	2019-10-25	3	3
71	2020-01-01	20	26	1	C19	0	40342	2019-10-25	3	3
72	2020-02-01	20	26	1	C19	0	40342	2019-10-25	3	3
73	2020-03-01	20	26	1	C19	0	40342	2019-10-25	3	3

In [18]:
```
df.isna().sum()
```

Out[18]:
```
MMM-YY                     0
Driver_ID                  0
Age                        0
Gender                     0
City                       0
Education_Level            0
Income                     0
Dateofjoining              0
Joining Designation        0
Grade                      0
Total Business Value       0
Quarterly Rating           0
is_churn                   0
tenture                    0
monthly_income_increase    0
Quarter                    0
quarterly_rating_increase  0
dtype: int64
```

In [19]:
```
df_processed = df.groupby('Driver_ID').agg(
    {
        'Age': 'last',
        'Gender': 'last',
        'City': 'last',
        'Education_Level': 'last',
        'Income': 'mean',
        'Dateofjoining': 'last',
        'Joining Designation': 'last',
        'Grade': 'last',
        'Total Business Value': 'sum',
        'is_churn': 'last',
        'monthly_income_increase': 'sum',
        'quarterly_rating_increase': 'sum',
        'tenture': 'last'
        }
    ).reset_index()
```

In [20]:
```
df_processed.describe().T
```

Out[20]:

	count	mean	min	25%	50%	75%	max
Driver_ID	2381.0	1397.559009	1.0	695.0	1400.0	2100.0	2788.0
Age	2381.0	33.661907	21.0	29.0	33.0	37.0	58.0
Income	2381.0	59232.460484	10747.0	39104.0	55285.0	75835.0	188418.0
Dateofjoining	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
Grade	2381.0	2.096598	1.0	1.0	2.0	3.0	5.0
Total Business Value	2381.0	4586741.822764	-1385530.0	0.0	817680.0	4173650.0	95331060.0
is_churn	2381.0	0.321294	0.0	0.0	0.0	1.0	1.0
monthly_income_increase	2381.0	0.01848	0.0	0.0	0.0	0.0	1.0
quarterly_rating_increase	2381.0	0.00336	0.0	0.0	0.0	0.0	1.0
tenture	2381.0	14.104998	0.0	3.0	6.0	16.0	94.0

In [21]:
```
print("Shape of the processed data: ", df_processed.shape)
print("The Processed Dataset has {} rows and {} columns".format(df_processed.shape[0], df_proces
```

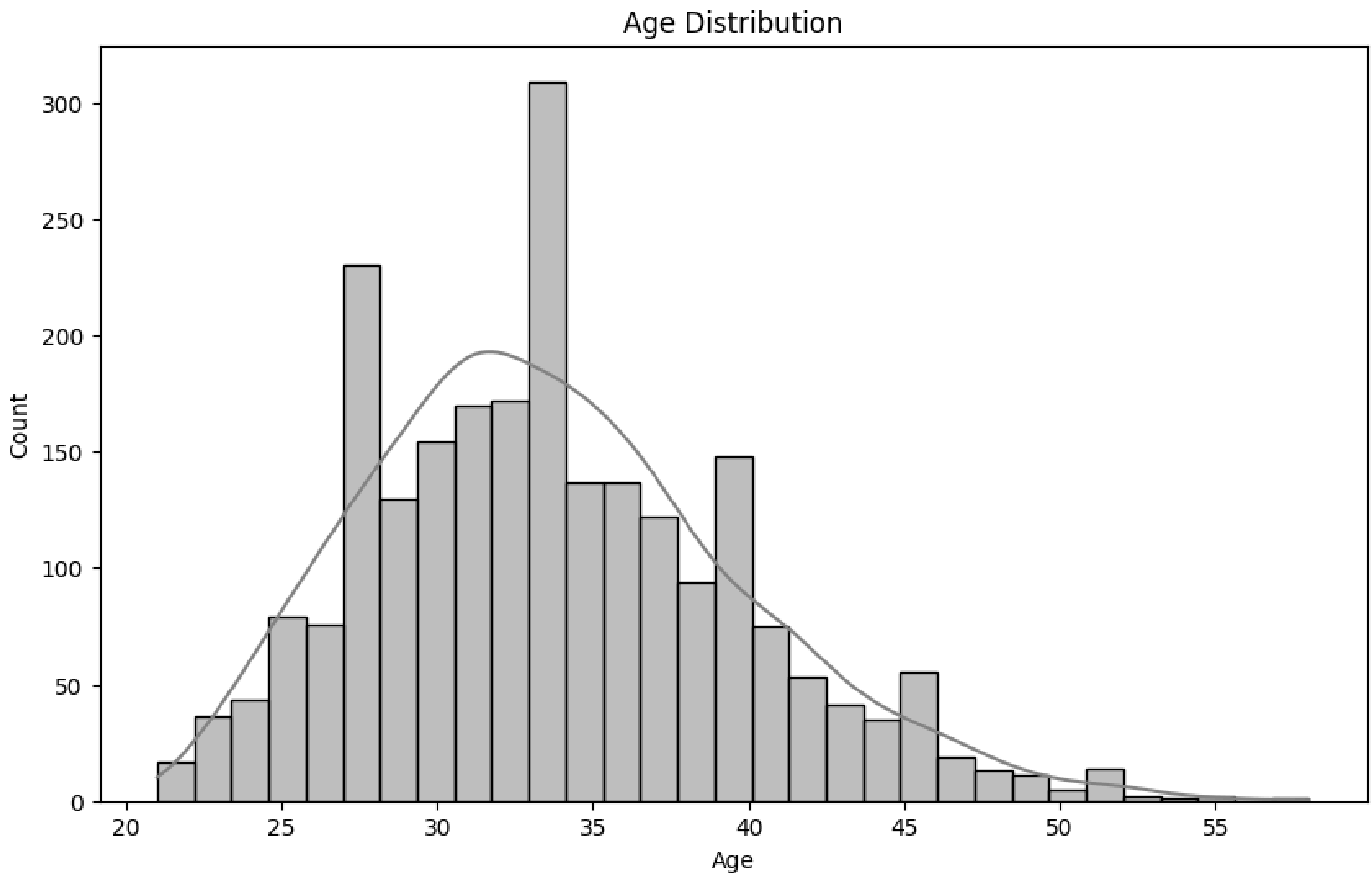
```
Shape of the processed data:  (2381, 14)
The Processed Dataset has 2381 rows and 14 columns
```

Shape 🕵️

- The dataset comprises 2381 rows and 13 columns, representing a volume of preprocessed data.
- Each row corresponds to information for each drivers.

Exploratory Data Analysis (EDA) 💹

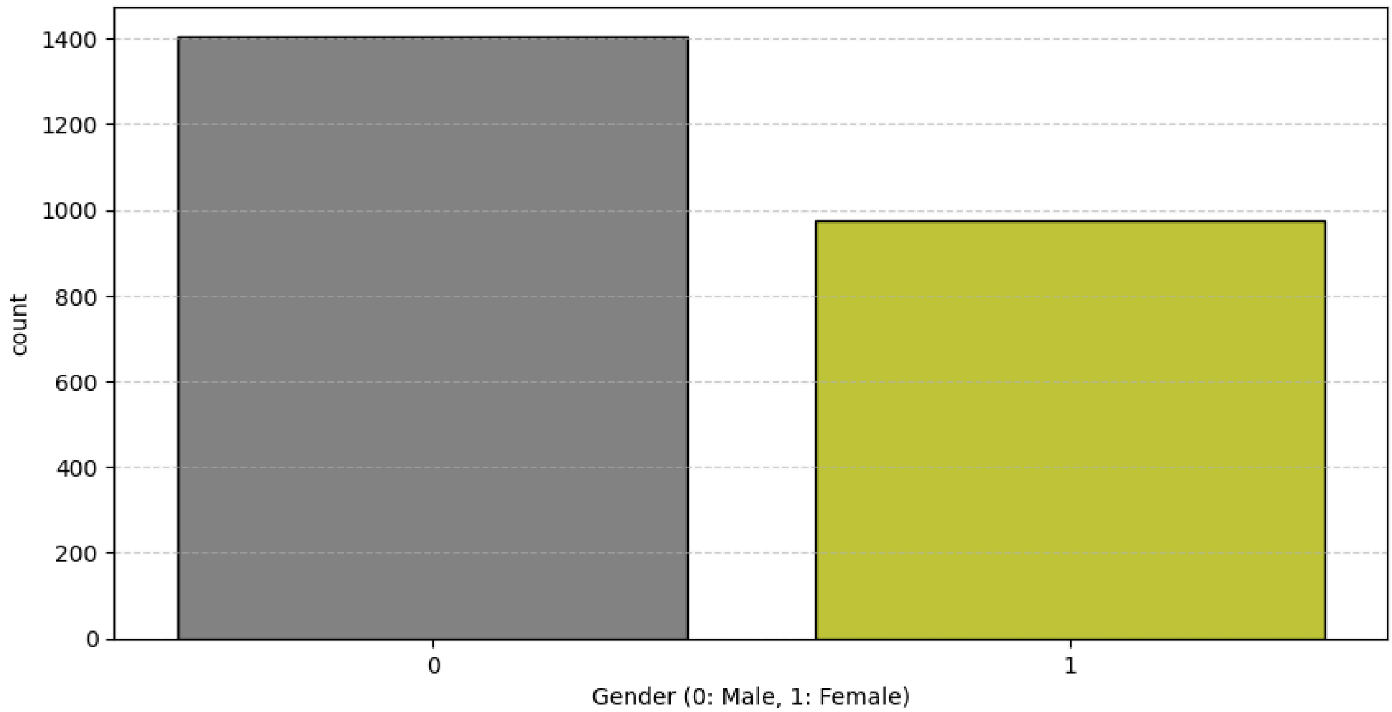
In [22]:
```python
# Age distribution
plt.figure(figsize=(10, 6))
sns.histplot(df_processed['Age'], kde=True, color=palette[0])
plt.title('Age Distribution')
plt.show()
```



Insight 🕵🏽

- The ages of individuals in the dataset range from approximately 20 to 55 years.
- The distribution appears to be right-skewed, with the majority of individuals clustered around the ages of 30 to 40 years.
- The peak of the distribution is around the age of 35, indicating that this is the most common age group in the dataset.
- The number of individuals decreases steadily for ages above 40, with very few individuals above the age of 50.

In [23]:
```python
# Gender distribution
plt.figure(figsize=(10, 5))
sns.countplot(df_processed, x='Gender', palette=palette, edgecolor='black')
plt.xlabel('Gender (0: Male, 1: Female)')
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.show()

print(df_processed['Gender'].value_counts(normalize=True))
```

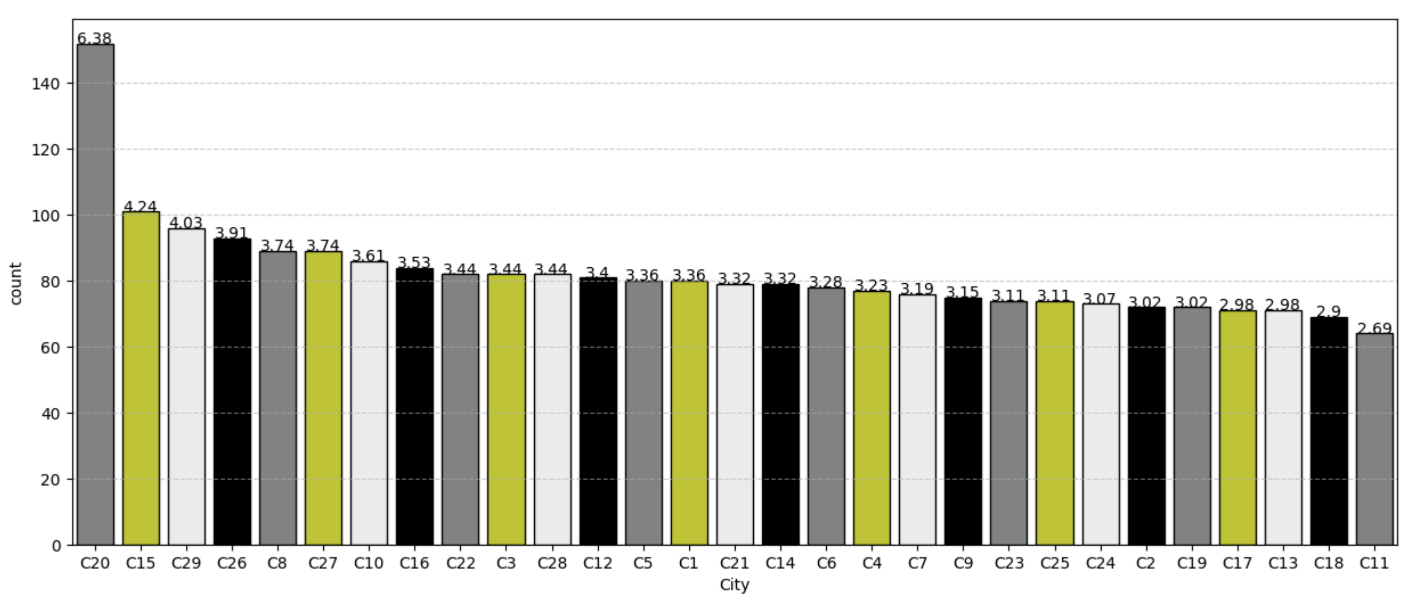



```
Gender
0    0.589668
1    0.410332
Name: proportion, dtype: float64
```

Insight 🕵️

- The distribution of gender, around 59% of drivers are Male, 41% are Females.

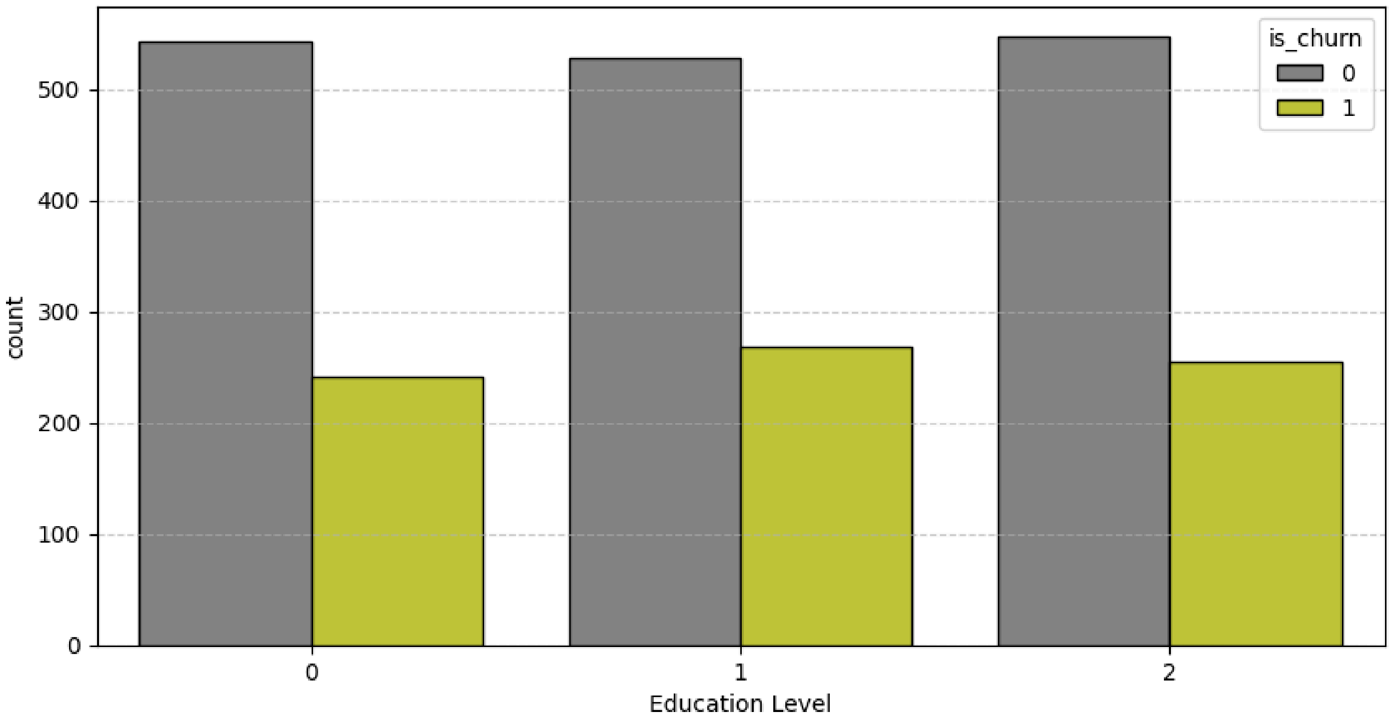
In [24]:
```python
# City distribution
plt.figure(figsize=(15, 6))
sns.countplot(df_processed, x='City', palette=palette, order=df_processed['City'].value_counts()
plt.xlabel('City')
plt.grid(axis='y', linestyle='--', alpha=0.6)

for i in range(len(df_processed['City'].value_counts())):
    plt.text(i, df_processed['City'].value_counts()[i], round(df_processed['City'].value_counts(

plt.show()
```



Insight 🕵🏽

- Around 6.4% of the driver from city code C20.
- Rest of the city has driver representating 2.5% to 4.2%

In [25]:
```
# Education Level distribution
plt.figure(figsize=(10, 5))
sns.countplot(df_processed, x='Education_Level', hue='is_churn' ,palette=palette, edgecolor='bla
plt.xlabel('Education Level')
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.show()

df_processed['Education_Level'].value_counts(normalize=True)
```

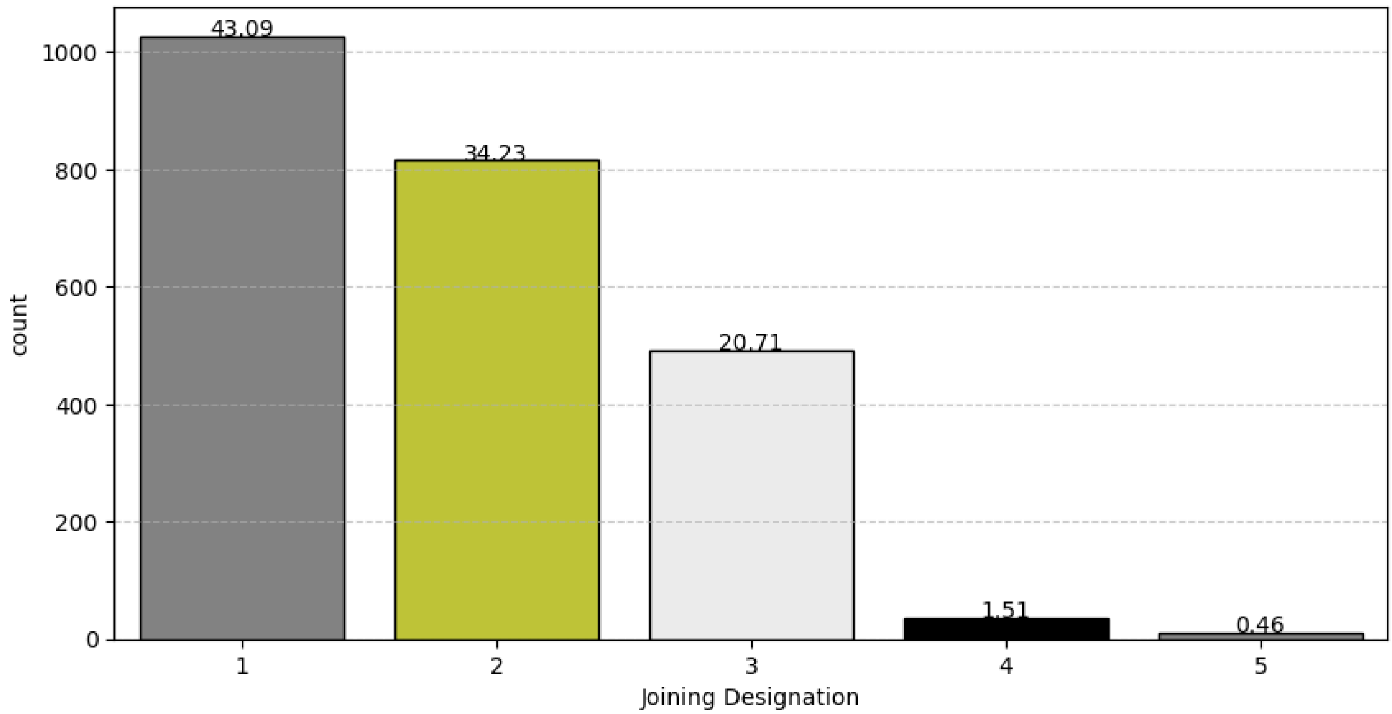


Out[25]:
```
Education_Level
2    0.336833
1    0.333893
0    0.329273
Name: proportion, dtype: float64
```

Insight 🕵🏽

- Almost 33% percentages of drivers has in each education level

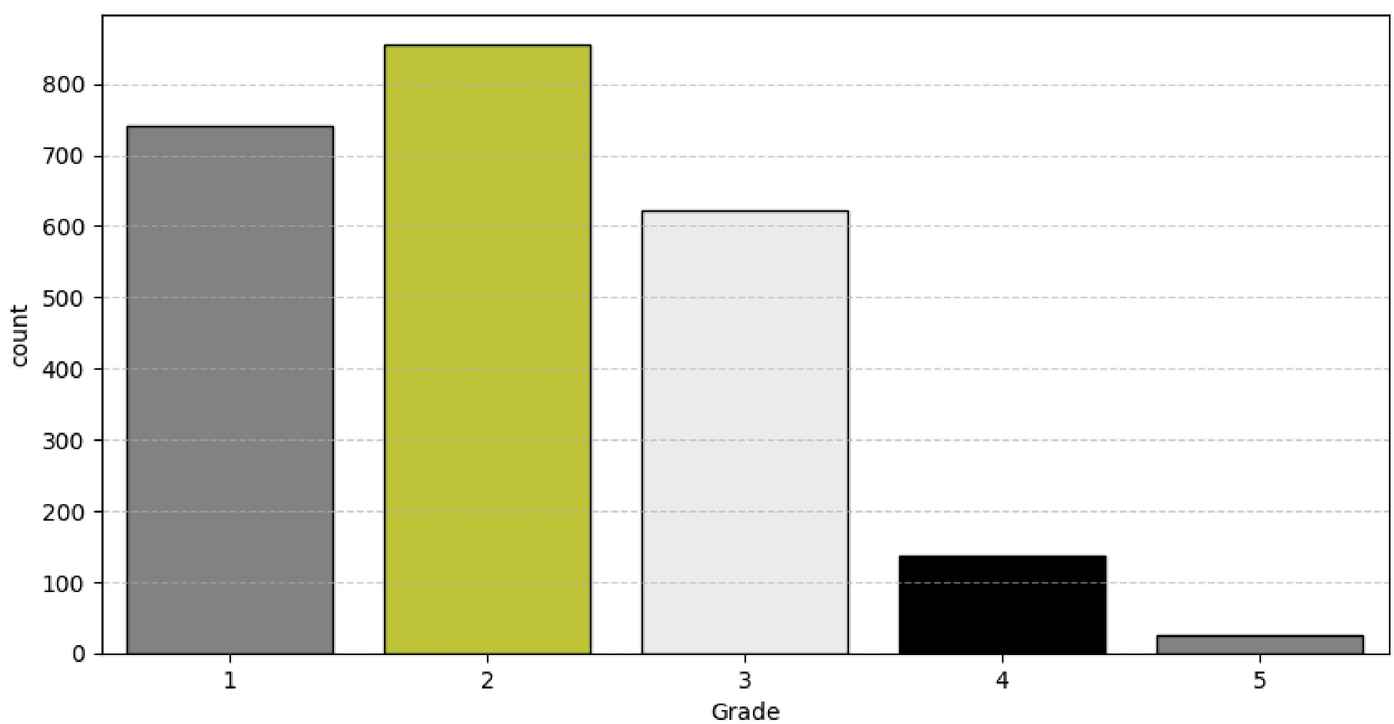
In [26]:
```
# Joining Designation distribution
plt.figure(figsize=(10, 5))
sns.countplot(df_processed, x='Joining Designation', palette=palette, order=df_processed['Joinin
plt.xlabel('Joining Designation')
plt.grid(axis='y', linestyle='--', alpha=0.6)

for i, val in enumerate(df_processed['Joining Designation'].value_counts().index):
    count = df_processed['Joining Designation'].value_counts()[val]
    percentage = round(df_processed['Joining Designation'].value_counts(normalize=True)[val] * 10
    plt.text(i, count, percentage, ha='center')

plt.show()
```



Insight 🕵🏽‍♂️

- Joining Designation distribution is around 43% with Designation 1 and ~1.5% with Designation 4 and 0.5% with Designation 5.
- Joining Designation distribution indicate that only ~2% joins with higher designation.

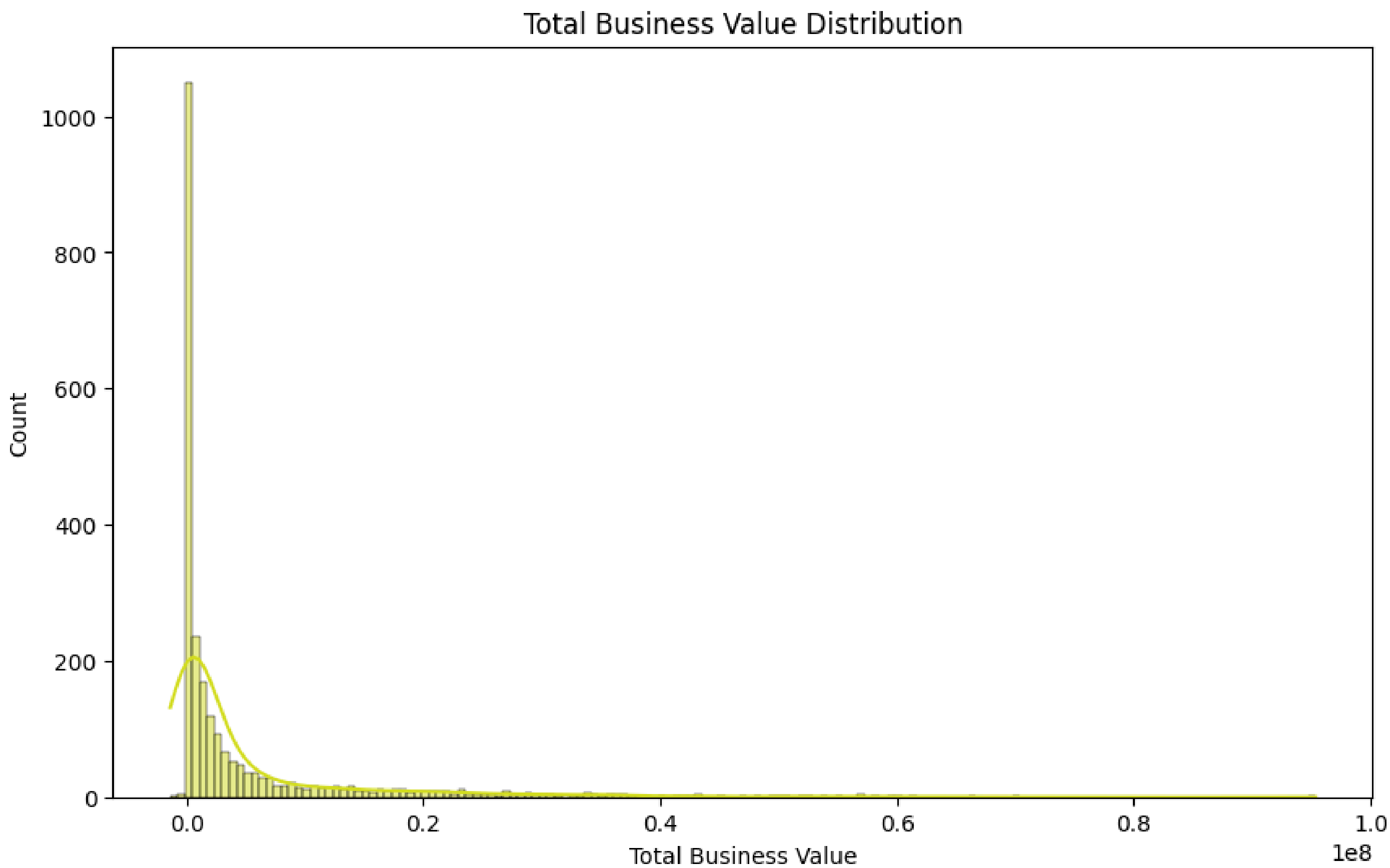
In [27]:
```
# Grade distribution
plt.figure(figsize=(10, 5))
sns.countplot(df_processed, x='Grade', palette=palette, edgecolor='black')
plt.xlabel('Grade')
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.show()
```



Insight 🕵🏽‍♂️

- More number of driver falls with Grade 2.
- Driver in grade 4 and 5 are relavatively lower than other grades

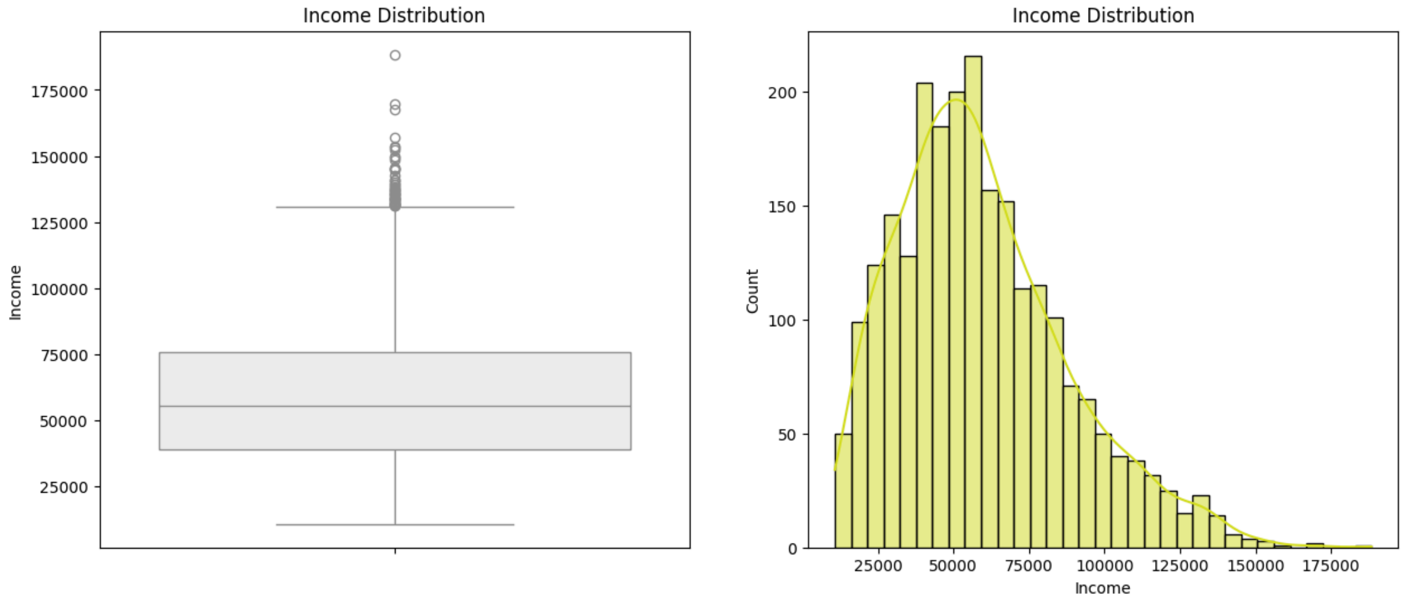
In [28]:
```
# Total Business Value distribution
plt.figure(figsize=(10, 6))
sns.histplot(df_processed['Total Business Value'], kde=True, color=palette[1])
plt.title('Total Business Value Distribution')
plt.show()
```



Insight 🕵🏽‍♂️

- The distribution is highly right-skewed. This indicates that a large number of users generate relatively low business value, while a small number of users generate very high business value.
- The majority of users generate a total business value close to zero.

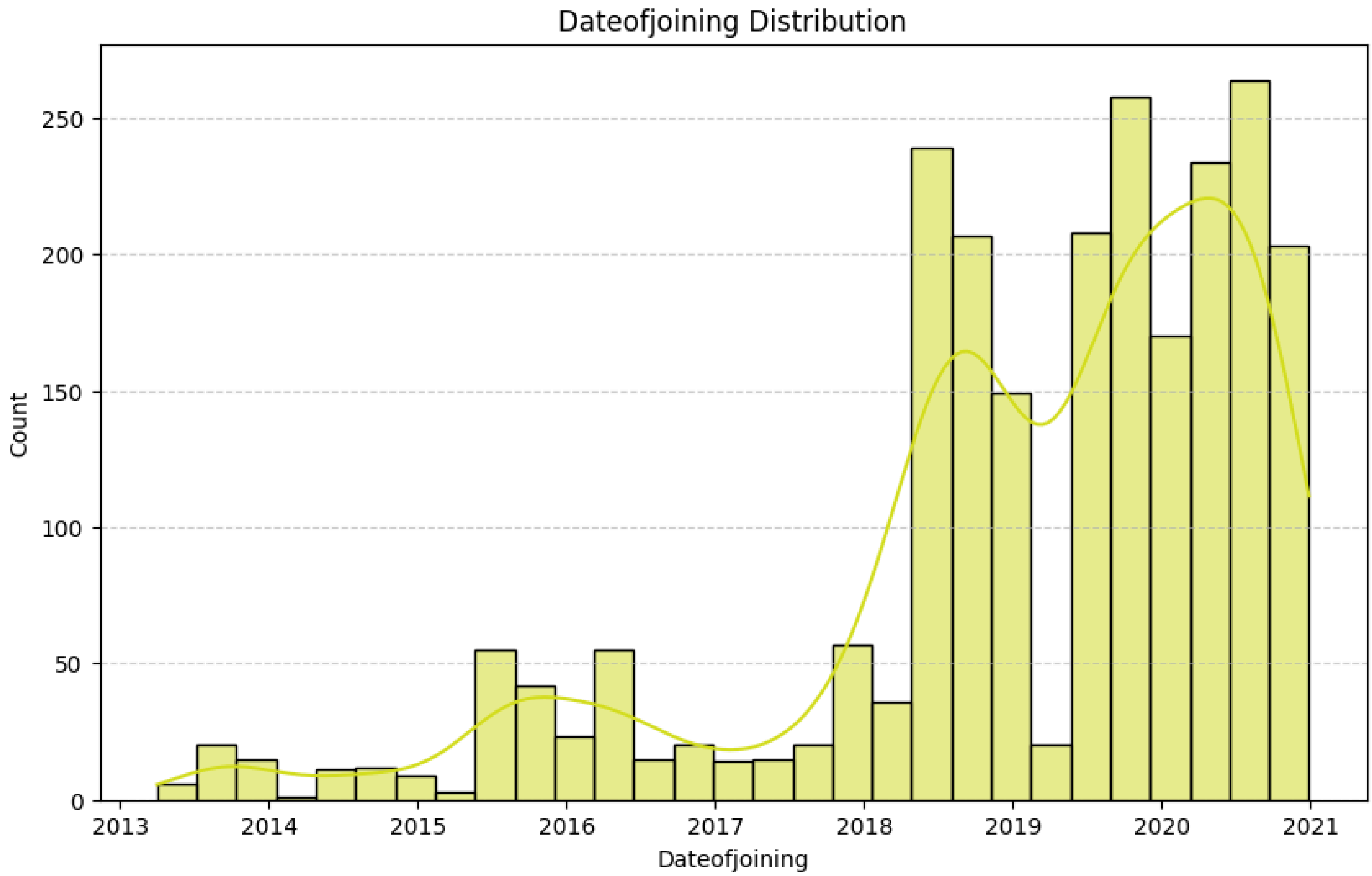
In [29]:
```
# Income distribution using boxplot and histogram
plt.figure(figsize=(15, 6))
plt.subplot(1, 2, 1)
sns.boxplot(df_processed['Income'], color=palette[2])
plt.title('Income Distribution')

plt.subplot(1, 2, 2)
sns.histplot(df_processed['Income'], kde=True, color=palette[1])
plt.title('Income Distribution')
plt.show()
```



Insight 🕵️

- The median income is approximately 50,000, as indicated by the line inside the box.
- The IQR, which represents the middle 50% of the data, ranges from approximately 30,000 to 75,000. This suggests that most incomes are within this range.
- The lower whisker extends to around 10,000, showing the minimum income in the dataset. The upper whisker extends to about 125,000, indicating the maximum income before outliers.
- There are several outliers above the upper whisker, indicating that some individuals have significantly higher incomes than the rest. These outliers start appearing above approximately 125,000.

In [30]:
```
# Dateofjoining distribution
plt.figure(figsize=(10, 6))
sns.histplot(df_processed['Dateofjoining'], kde=True, color=palette[1])
plt.title('Dateofjoining Distribution')
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.show()
```

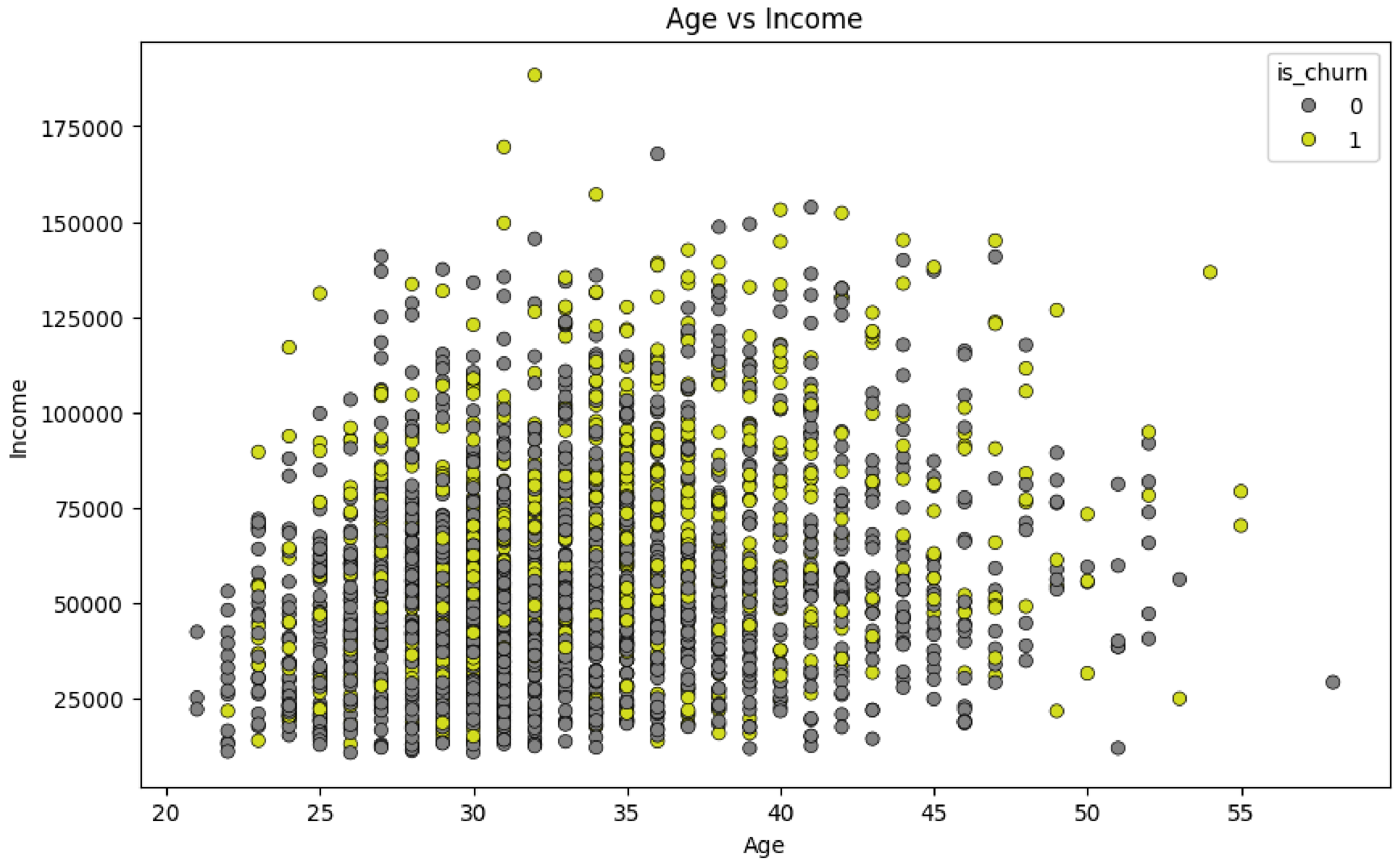


Insight 🕵️

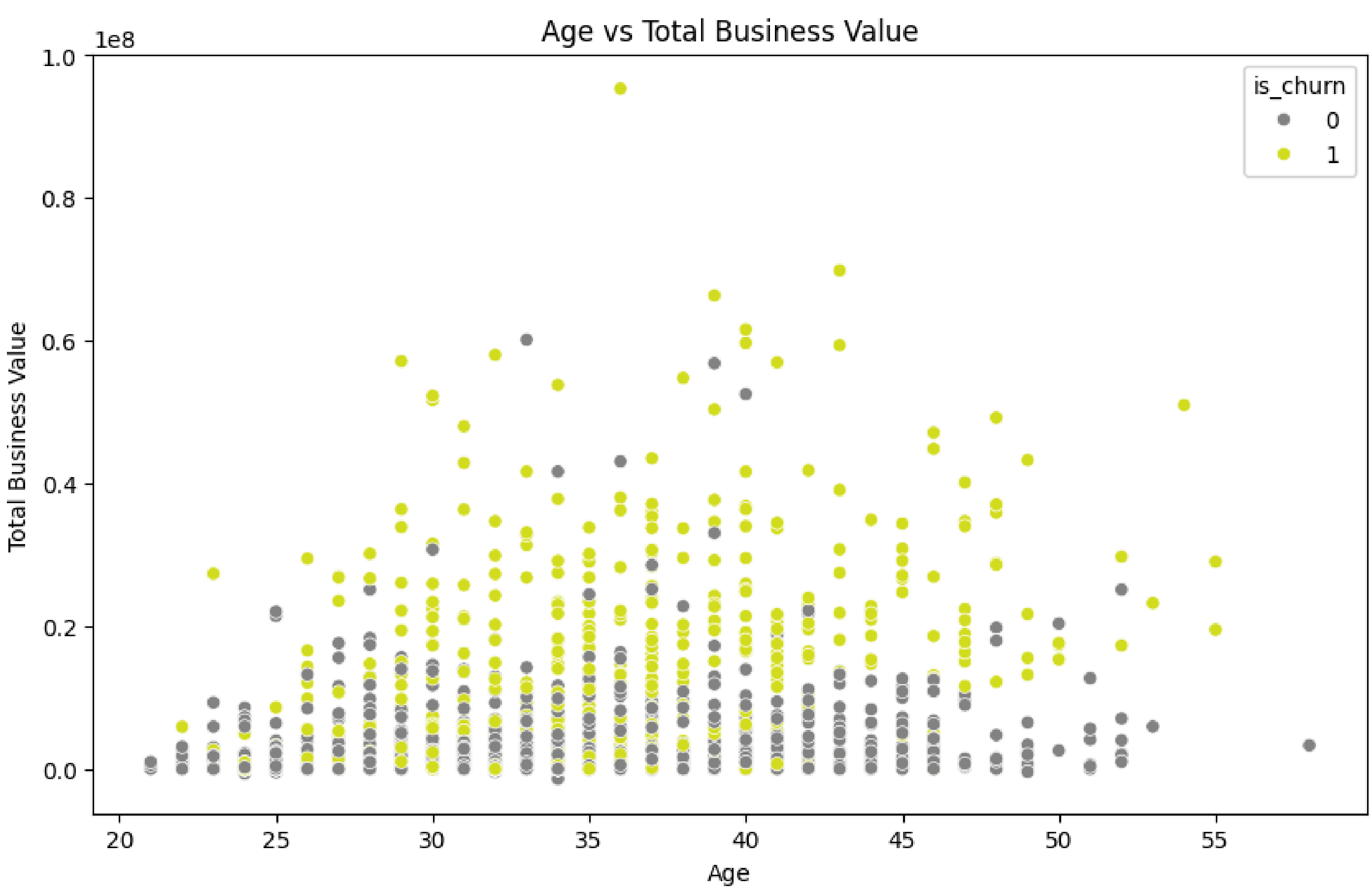
- There is significant number of driver joined after mid 2018.
- 2020 Q3 has highest number of driver onboarded.

In [31]:

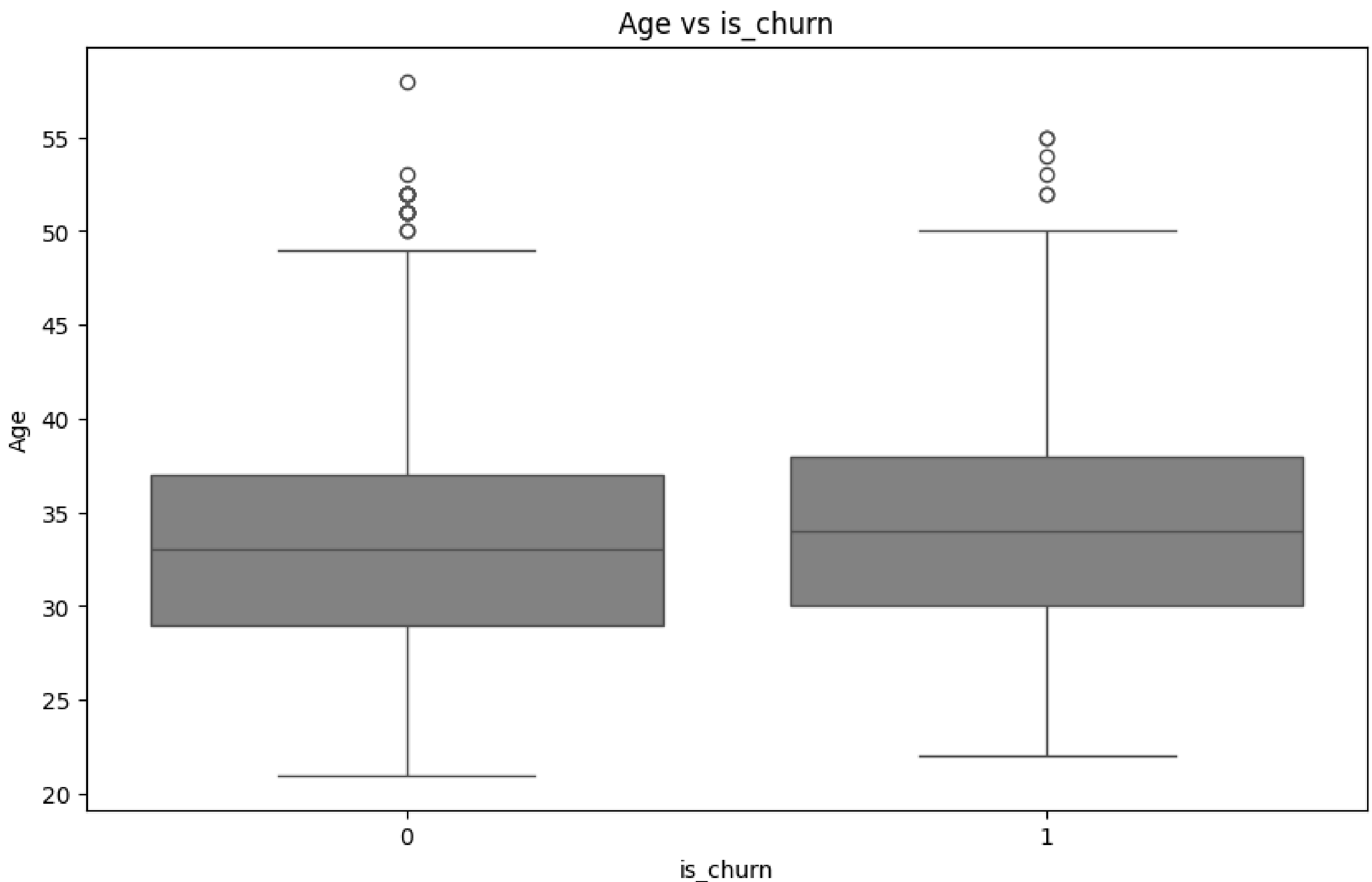
```
# Age vs Income
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_processed, x='Age', y='Income', hue='is_churn', palette=palette, edgecol
plt.title('Age vs Income')
plt.show()
```



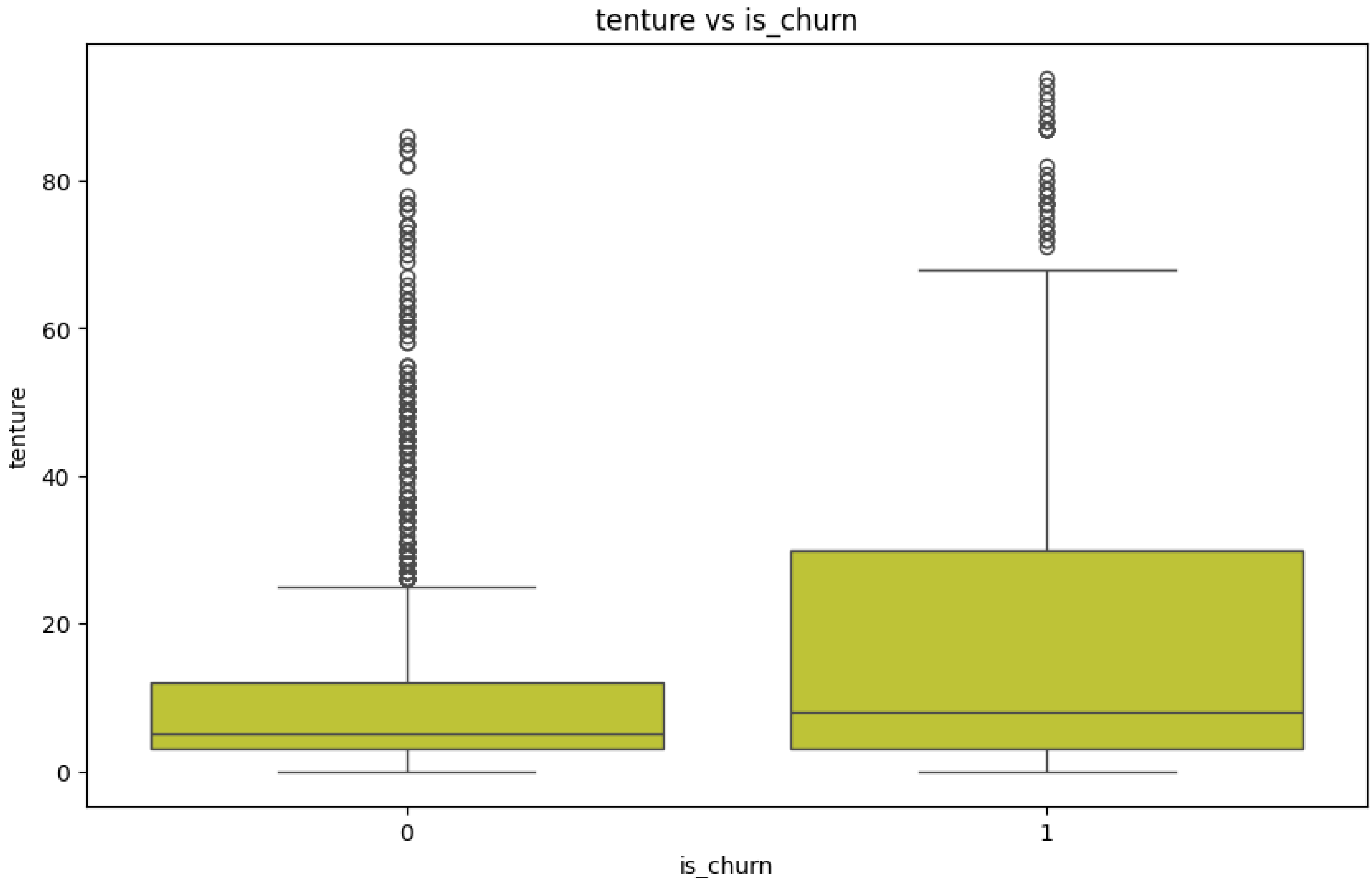
In [32]:
```
# Age vs Total Business Value
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_processed, x='Age', y='Total Business Value', hue='is_churn', palette=pa
plt.title('Age vs Total Business Value')
plt.show()
```



In [33]:
```
# Age vs is_churn
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_processed, x='is_churn', y='Age', color=palette[0])
plt.title('Age vs is_churn')
plt.show()
```



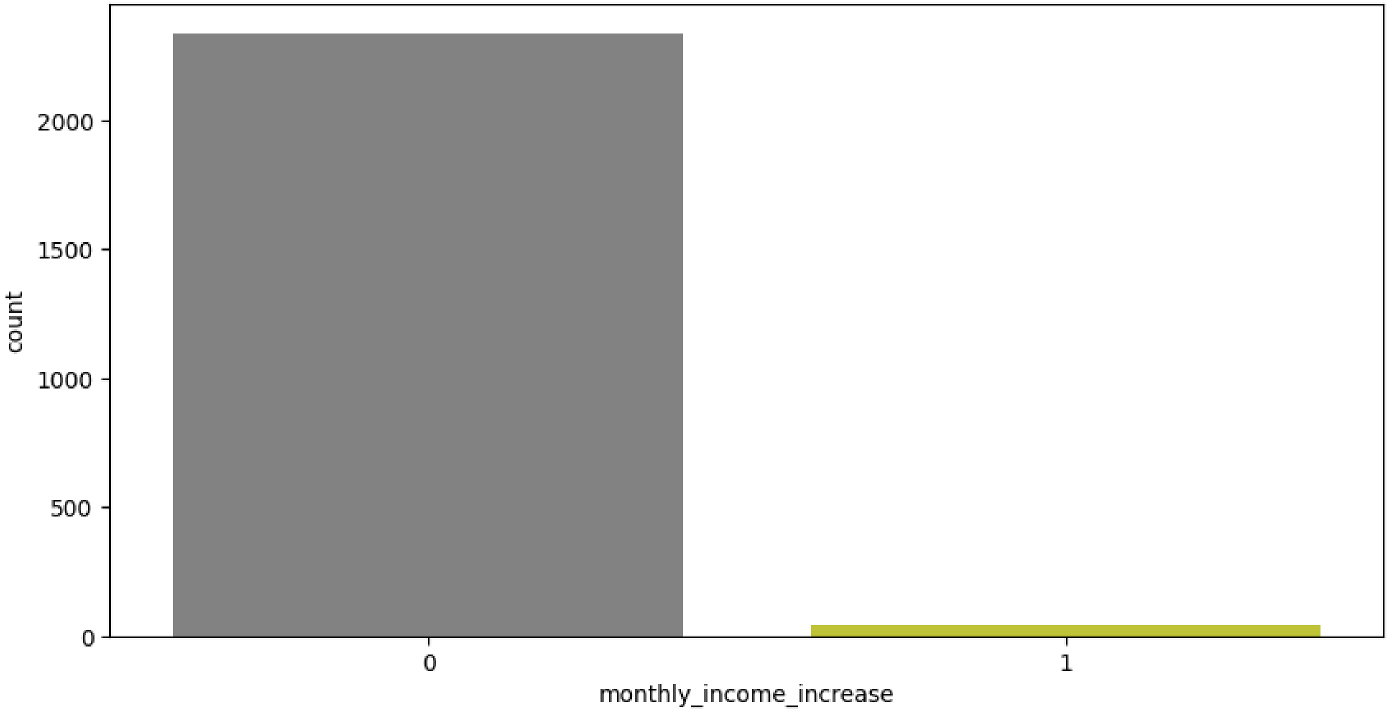
In [34]:
```
# tenture vs is_churn
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_processed, x='is_churn', y='tenture', color=palette[1])
plt.title('tenture vs is_churn')
plt.show()
```



Insight 🕵️

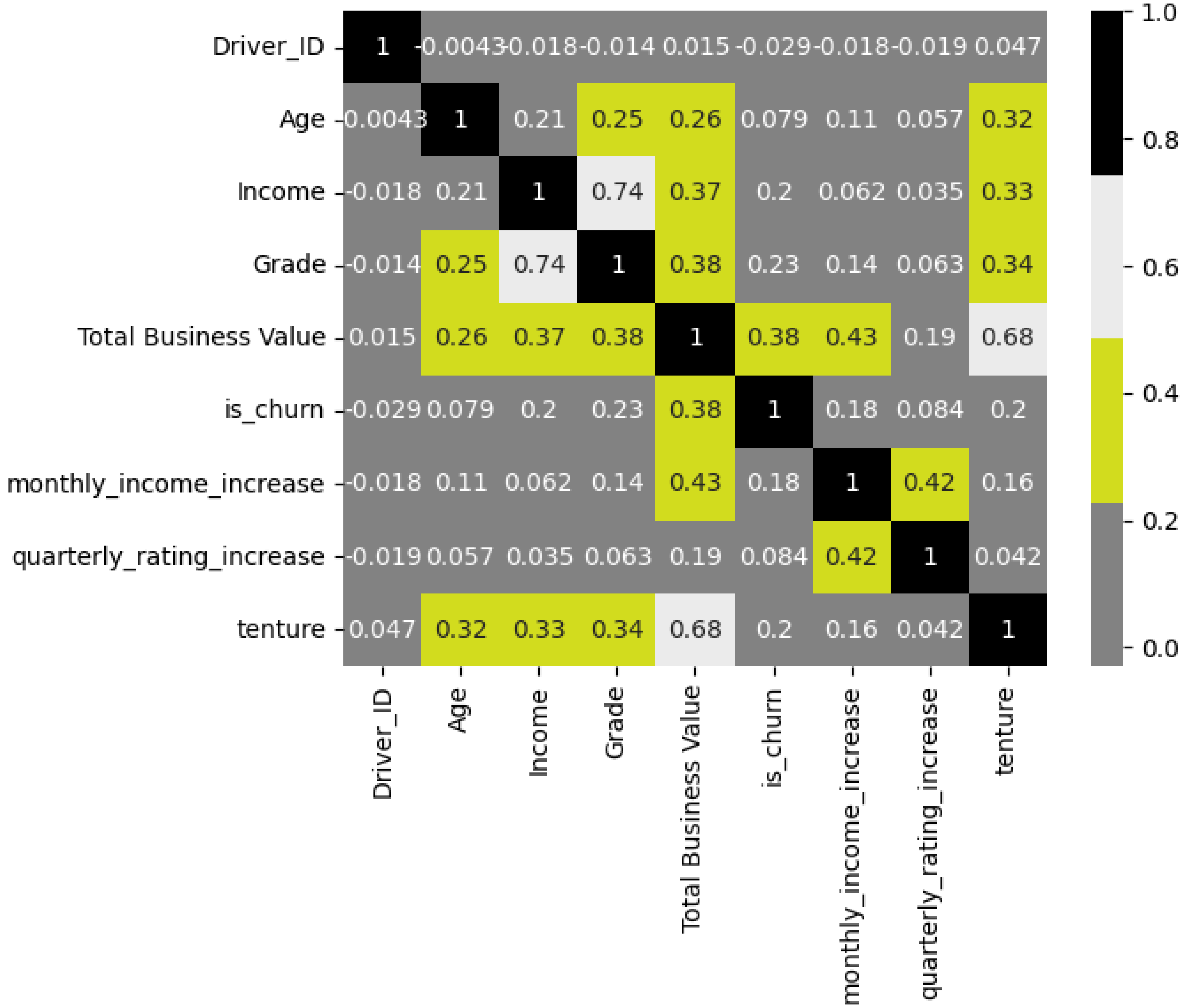
- The median tenure for customers who have not churned (is_churn = 0) is significantly lower compared to those who have churned (is_churn = 1).
- This suggests that customers who stay with the service tend to have shorter tenures on average compared to those who eventually churn.
- There are more outliers with higher tenures in the non-churn group. This indicates that while most non-churned customers have shorter tenures, there are a few who have stayed for a significantly longer period.

In [35]:
```
# monthly_income_increase distribution
plt.figure(figsize=(10, 5))
sns.countplot(df_processed, x='monthly_income_increase', palette=palette)
plt.xlabel('monthly_income_increase')
plt.show()
```



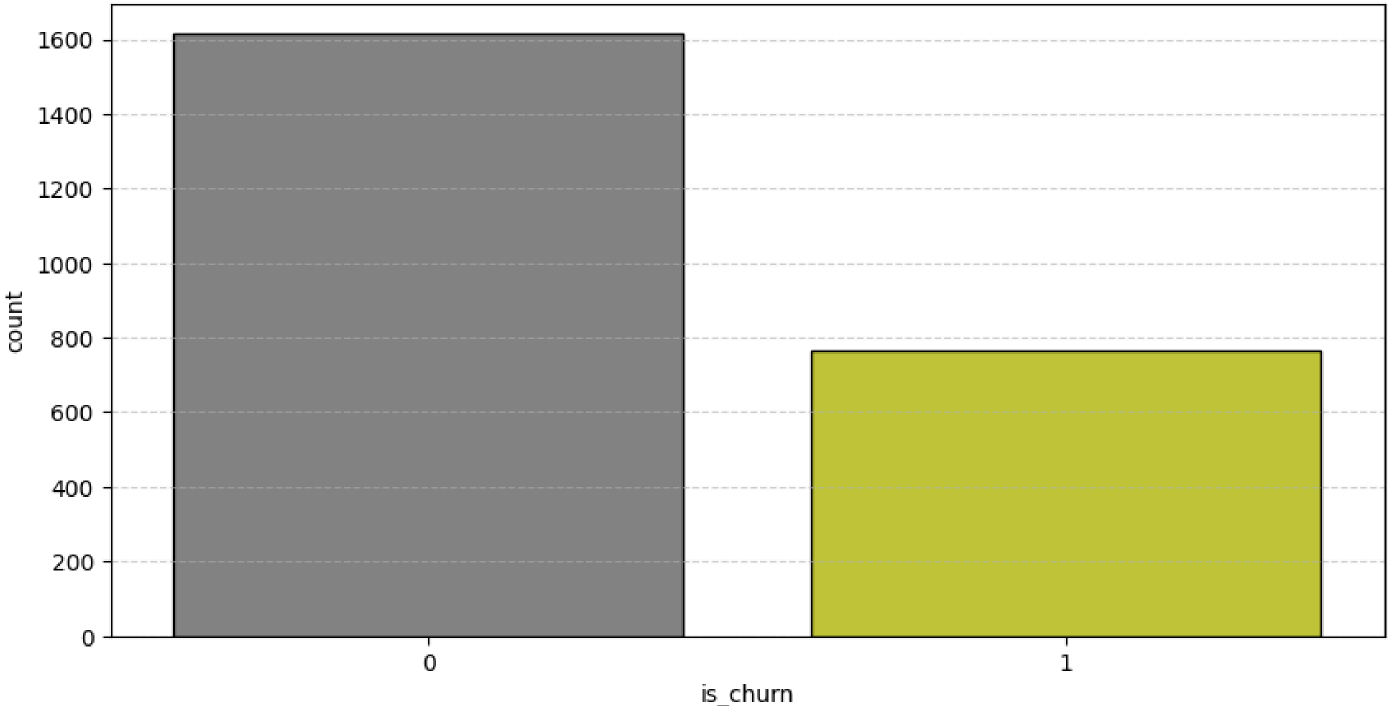
In [36]:
```
numeric_df = df_processed.select_dtypes(include=[np.number])
sns.heatmap(numeric_df.corr(), annot=True, cmap=palette)
```

Out[36]: <Axes: >



- Income and Grade has significant correlations between them.

In [37]:
```
# is_churn distribution
plt.figure(figsize=(10, 5))
sns.countplot(df_processed, x='is_churn', palette=palette, edgecolor='black')
plt.xlabel('is_churn')
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.show()

df_processed['is_churn'].value_counts()
```



Out[37]:
```
is_churn
0    1616
1     765
Name: count, dtype: int64
```

Insight 🕵️

- The Dataset contain information about 765 Driver left the organisation.
- 1616 Driver are associated with Ola.
- This distrubution indication that the dataset is highly imbalance.

Data preparation for Modelling 🤼

In [38]:
```
# One hot encoding of the categorical variable
df_processed = pd.get_dummies(df_processed, columns=['City', 'Gender'], dtype=int, drop_first=Tru
df_processed.head()

# df_processed['Gender'] = df_processed['Gender'].astype('int64')
```

Out[38]:

	Driver_ID	Age	Education_Level	Income	Dateofjoining	Joining Designation	Grade	Total Business Value	is_churn	mont
0	1	28	2	57387.0	2018-12-24	1	1	1715580	0	
1	2	31	2	67016.0	2020-11-06	2	2	0	1	
2	4	43	2	65603.0	2019-12-07	2	2	350000	0	
3	5	29	0	46368.0	2019-01-09	1	1	120360	0	
4	6	31	1	78728.0	2020-07-31	3	3	1265000	1	

5 rows × 41 columns

In [39]:
```
# Label Encoding for 'Education_Level', 'Joining Designation', 'Grade'
le = LabelEncoder()
df_processed['Education_Level'] = le.fit_transform(df_processed['Education_Level'])
df_processed['Joining Designation'] = le.fit_transform(df_processed['Joining Designation'])
df_processed['Grade'] = le.fit_transform(df_processed['Grade'])
```

In [40]:
```
# Class imbalance Treatment
X = df_processed.drop(['Driver_ID', 'is_churn', 'Dateofjoining'], axis=1)
y = df_processed['is_churn']

print('Original dataset shape %s' % Counter(y))

sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)
print('Resampled dataset shape %s' % Counter(y_res))

df_resampled = pd.concat([X_res, y_res], axis=1)
df_resampled.head()
```

```
Original dataset shape Counter({0: 1616, 1: 765})
Resampled dataset shape Counter({0: 1616, 1: 1616})
```

Out[40]:

	Age	Education_Level	Income	Joining Designation	Grade	Total Business Value	monthly_income_increase	quarterly_ratin
0	28	2	57387.0	0	0	1715580	0	
1	31	2	67016.0	1	1	0	0	
2	43	2	65603.0	1	1	350000	0	
3	29	0	46368.0	0	0	120360	0	
4	31	1	78728.0	2	2	1265000	0	

5 rows × 39 columns

In [41]:
```
# # Scaling the data - Not required for tree based models
# scaler = StandardScaler()
# X_scaled = scaler.fit_transform(X_res)
# X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
# X_scaled.head()
```

In [42]:
```
# Splitting the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=21

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[42]:
```
((2585, 38), (647, 38), (2585,), (647,))
```

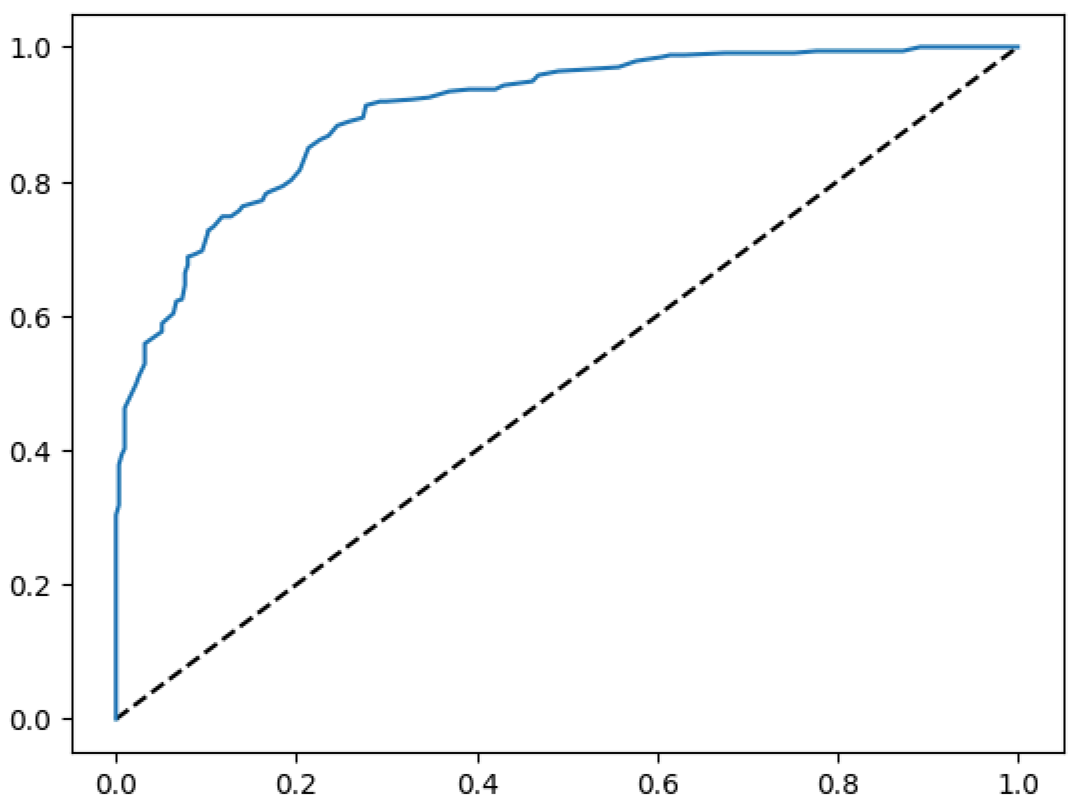
Bagging Algorithm

In [43]:
```
# Model Building - Random Forest

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

```python
print('Parameters: ', rf.get_params())

print('Training Score: ', rf.score(X_train, y_train))
print('Test Score: ', accuracy_score(y_test, y_pred))
print('Classification Report: \n', classification_report(y_test, y_pred))
print('Confusion Matrix: \n', confusion_matrix(y_test, y_pred))

# ROC Curve
y_pred_prob = rf.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
```

```
Parameters:  {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'ma
x_depth': None, 'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurit
y_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'monotonic_cst': None, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 4
2, 'verbose': 0, 'warm_start': False}
Training Score:  1.0
Test Score:  0.8068006182380216
Classification Report: 
               precision    recall  f1-score   support

           0       0.78      0.83      0.81       314
           1       0.83      0.78      0.81       333

    accuracy                           0.81       647
   macro avg       0.81      0.81      0.81       647
weighted avg       0.81      0.81      0.81       647

Confusion Matrix: 
 [[261  53]
 [ 72 261]]
```

Out[43]: [<matplotlib.lines.Line2D at 0x19ca36e2170>]



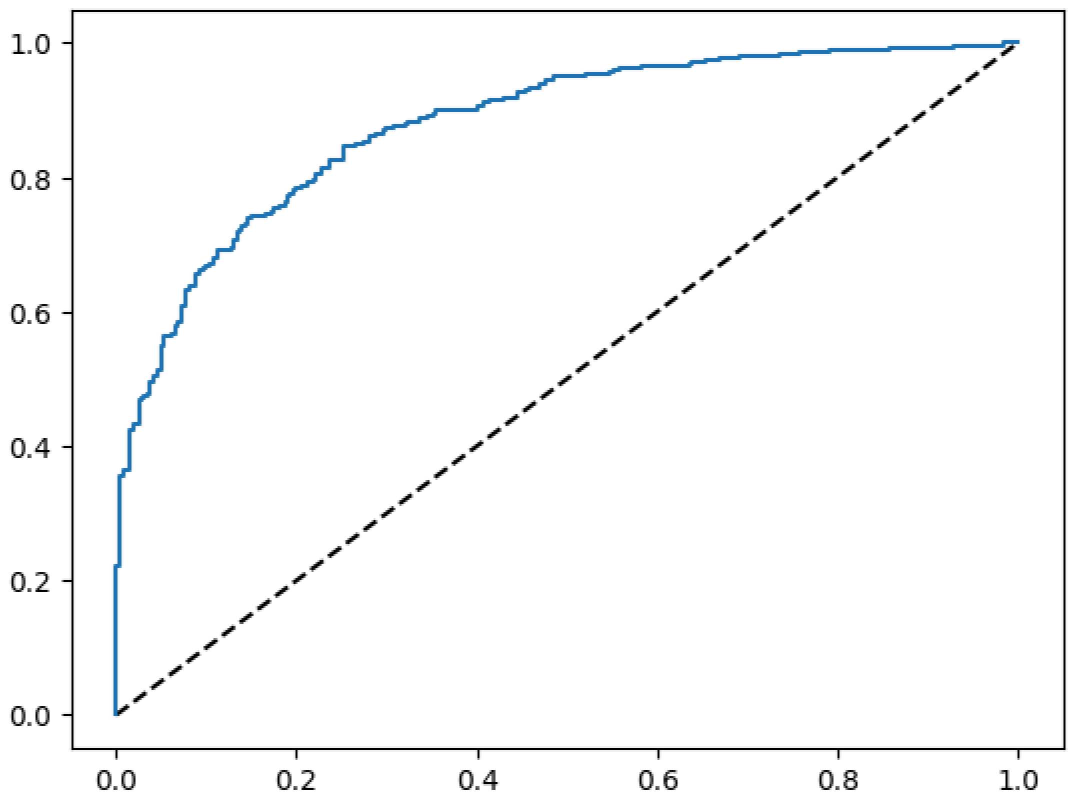
Insight:

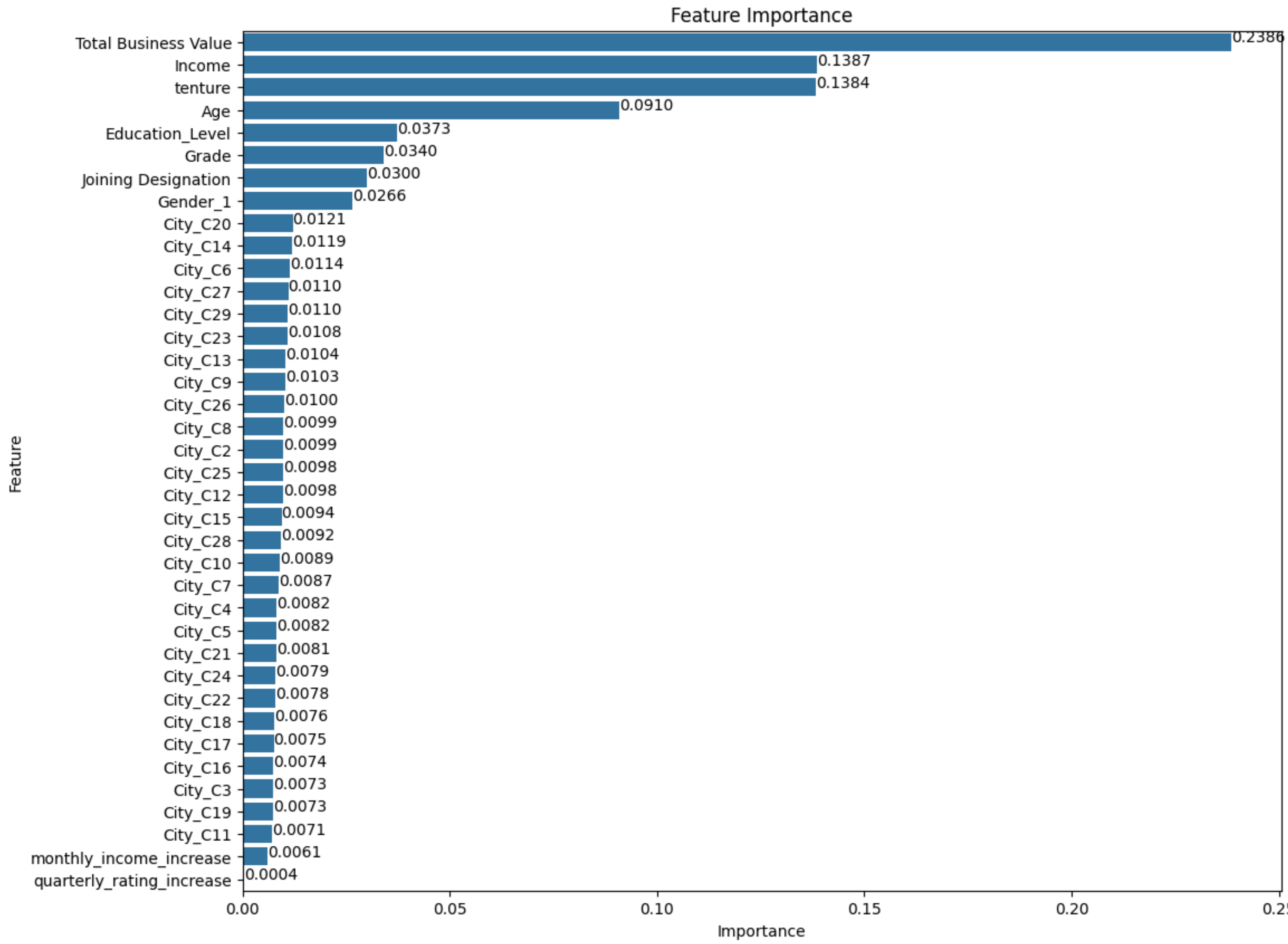
- Train accuracy is 100% which indicate model fits very well with data.
- Test accuracy is 80% which is good but lesser than the Train accuracy, indicating potiential overfitting.
- Additional tuning of hyperparameters could improve the model's generalization ability.

In [64]:

```
# Hyperparameter Tuning - RandomizedSearchCV

# Define a more extensive parameter grid for Random Search
params = {
    'n_estimators': [int(x) for x in range(100, 1000, 100)],
    'max_depth': [None] + [int(x) for x in range(10, 110, 10)],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['log2', 'sqrt'],
    'bootstrap': [True, False]
}

# Initialize the RandomForest classifier
rf = RandomForestClassifier(random_state=42)

cv = RandomizedSearchCV(estimator=rf, param_distributions=params, n_iter=100, cv=5, verbose=1, ra
cv.fit(X_train, y_train)

print("Best parameters found: ", cv.best_params_)
print("Best cross-validation score achieved: ", cv.best_score_)

# Model Building - With Best Hyperparameters
params = cv.best_params_

rf = RandomForestClassifier(**params, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print('Parameters: ', rf.get_params())
print('Training Score: ', rf.score(X_train, y_train))
print('Test Score: ', accuracy_score(y_test, y_pred))

print('Classification Report: \n', classification_report(y_test, y_pred))
print('Confusion Matrix: \n', confusion_matrix(y_test, y_pred))

# ROC Curve
y_pred_prob = rf.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)

# Feature Importance
feature_importance = pd.DataFrame({'Feature': X.columns, 'Importance': rf.feature_importances_})
feature_importance = feature_importance.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(12, 10))
sns.barplot(data=feature_importance, x='Importance', y='Feature')
plt.title('Feature Importance')

# Show the value at the end of each bar
for index, value in enumerate(feature_importance['Importance']):
    plt.text(value, index, f'{value:.4f}')

plt.show()
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best parameters found:  {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max
_features': 'log2', 'max_depth': 40, 'bootstrap': True}
Best cross-validation score achieved:  0.8096711798839458
Parameters:  {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'ma
x_depth': 40, 'max_features': 'log2', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_
decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 5, 'min_weight_fraction_leaf': 0.0,
'monotonic_cst': None, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 4
2, 'verbose': 0, 'warm_start': False}
Training Score:  0.988394584139265
Test Score:  0.7882534775888718
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.79      0.79       321
           1       0.79      0.79      0.79       326

    accuracy                           0.79       647
   macro avg       0.79      0.79      0.79       647
weighted avg       0.79      0.79      0.79       647

Confusion Matrix:
 [[253  68]
 [ 69 257]]
```





Key Insights

Overall Accuracy

The model achieved an accuracy of approximately **79.75%**. This indicates that the model correctly predicts the class labels for about 80% of the instances in the validation set.

Class-wise Performance

Class 0 (Non-Churn)

- **Precision:** 0.77
- **Recall:** 0.83
- **F1-Score:** 0.80

The model performs well in identifying non-churn instances, with a high recall of 0.83, indicating that 83% of the actual non-churn instances are correctly identified. However, the precision is slightly lower at 0.77, meaning that 77% of the instances predicted as non-churn are actually non-churn.

Class 1 (Churn)

- **Precision:** 0.83
- **Recall:** 0.77
- **F1-Score:** 0.80

The model shows a higher precision for churn instances at 0.83, indicating that 83% of the instances predicted as churn are actually churn. However, the recall is lower at 0.77, meaning that only 77% of the actual churn instances are correctly identified.

Feature Important

- **Total Business Value**
- **tenture**
- **Income**
- **Age**

Are the top features for our model predictation.

Boosting Algorithm

In [48]:

```
# Splitting the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[48]:

```
((2585, 38), (647, 38), (2585,), (647,))
```

In [49]:

```
models = {
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100, random_state=42),
    'XGBoost': XGBClassifier(n_estimators=100, random_state=42, use_label_encoder=False, eval_me
    'LightGBM': LGBMClassifier(n_estimators=100, random_state=42),
    'CatBoost': CatBoostClassifier(n_estimators=100, random_state=42, verbose=0)
}
```

In [50]:

```
# Evaluating different models
results = {}

for name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)

    # Evaluate the model
    train_accuracy = model.score(X_train, y_train)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)

    # Store the results
    results[name] = {
        'Train Accuracy': train_accuracy,
        'Test Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
        'ROC AUC': roc_auc
    }
```

```
results_df = pd.DataFrame(results).T
results_df
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 1290, number of negative: 1295
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000839 se
conds.
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 701
[LightGBM] [Info] Number of data points in the train set: 2585, number of used features: 37
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499033 -> initscore=-0.003868
[LightGBM] [Info] Start training from score -0.003868
```

Out[50]:

	Train Accuracy	Test Accuracy	Precision	Recall	F1 Score	ROC AUC
Gradient Boosting	0.845648	0.778980	0.788644	0.766871	0.777605	0.779074
XGBoost	0.974468	0.800618	0.797583	0.809816	0.803653	0.800547
LightGBM	0.952805	0.799073	0.816129	0.776074	0.795597	0.799252
CatBoost	0.893617	0.786708	0.793750	0.779141	0.786378	0.786767

Insight:

XGBoost demonstrates the highest training accuracy, which suggests it fits the training data very well. However, the significant difference between train and test accuracy indicates overfitting. Despite this, XGBoost achieves the highest test accuracy and F1 Score among the models, making it a strong candidate for predicting churn, provided overfitting is addressed.

Overall Comparison

- **Best Performing Model:** XGBoost, with the highest test accuracy (80.06%) and F1 Score (80.36%). However, it shows signs of overfitting that need to be addressed.
- **Most Balanced Model:** CatBoost, with close train and test accuracies, indicating good generalization.
- **Highest Precision:** LightGBM, making it effective in correctly identifying true positives.

XGBoost Classifier

In [62]:

```
# Define the parameter grid based on RandomizedSearchCV results
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [6, 7, 8],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 0.2],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

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

# Set up the RandomizedSearchCV
cv = RandomizedSearchCV(estimator=xgb_model, param_distributions=param_grid, cv=5, n_jobs=-1, ve

# Fit the model
cv.fit(X_train, y_train)
```

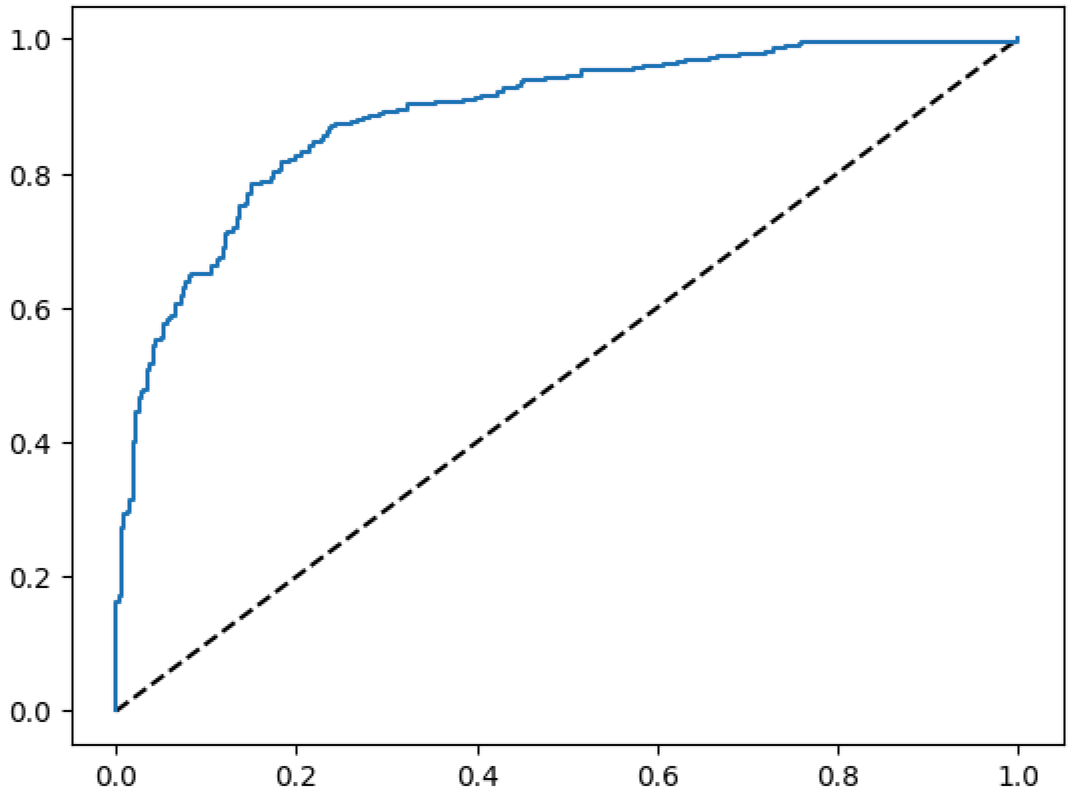
```
# Best parameters and score
print("Best parameters found: ", cv.best_params_)
print("Best cross-validation score achieved: ", cv.best_score_)
```

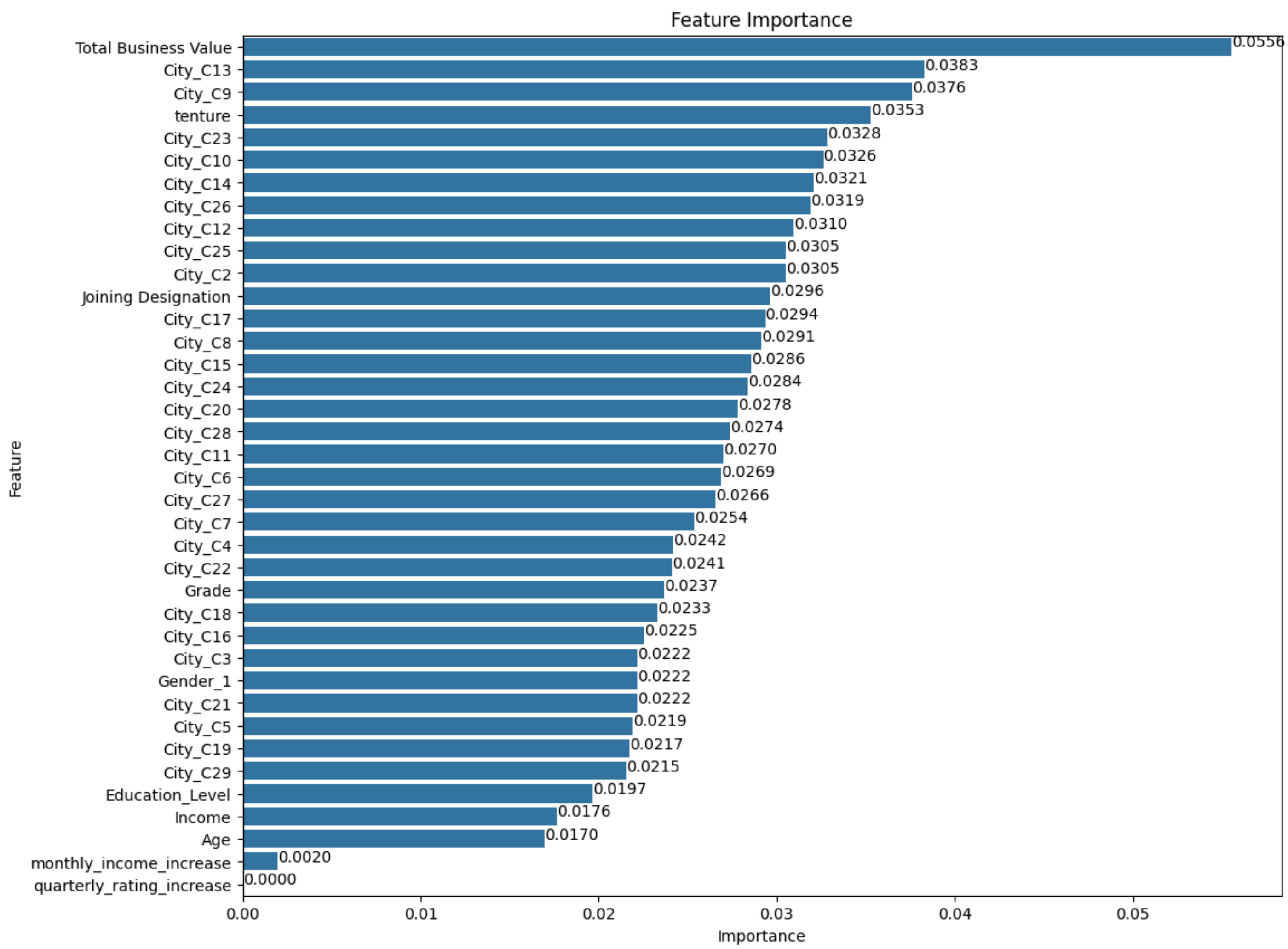
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters found:  {'subsample': 0.9, 'n_estimators': 300, 'min_child_weight': 1, 'max_dept
h': 7, 'learning_rate': 0.05, 'gamma': 0.1, 'colsample_bytree': 1.0}
Best cross-validation score achieved:  0.8123791102514506
```

In [63]:

```
# Train the model with the best parameters
params = cv.best_params_
xgb_model = XGBClassifier(**params, use_label_encoder=False, eval_metric='logloss', random_state
xgb_model.fit(X_train, y_train)
y_pred = xgb_model.predict(X_test)

print("Training Score: ", xgb_model.score(X_train, y_train))
print("Test Score: ", xgb_model.score(X_test, y_test))
print("Classification Report: \n", classification_report(y_test, y_pred))
print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))

# ROC Curve
y_pred_prob = xgb_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)

# Feature Importance
feature_importance = pd.DataFrame({'Feature': X.columns, 'Importance': xgb_model.feature_importa
feature_importance = feature_importance.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(12, 10))
sns.barplot(data=feature_importance, x='Importance', y='Feature')
plt.title('Feature Importance')

# Show the value at the end of each bar
for index, value in enumerate(feature_importance['Importance']):
    plt.text(value, index, f'{value:.4f}')

plt.show()
```

```
Training Score:  0.9620889748549323
Test Score:  0.8145285935085008
Classification Report: 
               precision    recall  f1-score   support

           0       0.81      0.82      0.81       321
           1       0.82      0.81      0.82       326

    accuracy                           0.81       647
   macro avg       0.81      0.81      0.81       647
weighted avg       0.81      0.81      0.81       647

Confusion Matrix: 
 [[262  59]
 [ 61 265]]
```





Key Insights:

- The model achieved a training accuracy of 96.21%, indicating that it fits the training data very well. However, this high training score compared to the test score suggests potential overfitting.

- The test accuracy is 81.45%, which is a strong performance, indicating that the model generalizes well to unseen data, though not as perfectly as it does with the training data.
- Both classes (0 and 1) have similar precision and recall values (~0.81-0.82), indicating that the model is equally good at identifying both churners and non-churners. This balance is crucial for maintaining a low false positive and false negative rate.
- Both macro and weighted averages for precision, recall, and F1-score are 0.81, reinforcing the model's balanced performance across both classes.

Actionable Insight and Recommendations

- **Retention Strategies:**

Focus retention efforts on newer drivers, as those with shorter tenures are more likely to stay. Implement additional engagement strategies for drivers with longer tenures to prevent churn.

- **Driver Segmentation:**

Segment drivers based on tenure and age to tailor specific interventions. For example, provide loyalty rewards or personalized offers to long-tenured drivers.

- **Training and Support:**

Provide robust training and support during the initial months to help retain new drivers. Ensuring they feel valued and supported could reduce early churn.

- **Incentive Programs:**

Develop and refine incentive programs that reward drivers for increases in monthly income and quarterly ratings, as these are linked to higher business value and lower churn rates.

In [ ]:

```

```