Software Development for Data Science

Module Code: MMI226822

Coursework 2: Predictive Analysis of the Telco Customer Churn Dataset

Abstract

This study navigates through the essential stages of a data science project using the Telco customer churn dataset. Commencing with an introduction to the dataset, we conduct thorough data exploration, examining its structure, summary statistics, and unique features. The subsequent focus is on data cleaning and preprocessing, addressing missing values, encoding categorical variables, and handling class imbalance. Visualizations and statistical tests are deployed to discern patterns and relationships within the data, fostering a deeper comprehension of customer behavior.

The analysis extends to visualizations uncovering trends related to customer churn based on factors such as contract type, online security, and paperless billing. These visualizations, accompanied by insightful comments, provide a comprehensive overview of the factors influencing customer churn.

The modeling phase entails the application of machine learning algorithms, including Random Forest and XGBoost classifiers, for predicting customer churn. Rigorous evaluation metrics such as accuracy, precision, recall, and ROC AUC gauge model performance are utilized in the process which includes hyperparameter tuning via GridSearchCV to optimize model parameters.

Class imbalance is addressed through the implementation of oversampling techniques like BorderlineSMOTE. The final models undergo training and evaluation, with performance scrutiny using classification reports, confusion matrices, and ROC curves.

The project concludes with an exploration of the interpretability of machine learning models, shedding light on feature importance and their impact on predictions. The findings offer valuable insights for Telco companies seeking to understand and mitigate customer churn, contributing to strategic decision-making processes.

1.0 Introduction

In today's dynamic telecommunications landscape, customer churn stands as a pivotal challenge for service providers. Defined as the discontinuation of services by customers or subscribers, the yearly churn rate can range between 20% and 40%. According to the SAS Institute's publication on churn prediction, a 25% churn rate stands as the industry's global average (Nath, 2003). In the telecommunications sectors of the United States and Europe, customer attrition results in an approximate annual loss of nearly $4 billion (Madden, Savage & Neal, 2014).

The ability to predict and prevent customer churn holds significant value, given the high costs associated with acquiring new customers compared to retaining existing ones.

This coursework delves into the realm of customer churn analysis within the telecom sector using a comprehensive dataset sourced from a fictional telco company in California. The primary objective is to leverage exploratory data analysis (EDA) and machine learning techniques to scrutinize patterns and unveil insights crucial for predicting and mitigating customer churn.

The telecom industry's survival hinges on effectively identifying customers at risk of churn and deploying proactive retention strategies. However, personalized retention strategies pose challenges due to the sheer volume of customers. Consequently, the strategic focus revolves around predicting potential churners, enabling targeted efforts towards retaining these 'high-risk' customers.

1.1 Objectives

This exploration aims to address several pivotal inquiries:

- What proportion of customers exhibit churn versus those retaining active services?
- Are discernible patterns in churn rates observable based on gender demographics?
- Do preferences or patterns in churn exist concerning the types of services provided?
- Which services or features contribute significantly to profitability?
- Which evaluations of the models, taking into account both accuracy and quality of prediction by uncovering subtle trade-offs across different algorithms?

By unraveling these insights, this analysis seeks to empower telecom companies with actionable intelligence to not only stem customer attrition but also foster growth and competitive advantage. The ultimate goal lies in optimizing customer retention strategies, thereby fortifying market positions and cultivating sustained profitability.

Through a meticulous exploration of this dataset, this coursework endeavors to illuminate the critical factors driving customer churn, laying the groundwork for informed decision-making and strategic interventions to safeguard customer loyalty and enhance business viability within the telecommunications sphere.

2.0 Importing the Libraries

```
import pandas as pd  # Imports the pandas library and assigns it the alias 'pd' for ease of use
import matplotlib.pyplot as plt  # Imports the matplotlib library for plotting and assigns the pyplot module to the alias 'p
import scipy  # Imports the scipy library for scientific and technical computing
import seaborn as sns  # Imports the seaborn library for statistical data visualization
import missingno as msno  # Imports the missingno library for visualizing missing data
import sklearn  # Imports the scikit-learn library for machine learning
from sklearn.model_selection import train_test_split, GridSearchCV  # Imports specific functions (train_test_split and GridS
from sklearn import linear_model, metrics  # Imports the linear model and metrics modules from scikit-learn
from sklearn.linear_model import LogisticRegression  # Imports the Logistic Regression classifier from scikit-learn
from sklearn.metrics import classification_report, accuracy_score, roc_curve,roc_auc_score, recall_score, confusion_matrix
from sklearn.tree import DecisionTreeClassifier  # Imports the Decision Tree classifier from scikit-learn
from sklearn.utils import resample  # Imports the resample function from scikit-learn for handling imbalanced datasets
from sklearn.ensemble import RandomForestClassifier, VotingClassifier, StackingClassifier  # Imports ensemble classifiers li
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, StandardScaler, LabelEncoder  # Imports different types of
from xgboost import XGBClassifier  # Imports the XGBoost classifier from the XGBoost library
from scipy.stats import shapiro  # Imports the Shapiro-Wilk test for normality from the scipy library
from catboost import CatBoostClassifier  # Imports the CatBoost classifier from the CatBoost library.
import warnings  # Imports the warnings module for handling warnings
warnings.filterwarnings('ignore')  # Configures the script to ignore warning messages
```

```
pip install catboost  # Installed to enable the use CatBoost for tasks like classification, regression, and ranking
```

3.0 About the Telco Customer Churn Data

```
from google.colab import drive   # The code enables imports the Google Drive
drive.mount('/content/drive')    # The code mounts the Google Drive to '/content/drive' allowing you to access files and fo

TelcoData = pd.read_csv('/content/drive/MyDrive/Software_Dev/Telco-Customer-Churn.csv')
```

```
Mounted at /content/drive
```

```
TelcoData.head(3)
```

	customerID	customerID.1	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	Onlin
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	

3 rows × 21 columns

```
print("\n The Telco Customer Dataset contains 7043 Customers (Rows) and 21 Customer's attributes (Columns)")
TelcoData.shape # Prints the numbers of rows and columns in a dataframe
```

```

 The Telco Customer Dataset contains 7043 Customers (Rows) and 21 Customer's attributes (Columns)
(7043, 21)
```

```
TelcoData.columns # prints the column names in the dataframe
```

```
Index(['customerID', 'customerID.1', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

- **CustomerId:** Customer ID
- **Gender:** Customer gender
- **SeniorCitizen:** Whether the customer is a senior citizen (1, 0)
- **Partner:** Whether the client has a partner (Yes, No) - Indicates if the customer is married or not

- **Dependents:** Whether the client has dependents (Yes, No) - Indicates if the customer has children, parents, grandparents, etc.
- **Tenure:** Number of months the customer has stayed with the company
- **PhoneService:** Whether the customer has phone service (Yes, No)
- **MultipleLines:** Whether the customer has more than one line (Yes, No, No phone service)
- **InternetService:** Customer's internet service provider (DSL, Fiber optic, No)
- **OnlineSecurity:** Whether the customer has online security (Yes, No, No * Internet service)
- **OnlineBackup:** Whether the customer has online backup (Yes, No, No * Internet service)
- **DeviceProtection:** Whether the customer has device protection (Yes, No, No Internet service)
- **TechSupport:** Whether the customer receives technical support (Yes, No, No Internet service)
- **StreamingTV:** Indicates whether the customer has streaming TV (Yes, No, No Internet service) Indicates if the customer uses the Internet service to stream television programs from a third-party provider
- **StreamingMovies:** Whether the customer has streaming movies (Yes, No, No Internet service) - Indicates if the customer uses the Internet service to stream movies from a third-party provider
- **Contract:** Duration of the customer's contract (Month to month, One year, Two years)
- **PaperlessBilling:** Whether the customer receives a paperless bill (Yes, No)
- **PaymentMethod:** Customer's payment method (Electronic check, Postal check, Bank transfer (automatic), Credit card (automatic))
- **MonthlyCharges:** Amount charged to the customer on a monthly basis
- **TotalCharges:** Total amount charged to the customer
- **Churn:** Whether the customer is using the service or not (Yes or No) - Refers to customers who left in the last month or quarter

```
print("The Telco Customer Dataset Datatype information are:")
TelcoData.info()
```

```
The Telco Customer Dataset Datatype information are:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   customerID.1      7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

The Telco Customer Dataset includes **categorical information** such as customer ID, gender, partnership status, presence of dependents, phone service, multiple lines, internet service type, security, backup, device protection, tech support, streaming services, contract type, paperless billing, payment method, and the churn indicator.

On the **numerical values** side, it contains data on senior citizen status, tenure, monthly charges, and total charges.

3.1 Justification for Employing the Classification Prediction Method for Analysis:

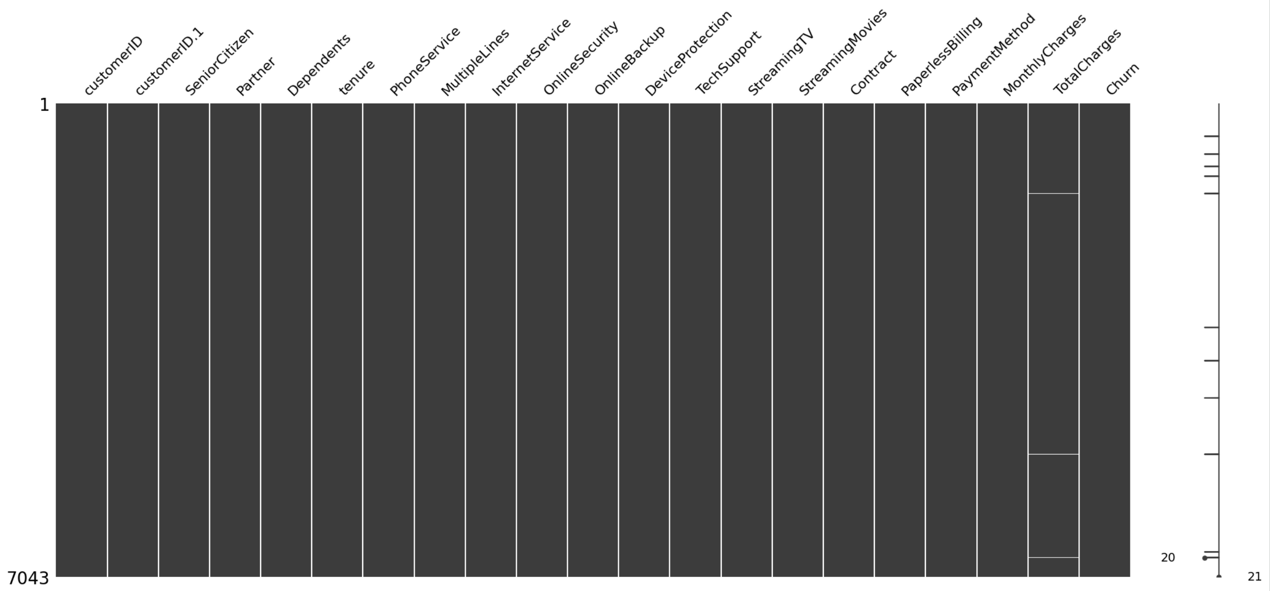
The utilization of the classification prediction method for analyzing customer churn within the telecom dataset is justified due to the categorical nature of the dataset's variables. The dataset comprises categorical information like gender, service types, contract details, and billing preferences, making it conducive for classification analysis. Classification is apt for addressing pivotal inquiries, allowing for the prediction of binary outcomes—specifically(Dixon and Oyebode, 2007), identifying churners versus non-churners. This method facilitates the exploration of discernible patterns in churn rates based on categorical demographics, service preferences, and features, unveiling correlations between customer behaviors across various service channels and their propensity to churn. Its suitability stems from the dataset's categorical variables, enabling the formulation of predictive models that can aid in proactive customer retention strategies and optimizing business decisions in the telecom industry.

```
TelcoData['TotalCharges'] = pd.to_numeric(TelcoData.TotalCharges, errors='coerce')
TelcoData.isnull().sum()
```

```
customerID           0
customerID.1         0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

```
#msno.matrix(TelcoData, color=(0.0, 0.0, 0.9)) # Visualizes missing values as a matrix
msno.matrix(TelcoData) # Visualizes missing values as a matrix
```

```
<Axes: >
```



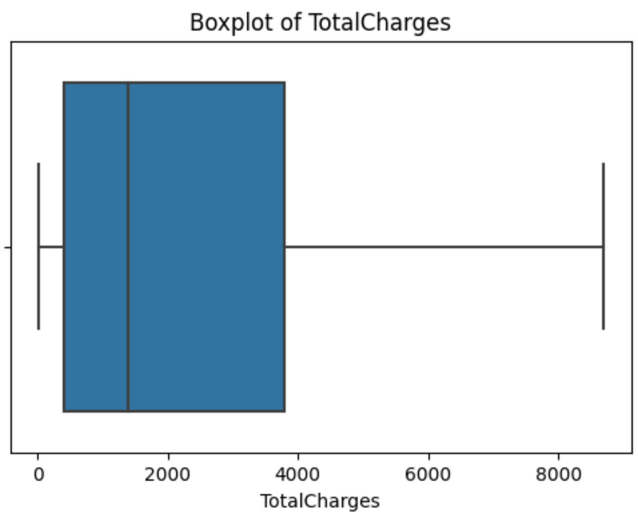
3.2 Outliers in the Telco Customer Churn Dataset

```
print("The Quantiles detail are:")
print(TelcoData.quantile([0, 0.05, 0.50, 0.95, 0.99, 1]).T)
```

```
The Quantiles detail are:
                 0.00    0.05      0.50     0.95      0.99     1.00
SeniorCitizen    0.00   0.000     0.000     1.00     1.000     1.00
tenure           0.00   1.000    29.000    72.00    72.000    72.00
MonthlyCharges  18.25  19.650    70.350   107.40   114.729   118.75
TotalCharges    18.80  49.605  1397.475  6923.59  8039.883  8684.80
```

Looking at the quantiles provided, a potential outlier can be observed in the 'TotalCharges' column. The maximum value (1.00 quantile) of 'TotalCharges' stands at 8684.80, significantly higher than the 95th percentile (6923.59). This substantial difference suggests the presence of extreme values or outliers in the upper range of the 'TotalCharges' column.
The following graph shows a boxplot visualising the distribution and identifing outliers in the 'TotalCharges' column:

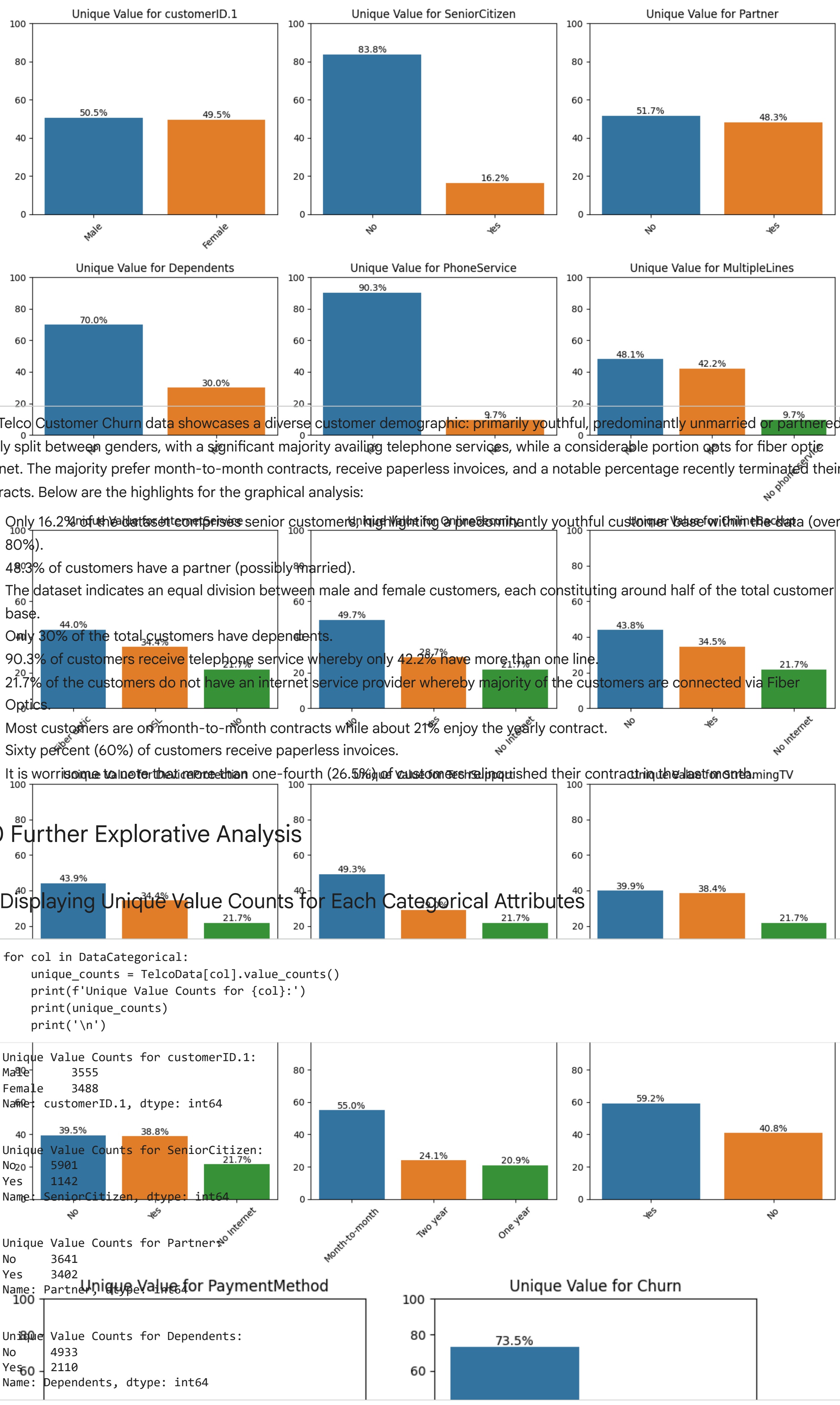
```
plt.figure(figsize=(6, 4))
sns.boxplot(x=TelcoData['TotalCharges'])
plt.title('Boxplot of TotalCharges')
plt.xlabel('TotalCharges')
plt.show()
```



The boxplot for 'TotalCharges' indicates a predominant concentration of values within a confined range, yet outliers beyond the upper whisker depict a subset of customers with notably higher total charges, indicating potential anomalies in the dataset!

```
TelcoData['SeniorCitizen'] = TelcoData['SeniorCitizen'].replace({1: 'Yes', 0: 'No'})  #This code will replace the values 1
```

```
DataSkip = ['customerID', 'tenure', 'MonthlyCharges', 'TotalCharges'] # This line excludes the listed numerical customer's

DataCategorical = [col for col in TelcoData.columns if col not in DataSkip and TelcoData[col].dtype == 'object']
PlotPerRow = 3
TotalPlot = len(DataCategorical)
rows = (TotalPlot + PlotPerRow - 1) // PlotPerRow

for i in range(0, TotalPlot, PlotPerRow):
  # Plotting bar graphs for each categorical column
    plt.figure(figsize=(13, 4))
    for j, col in enumerate(DataCategorical[i:i + PlotPerRow]):
        plt.subplot(1, PlotPerRow, j + 1)
        unique_counts = TelcoData[col].value_counts(normalize=True) * 100  # Calculate percentages
        sns.barplot(x=unique_counts.index, y=unique_counts.values)
        plt.title(f'Unique Value for {col}')
        plt.xlabel('')
        plt.ylabel('')
        plt.xticks(rotation=45)
        plt.ylim(0, 100)  # Set the y-axis limit from 0 to 100 for percentages
        for idx, val in enumerate(unique_counts.values):
            plt.text(idx, val + 1, f'{val:.1f}%', ha='center')  # Display percentages on bars
    plt.tight_layout()
    plt.show()
```



The Telco Customer Churn data showcases a diverse customer demographic: primarily youthful, predominantly unmarried or partnered, evenly split between genders, with a significant majority availing telephone services, while a considerable portion opts for fiber optic internet. The majority prefer month-to-month contracts, receive paperless invoices, and a notable percentage recently terminated their contracts. Below are the highlights for the graphical analysis:

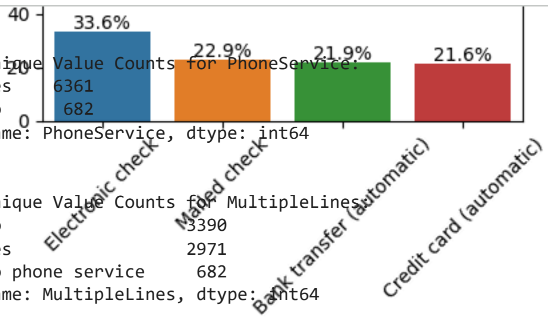
- Only 16.2% of the dataset comprises senior customers, highlighting a predominantly youthful customer base within the data (over 80%).
- 48.3% of customers have a partner (possibly married).
- The dataset indicates an equal division between male and female customers, each constituting around half of the total customer base.
- Only 30% of the total customers have dependents.
- 90.3% of customers receive telephone service whereby only 42.2% have more than one line.
- 21.7% of the customers do not have an internet service provider whereby majority of the customers are connected via Fiber Optics.
- Most customers are on month-to-month contracts while about 21% enjoy the yearly contract.
- Sixty percent (60%) of customers receive paperless invoices.
- It is worrisome to note that more than one-fourth (26.5%) of customers relinquished their contract in the last month.

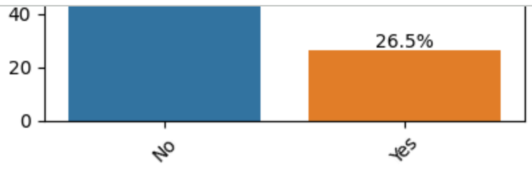
4.0 Further Explorative Analysis

4.1 Displaying Unique Value Counts for Each Categorical Attributes

```
for col in DataCategorical:
    unique_counts = TelcoData[col].value_counts()
    print(f'Unique Value Counts for {col}:')
    print(unique_counts)
    print('\n')
```

```
Unique Value Counts for customerID.1:
Male      3555
Female    3488
Name: customerID.1, dtype: int64


Unique Value Counts for SeniorCitizen:
No     5901
Yes    1142
Name: SeniorCitizen, dtype: int64


Unique Value Counts for Partner:
No     3641
Yes    3402
Name: Partner, dtype: int64


Unique Value Counts for Dependents:
No     4933
Yes    2110
Name: Dependents, dtype: int64
```





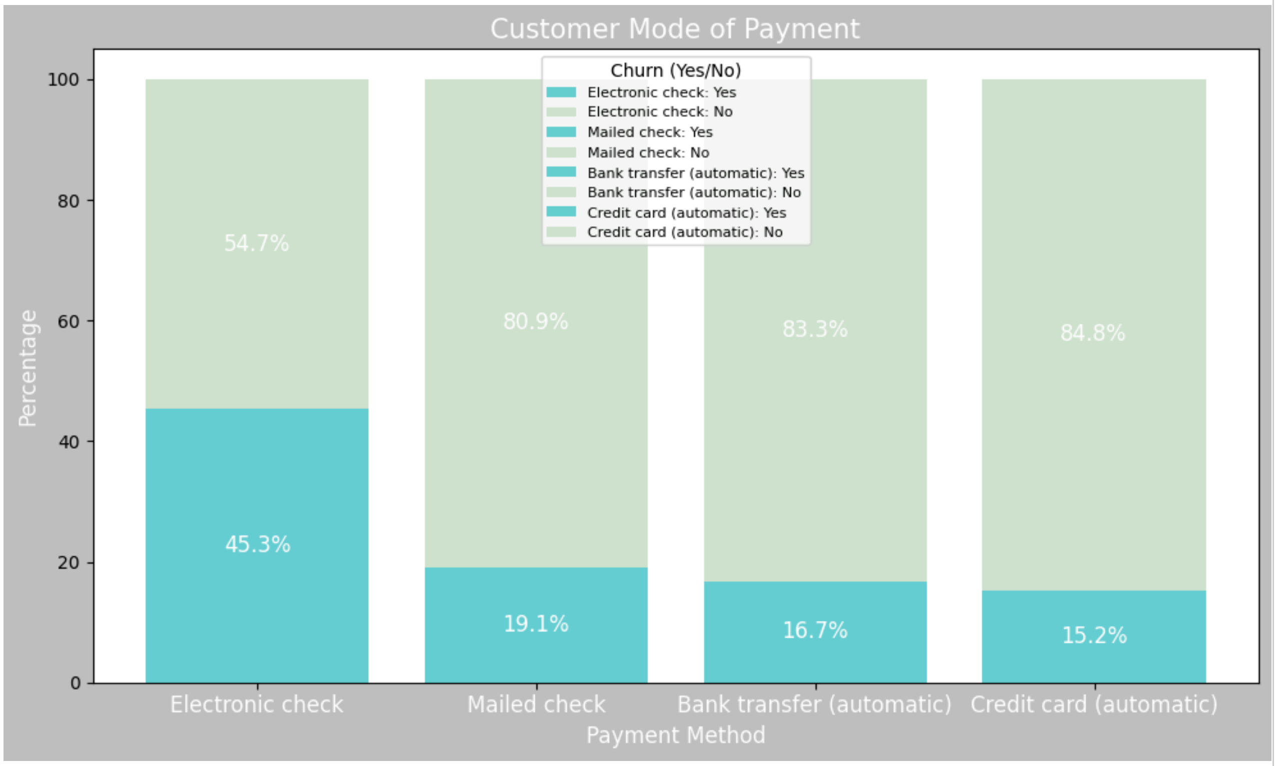
```
Unique Value Counts for PhoneService:
Yes    6361
No      682
Name: PhoneService, dtype: int64


Unique Value Counts for MultipleLines:
No                  3390
Yes                 2971
No phone service     682
Name: MultipleLines, dtype: int64


Unique Value Counts for InternetService:
Fiber optic    3096
DSL            2421
No             1526
Name: InternetService, dtype: int64


Unique Value Counts for OnlineSecurity:
No             3498
Yes            2019
No Internet    1526
Name: OnlineSecurity, dtype: int64


Unique Value Counts for OnlineBackup:
No             3088
Yes            2429
No Internet    1526
Name: OnlineBackup, dtype: int64
```

```
print("Count of 'Yes' Churn by Gender:")
print(TelcoData["Churn"][TelcoData["Churn"]=="Yes"].groupby(by=TelcoData["customerID.1"]).count())

print("\nCount of 'No' Churn by Gender:")
print(TelcoData["Churn"][TelcoData["Churn"]=="No"].groupby(by=TelcoData["customerID.1"]).count())
```

```
Count of 'Yes' Churn by Gender:
customerID.1
Female    939
Male      930
Name: Churn, dtype: int64

Count of 'No' Churn by Gender:
customerID.1
Female    2549
Male      2625
Name: Churn, dtype: int64
```

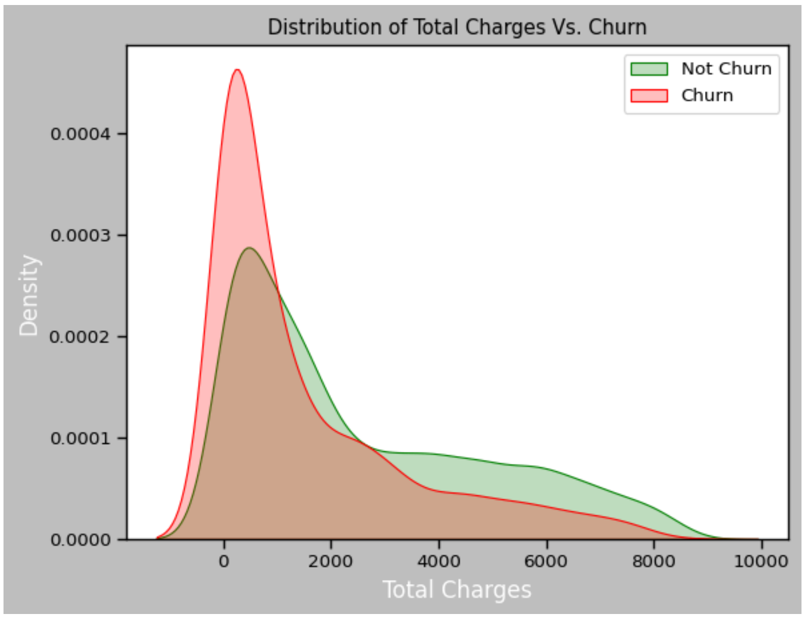
```
# Data for the histogram
Churnlabel = TelcoData['Churn']
PaymentMode = TelcoData['PaymentMethod'].unique()

plt.style.use('grayscale') # Sets the background colour

fig, ax = plt.subplots(figsize=(10, 6)) # Creates subplots with matplotlib

# Iterate through each contract type
for i, contract in enumerate(PaymentMode):
    Data = Churnlabel[TelcoData['PaymentMethod'] == contract]
    Count = Data.value_counts(normalize=True) * 100  # Calculate percentages directly

    # Plotting histogram
    ax.bar(i, Count['Yes'], alpha=0.9, label=f'{contract}: Yes', color='#55cbcd')
    ax.bar(i, Count['No'], alpha=0.9, bottom=Count['Yes'], label=f'{contract}: No', color='#cce2cb')

    # Annotate percentages on the plot
    ax.text(i, Count['Yes'] / 2, f"{Count['Yes']:.1f}%", ha='center', va='center', color='white', fontsize=12)
    ax.text(i, Count['Yes'] + Count['No'] / 2, f"{Count['No']:.1f}%", ha='center', va='center', color='white', fontsize=12

# Adding title and labels
plt.title('Customer Mode of Payment', fontsize=15, color='white')
plt.xlabel('Payment Method', fontsize=12, color='white')
plt.ylabel('Percentage', fontsize=12, color='white')

# Adding legend and adjusting its properties
plt.legend(title='Churn (Yes/No)', fontsize=8, title_fontsize='10', loc='upper center')

# Set x-axis ticks and labels
plt.xticks(range(len(PaymentMode)), PaymentMode, fontsize=12, color='white')
```

```
# Show plot
plt.tight_layout()
plt.show()
```



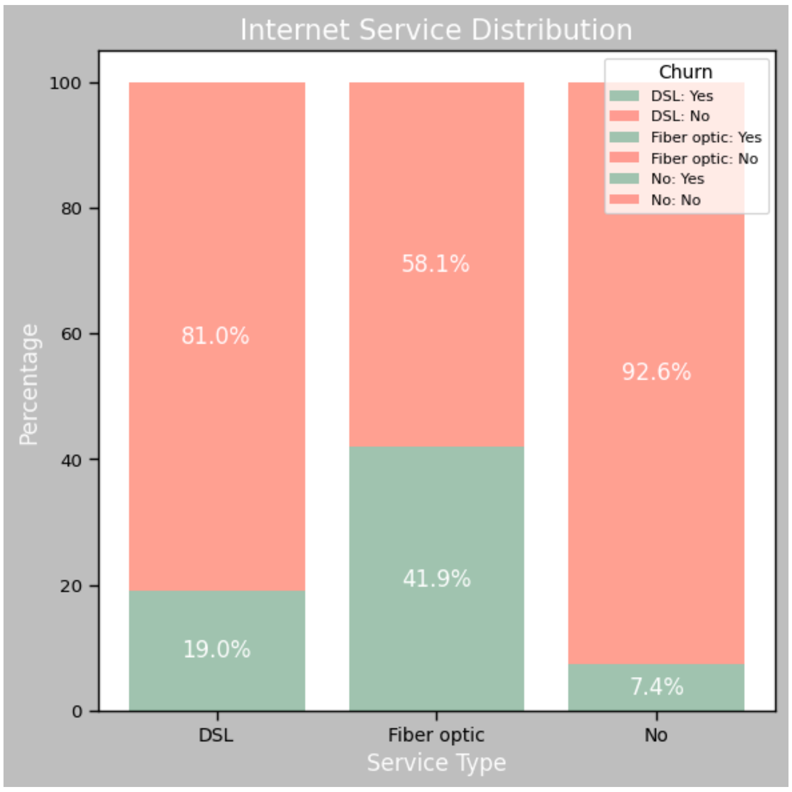
The visualization effectively presents the churn distribution across various payment methods. It's apparent that customers using Electronic Check as their payment method have a significantly higher churn rate compared to other payment modes. Credit-Card automatic transfer, Bank Automatic Transfer, and Mailed Check display notably lower churn rates, suggesting that customers utilizing these payment methods tend to be more loyal.

```
if TelcoData['TotalCharges'].isnull().any():  # Checks for null values after conversion
    TelcoData.dropna(subset=['TotalCharges'], inplace=True)   # This code drops rows with missing 'TotalCharges'

plt.style.use('grayscale') # Sets the background colour
sns.set_context("paper", font_scale=1.1)
ax = sns.kdeplot(TelcoData['TotalCharges'][(TelcoData["Churn"] == 'No')],
                 color="green", shade=True)
ax = sns.kdeplot(TelcoData['TotalCharges'][(TelcoData["Churn"] == 'Yes')],
                 ax=ax, color="red", shade=True)

plt.xlabel('Churn', fontsize=12, color='white')
plt.ylabel('Count', fontsize=12, color='white')

ax.legend(["Not Churn", "Churn"], loc='upper right')
ax.set_ylabel('Density')
ax.set_xlabel('Total Charges')
ax.set_title('Distribution of Total Charges Vs. Churn')
plt.show()
```



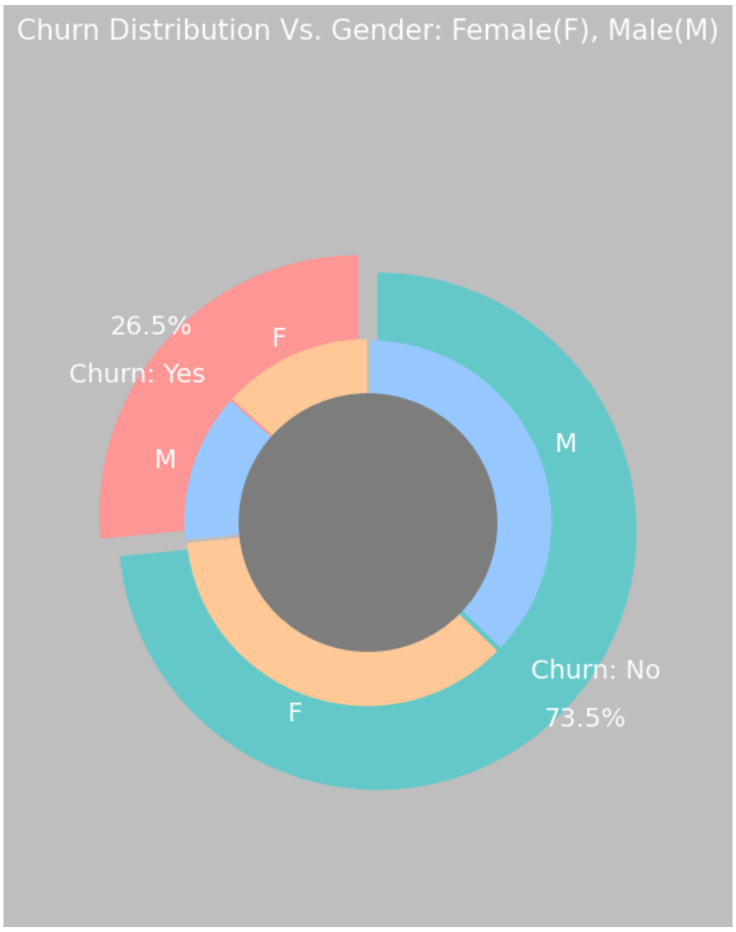
The Kernel Density Estimation (KDE) plot shows the color green representing customers who didn't churn, while the red represents those who did. It is helpful to note that a positive skewness of the tail on the right side of the distribution shows that more customers remained with their service provider even as the total charge increases as against the churn density that peaked at lower total charges.

```
# Data for the histogram
Churnlabel = TelcoData['Churn']
Service = TelcoData['InternetService'].unique()

plt.style.use('grayscale') # Sets the background colour
fig, ax = plt.subplots(figsize=(6, 6)) # Create subplots with matplotlib

# Iterate through each contract type
for i, contract in enumerate(Service):
    Data = Churnlabel[TelcoData['InternetService'] == contract]
    Count = Data.value_counts(normalize=True) * 100  # Calculate percentages directly

    # Plotting histogram
    ax.bar(i, Count['Yes'], alpha=0.9, label=f'{contract}: Yes', color='#97c1a9')
    ax.bar(i, Count['No'], alpha=0.9, bottom=Count['Yes'], label=f'{contract}: No', color='#ff968a')

    # Annotate percentages on the plot
    ax.text(i, Count['Yes'] / 2, f"{Count['Yes']:.1f}%", ha='center', va='center', color='white', fontsize=12)
    ax.text(i, Count['Yes'] + Count['No'] / 2, f"{Count['No']:.1f}%", ha='center', va='center', color='white', fontsize=12

# Adding title and labels
plt.title('Internet Service Distribution', fontsize=15, color='white')
plt.xlabel('Service Type', fontsize=12, color='white')
plt.ylabel('Percentage', fontsize=12, color='white')

# Adding legend and adjusting its properties
plt.legend(title='Churn', fontsize=8, title_fontsize='10', loc='upper right')

# Set x-axis ticks and labels
plt.xticks(range(len(Service)), Service, fontsize=10, color='black')

# Show plot
plt.tight_layout()
plt.show()
```



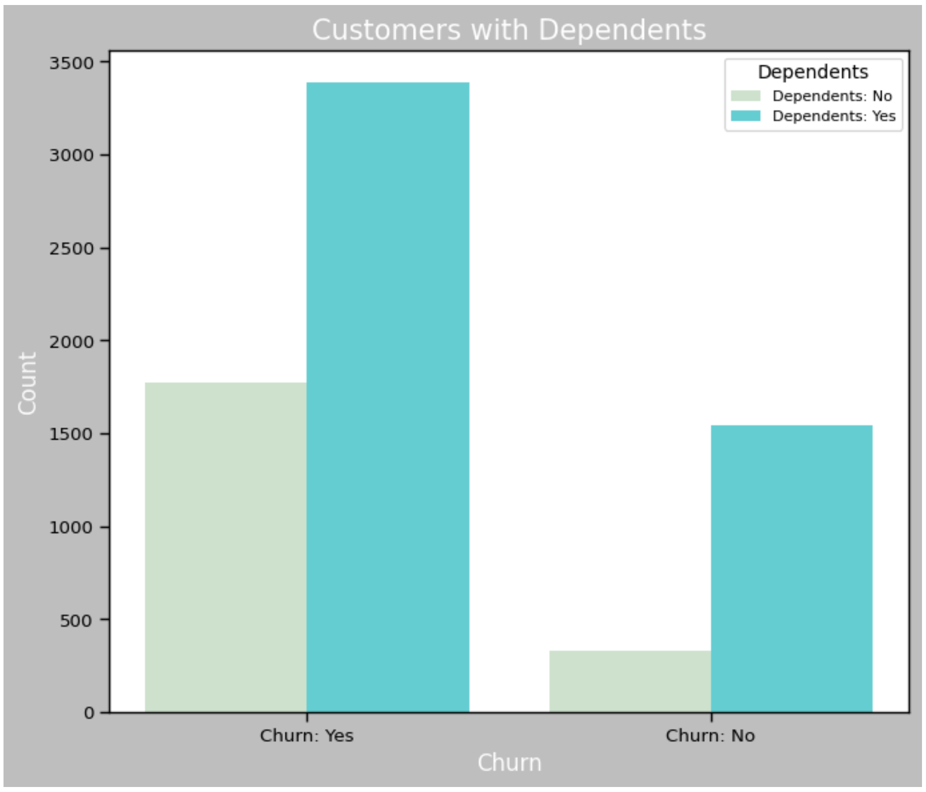
This histogram gives a clear overview of churn rates across different internet service types. It's conspicuous that Fiber optic service users exhibit a notably higher churn rate compared to DSL service users, indicating potential dissatisfaction with this specific internet service. Despite the majority of customers opting for Fiber optic service, the higher churn rate might suggest underlying issues or dissatisfaction with its performance or offerings. Conversely, DSL service, while less popular among customers, showcases a relatively lower churn rate, indicating better customer retention or satisfaction within this user segment. This insight can guide strategic decisions for the telecom company, highlighting areas for improvement in their Fiber optic service and potentially leveraging the satisfaction factors of DSL service to enhance overall customer retention.

```
# Data for the pie charts
Label = ["Churn: Yes", "Churn: No"]
Value = [1869, 5174]  # 1869 customers are labeled as churned and 5174 as not churned
GenderLabel = ["F", "M", "F", "M"]
GenderSize = [939, 930, 2544, 2619]  #Count of 'Yes' & 'No' Churn by Gender
Color =  ['#ff9999', '#66cccc'] # ['#ffcc99', '#ffcc99']   #
GenderColor = ['#ffcc99', '#99ccff', '#ffcc99', '#99ccff'] #['#ffcc99', '#99ccff', '#ffcc99', '#99ccff']
Explode = (0.5, 0.5)
GenderExplode = (0.1, 0.1, 0.1, 0.1)
TextAttrib = {"fontsize": 14, "color": "white"}

plt.style.use('grayscale') # Sets the background colour

fig, axs = plt.subplots(figsize=(5, 7)) # Creates subplots with matplotlib

# Plotting Churn distribution pie chart with percentages
axs.pie(Value, labels=Label, autopct='%1.1f%%', pctdistance=1.08, labeldistance=0.8, colors=Color,
        startangle=90, frame=True, explode=Explode, radius=10, textprops=TextAttrib, counterclock=True)
axs.pie(GenderSize, labels=GenderLabel, colors=GenderColor, startangle=90, explode=GenderExplode,
        radius=7, textprops=TextAttrib, counterclock=True)

# Draws the circle
centre_circle = plt.Circle((0, 0), 5, color='grey', fc='grey', linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Churn Distribution Vs. Gender: Female(F), Male(M)', fontsize=15, color='white', y=1.1)

axs.axis('equal') # This ensures that pie is drawn as a circle.

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



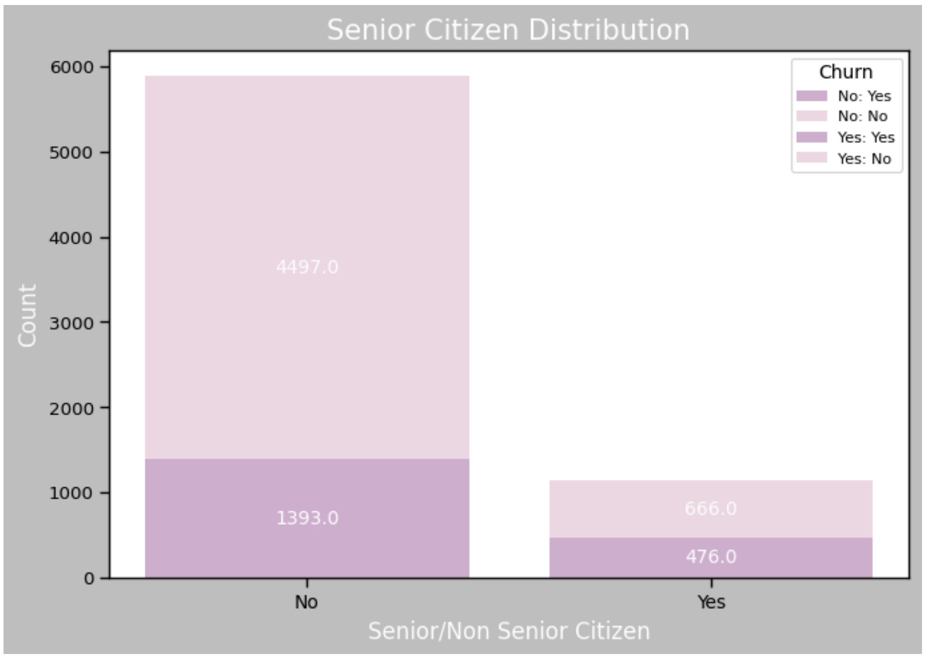
The data showcased through pie charts indicates that the proportion of churned female customers is slightly higher compared to males. The labels "Churn: Yes" and "Churn: No" respectively depict the proportion of customers who churned against those who didn't. This highlights a marginally greater percentage of churn among female customers compared to their male counterparts.

```
IbroColor = {'Yes': '#55cbcd', 'No': '#cce2cb'}
DependentY = TelcoData[TelcoData['Dependents'] == 'Yes']['Churn']
DependentX = TelcoData[TelcoData['Dependents'] == 'No']['Churn']

plt.style.use('grayscale') # Sets the background colour

fig, ax = plt.subplots(figsize=(7, 6)) # Creates subplots with Matplotlib

ax.hist([DependentY, DependentX], bins=2, alpha=0.9, label=['Dependents: No', 'Dependents: Yes'], color=['#cce2cb', '#55cb

plt.title('Customers with Dependents', fontsize=15, color='white')
plt.xlabel('Churn', fontsize=12, color='white')
plt.ylabel('Count', fontsize=12, color='white')

# Setting x-axis ticks and labels
plt.xticks([0.25, 0.75], ['Churn: Yes', 'Churn: No'], color='black')

plt.legend(title='Dependents', fontsize=8, title_fontsize='10', loc='upper right') # Adding legend and adjusting its proper

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



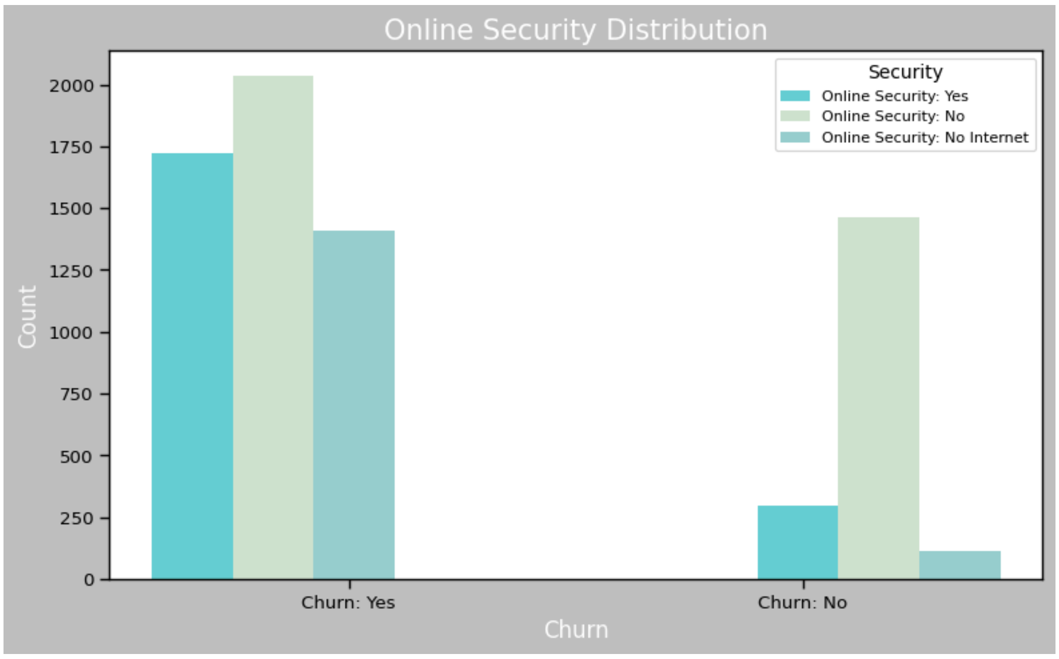
A significant distinction in churn rates between customers with and without dependents is shown. Those without dependents exhibit a notably higher tendency to churn compared to those with dependents. This suggests a potential trend where customers without dependents might be more inclined to switch service providers, possibly due to unmet needs or preferences.

```
Churnlabel = TelcoData['Churn']
Service = TelcoData['SeniorCitizen'].unique()
plt.style.use('grayscale') # Sets the background colour
fig, ax = plt.subplots(figsize=(7, 5))   # Creating subplots with matplotlib

for i, contract in enumerate(Service):
  # Iterate through each contract type
    Data = Churnlabel[TelcoData['SeniorCitizen'] == contract]
    Count = Data.value_counts()
    Total = Count.sum()
    Percentages = Count / Total * 100  # Calculate percentages

  # Plotting histogram
    ax.bar(i, Count['Yes'], alpha=0.9, label=f'{contract}: Yes', color='#cbaacb')
    ax.bar(i, Count['No'], alpha=0.9, bottom=Count['Yes'], label=f'{contract}: No', color='#ecd5e3')

    ax.text(i, Count['Yes'] / 2, f"{Count['Yes']:.1f}", ha='center', va='center', color='white', fontsize=10) # Annotate pe
    ax.text(i, Count['Yes'] + Count['No'] / 2, f"{Count['No']:.1f}", ha='center', va='center', color='white', fontsize=10)

plt.title('Senior Citizen Distribution', fontsize=15, color='white') # Adding title and labels
plt.xlabel('Senior/Non Senior Citizen', fontsize=12, color='white')
plt.ylabel('Count', fontsize=12, color='white')

plt.legend(title='Churn', fontsize=8, title_fontsize='10', loc='upper right') # Adding legend and adjusting its properties

plt.xticks(range(len(Service)), Service, fontsize=10, color='black') # Set x-axis ticks and labels

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



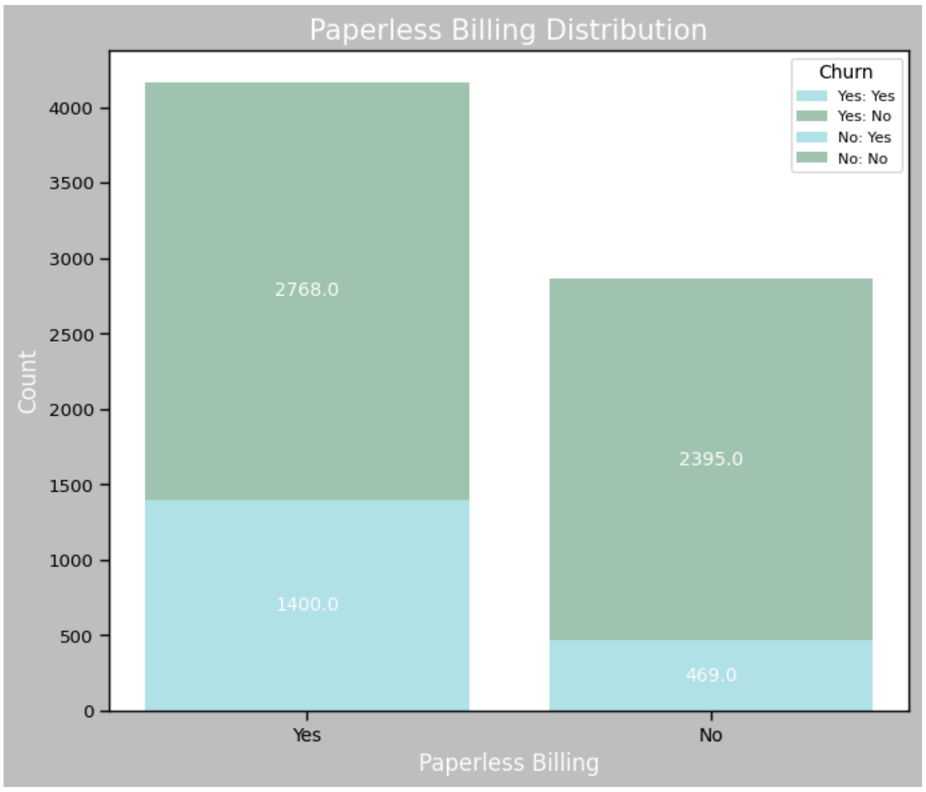
The histogram highlights the distribution of churn among senior citizens, showcasing a notably lower representation of senior citizens within the customer base. Despite their lower numbers, there's a substantial churn rate among this demographic, indicating a trend where senior citizens are more likely to discontinue services compared to non-senior customers. This trend might suggest specific issues or dissatisfaction points within the services offered to this segment, potentially requiring further investigation and tailored strategies to improve service satisfaction and retention among senior customers.

```
IbroColor = {'Yes': '#55cbcd', 'No': '#cce2cb', 'No Internet':'#8fcaca'}
dependentX = TelcoData[TelcoData['OnlineSecurity'] == 'Yes']['Churn']
dependentY = TelcoData[TelcoData['OnlineSecurity'] == 'No']['Churn']
dependentZ = TelcoData[TelcoData['OnlineSecurity'] == 'No Internet']['Churn']
plt.style.use('grayscale') # Sets the background colour
fig, ax = plt.subplots(figsize=(8, 5)) # Creates subplots with Matplotlib

# Plotting histogram
ax.hist([dependentX, dependentY, dependentZ], bins=3, alpha=0.9, label=['Online Security: Yes', 'Online Security: No', 'On

# Adding title and labels
plt.title('Online Security Distribution', fontsize=15, color='white')
plt.xlabel('Churn', fontsize=12, color='white')
plt.ylabel('Count', fontsize=12, color='white')

# Setting x-axis ticks and labels
plt.xticks([0.25, 0.75], ['Churn: Yes', 'Churn: No'], color='black')

# Adding legend and adjusting its properties
plt.legend(title='Security', fontsize=8, title_fontsize='10', loc='upper right')

# Show plot
plt.tight_layout()
plt.show()
```



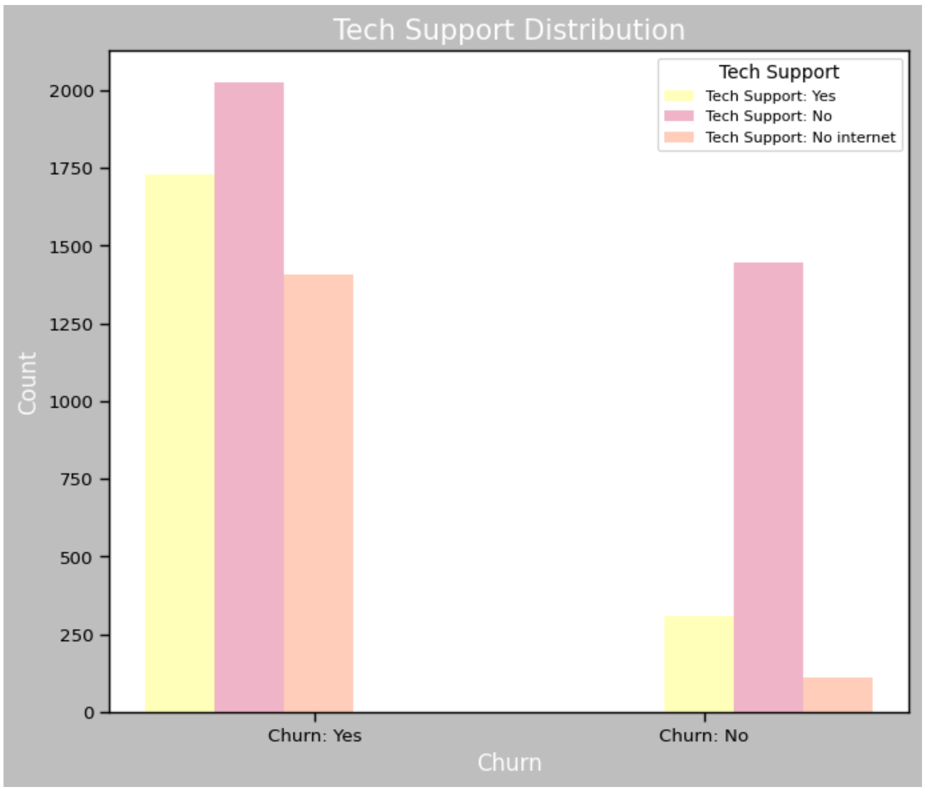
The histogram provides an insight into the churn distribution concerning online security among customers. It vividly illustrates that a substantial number of customers who lack online security features are more inclined to churn compared to those with online security or without internet services. This trend emphasizes the significant impact of online security on customer retention, suggesting that the absence of adequate security measures might be a critical factor leading to higher churn rates.

```
Churnlabel = TelcoData['Churn']
Service = TelcoData['PaperlessBilling'].unique()
plt.style.use('grayscale') # Sets the background colour

# Create subplots with matplotlib
fig, ax = plt.subplots(figsize=(7, 6))

# Iterate through each contract type
for i, contract in enumerate(Service):
    Data = Churnlabel[TelcoData['PaperlessBilling'] == contract]
    Count = Data.value_counts()
    Total = Count.sum()
    Percentages = Count / Total * 100  # Calculate percentages

    # Plotting histogram
    ax.bar(i, Count['Yes'], alpha=0.9, label=f'{contract}: Yes', color='#abdee6')
    ax.bar(i, Count['No'], alpha=0.9, bottom=Count['Yes'], label=f'{contract}: No', color='#97c1a9')

    # Annotate percentages on the plot
    ax.text(i, Count['Yes'] / 2, f"{Count['Yes']:.1f}", ha='center', va='center', color='white', fontsize=10)
    ax.text(i, Count['Yes'] + Count['No'] / 2, f"{Count['No']:.1f}", ha='center', va='center', color='white', fontsize=10)

# Adding title and labels
plt.title('Paperless Billing Distribution', fontsize=15, color='white')
plt.xlabel('Paperless Billing', fontsize=12, color='white')
plt.ylabel('Count', fontsize=12, color='white')

# Adding legend and adjusting its properties
plt.legend(title='Churn', fontsize=8, title_fontsize='10', loc='upper right')

# Set x-axis ticks and labels
plt.xticks(range(len(Service)), Service, fontsize=10, color='black')

# Show plot
plt.tight_layout()
plt.show()
```



The code reveals that customers opting for paperless billing are more prone to churn compared to those who don't. This could suggest that the method of paperless billing might somehow contribute to a higher churn rate. Further analysis might be necessary to understand the reasons behind this correlation and explore potential strategies to reduce churn among customers using paperless billing.

```
IbroColor = {'Yes': '#ffffb5', 'No': '#f3b0c3', 'No Internet':'#ffccb6'}
dependentX = TelcoData[TelcoData['TechSupport'] == 'Yes']['Churn']
dependentY = TelcoData[TelcoData['TechSupport'] == 'No']['Churn']
dependentZ = TelcoData[TelcoData['TechSupport'] == 'No Internet']['Churn']

plt.style.use('grayscale') # Sets the background colour
fig, ax = plt.subplots(figsize=(7, 6))  # Creates subplots with Matplotlib

ax.hist([dependentX, dependentY, dependentZ], bins=3, alpha=0.9, label=['Tech Support: Yes', 'Tech Support: No', 'Tech Sup

plt.title('Tech Support Distribution', fontsize=15, color='white') # Adding the title
plt.xlabel('Churn', fontsize=12, color='white') # Adding the labels on the x-axis
plt.ylabel('Count', fontsize=12, color='white') # Adding the labels on the y-axis
plt.xticks([0.25, 0.75], ['Churn: Yes', 'Churn: No'], color='black') # Setting x-axis ticks and labels
plt.legend(title='Tech Support', fontsize=8, title_fontsize='10', loc='upper right')  # Adding legend and adjusting its pro

# Show plot
plt.tight_layout()
plt.show()
```



Expectedly, the graph shows that customers without technical support are more inclined to switch service providers. This insight could suggest that the availability of technical assistance plays a crucial role in customer retention. Companies might benefit from enhancing their tech support services to reduce churn, potentially improving customer satisfaction and loyalty.

5.0 Telco Customer Churn Data Preprocessing for Modelling

```
Dropping=['customerID'] #To drop the customerID column
TelcoData1=TelcoData.drop(Dropping,axis=1)
TelcoData1.head(3)
```

	customerID.1	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	On
0	Female	No	Yes	No	1	No	No phone service	DSL	No	
1	Male	No	No	No	34	Yes	No	DSL	Yes	
2	Male	No	No	No	2	Yes	No	DSL	Yes	

5.1 Feature Encoding of the Telco Customer Churn Dataset

```
EncodedCols = [      # Defining the encoded dataset
    'customerID.1', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService','MultipleLines', 'OnlineSecurity',
    'OnlineBackup','DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies','PaperlessBilling', 'Churn',
]

EncodedData = TelcoData1.copy()  # Create a copy of the TelcoData1 dataframe
OEncoder = OrdinalEncoder() # Initializes an ordinal encoder

EncodedVals = OEncoder.fit_transform(EncodedData[EncodedCols]) + 1  # Fits and transforms the listed columns

EncodedData[EncodedCols] = EncodedVals # Replace the original columns with the encoded values in the DataFrame
```

The 'EncodedData' now, contains the specified columns with values starting from 1, as we divide the encoded data for *'Ordinal Encoding'* and *'Hot Encoding'*.

```
EncodedData.head()
#EncodedData.shape
```

	customerID.1	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	O
0	1.0	1.0	2.0	1.0	1	1.0	2.0	DSL	1.0	
1	2.0	1.0	1.0	1.0	34	2.0	1.0	DSL	3.0	
2	2.0	1.0	1.0	1.0	2	2.0	1.0	DSL	3.0	
3	2.0	1.0	1.0	1.0	45	1.0	2.0	DSL	3.0	
4	1.0	1.0	1.0	1.0	2	2.0	1.0	Fiber optic	1.0	

```
# To encode the target variable 'Churn' as 0 and 1
EncodedData['Churn'] = EncodedData['Churn'].map({1: 0, 2: 1}) # This code maps '1' to 0 and '2' to 1 in the 'Churn' variab

EncodedData.head()
```

	customerID.1	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	O
0	1.0	1.0	2.0	1.0	1	1.0	2.0	DSL	1.0	
1	2.0	1.0	1.0	1.0	34	2.0	1.0	DSL	3.0	
2	2.0	1.0	1.0	1.0	2	2.0	1.0	DSL	3.0	
3	2.0	1.0	1.0	1.0	45	1.0	2.0	DSL	3.0	
4	1.0	1.0	1.0	1.0	2	2.0	1.0	Fiber optic	1.0	

```
EncodedCols2 = ['PaymentMethod', 'Contract', 'InternetService']
DataEncode = EncodedData[EncodedCols2] # Subset the DataFrame with the columns to encode

OHEncoder = OneHotEncoder(sparse=False, drop='first')  # Initializes OneHotEncoder

EncodedVal2 = OHEncoder.fit_transform(DataEncode)  # Fits and transforms the data

ColsEncode = OHEncoder.get_feature_names_out(input_features=EncodedCols2) # Creating new columnnames

EncodedData2 = pd.DataFrame(EncodedVal2, columns=ColsEncode, index=DataEncode.index) # Create a DataFrame from the encoded

EncodedData.drop(EncodedCols2, axis=1, inplace=True) # Drops the original columns from the EncodedData
EncodedData = pd.concat([EncodedData, EncodedData2], axis=1)   #Concatenates the encoded columns
```

```
EncodedData.head()
```

	customerID.1	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurity	OnlineBackup	Devi
0	1.0	1.0	2.0	1.0	1	1.0	2.0	1.0	3.0	
1	2.0	1.0	1.0	1.0	34	2.0	1.0	3.0	1.0	
2	2.0	1.0	1.0	1.0	2	2.0	1.0	3.0	3.0	
3	2.0	1.0	1.0	1.0	45	1.0	2.0	3.0	1.0	
4	1.0	1.0	1.0	1.0	2	2.0	1.0	1.0	1.0	

5 rows × 24 columns

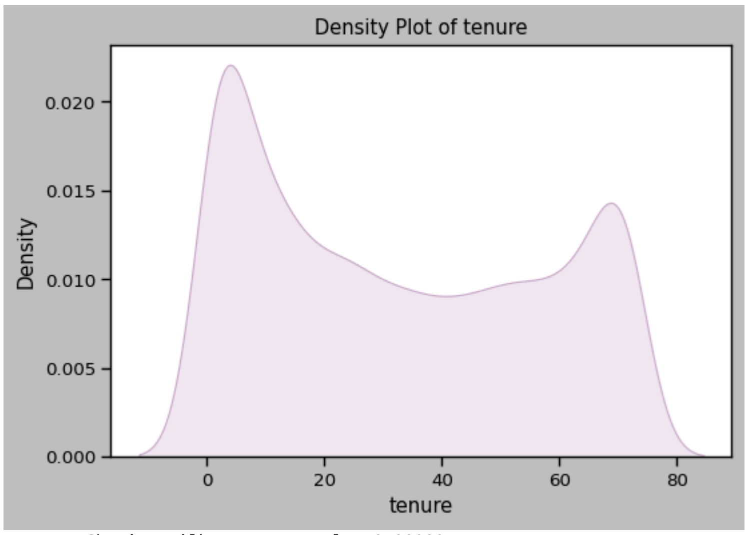
```
# Check for normality using KDE plots and Shapiro-Wilk test for each column
for col in columns_to_check:
    # Plot KDE plot
    plt.figure(figsize=(6, 4))
    sns.kdeplot(df_encoded[col], shade=True)
    plt.title(f'KDE Plot of {col}')
    plt.xlabel(col)
    plt.ylabel('Density')
    plt.show()
# Shapiro-Wilk test for normality
    stat, p = shapiro(df_encoded[col])
```

```
        alpha = 0.05
        print(f'{col}: Shapiro-Wilk Test - p-value={p:.4f}')
        if p > alpha:
            print(f'  The {col} column looks normally distributed (fail to reject H0)\n')
        else:
            print(f'  The {col} column does not look normally distributed (reject H0)\n')
```

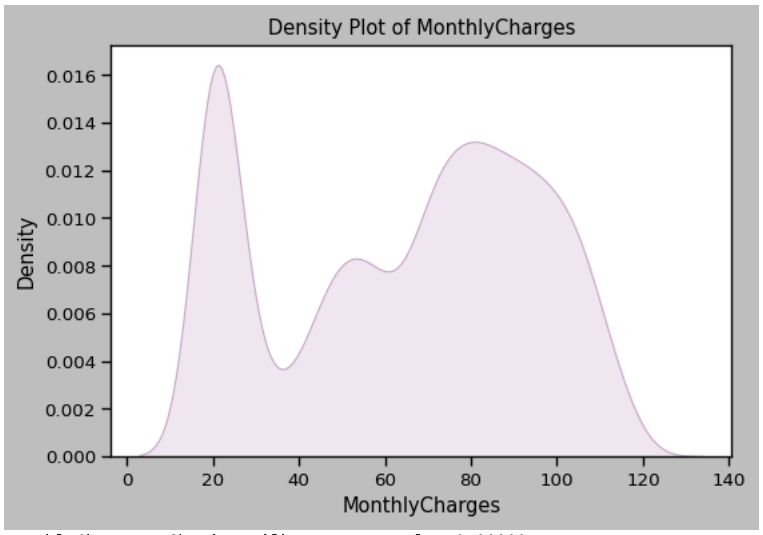
5.2 Test for Normal Distribution of the Telco Customer Data

In this session, the 'tenure','MonthlyCharges' and 'TotalCharges'attributes will be checked for normally using KDE plots while employing Shapiro-Wilk test for atttibute.

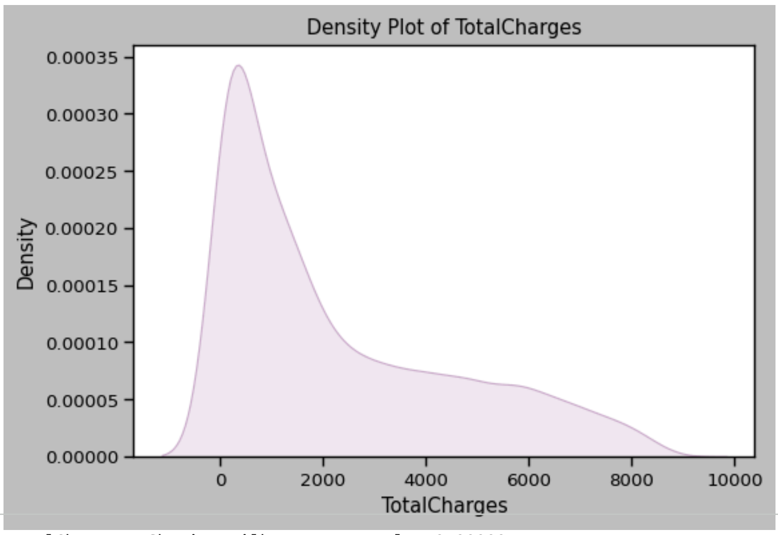
```
    NormalityCol = ['tenure', 'MonthlyCharges', 'TotalCharges']  # Checks the columns to check for normality

    for col in NormalityCol:
        plt.figure(figsize=(6, 4))
        sns.kdeplot(EncodedData[col], shade=True, color='#cbaacb')  # Adding color to the KDE plot
        plt.title(f'Density Plot of {col}')
        plt.xlabel(col)
        plt.ylabel('Density')
        plt.show()

        stat, p = shapiro(EncodedData[col])  # Shapiro-Wilk test for normality
        alpha = 0.05
        print(f'{col}: Shapiro-Wilk Test - p-value={p:.5f}')
        if p > alpha:
            print(f'  The {col} column looks normally distributed (fail to reject H0)\n')
        else:
            print(f'  The {col} column does not look normally distributed (reject H0)\n')
```



```
tenure: Shapiro-Wilk Test - p-value=0.00000
  The tenure column does not look normally distributed (reject H0)
```



```
MonthlyCharges: Shapiro-Wilk Test - p-value=0.00000
  The MonthlyCharges column does not look normally distributed (reject H0)
```



```
TotalCharges: Shapiro-Wilk Test - p-value=0.00000
  The TotalCharges column does not look normally distributed (reject H0)
```

None of the customer's attributes (tenure, MonthlyCharges, TotalCharges) passed the normality test based on the Shapiro-Wilk test. The small p-values (all zero) indicate a rejection of the null hypothesis (H0) that the data is normally distributed. This revealed that these columns significantly deviate from a normal distribution.

5.3 Rescaling the Customer's Data for Prediction Analysis

Standard scaling, applied to 'tenure', 'MonthlyCharges', and 'TotalCharges', normalizes these numerical attributes by centering their values around 0 and scaling them to have a standard deviation of 1. This normalization process is vital for ensuring comparability between variables measured on different scales, preventing dominance of larger-scale features, and facilitating better performance in machine learning models. By standardizing the data, it enhances the model's stability and ensures fairer comparisons among features, benefiting algorithms that rely on distance calculations or gradient-based optimization methods.

```
ColsToCheck = ['tenure', 'MonthlyCharges', 'TotalCharges']

DataToScale = EncodedData[ColsToCheck] # Subsets the EncodedData with the attributes to standardize

StdScaler = StandardScaler() # Initializes StandardScaler

ScaledData = StdScaler.fit_transform(DataToScale) # Fits and transforms the data

ScaledCol = [col + '_scaled' for col in ColsToCheck] # Creates new column names for the scaled columns

ScaledData2 = pd.DataFrame(ScaledData, columns=ScaledCol, index=DataToScale.index) # Creates a DataFrame from the scaled da

EncodedData.drop(ColsToCheck, axis=1, inplace=True)  # Drops the original columns from EncodedData
EncodedData = pd.concat([EncodedData, ScaledData2], axis=1)  # Concatenates the scaled attributes
```

```
EncodedData.head(3)
```

	customerID.1	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	OnlineSecurity	OnlineBackup	DeviceProtec
0	1.0	1.0	2.0	1.0	1.0	2.0	1.0	3.0	
1	2.0	1.0	1.0	1.0	2.0	1.0	3.0	1.0	
2	2.0	1.0	1.0	1.0	2.0	1.0	3.0	3.0	

3 rows × 24 columns

5.4 Train-Test Split of the Dataset

```
TargetColumn = 'Churn'

y = EncodedData[TargetColumn]
X = EncodedData.drop(TargetColumn, axis=1)  # This code drops the target column (churn) from the features

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=23)  # Splits the dataset into train

print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (5625, 23)
X_test shape: (1407, 23)
y_train shape: (5625,)
y_test shape: (1407,)
```

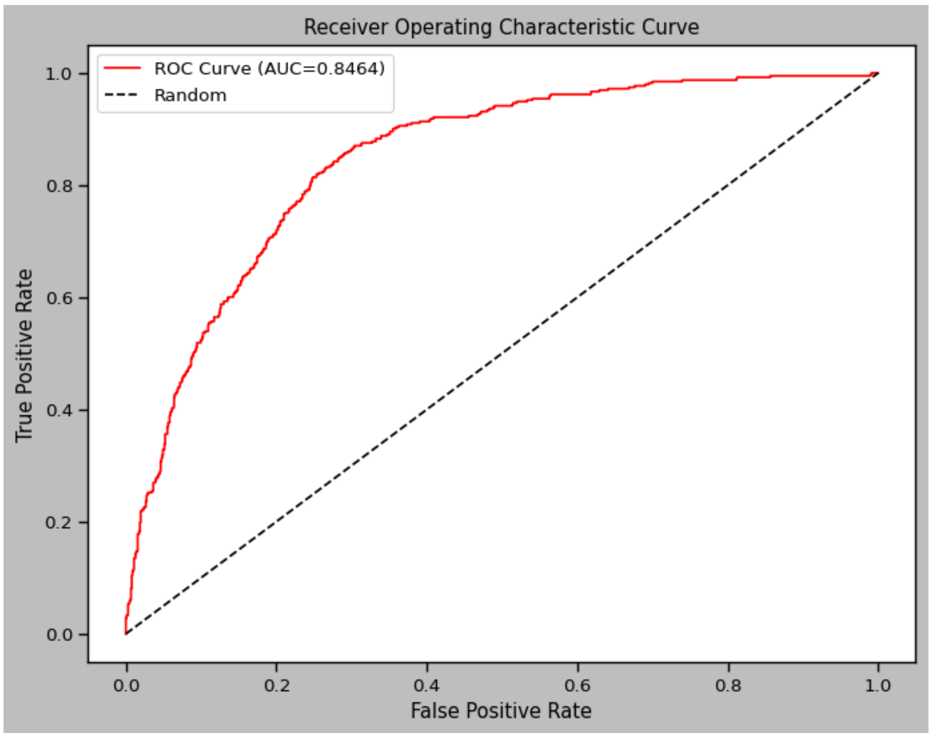
```
RfClassifier = RandomForestClassifier(random_state=23)

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

GridSearch = GridSearchCV(
    RfClassifier,
    param_grid,
    cv=5,
    scoring='roc_auc',
    n_jobs=-1  # Sets n_jobs to -1 to use all available cores
)

GridSearch.fit(X_train, y_train)

BestParam = GridSearch.best_params_
BestEstimator = GridSearch.best_estimator_

y_pred = BestEstimator.predict(X_test)
y_prob = BestEstimator.predict_proba(X_test)[:, 1]

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
ROCScore = roc_auc_score(y_test, y_prob)
print(f"ROC AUC: {ROCScore:.4f}")
```

```
Accuracy: 0.7932
ROC AUC: 0.8464
```

This code employs a Random Forest Classifier with hyperparameter tuning using GridSearchCV to optimize its performance in predicting customer churn, providing insights into accuracy and ROC AUC. The use of parallel processing enhances the efficiency of the hyperparameter search. The Receiver Operating Characteristic Curve is shown below

```
fpr, tpr, thresholds = roc_curve(y_test, y_prob) #This code will plot Reciever Operating Characteristic (ROC) curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='red', label=f'ROC Curve (AUC={ROCScore:.4f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve')
plt.legend()
plt.show()
```

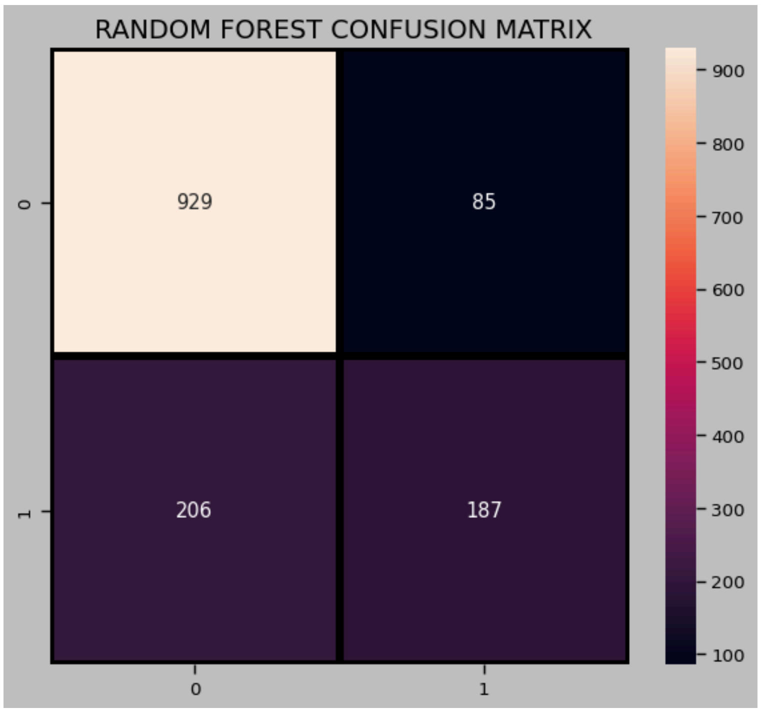


With a ROC AUC of 0.8464, the model is demonstrating a good balance between sensitivity and specificity.

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.92      0.86      1014
           1       0.69      0.48      0.56       393

    accuracy                           0.79      1407
   macro avg       0.75      0.70      0.71      1407
weighted avg       0.78      0.79      0.78      1407
```

```
plt.figure(figsize=(7,6))
sns.heatmap(confusion_matrix(y_test, y_pred),
                annot=True,fmt = "d",linecolor="k",linewidths=3)

plt.title(" RANDOM FOREST CONFUSION MATRIX",fontsize=14)
plt.show()
```



The classification report delves into the model's performance across two distinctive classes: '0' and '1'. It showcased an impressive precision of 83% and commendable recall of 91% for class '0', illustrating the model's proficiency in accurately predicting this category. Conversely, for class '1', the model's performance was comparatively weaker, demonstrating a precision of 66% and recall of 50%, indicating notable challenges in correctly identifying this class. The overall accuracy of 80% signifies the proportion of accurately classified samples. Detailed in the confusion matrix, the model correctly classified 929 instances of class '0' while misclassifying 95 instances as class '1'. Conversely, for class '1', it correctly identified 187 instances but erroneously classified an equal number as class '0'. Enhancing the model's ability to distinguish and capture instances of class '1' could significantly bolster its overall accuracy and predictive performance.

5.5 Handling Class Imbalance by BorderlineSMOTE

BorderlineSMOTE is a method used for oversampling in dealing with class imbalance. It is available in the imblearn.over_sampling module, a part of the imbalanced-learn library. It aims to improve the classifier's ability to capture the complex decision boundaries present in the dataset, potentially enhancing the performance of the model in correctly predicting the minority class.

```
from imblearn.over_sampling import BorderlineSMOTE

BLSmote = BorderlineSMOTE(random_state=23, kind = 'borderline-2') # This code runs the BorderlineSMOTE

print("Shapes before BorderlineSMOTE:")  # Prints shapes before applying BorderlineSMOTE
print(f"X_train shape: {X_train.shape}")
print(f"y_train shape: {y_train.shape}")

X_train_balanced, y_train_balanced = BLSmote.fit_resample(X_train, y_train)  # Applying BorderlineSMOTE on the training dat

print("\nShapes after BorderlineSMOTE:") # Print shapes after applying BorderlineSMOTE
print(f"X_train_balanced shape: {X_train_balanced.shape}")
print(f"y_train_balanced shape: {y_train_balanced.shape}")
```

```
Shapes before BorderlineSMOTE:
X_train shape: (5625, 23)
y_train shape: (5625,)

Shapes after BorderlineSMOTE:
X_train_balanced shape: (8298, 23)
y_train_balanced shape: (8298,)
```

The balanced training set now contains more instances, specifically targeting the minority class to alleviate the class imbalance issue. The labels for the corresponding samples also increased to 8259 to match the augmented dataset.

6.0 Model Training and Evaluation Algorithms

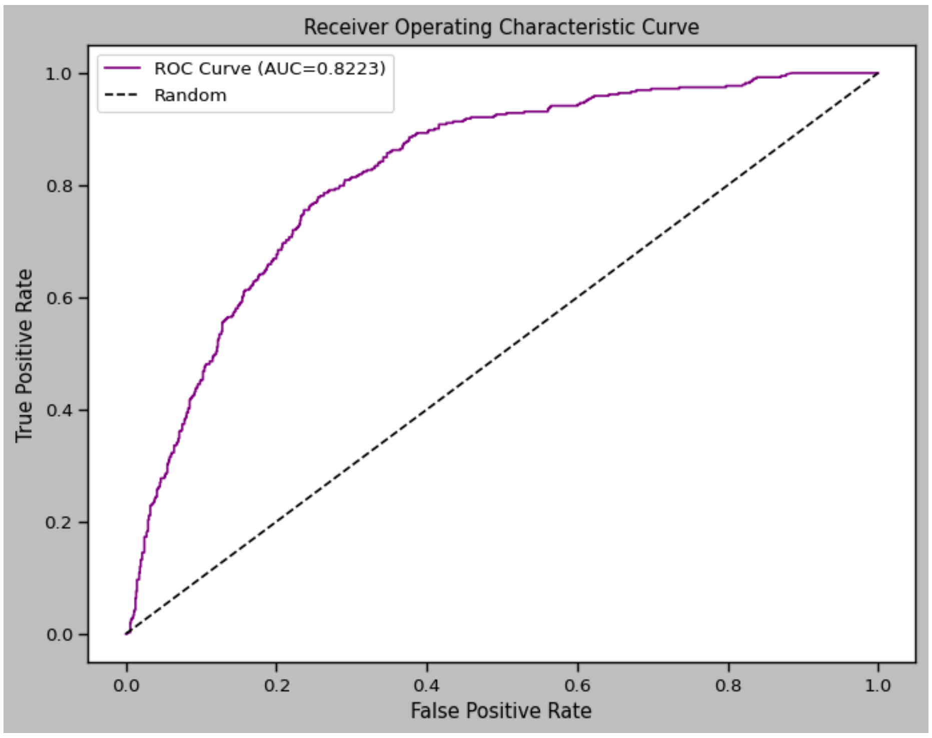
XGBoost, CatBoost, Voting Classifier, and Stacking Classifier algorithms will be considered in this context of model training and ensemble learning to improve the model performance while opting for the algorithm with the best performance at the end of the excercise.

6.1. The XGBoost Algorithm

```
XGB_Classifier = XGBClassifier(random_state=23)    # Initializes the XGBoost Classifier

param_grid = {      # Defines the parameter grid for GridSearchCV
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7, 10],
    'learning_rate': [0.001, 0.01, 0.1, 0.2]
}

GridSearch = GridSearchCV(XGB_Classifier, param_grid, cv=5, scoring='roc_auc')  # Initializes GridSearchCV


GridSearch.fit(X_train_balanced, y_train_balanced)  # Fits the model

BestParam = GridSearch.best_params_  # Obtains the best parameters and best estimator
BestEstimator = GridSearch.best_estimator_  # Obtains the best estimator

y_pred = BestEstimator.predict(X_test)  # Predicts on the test set
y_prob = BestEstimator.predict_proba(X_test)[:, 1]  # Predicts on the test set

accuracy = accuracy_score(y_test, y_pred)  # To calculate accuracy
print(f"Accuracy: {accuracy:.4f}")

ROCScore = roc_auc_score(y_test, y_prob)  # To calculate ROC AUC
print(f"ROC AUC: {ROCScore:.4f}")
```

```
Accuracy: 0.7939
ROC AUC: 0.8398
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_prob) # Plots Reciever Operating Characteristic (ROC) curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='purple', label=f'ROC Curve (AUC={ROCScore:.4f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve')
plt.legend()
plt.show()
```


Receiver Operating Characteristic Curve
ROC Curve (AUC=0.8223)
Random
True Positive Rate
False Positive Rate