Telecom Customer Churn Prediction

Library Importation

In [1]:
```
#!pip install folium
```

In [2]:
```
#!pip install imbalanced-learn
```

In [3]:
```
#Making necessary imports
import numpy as np  # linear algebra
import pandas as pd # data processing
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns
import os, sys
import folium
import sklearn
from sklearn import svm
from sklearn.svm import SVC
from sklearn import metrics
from sklearn import tree
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.neighbors import NeighborhoodComponentsAnalysis,KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import precision_score, recall_score , f1_score
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.naive_bayes import GaussianNB
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from scipy.stats import chi2_contingency
from sklearn.utils import shuffle
from sklearn.preprocessing import MinMaxScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler
import warnings
from matplotlib.colors import LinearSegmentedColormap
from sklearn.metrics import precision_score, recall_score, ConfusionMatrixDisplay
from itertools import cycle
from sklearn.tree import plot_tree
from scipy.stats import randint
from sklearn import model_selection
from sklearn.pipeline import make_pipeline
from sklearn.metrics import precision_recall_curve
import warnings
from sklearn.exceptions import ConvergenceWarning
```

Data Importation

In [4]:
```
# Loading in the Dataset
Churn =pd.read_csv("Churn.csv")
```

Initial Data Analysis

In [5]:
```
#Checking head of dataset (first 10)
Churn.head(n=10)
```

Out[5]:

	Customer ID	Gender	Age	Married	Number of Dependents	City	Zip Code	Latitude	Longitude	Num Refer
0	0002-ORFBO	Female	37	Yes	0	Frazier Park	93225	34.827662	-118.999073	
1	0003-MKNFE	Male	46	No	0	Glendale	91206	34.162515	-118.203869	
2	0004-TLHLJ	Male	50	No	0	Costa Mesa	92627	33.645672	-117.922613	
3	0011-IGKFF	Male	78	Yes	0	Martinez	94553	38.014457	-122.115432	
4	0013-EXCHZ	Female	75	Yes	0	Camarillo	93010	34.227846	-119.079903	
5	0013-MHZWF	Female	23	No	3	Midpines	95345	37.581496	-119.972762	
6	0013-SMEOE	Female	67	Yes	0	Lompoc	93437	34.757477	-120.550507	
7	0014-BMAQU	Male	52	Yes	0	Napa	94558	38.489789	-122.270110	
8	0015-UOCOJ	Female	68	No	0	Simi Valley	93063	34.296813	-118.685703	
9	0016-QLJIS	Female	43	Yes	1	Sheridan	95681	38.984756	-121.345074	

10 rows × 38 columns

In [6]:
```
# Checking size of dataset
Churn.shape
```

Out[6]:
```
(7043, 38)
```

In [7]:
```
# Checking summary statistics of numerical features
Churn.describe()
```

Out[7]:

	Age	Number of Dependents	Zip Code	Latitude	Longitude	Number of Referrals	Tenure ir Months
count	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000
mean	46.509726	0.468692	93486.070567	36.197455	-119.756684	1.951867	32.386767
std	16.750352	0.962802	1856.767505	2.468929	2.154425	3.001199	24.542061
min	19.000000	0.000000	90001.000000	32.555828	-124.301372	0.000000	1.000000
25%	32.000000	0.000000	92101.000000	33.990646	-121.788090	0.000000	9.000000
50%	46.000000	0.000000	93518.000000	36.205465	-119.595293	0.000000	29.000000
75%	60.000000	0.000000	95329.000000	38.161321	-117.969795	3.000000	55.000000
max	80.000000	9.000000	96150.000000	41.962127	-114.192901	11.000000	72.000000

In [8]:
```
#Checking data types of each features and the number of non-null values.
Churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 38 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   Customer ID                        7043 non-null   object 
 1   Gender                             7043 non-null   object 
 2   Age                                7043 non-null   int64  
 3   Married                            7043 non-null   object 
 4   Number of Dependents               7043 non-null   int64  
 5   City                               7043 non-null   object 
 6   Zip Code                           7043 non-null   int64  
 7   Latitude                           7043 non-null   float64
 8   Longitude                          7043 non-null   float64
 9   Number of Referrals                7043 non-null   int64  
 10  Tenure in Months                   7043 non-null   int64  
 11  Offer                              3166 non-null   object 
 12  Phone Service                      7043 non-null   object 
 13  Avg Monthly Long Distance Charges  6361 non-null   float64
 14  Multiple Lines                     6361 non-null   object 
 15  Internet Service                   7043 non-null   object 
 16  Internet Type                      5517 non-null   object 
 17  Avg Monthly GB Download            5517 non-null   float64
 18  Online Security                    5517 non-null   object 
 19  Online Backup                      5517 non-null   object 
 20  Device Protection Plan             5517 non-null   object 
 21  Premium Tech Support               5517 non-null   object 
 22  Streaming TV                       5517 non-null   object 
 23  Streaming Movies                   5517 non-null   object 
 24  Streaming Music                    5517 non-null   object 
 25  Unlimited Data                     5517 non-null   object 
 26  Contract                           7043 non-null   object 
 27  Paperless Billing                  7043 non-null   object 
 28  Payment Method                     7043 non-null   object 
 29  Monthly Charge                     7043 non-null   float64
 30  Total Charges                      7043 non-null   float64
 31  Total Refunds                      7043 non-null   float64
 32  Total Extra Data Charges           7043 non-null   int64  
 33  Total Long Distance Charges        7043 non-null   float64
 34  Total Revenue                      7043 non-null   float64
 35  Customer Status                    7043 non-null   object 
 36  Churn Category                     1869 non-null   object 
 37  Churn Reason                       1869 non-null   object 
dtypes: float64(9), int64(6), object(23)
memory usage: 2.0+ MB
```

In [9]:
```
# Checking data types
Churn.dtypes
```

Out[9]:
```
Customer ID                         object
Gender                              object
Age                                  int64
Married                             object
Number of Dependents                 int64
City                                object
Zip Code                             int64
Latitude                           float64
Longitude                          float64
Number of Referrals                  int64
Tenure in Months                     int64
Offer                               object
Phone Service                       object
Avg Monthly Long Distance Charges  float64
Multiple Lines                      object
Internet Service                    object
Internet Type                       object
Avg Monthly GB Download            float64
Online Security                     object
Online Backup                       object
Device Protection Plan              object
Premium Tech Support                object
Streaming TV                        object
Streaming Movies                    object
Streaming Music                     object
Unlimited Data                      object
Contract                            object
Paperless Billing                   object
Payment Method                      object
Monthly Charge                     float64
Total Charges                      float64
Total Refunds                      float64
Total Extra Data Charges             int64
Total Long Distance Charges        float64
Total Revenue                      float64
Customer Status                     object
Churn Category                      object
Churn Reason                        object
dtype: object
```

Removing Joined from Target Variables

The joined observation would be removed from the target variable because the aim of the analysis is to predict telecom customer churn & not whether new customer joined. To predict the customer churn we would be using the "Stayed" or "churned".

In [11]:
```
Churn = Churn[Churn['Customer Status'] != 'Joined']
```

In [12]:
```
print(Churn['Customer Status'])
```

```
0        Stayed
1        Stayed
2       Churned
3       Churned
4       Churned
         ...
7037    Churned
7038     Stayed
7039    Churned
7041     Stayed
7042     Stayed
Name: Customer Status, Length: 6589, dtype: object
```

In [14]:
```
# Checking shape of datset after removing joined
Churn.shape
```

Out[14]:
```
(6589, 38)
```

In [15]:
```
#Checking tail of dataset (last 10)
Churn.tail(n=10)
```

Out[15]:

	Customer ID	Gender	Age	Married	Number of Dependents	City	Zip Code	Latitude	Longitude	F
7031	9974-JFBHQ	Male	31	No	1	Del Rey	93616	36.657462	-119.595293	
7032	9975-GPKZU	Male	62	Yes	3	Alameda	94501	37.774633	-122.274434	
7034	9978-HYCIN	Male	72	Yes	1	Bakersfield	93301	35.383937	-119.020428	
7035	9979-RGMZT	Female	20	No	0	Los Angeles	90022	34.023810	-118.156582	
7036	9985-MWVIX	Female	53	No	0	Hume	93628	36.807595	-118.901544	
7037	9986-BONCE	Female	36	No	0	Fallbrook	92028	33.362575	-117.299644	
7038	9987-LUTYD	Female	20	No	0	La Mesa	91941	32.759327	-116.997260	
7039	9992-RRAMN	Male	40	Yes	0	Riverbank	95367	37.734971	-120.954271	
7041	9993-LHIEB	Male	21	Yes	0	Solana Beach	92075	33.001813	-117.263628	
7042	9995-HOTOH	Male	36	Yes	0	Sierra City	96125	39.600599	-120.636358	

10 rows × 38 columns

Exploratory Data Analysis(EDA) / Data Visualisation

Univariate Analysis

Dividing features into Categorical and Numerical for easy Visualisation

In [16]:
```
categorical_columns = ['Gender','Married','City', 'Offer', 'Phone Service', 'Multip
                       'Internet Type', 'Online Security', 'Number of Dependents',
                       'Device Protection Plan', 'Premium Tech Support', 'Streaming
                       'Streaming Movies', 'Streaming Music', 'Unlimited Data', 'Cc
numerical_columns = ['Age', 'Tenure in Months', 'Avg Monthly Long Distance Charges'
                     'Avg Monthly GB Download', 'Monthly Charge', 'Total Charges',
                     'Total Long Distance Charges', 'Total Revenue']
```

In [17]:
```
Categorical = Churn[categorical_columns]
Numerical = Churn[numerical_columns]
```

In [18]:
```
#Checking head of datase
Categorical.head()
```

Out[18]:

	Gender	Married	City	Offer	Phone Service	Multiple Lines	Internet Service	Internet Type	Online Security	Number of Dependents
0	Female	Yes	Frazier Park	NaN	Yes	No	Yes	Cable	No	0
1	Male	No	Glendale	NaN	Yes	Yes	Yes	Cable	No	0
2	Male	No	Costa Mesa	Offer E	Yes	No	Yes	Fiber Optic	No	0
3	Male	Yes	Martinez	Offer D	Yes	No	Yes	Fiber Optic	No	0
4	Female	Yes	Camarillo	NaN	Yes	No	Yes	Fiber Optic	No	0

5 rows × 25 columns

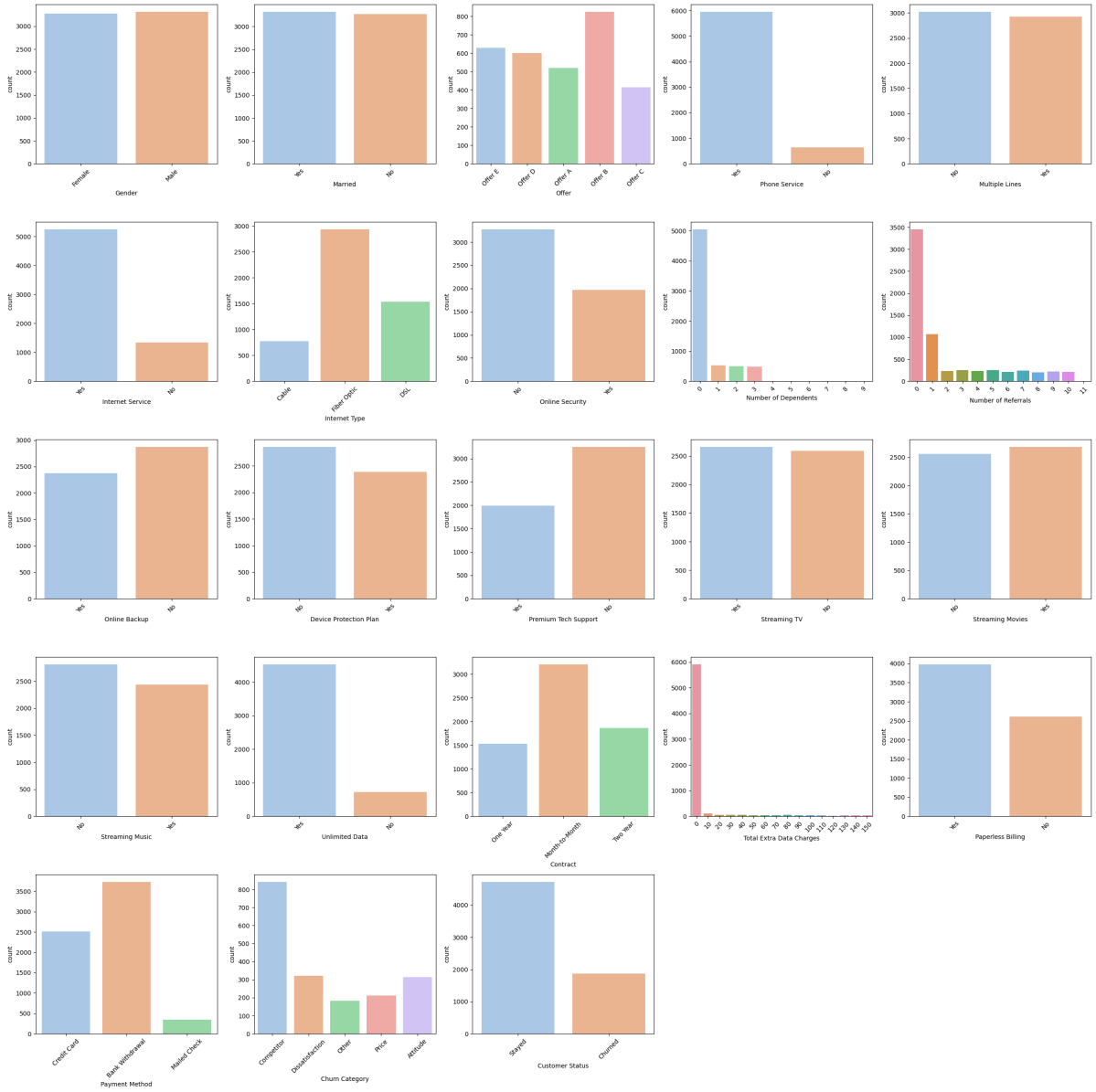
In [19]:
```
#Checking head of dataset
Numerical.head()
```

Out[19]:

	Age	Tenure in Months	Avg Monthly Long Distance Charges	Avg Monthly GB Download	Monthly Charge	Total Charges	Total Refunds	Total Long Distance Charges	Total Revenue
0	37	9	42.39	16.0	65.6	593.30	0.00	381.51	974.81
1	46	9	10.69	10.0	-4.0	542.40	38.33	96.21	610.28
2	50	4	33.65	30.0	73.9	280.85	0.00	134.60	415.45
3	78	13	27.82	4.0	98.0	1237.85	0.00	361.66	1599.51
4	75	3	7.38	11.0	83.9	267.40	0.00	22.14	289.54

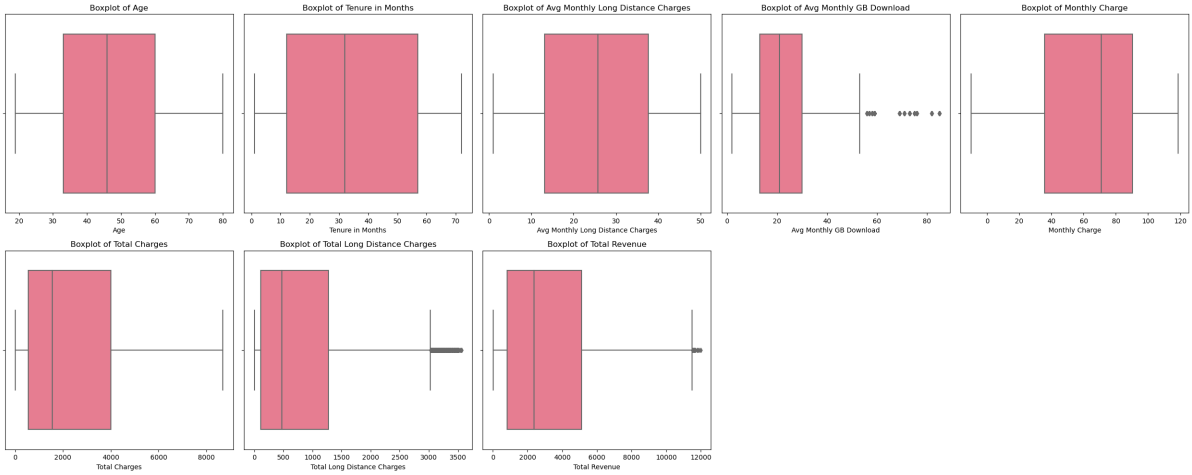
In [20]:

```
# Defining the number of rows and columns for the grid
num_cols = 5
num_rows = (len(Categorical.columns) - 2 + num_cols - 1) // num_cols
# Creating a grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(5*num_cols, 5*num_rows))
# Flatten the axes array
axes = axes.flatten()
# Setting color palette
color = sns.set_palette("pastel")
# Columns to iterate over excluding "city" and "churn reason"
columns_to_plot = [col for col in Categorical.columns if col not in ["City", "Churr
# Iterating over each column in the DataFrame
for i, column in enumerate(columns_to_plot):
    ax = axes[i]  # Get the subplot axes
    sns.countplot(x=column, data=Categorical, ax=ax, palette= color)
    ax.tick_params(axis='x', rotation=45)  # Rotate x-axis labels for better readab
# Remove empty subplots
for i in range(len(columns_to_plot), num_rows * num_cols):
    fig.delaxes(axes[i])
# Adjust layout
plt.tight_layout()
# Save image to include in PDF
plt.savefig('Distribution_of_Categorical_Feature.jpg')
# Show the plot
plt.show()
```



In [21]:

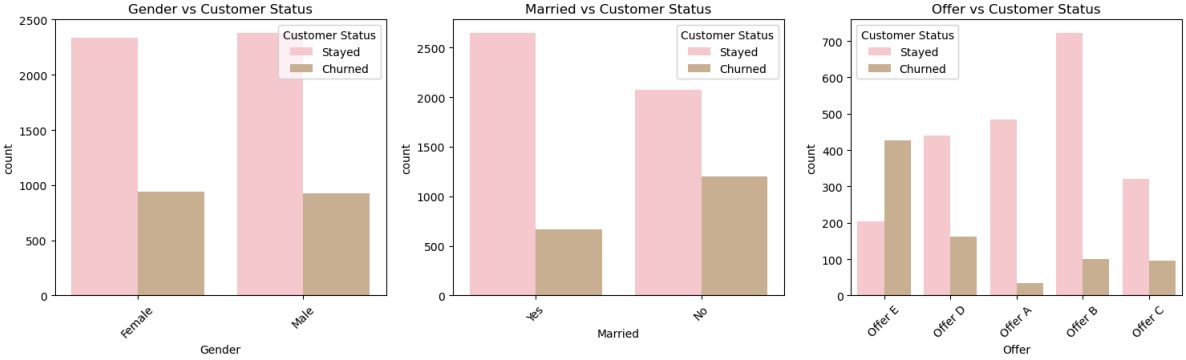
```
# Setting color palette
sns.set_palette("husl", 9)
# Defining the number of rows and columns for the grid
num_cols = 5
num_rows = (len(Numerical.columns) + num_cols - 1) // num_cols
# Creating a grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(5*num_cols, 5*num_rows))
# Flatten the axes array
axes = axes.flatten()
# Columns to iterate over excluding "Total Refunds"
columns_to_plot = [col for col in Numerical.columns if col != "Total Refunds"]
# Iterating over each column in the DataFrame
for i, column in enumerate(columns_to_plot):
    ax = axes[i]  # Get the subplot axes
    # Plot boxplot for numerical columns
    sns.boxplot(x=column, data=Numerical, ax=ax)
    ax.set_title(f"Boxplot of {column}")
# Remove empty subplots
for i in range(len(columns_to_plot), num_rows * num_cols):
    fig.delaxes(axes[i])
# Adjust layout
plt.tight_layout()
# Save image to include in PDF
plt.savefig('Boxplot_of_each_Numerical_Feature.jpg')
plt.show()
```



Bi-Variate Analysis

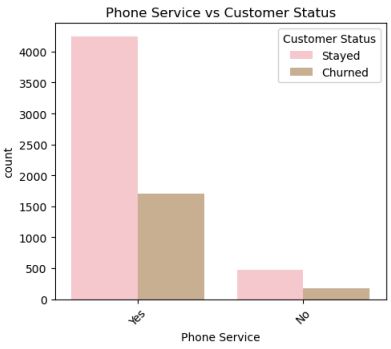
Visualising relationship between Categorical features and target variable

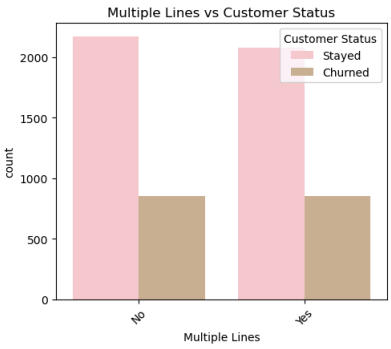
In [22]:
```
# Define the target variable
target_variable = 'Customer Status'
# Exclude columns from categorical_columns
columns_to_plot = [col for col in categorical_columns if col not in ['City', 'Churr
# Calculate the number of columns for the subplot grid
num_cols = 3  # You may adjust this value as needed
# Calculate the actual number of rows needed
num_rows = (len(columns_to_plot) + num_cols - 1) // num_cols
# Create a grid of subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows))
# Flatten the axes array
axes = axes.flatten()
# Iterate over each categorical column
for i, column in enumerate(columns_to_plot):
    # Get the subplot axes
    ax = axes[i]
    # Plot the relationship using a bar plot
    colour = ['pink','tan']
    sns.set_palette(colour)
    sns.countplot(x=column, hue=target_variable, data=Categorical, ax=ax)
    # Set title
    ax.set_title(f'{column} vs {target_variable}')
  # Rotate x-axis labels for better readability
    ax.tick_params(axis='x', rotation=45)
# Remove any remaining empty subplots
for j in range(len(columns_to_plot), num_rows * num_cols):
    fig.delaxes(axes[j])
# Adjust layout
plt.tight_layout()
plt.savefig("Bivariate Analysis.jpg")
plt.show()
```

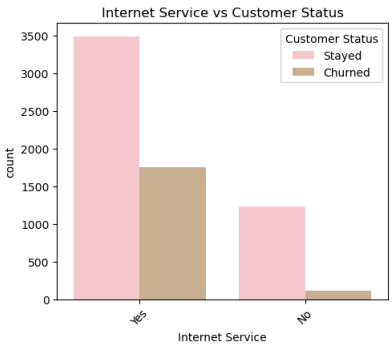

Gender vs Customer Status
Customer Status
Stayed
Churned
count
Female
Male
Gender

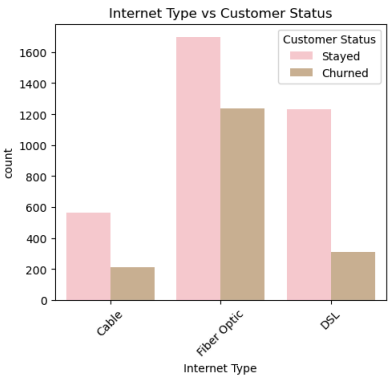
Married vs Customer Status
Customer Status
Stayed
Churned
count
Yes
No
Married

Offer vs Customer Status
Customer Status
Stayed
Churned
count
Offer E
Offer D
Offer A
Offer B
Offer C
Offer

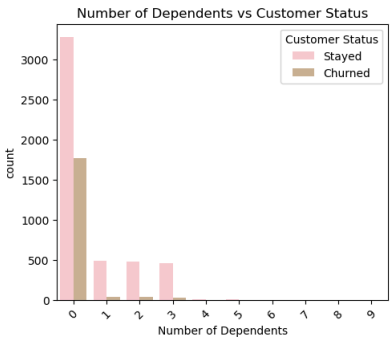

Phone Service vs Customer Status
Customer Status
Stayed
Churned
count
Yes
No
Phone Service

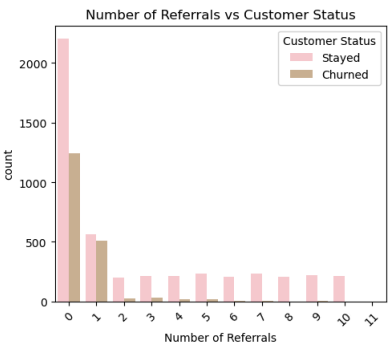

Multiple Lines vs Customer Status
Customer Status
Stayed
Churned
count
No
Yes
Multiple Lines

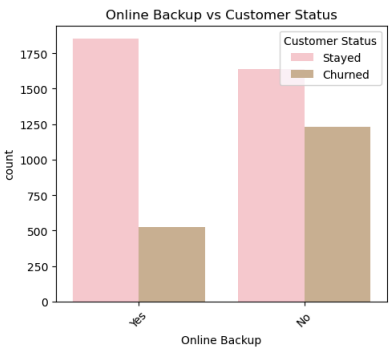

Internet Service vs Customer Status
Customer Status
Stayed
Churned
count
Yes
No
Internet Service

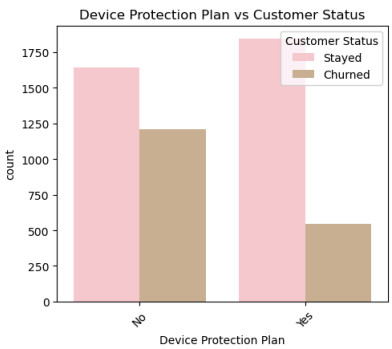

Internet Type vs Customer Status
Customer Status
Stayed
Churned
count
Cable
Fiber Optic
DSL
Internet Type

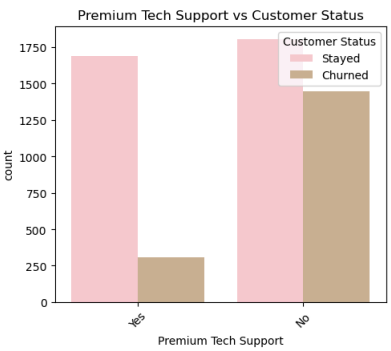

Online Security vs Customer Status
Customer Status
Stayed
Churned
count
No
Yes
Online Security

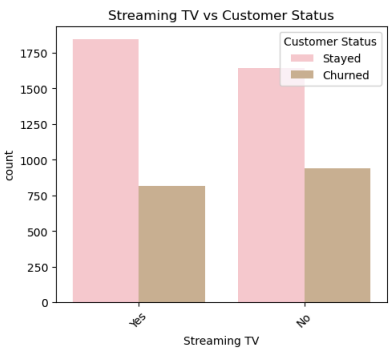

Number of Dependents vs Customer Status
Customer Status
Stayed
Churned
count
Number of Dependents

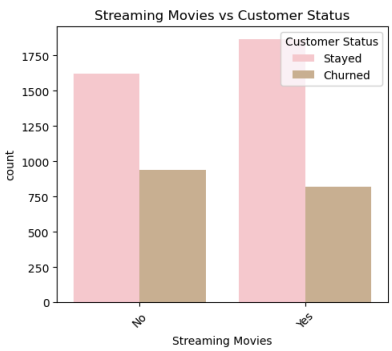

Number of Referrals vs Customer Status
Customer Status
Stayed
Churned
count
Number of Referrals

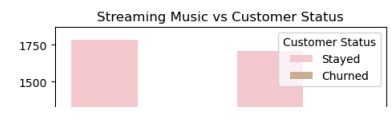

Online Backup vs Customer Status
Customer Status
Stayed
Churned
count
Yes
No
Online Backup

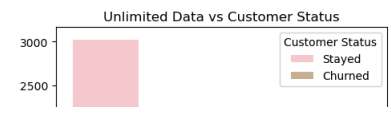

Device Protection Plan vs Customer Status
Customer Status
Stayed
Churned
count
No
Yes
Device Protection Plan

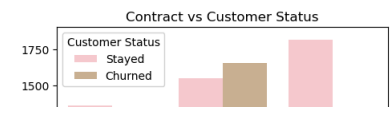

Premium Tech Support vs Customer Status
Customer Status
Stayed
Churned
count
Yes
No
Premium Tech Support

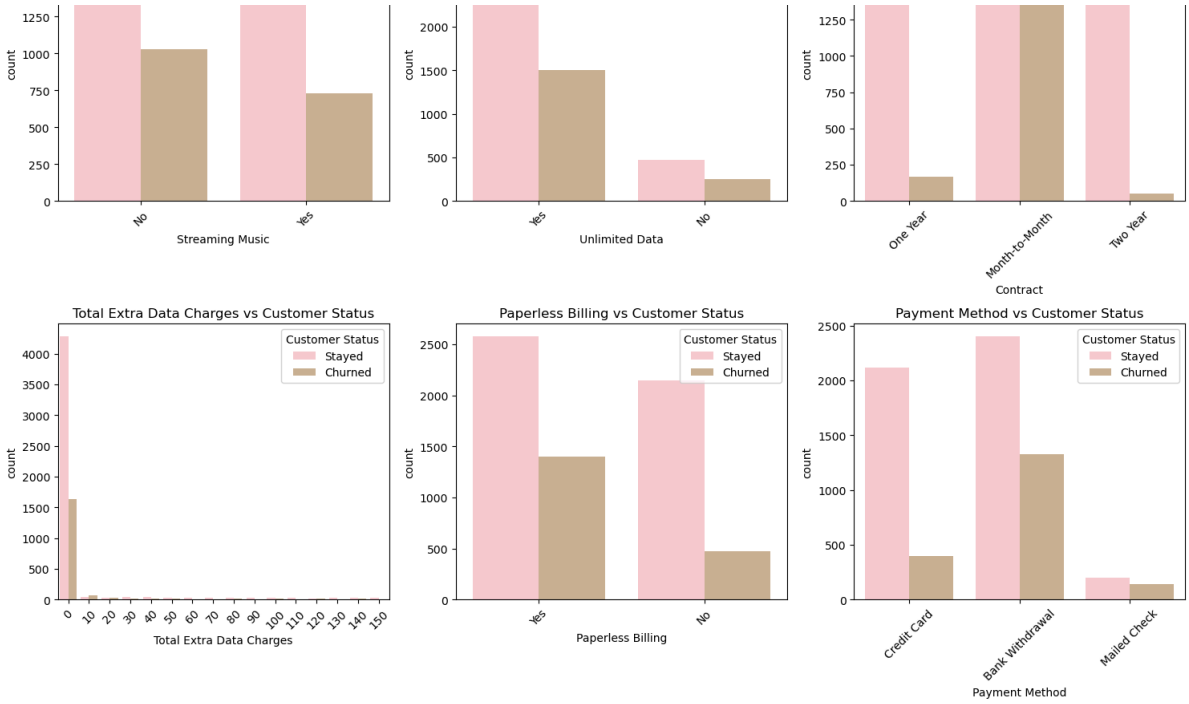

Streaming TV vs Customer Status
Customer Status
Stayed
Churned
count
Yes
No
Streaming TV


Streaming Movies vs Customer Status
Customer Status
Stayed
Churned
count
No
Yes
Streaming Movies


Streaming Music vs Customer Status
Customer Status
Stayed
Churned


Unlimited Data vs Customer Status
Customer Status
Stayed
Churned


Contract vs Customer Status
Customer Status
Stayed
Churned



In [23]:
```
# Group by city and churn status
city_churn_counts = Churn.groupby(['City', 'Customer Status']).size().unstack().res
city_churn_counts.fillna(0, inplace=True)  # Fill NaN values with 0 for cities with

# Create a map centered around a latitude and longitude
churn_map = folium.Map(location=[Churn['Latitude'].mean(), Churn['Longitude'].mean(

# Add markers for each city
for index, row in city_churn_counts.iterrows():
    City = row['City']
    Churned = row['Churned'] if 'Churned' in row else 0
    Stayed = row['Stayed'] if 'Stayed' in row else 0
    marker_popup = f"{City}<br>Churned: {Churned}<br>Stayed: {Stayed}"
    folium.Marker(
        location=[Churn[Churn['City'] == City]['Latitude'].values[0], Churn[Churn['
        popup=marker_popup,
        icon=folium.Icon(color='red' if Churned > Stayed else 'green')
    ).add_to(churn_map)

# Add a tooltip to show city name and churned/stayed count when hovering over a mar
folium.Marker(
    location=[0, 0],  # Dummy location
    icon=None,
    tooltip=folium.Tooltip("<b>Click a marker to see details</b>")
).add_to(churn_map)

# Display the map
churn_map
```

Out[23]:

Data Cleaning

Removing Unnecessary Columns

In [24]:
```
#Removing unnecessary Columns
Churn.drop(["Customer ID", "Zip Code", "City", "Latitude", "Longitude", "Churn Reas
```

In [25]:
```
# Checking head of dataset after removing columns
Churn.head(n=30)
```

Out[25]:

	Gender	Age	Married	Number of Dependents	Number of Referrals	Tenure in Months	Offer	Phone Service	Avg Monthly Long Distance Charges	Multiple Lines	...
0	Female	37	Yes	0	2	9	NaN	Yes	42.39	No	...
1	Male	46	No	0	0	9	NaN	Yes	10.69	Yes	...
2	Male	50	No	0	0	4	Offer E	Yes	33.65	No	...
3	Male	78	Yes	0	1	13	Offer D	Yes	27.82	No	...
4	Female	75	Yes	0	3	3	NaN	Yes	7.38	No	...
5	Female	23	No	3	0	9	Offer E	Yes	16.77	No	...
6	Female	67	Yes	0	1	71	Offer A	Yes	9.96	No	...
7	Male	52	Yes	0	8	63	Offer B	Yes	12.96	Yes	...
8	Female	68	No	0	0	7	Offer E	Yes	10.53	No	...
9	Female	43	Yes	1	3	65	NaN	Yes	28.46	Yes	...
10	Male	47	No	0	0	54	NaN	No	NaN	NaN	...
11	Female	25	Yes	2	2	72	NaN	Yes	16.01	Yes	...
12	Female	58	Yes	0	0	5	NaN	Yes	18.65	No	...
13	Female	32	No	0	0	72	Offer A	Yes	2.25	Yes	...
14	Female	39	No	0	0	56	NaN	No	NaN	NaN	...
15	Female	58	Yes	2	9	71	Offer A	Yes	27.26	Yes	...
16	Female	52	Yes	1	0	34	NaN	No	NaN	NaN	...
18	Male	79	No	0	0	45	NaN	Yes	10.67	No	...
19	Male	67	No	0	0	1	NaN	No	NaN	NaN	...
20	Female	79	Yes	0	0	50	NaN	Yes	31.43	Yes	...
21	Female	26	Yes	0	1	13	NaN	Yes	43.56	Yes	...

	Gender	Age	Married	Number of Dependents	Number of Referrals	Tenure in Months	Offer	Phone Service	Avg Monthly Long Distance Charges	Multiple Lines	...
22	Female	30	Yes	2	1	23	Offer D	Yes	34.91	Yes	...
24	Female	34	Yes	0	0	4	Offer E	Yes	12.43	No	...
25	Female	37	Yes	1	1	1	NaN	No	NaN	NaN	...
26	Female	37	Yes	3	10	55	NaN	Yes	35.04	No	...
27	Male	42	Yes	3	3	54	NaN	Yes	19.70	No	...
28	Male	64	No	0	0	26	Offer C	Yes	37.70	No	...
29	Female	47	Yes	3	5	69	NaN	Yes	38.39	No	...
30	Male	23	Yes	3	1	37	Offer C	Yes	43.01	No	...
31	Female	48	Yes	0	5	49	Offer B	Yes	49.58	No	...

30 rows × 31 columns

Checking for Null in Columns and replacing with median and mode

In [26]:
```
#Checking number of null in each column
Churn.isnull().sum()
```

Out[26]:

```
Gender                                0
Age                                   0
Married                               0
Number of Dependents                  0
Number of Referrals                   0
Tenure in Months                      0
Offer                              3598
Phone Service                         0
Avg Monthly Long Distance Charges   644
Multiple Lines                      644
Internet Service                      0
Internet Type                      1344
Avg Monthly GB Download            1344
Online Security                    1344
Online Backup                      1344
Device Protection Plan             1344
Premium Tech Support               1344
Streaming TV                       1344
Streaming Movies                   1344
Streaming Music                    1344
Unlimited Data                     1344
Contract                              0
Paperless Billing                     0
Payment Method                        0
Monthly Charge                        0
Total Charges                         0
Total Refunds                         0
Total Extra Data Charges              0
Total Long Distance Charges           0
Total Revenue                         0
Customer Status                       0
dtype: int64
```

In [27]:

```
# Calculate total number of rows
total_rows = Churn.shape[0]
# Calculate number of missing values in each column
missing_values = Churn.isnull().sum()
# Calculate percentage of missing values in each column
percentage_missing = (missing_values / total_rows) * 100
# Display the percentage of missing values in each column
print("Percentage of missing values in each column:")
print(percentage_missing)
```

```
Percentage of missing values in each column:
Gender                                 0.000000
Age                                    0.000000
Married                                0.000000
Number of Dependents                   0.000000
Number of Referrals                    0.000000
Tenure in Months                       0.000000
Offer                                 54.606162
Phone Service                          0.000000
Avg Monthly Long Distance Charges      9.773866
Multiple Lines                         9.773866
Internet Service                       0.000000
Internet Type                         20.397632
Avg Monthly GB Download               20.397632
Online Security                       20.397632
Online Backup                         20.397632
Device Protection Plan                20.397632
Premium Tech Support                  20.397632
Streaming TV                          20.397632
Streaming Movies                      20.397632
Streaming Music                       20.397632
Unlimited Data                        20.397632
Contract                               0.000000
Paperless Billing                      0.000000
Payment Method                         0.000000
Monthly Charge                         0.000000
Total Charges                          0.000000
Total Refunds                          0.000000
Total Extra Data Charges               0.000000
Total Long Distance Charges            0.000000
Total Revenue                          0.000000
Customer Status                        0.000000
dtype: float64
```

In [28]:
```
#Removing unnecessary Columns
Churn.drop(["Offer"], axis =1, inplace= True)
```

In [29]:
```
# Replacing missing values in numerical columns with the median
numerical_columns = ['Avg Monthly Long Distance Charges', 'Avg Monthly GB Download'
for column in numerical_columns:
    median_value = Churn[column].median()
    Churn[column].fillna(median_value, inplace=True)
```

In [30]:
```
# Replacing missing values in categorical columns with the mode
categorical_columns = ['Internet Type', 'Online Security', 'Multiple Lines', 'Inter
                       'Online Backup', 'Device Protection Plan', 'Premium Tech Sup
                       'Streaming Music', 'Unlimited Data']
for column in categorical_columns:
    mode_value = Churn[column].mode()[0]
    Churn[column].fillna(mode_value, inplace=True)
```

In [31]:
```
#Checking number of null in each column after replacing missing values
Churn.isnull().sum()
```

Out[31]:
```
Gender                                0
Age                                   0
Married                               0
Number of Dependents                  0
Number of Referrals                   0
Tenure in Months                      0
Phone Service                         0
Avg Monthly Long Distance Charges     0
Multiple Lines                        0
Internet Service                      0
Internet Type                         0
Avg Monthly GB Download               0
Online Security                       0
Online Backup                         0
Device Protection Plan                0
Premium Tech Support                  0
Streaming TV                          0
Streaming Movies                      0
Streaming Music                       0
Unlimited Data                        0
Contract                              0
Paperless Billing                     0
Payment Method                        0
Monthly Charge                        0
Total Charges                         0
Total Refunds                         0
Total Extra Data Charges              0
Total Long Distance Charges           0
Total Revenue                         0
Customer Status                       0
dtype: int64
```

In [32]:
```
Churn.duplicated().sum()
```

Out[32]:
```
0
```

In [33]:
```
# Checking head of dataset after replacing NaN Values
Churn.head(n=30)
```

Out[33]:

	Gender	Age	Married	Number of Dependents	Number of Referrals	Tenure in Months	Phone Service	Avg Monthly Long Distance Charges	Multiple Lines	Internet Service
0	Female	37	Yes	0	2	9	Yes	42.39	No	Yes
1	Male	46	No	0	0	9	Yes	10.69	Yes	Yes
2	Male	50	No	0	0	4	Yes	33.65	No	Yes
3	Male	78	Yes	0	1	13	Yes	27.82	No	Yes
4	Female	75	Yes	0	3	3	Yes	7.38	No	Yes
5	Female	23	No	3	0	9	Yes	16.77	No	Yes
6	Female	67	Yes	0	1	71	Yes	9.96	No	Yes
7	Male	52	Yes	0	8	63	Yes	12.96	Yes	Yes
8	Female	68	No	0	0	7	Yes	10.53	No	Yes
9	Female	43	Yes	1	3	65	Yes	28.46	Yes	Yes
10	Male	47	No	0	0	54	No	25.72	No	Yes
11	Female	25	Yes	2	2	72	Yes	16.01	Yes	Yes
12	Female	58	Yes	0	0	5	Yes	18.65	No	Yes
13	Female	32	No	0	0	72	Yes	2.25	Yes	Yes
14	Female	39	No	0	0	56	No	25.72	No	Yes
15	Female	58	Yes	2	9	71	Yes	27.26	Yes	Yes
16	Female	52	Yes	1	0	34	No	25.72	No	Yes
18	Male	79	No	0	0	45	Yes	10.67	No	Yes
19	Male	67	No	0	0	1	No	25.72	No	Yes
20	Female	79	Yes	0	0	50	Yes	31.43	Yes	No
21	Female	26	Yes	0	1	13	Yes	43.56	Yes	Yes

	Gender	Age	Married	Number of Dependents	Number of Referrals	Tenure in Months	Phone Service	Avg Monthly Long Distance Charges	Multiple Lines	Internet Service
22	Female	30	Yes	2	1	23	Yes	34.91	Yes	Yes
24	Female	34	Yes	0	0	4	Yes	12.43	No	No
25	Female	37	Yes	1	1	1	No	25.72	No	Yes
26	Female	37	Yes	3	10	55	Yes	35.04	No	Yes
27	Male	42	Yes	3	3	54	Yes	19.70	No	No
28	Male	64	No	0	0	26	Yes	37.70	No	No
29	Female	47	Yes	3	5	69	Yes	38.39	No	No
30	Male	23	Yes	3	1	37	Yes	43.01	No	Yes
31	Female	48	Yes	0	5	49	Yes	49.58	No	No

30 rows × 30 columns

Encoding Categorical Features

In [34]:
```
# Checking data types before encoding
Churn.dtypes
```

Out[34]:
```
Gender                              object
Age                                  int64
Married                             object
Number of Dependents                 int64
Number of Referrals                  int64
Tenure in Months                     int64
Phone Service                       object
Avg Monthly Long Distance Charges  float64
Multiple Lines                      object
Internet Service                    object
Internet Type                       object
Avg Monthly GB Download            float64
Online Security                     object
Online Backup                       object
Device Protection Plan              object
Premium Tech Support                object
Streaming TV                        object
Streaming Movies                    object
Streaming Music                     object
Unlimited Data                      object
Contract                            object
Paperless Billing                   object
Payment Method                      object
Monthly Charge                     float64
Total Charges                      float64
Total Refunds                      float64
Total Extra Data Charges             int64
Total Long Distance Charges        float64
Total Revenue                      float64
Customer Status                     object
dtype: object
```

In [35]:
```
# Convert categorical columns (with 2 unique values) to numerical
# List of categorical columns excluding specified columns
categorical_columns = [col for col in Churn.select_dtypes(include=['object']).colum
                       if col not in ['Internet Type', 'Contract', 'Payment Method'
# Convert categorical columns to numerical using Label Encoding
for column in categorical_columns:
    le = LabelEncoder()
    Churn[column] = le.fit_transform(Churn[column])
```

In [36]:
```
# Checking head of dataset after Encoding features with 2 unique values
Churn.head()
```

Out[36]:

	Gender	Age	Married	Number of Dependents	Number of Referrals	Tenure in Months	Phone Service	Avg Monthly Long Distance Charges	Multiple Lines	Internet Service
0	0	37	1	0	2	9	1	42.39	0	1
1	1	46	0	0	0	9	1	10.69	1	1
2	1	50	0	0	0	4	1	33.65	0	1
3	1	78	1	0	1	13	1	27.82	0	1
4	0	75	1	0	3	3	1	7.38	0	1

5 rows × 30 columns

In [37]:
```
# Extract the columns you want to encode
columns_to_encode = ['Internet Type', 'Contract', 'Payment Method']
# Apply One-Hot Encoding
encoded_columns = pd.get_dummies(Churn[columns_to_encode])
# Drop the original categorical columns from the DataFrame
Churn = Churn.drop(columns_to_encode, axis=1)
# Concatenate the encoded columns with the original DataFrame
Churn = pd.concat([Churn, encoded_columns], axis=1)
```

In [38]:
```
# Checking head of dataset after Encoding
Churn.head(n=10)
```

Out[38]:

	Gender	Age	Married	Number of Dependents	Number of Referrals	Tenure in Months	Phone Service	Avg Monthly Long Distance Charges	Multiple Lines	Internet Service
0	0	37	1	0	2	9	1	42.39	0	1
1	1	46	0	0	0	9	1	10.69	1	1
2	1	50	0	0	0	4	1	33.65	0	1
3	1	78	1	0	1	13	1	27.82	0	1
4	0	75	1	0	3	3	1	7.38	0	1
5	0	23	0	3	0	9	1	16.77	0	1
6	0	67	1	0	1	71	1	9.96	0	1
7	1	52	1	0	8	63	1	12.96	1	1
8	0	68	0	0	0	7	1	10.53	0	1
9	0	43	1	1	3	65	1	28.46	1	1

10 rows × 36 columns

In [39]:
```
# Checking data types after encoding
Churn.dtypes
```

Out[39]:
```
Gender                                int32
Age                                   int64
Married                               int32
Number of Dependents                  int64
Number of Referrals                   int64
Tenure in Months                      int64
Phone Service                         int32
Avg Monthly Long Distance Charges   float64
Multiple Lines                        int32
Internet Service                      int32
Avg Monthly GB Download             float64
Online Security                       int32
Online Backup                         int32
Device Protection Plan                int32
Premium Tech Support                  int32
Streaming TV                          int32
Streaming Movies                      int32
Streaming Music                       int32
Unlimited Data                        int32
Paperless Billing                     int32
Monthly Charge                      float64
Total Charges                       float64
Total Refunds                       float64
Total Extra Data Charges              int64
Total Long Distance Charges         float64
Total Revenue                       float64
Customer Status                       int32
Internet Type_Cable                    bool
Internet Type_DSL                      bool
Internet Type_Fiber Optic              bool
Contract_Month-to-Month                bool
Contract_One Year                      bool
Contract_Two Year                      bool
Payment Method_Bank Withdrawal         bool
Payment Method_Credit Card             bool
Payment Method_Mailed Check            bool
dtype: object
```

In [40]:
```
# Select boolean columns
boolean_columns = Churn.select_dtypes(include='bool').columns
# Convert boolean columns to integers (0 and 1)
Churn[boolean_columns] = Churn[boolean_columns].astype(int)
```

In [41]:
```
# Checking head of dataset after Encoding
Churn.head(n=10)
```

Out[41]:

	Gender	Age	Married	Number of Dependents	Number of Referrals	Tenure in Months	Phone Service	Avg Monthly Long Distance Charges	Multiple Lines	Internet Service
0	0	37	1	0	2	9	1	42.39	0	1
1	1	46	0	0	0	9	1	10.69	1	1
2	1	50	0	0	0	4	1	33.65	0	1
3	1	78	1	0	1	13	1	27.82	0	1
4	0	75	1	0	3	3	1	7.38	0	1
5	0	23	0	3	0	9	1	16.77	0	1
6	0	67	1	0	1	71	1	9.96	0	1
7	1	52	1	0	8	63	1	12.96	1	1
8	0	68	0	0	0	7	1	10.53	0	1
9	0	43	1	1	3	65	1	28.46	1	1

10 rows × 36 columns

Normlisation

In [42]:
```
# Select the features to be normalized
features_to_normalize = ['Avg Monthly Long Distance Charges', 'Tenure in Months', '

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Apply Min-Max scaling to the selected features
Churn[features_to_normalize] = scaler.fit_transform(Churn[features_to_normalize])
```

In [43]:
```
# Checking head of dataset after normalisation
Churn.head(n=10)
```

Out[43]:

	Gender	Age	Married	Number of Dependents	Number of Referrals	Tenure in Months	Phone Service	Avg Monthly Long Distance Charges	Multiple Lines	Interr Serv
0	0	0.295082	1	0.000000	0.181818	0.112676	1	0.844835	0	
1	1	0.442623	0	0.000000	0.000000	0.112676	1	0.197632	1	
2	1	0.508197	0	0.000000	0.000000	0.042254	1	0.666394	0	
3	1	0.967213	1	0.000000	0.090909	0.169014	1	0.547366	0	
4	0	0.918033	1	0.000000	0.272727	0.028169	1	0.130053	0	
5	0	0.065574	0	0.333333	0.000000	0.112676	1	0.321764	0	
6	0	0.786885	1	0.000000	0.090909	0.985915	1	0.182728	0	
7	1	0.540984	1	0.000000	0.727273	0.873239	1	0.243977	1	
8	0	0.803279	0	0.000000	0.000000	0.084507	1	0.194365	0	
9	0	0.393443	1	0.111111	0.272727	0.901408	1	0.560433	1	

10 rows × 36 columns

Correlation Analysis for Multi-Collinearity Assessment

In [44]:
```
# Correlation Analysis
# Selecting only numerical features (excluding the target variable)
numerical_features = Churn.select_dtypes(include='number').drop(columns=['Customer
# Compute correlation matrix
correlation_matrix = numerical_features.corr()
# Display correlation matrix
print(correlation_matrix)
```

	Gender	Age	Married
Gender	1.000000	0.007664	0.004971
Age	0.007664	1.000000	-0.014409
Married	0.004971	-0.014409	1.000000
Number of Dependents	0.006321	-0.125500	0.322628
Number of Referrals	-0.004353	-0.032019	0.663598
Tenure in Months	0.011547	-0.009574	0.359737
Phone Service	-0.010518	0.007853	0.023813
Avg Monthly Long Distance Charges	0.015720	-0.015403	-0.004581
Multiple Lines	-0.004748	0.098179	0.123751
Internet Service	-0.002584	0.110758	-0.021248
Avg Monthly GB Download	-0.014879	-0.505260	0.066664
Online Security	-0.017059	-0.046073	0.132193
Online Backup	-0.009362	0.028884	0.125829
Device Protection Plan	-0.001482	0.020439	0.136040
Premium Tech Support	-0.012419	-0.052058	0.109226
Streaming TV	-0.001634	-0.037936	0.122905
Streaming Movies	-0.009462	-0.024728	0.117148
Streaming Music	-0.008470	-0.170316	0.071881
Unlimited Data	0.004778	-0.025313	0.029683
Paperless Billing	-0.010891	0.100351	-0.032990
Monthly Charge	-0.017288	0.124422	0.070009
Total Charges	0.004251	0.047788	0.296830
Total Refunds	0.008031	0.020570	0.031884
Total Extra Data Charges	-0.000064	0.021759	0.012018
Total Long Distance Charges	0.015564	-0.009976	0.243303
Total Revenue	0.008006	0.035080	0.308360
Internet Type_Cable	-0.008537	-0.041248	0.006825
Internet Type_DSL	0.014655	-0.056854	0.003602
Internet Type_Fiber Optic	-0.007227	0.078216	-0.007797
Contract_Month-to-Month	-0.005772	-0.001283	-0.252296
Contract_One Year	0.008596	-0.010480	0.069478
Contract_Two Year	-0.001647	0.011245	0.214997
Payment Method_Bank Withdrawal	-0.005451	0.114948	-0.003856
Payment Method_Credit Card	0.007691	-0.101897	0.033121
Payment Method_Mailed Check	-0.004661	-0.033586	-0.063845

	Number of Dependents	Number of Referrals
Gender	0.006321	-0.004353
Age	-0.125500	-0.032019
Married	0.322628	0.663598
Number of Dependents	1.000000	0.276880
Number of Referrals	0.276880	1.000000
Tenure in Months	0.107311	0.323826
Phone Service	-0.008652	0.012590
Avg Monthly Long Distance Charges	-0.005649	-0.003455
Multiple Lines	-0.035975	0.067451
Internet Service	-0.164563	-0.051425
Avg Monthly GB Download	0.229396	0.064592
Online Security	0.036838	0.137056
Online Backup	-0.005134	0.104668
Device Protection Plan	-0.024079	0.108669
Premium Tech Support	0.019176	0.104969
Streaming TV	0.076270	0.110200
Streaming Movies	0.061325	0.089334
Streaming Music	-0.039533	0.042108
Unlimited Data	0.029913	0.042395
Paperless Billing	-0.112755	-0.056960
Monthly Charge	-0.136297	0.011015
Total Charges	0.016118	0.241943
Total Refunds	0.012211	0.020372
Total Extra Data Charges	-0.015797	-0.002547
Total Long Distance Charges	0.065869	0.207028
Total Revenue	0.032315	0.253851

Internet Type_Cable	0.006085	0.009476
Internet Type_DSL	0.010465	0.027511
Internet Type_Fiber Optic	-0.013380	-0.030775
Contract_Month-to-Month	-0.128389	-0.272251
Contract_One Year	0.004264	0.066514
Contract_Two Year	0.138544	0.239930
Payment Method_Bank Withdrawal	-0.089444	-0.053171
Payment Method_Credit Card	0.099719	0.077497
Payment Method_Mailed Check	-0.018555	-0.050879

	Tenure in Months	Phone Service	\
Gender	0.011547	-0.010518	
Age	-0.009574	0.007853	
Married	0.359737	0.023813	
Number of Dependents	0.107311	-0.008652	
Number of Referrals	0.323826	0.012590	
Tenure in Months	1.000000	0.013282	
Phone Service	0.013282	1.000000	
Avg Monthly Long Distance Charges	0.006230	-0.004927	
Multiple Lines	0.299944	0.294161	
Internet Service	-0.000288	-0.166607	
Avg Monthly GB Download	0.034121	-0.032934	
Online Security	0.315726	-0.097259	
Online Backup	0.346104	-0.048827	
Device Protection Plan	0.336471	-0.073776	
Premium Tech Support	0.312314	-0.093205	
Streaming TV	0.251734	0.113186	
Streaming Movies	0.258661	0.103364	
Streaming Music	0.211155	-0.040135	
Unlimited Data	-0.003243	0.041242	
Paperless Billing	-0.024360	0.012967	
Monthly Charge	0.199019	0.243944	
Total Charges	0.812795	0.123390	
Total Refunds	0.042729	0.006675	
Total Extra Data Charges	0.071224	-0.029174	
Total Long Distance Charges	0.653170	0.307688	
Total Revenue	0.840935	0.189668	
Internet Type_Cable	-0.005228	-0.224363	
Internet Type_DSL	0.016244	-0.334472	
Internet Type_Fiber Optic	-0.010868	0.447803	
Contract_Month-to-Month	-0.614856	-0.002087	
Contract_One Year	0.168878	0.000181	
Contract_Two Year	0.524372	0.002147	
Payment Method_Bank Withdrawal	-0.025183	0.009661	
Payment Method_Credit Card	0.084859	0.004319	
Payment Method_Mailed Check	-0.129430	-0.031003	

	Avg Monthly Long Distance Charges	\
Gender	0.015720	
Age	-0.015403	
Married	-0.004581	
Number of Dependents	-0.005649	
Number of Referrals	-0.003455	
Tenure in Months	0.006230	
Phone Service	-0.004927	
Avg Monthly Long Distance Charges	1.000000	
Multiple Lines	-0.005364	
Internet Service	0.011725	
Avg Monthly GB Download	0.013169	
Online Security	0.014592	
Online Backup	0.010468	
Device Protection Plan	-0.001754	
Premium Tech Support	-0.005476	
Streaming TV	0.000106	

Streaming Movies	0.009270
Streaming Music	0.028029
Unlimited Data	-0.012503
Paperless Billing	0.015637
Monthly Charge	0.012426
Total Charges	0.011565
Total Refunds	-0.024660
Total Extra Data Charges	0.019051
Total Long Distance Charges	0.548711
Total Revenue	0.173532
Internet Type_Cable	0.014270
Internet Type_DSL	-0.008801
Internet Type_Fiber Optic	-0.001829
Contract_Month-to-Month	-0.002238
Contract_One Year	0.030331
Contract_Two Year	-0.025937
Payment Method_Bank Withdrawal	0.013207
Payment Method_Credit Card	-0.010261
Payment Method_Mailed Check	-0.007024

	Multiple Lines	Internet Service	...	\
Gender	-0.004748	-0.002584	...	
Age	0.098179	0.110758	...	
Married	0.123751	-0.021248	...	
Number of Dependents	-0.035975	-0.164563	...	
Number of Referrals	0.067451	-0.051425	...	
Tenure in Months	0.299944	-0.000288	...	
Phone Service	0.294161	-0.166607	...	
Avg Monthly Long Distance Charges	-0.005364	0.011725	...	
Multiple Lines	1.000000	0.199997	...	
Internet Service	0.199997	1.000000	...	
Avg Monthly GB Download	-0.001234	0.119774	...	
Online Security	0.083926	0.330946	...	
Online Backup	0.187246	0.380024	...	
Device Protection Plan	0.184029	0.381903	...	
Premium Tech Support	0.087188	0.333822	...	
Streaming TV	0.076823	-0.406993	...	
Streaming Movies	0.081905	-0.403762	...	
Streaming Music	0.180489	0.387690	...	
Unlimited Data	-0.037594	-0.177853	...	
Paperless Billing	0.154991	0.318405	...	
Monthly Charge	0.458214	0.721391	...	
Total Charges	0.451265	0.376125	...	
Total Refunds	0.036225	0.012650	...	
Total Extra Data Charges	0.057678	0.140662	...	
Total Long Distance Charges	0.300490	-0.042382	...	
Total Revenue	0.448428	0.287064	...	
Internet Type_Cable	-0.101236	0.184681	...	
Internet Type_DSL	-0.149177	0.279211	...	
Internet Type_Fiber Optic	0.200512	-0.372054	...	
Contract_Month-to-Month	-0.070235	0.223913	...	
Contract_One Year	-0.020749	-0.035474	...	
Contract_Two Year	0.097419	-0.215349	...	
Payment Method_Bank Withdrawal	0.132185	0.259444	...	
Payment Method_Credit Card	-0.093775	-0.225085	...	
Payment Method_Mailed Check	-0.089814	-0.086532	...	

	Total Revenue	Internet Type_Cable	\
Gender	0.008006	-0.008537	
Age	0.035080	-0.041248	
Married	0.308360	0.006825	
Number of Dependents	0.032315	0.006085	
Number of Referrals	0.253851	0.009476	
Tenure in Months	0.840935	-0.005228	

Phone Service	0.189668	-0.224363
Avg Monthly Long Distance Charges	0.173532	0.014270
Multiple Lines	0.448428	-0.101236
Internet Service	0.287064	0.184681
Avg Monthly GB Download	0.063804	0.057459
Online Security	0.377673	0.148439
Online Backup	0.468395	0.076588
Device Protection Plan	0.461121	0.069854
Premium Tech Support	0.389047	0.151191
Streaming TV	0.213616	-0.129448
Streaming Movies	0.220091	-0.134447
Streaming Music	0.380614	0.041838
Unlimited Data	-0.057592	-0.034594
Paperless Billing	0.116670	-0.015241
Monthly Charge	0.554815	-0.058105
Total Charges	0.970266	-0.032446
Total Refunds	0.022574	-0.012800
Total Extra Data Charges	0.114636	0.011158
Total Long Distance Charges	0.765651	-0.070568
Total Revenue	1.000000	-0.046718
Internet Type_Cable	-0.046718	1.000000
Internet Type_DSL	-0.054856	-0.201234
Internet Type_Fiber Optic	0.080137	-0.496382
Contract_Month-to-Month	-0.451532	-0.006728
Contract_One Year	0.155290	0.010886
Contract_Two Year	0.355781	-0.002732
Payment Method_Bank Withdrawal	0.062803	-0.035113
Payment Method_Credit Card	-0.001403	0.012819
Payment Method_Mailed Check	-0.137061	0.050307

	Internet Type_DSL \
Gender	0.014655
Age	-0.056854
Married	0.003602
Number of Dependents	0.010465
Number of Referrals	0.027511
Tenure in Months	0.016244
Phone Service	-0.334472
Avg Monthly Long Distance Charges	-0.008801
Multiple Lines	-0.149177
Internet Service	0.279211
Avg Monthly GB Download	0.074321
Online Security	0.249760
Online Backup	0.121076
Device Protection Plan	0.120539
Premium Tech Support	0.236722
Streaming TV	-0.219388
Streaming Movies	-0.208604
Streaming Music	0.060038
Unlimited Data	-0.057505
Paperless Billing	-0.058684
Monthly Charge	-0.129892
Total Charges	-0.036622
Total Refunds	0.006062
Total Extra Data Charges	0.057450
Total Long Distance Charges	-0.087915
Total Revenue	-0.054856
Internet Type_Cable	-0.201234
Internet Type_DSL	1.000000
Internet Type_Fiber Optic	-0.750457
Contract_Month-to-Month	-0.047333
Contract_One Year	0.040011
Contract_Two Year	0.015057
Payment Method_Bank Withdrawal	-0.056924

Payment Method_Credit Card	0.053642
Payment Method_Mailed Check	0.009676

	Internet Type_Fiber Optic
Gender	-0.007227
Age	0.078216
Married	-0.007797
Number of Dependents	-0.013380
Number of Referrals	-0.030775
Tenure in Months	-0.010868
Phone Service	0.447803
Avg Monthly Long Distance Charges	-0.001829
Multiple Lines	0.200512
Internet Service	-0.372054
Avg Monthly GB Download	-0.104635
Online Security	-0.321500
Online Backup	-0.158977
Device Protection Plan	-0.153957
Premium Tech Support	-0.311804
Streaming TV	0.281770
Streaming Movies	0.275586
Streaming Music	-0.081437
Unlimited Data	0.074305
Paperless Billing	0.062292
Monthly Charge	0.154319
Total Charges	0.054347
Total Refunds	0.003264
Total Extra Data Charges	-0.058443
Total Long Distance Charges	0.125527
Total Revenue	0.080137
Internet Type_Cable	-0.496382
Internet Type_DSL	-0.750457
Internet Type_Fiber Optic	1.000000
Contract_Month-to-Month	0.046488
Contract_One Year	-0.042804
Contract_Two Year	-0.011501
Payment Method_Bank Withdrawal	0.074140
Payment Method_Credit Card	-0.056188
Payment Method_Mailed Check	-0.042518

	Contract_Month-to-Month	Contract_One Year
Gender	-0.005772	0.008596
Age	-0.001283	-0.010480
Married	-0.252296	0.069478
Number of Dependents	-0.128389	0.004264
Number of Referrals	-0.272251	0.066514
Tenure in Months	-0.614856	0.168878
Phone Service	-0.002087	0.000181
Avg Monthly Long Distance Charges	-0.002238	0.030331
Multiple Lines	-0.070235	-0.020749
Internet Service	0.223913	-0.035474
Avg Monthly GB Download	0.000653	0.019519
Online Security	-0.225947	0.091162
Online Backup	-0.154417	0.074896
Device Protection Plan	-0.203021	0.082671
Premium Tech Support	-0.256663	0.078670
Streaming TV	-0.276596	0.079672
Streaming Movies	-0.286515	0.082180
Streaming Music	-0.067183	0.032663
Unlimited Data	-0.050649	0.000779
Paperless Billing	0.171170	-0.051747
Monthly Charge	0.067649	-0.003094
Total Charges	-0.418397	0.145967
Total Refunds	-0.037443	0.004841

Total Extra Data Charges	-0.003023	0.021882
Total Long Distance Charges	-0.399747	0.131272
Total Revenue	-0.451532	0.155290
Internet Type_Cable	-0.006728	0.010886
Internet Type_DSL	-0.047333	0.040011
Internet Type_Fiber Optic	0.046488	-0.042804
Contract_Month-to-Month	1.000000	-0.533797
Contract_One Year	-0.533797	1.000000
Contract_Two Year	-0.610011	-0.344435
Payment Method_Bank Withdrawal	0.163780	-0.053275
Payment Method_Credit Card	-0.186632	0.054667
Payment Method_Mailed Check	0.042806	-0.000710

	Contract_Two Year \
Gender	-0.001647
Age	0.011245
Married	0.214997
Number of Dependents	0.138544
Number of Referrals	0.239930
Tenure in Months	0.524372
Phone Service	0.002147
Avg Monthly Long Distance Charges	-0.025937
Multiple Lines	0.097419
Internet Service	-0.215349
Avg Monthly GB Download	-0.019016
Online Security	0.165425
Online Backup	0.101252
Device Protection Plan	0.147928
Premium Tech Support	0.211232
Streaming TV	0.232423
Streaming Movies	0.241085
Streaming Music	0.043980
Unlimited Data	0.055501
Paperless Billing	-0.141546
Monthly Charge	-0.072206
Total Charges	0.327729
Total Refunds	0.037034
Total Extra Data Charges	-0.017149
Total Long Distance Charges	0.320794
Total Revenue	0.355781
Internet Type_Cable	-0.002732
Internet Type_DSL	0.015057
Internet Type_Fiber Optic	-0.011501
Contract_Month-to-Month	-0.610011
Contract_One Year	-0.344435
Contract_Two Year	1.000000
Payment Method_Bank Withdrawal	-0.131909
Payment Method_Credit Card	0.155975
Payment Method_Mailed Check	-0.046859

	Payment Method_Bank Withdrawal \
Gender	-0.005451
Age	0.114948
Married	-0.003856
Number of Dependents	-0.089444
Number of Referrals	-0.053171
Tenure in Months	-0.025183
Phone Service	0.009661
Avg Monthly Long Distance Charges	0.013207
Multiple Lines	0.132185
Internet Service	0.259444
Avg Monthly GB Download	-0.005962
Online Security	-0.039649
Online Backup	0.059466

Device Protection Plan	0.055254
Premium Tech Support	-0.035901
Streaming TV	-0.049095
Streaming Movies	-0.050527
Streaming Music	0.121618
Unlimited Data	-0.058125
Paperless Billing	0.179961
Monthly Charge	0.268894
Total Charges	0.082373
Total Refunds	0.026451
Total Extra Data Charges	0.046901
Total Long Distance Charges	-0.009761
Total Revenue	0.062803
Internet Type_Cable	-0.035113
Internet Type_DSL	-0.056924
Internet Type_Fiber Optic	0.074140
Contract_Month-to-Month	0.163780
Contract_One Year	-0.053275
Contract_Two Year	-0.131909
Payment Method_Bank Withdrawal	1.000000
Payment Method_Credit Card	-0.897752
Payment Method_Mailed Check	-0.267501

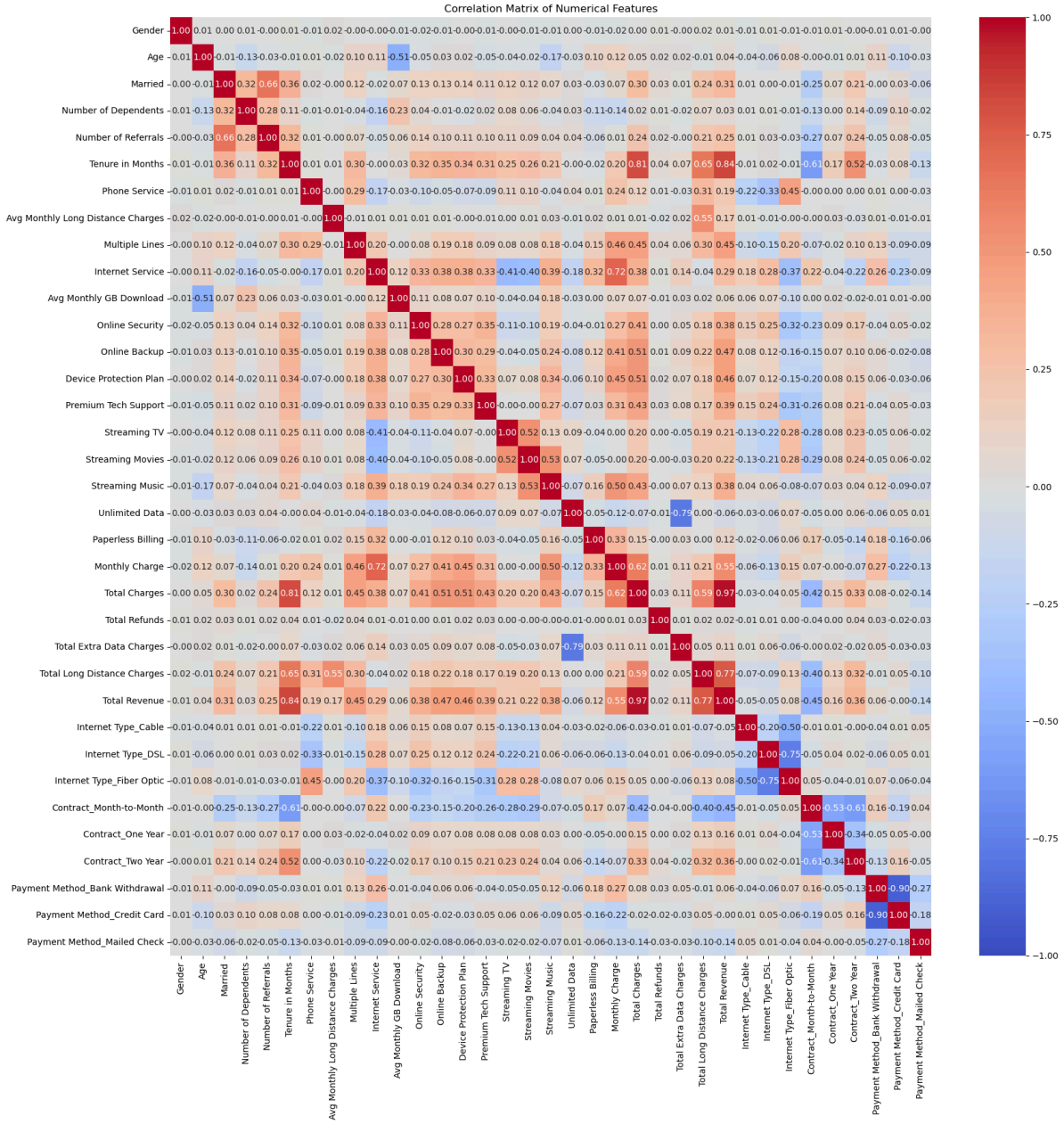
	Payment Method_Credit Card \
Gender	0.007691
Age	-0.101897
Married	0.033121
Number of Dependents	0.099719
Number of Referrals	0.077497
Tenure in Months	0.084859
Phone Service	0.004319
Avg Monthly Long Distance Charges	-0.010261
Multiple Lines	-0.093775
Internet Service	-0.225085
Avg Monthly GB Download	0.007092
Online Security	0.051155
Online Backup	-0.023816
Device Protection Plan	-0.028157
Premium Tech Support	0.052222
Streaming TV	0.057325
Streaming Movies	0.062196
Streaming Music	-0.092474
Unlimited Data	0.052610
Paperless Billing	-0.156198
Monthly Charge	-0.215994
Total Charges	-0.021773
Total Refunds	-0.015421
Total Extra Data Charges	-0.033446
Total Long Distance Charges	0.053947
Total Revenue	-0.001403
Internet Type_Cable	0.012819
Internet Type_DSL	0.053642
Internet Type_Fiber Optic	-0.056188
Contract_Month-to-Month	-0.186632
Contract_One Year	0.054667
Contract_Two Year	0.155975
Payment Method_Bank Withdrawal	-0.897752
Payment Method_Credit Card	1.000000
Payment Method_Mailed Check	-0.184299

	Payment Method_Mailed Check
Gender	-0.004661
Age	-0.033586
Married	-0.063845

```
Number of Dependents                          -0.018555
Number of Referrals                           -0.050879
Tenure in Months                              -0.129430
Phone Service                                 -0.031003
Avg Monthly Long Distance Charges             -0.007024
Multiple Lines                                -0.089814
Internet Service                              -0.086532
Avg Monthly GB Download                       -0.002210
Online Security                               -0.023430
Online Backup                                 -0.080588
Device Protection Plan                        -0.061693
Premium Tech Support                          -0.034127
Streaming TV                                  -0.015851
Streaming Movies                              -0.023309
Streaming Music                               -0.069081
Unlimited Data                                 0.014612
Paperless Billing                             -0.059869
Monthly Charge                                -0.127502
Total Charges                                 -0.136169
Total Refunds                                 -0.025287
Total Extra Data Charges                      -0.031488
Total Long Distance Charges                   -0.096225
Total Revenue                                 -0.137061
Internet Type_Cable                            0.050307
Internet Type_DSL                              0.009676
Internet Type_Fiber Optic                     -0.042518
Contract_Month-to-Month                        0.042806
Contract_One Year                             -0.000710
Contract_Two Year                             -0.046859
Payment Method_Bank Withdrawal                -0.267501
Payment Method_Credit Card                    -0.184299
Payment Method_Mailed Check                    1.000000

[35 rows x 35 columns]
```

In [45]:

```
# Correlation Plot
plt.figure(figsize=(18, 18))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", vmin=-1, vm
plt.xticks(rotation=90)
plt.yticks(rotation=0)
# Add title and adjust layout
plt.title('Correlation Matrix of Numerical Features')
plt.tight_layout()
# Show plot
plt.savefig('Correlation Matrix of Numerical Features.jpg')
plt.show()
```



In [46]:
```
# Checking for highly Correlated Features
# Setting a threshold for high correlation
threshold = 0.7
# Initializing a list to store highly correlated features pairs
highly_correlated_pairs = []
# Iterating through the correlation matrix
for i in range(len(correlation_matrix.columns)):
    for j in range(i+1, len(correlation_matrix.columns)):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            # Store the names of highly correlated features
            pair = (correlation_matrix.columns[i], correlation_matrix.columns[j])
            highly_correlated_pairs.append(pair)
if not highly_correlated_pairs:
    print("No highly correlated features found.")
else:
    print("Highly correlated feature pairs:")
    for pair in highly_correlated_pairs:
        print(pair)
```

```
Highly correlated feature pairs:
('Tenure in Months', 'Total Charges')
('Tenure in Months', 'Total Revenue')
('Internet Service', 'Monthly Charge')
('Unlimited Data', 'Total Extra Data Charges')
('Total Charges', 'Total Revenue')
('Total Long Distance Charges', 'Total Revenue')
('Internet Type_DSL', 'Internet Type_Fiber Optic')
('Payment Method_Bank Withdrawal', 'Payment Method_Credit Card')
```

In [47]:
```
# Removal of highly correlated features
Churn.drop(['Tenure in Months', 'Total Charges','Total Revenue', 'Internet Service'
            'Total Extra Data Charges', 'Total Long Distance Charges', 'Internet Ty
```

In [48]:
```
#Checking Columns in dataset after feature removals
print(Churn.columns)
```

```
Index(['Gender', 'Age', 'Married', 'Number of Dependents',
       'Number of Referrals', 'Phone Service',
       'Avg Monthly Long Distance Charges', 'Multiple Lines',
       'Avg Monthly GB Download', 'Online Security', 'Online Backup',
       'Device Protection Plan', 'Premium Tech Support', 'Streaming TV',
       'Streaming Movies', 'Streaming Music', 'Paperless Billing',
       'Total Refunds', 'Customer Status', 'Internet Type_Cable',
       'Contract_Month-to-Month', 'Contract_One Year', 'Contract_Two Year',
       'Payment Method_Mailed Check'],
      dtype='object')
```

Variance Inflation Factor (VIF) for Numerical Features

In [49]:
```
# Selecting only numerical columns
numerical_columns = Churn.select_dtypes(include=['int64', 'float64']).columns

# Calculate VIF
vif_data = pd.DataFrame()
vif_data["Feature"] = numerical_columns
vif_data["VIF"] = [variance_inflation_factor(Churn[numerical_columns].values, i) fo

print("Variance Inflation Factors:")
print(vif_data)
```

```
Variance Inflation Factors:
                            Feature       VIF
0                               Age  2.338980
1             Number of Dependents  1.413638
2              Number of Referrals  1.535375
3  Avg Monthly Long Distance Charges  3.295581
4           Avg Monthly GB Download  2.371388
5                     Total Refunds  1.062786
```

The Variance Inflation Factor showed that all the numerical features have a variance of less than 10, so none of them would be removed.

Chi Square Test for Categorical Features

In [50]:
```
# Iterate over each feature (excluding the target variable) in the dataset
for feature in Churn.columns[:-1]:
```

```
    # Check if the feature is integer-encoded (assuming categorical)
    if Churn[feature].dtype == 'int32':
        # Create a contingency table for the feature and the target variable
        contingency_table = pd.crosstab(Churn[feature], Churn['Customer Status'])

        # Perform the chi-square test
        chi2_statistic, p_value, _, _ = chi2_contingency(contingency_table)

        # Print the results
        print(f"Chi-square test results for {feature}:")
        print(f"   Chi-square statistic: {chi2_statistic}")
        print(f"   P-value: {p_value}")

        # Check for significance (e.g., p-value < 0.05)
        if p_value < 0.05:
            print(f"   {feature} is statistically significant for predicting the ta
        else:
            print(f"   {feature} is not statistically significant for predicting th

        print()
```

```
Chi-square test results for Gender:
   Chi-square statistic: 0.24007105217966312
   P-value: 0.6241548018547587
   Gender is not statistically significant for predicting the target variable.

Chi-square test results for Married:
   Chi-square statistic: 220.5060439643492
   P-value: 7.014602453362894e-50
   Married is statistically significant for predicting the target variable.

Chi-square test results for Phone Service:
   Chi-square statistic: 1.2551734871989866
   P-value: 0.26256666476273127
   Phone Service is not statistically significant for predicting the target variab
le.

Chi-square test results for Multiple Lines:
   Chi-square statistic: 1.1535971852605198
   P-value: 0.2827973969037025
   Multiple Lines is not statistically significant for predicting the target varia
ble.

Chi-square test results for Online Security:
   Chi-square statistic: 248.43749367640638
   P-value: 5.689634112881904e-56
   Online Security is statistically significant for predicting the target variabl
e.

Chi-square test results for Online Backup:
   Chi-square statistic: 73.07537713497942
   P-value: 1.247916969909602e-17
   Online Backup is statistically significant for predicting the target variable.

Chi-square test results for Device Protection Plan:
   Chi-square statistic: 56.671577391479005
   P-value: 5.1502608160860495e-14
   Device Protection Plan is statistically significant for predicting the target v
ariable.

Chi-square test results for Premium Tech Support:
   Chi-square statistic: 231.66761732586693
   P-value: 2.580290732051722e-52
   Premium Tech Support is statistically significant for predicting the target var
iable.

Chi-square test results for Streaming TV:
   Chi-square statistic: 135.09725737576034
   P-value: 3.1438574036725362e-31
   Streaming TV is statistically significant for predicting the target variable.

Chi-square test results for Streaming Movies:
   Chi-square statistic: 139.63452425587423
   P-value: 3.199893781319627e-32
   Streaming Movies is statistically significant for predicting the target variabl
e.

Chi-square test results for Streaming Music:
   Chi-square statistic: 4.274339099232788
   P-value: 0.03869200784160805
   Streaming Music is statistically significant for predicting the target variabl
e.

Chi-square test results for Paperless Billing:
   Chi-square statistic: 231.29426752260608
```

```
   P-value: 3.112345421939546e-52
   Paperless Billing is statistically significant for predicting the target variab
le.

Chi-square test results for Customer Status:
   Chi-square statistic: 6584.07952972205
   P-value: 0.0
   Customer Status is statistically significant for predicting the target variabl
e.

Chi-square test results for Internet Type_Cable:
   Chi-square statistic: 0.26359201971038076
   P-value: 0.6076630592581913
   Internet Type_Cable is not statistically significant for predicting the target
variable.

Chi-square test results for Contract_Month-to-Month:
   Chi-square statistic: 1665.0430834588672
   P-value: 0.0
   Contract_Month-to-Month is statistically significant for predicting the target
variable.

Chi-square test results for Contract_One Year:
   Chi-square statistic: 297.7642870461098
   P-value: 1.0112624954818955e-66
   Contract_One Year is statistically significant for predicting the target variab
le.

Chi-square test results for Contract_Two Year:
   Chi-square statistic: 846.9257076455871
   P-value: 3.3878596704718002e-186
   Contract_Two Year is statistically significant for predicting the target variab
le.
```

In [51]:
```
# Removal of non statistically significant features
Churn.drop(["Gender", "Phone Service", "Multiple Lines", "Internet Type_Cable"], ax
```

In [52]:
```
# Checking Shape of dataset after removal of irrelevant features
Churn.shape
```

Out[52]:
```
(6589, 20)
```

Data Splitting for Machine Learning

Seperating Features and Target Variables

In [53]:
```
#Splitting the dataset into features (X) and target variable (Y)
X = Churn.drop(columns=['Customer Status'], axis=1)
Y = Churn["Customer Status"]
```

In [54]:
```
# Shuffle your data
X,Y = shuffle(X, Y, random_state=42)
```

In [55]:
```
X.head()
```

Out[55]:

	Age	Married	Number of Dependents	Number of Referrals	Avg Monthly Long Distance Charges	Avg Monthly GB Download	Online Security	Online Backup	Devic Protectio Pla
2059	0.606557	0	0.000000	0.000000	0.111474	21.0	0	0	
4323	0.491803	1	0.000000	0.909091	0.903838	21.0	0	0	
1544	0.163934	1	0.222222	0.909091	0.504492	69.0	1	1	
2644	0.770492	1	0.000000	0.545455	0.634341	8.0	1	0	
4029	0.655738	1	0.111111	1.000000	0.841364	25.0	1	0	

In [56]:
```
Y.head()
```

Out[56]:
```
2059    0
4323    1
1544    1
2644    1
4029    1
Name: Customer Status, dtype: int32
```

Backward Stepwise Regression

In [57]:
```
import statsmodels.api as sm
logit_model=sm.Logit(Y,X)
result=logit_model.fit()
print(result.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.365385
         Iterations 8
                                Results: Logit
================================================================================
Model:                     Logit                  Method:                 MLE
Dependent Variable:        Customer Status        Pseudo R-squared:       0.387
Date:                      2024-05-05 21:37       AIC:                    4853.0382
No. Observations:          6589                   BIC:                    4982.1082
Df Model:                  18                     Log-Likelihood:         -2407.5
Df Residuals:              6570                   LL-Null:                -3929.5
Converged:                 1.0000                 LLR p-value:            0.0000
No. Iterations:            8.0000                 Scale:                  1.0000
--------------------------------------------------------------------------------
                                  Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
--------------------------------------------------------------------------------
Age                              -1.4799   0.1598  -9.2624 0.0000 -1.7931 -1.1668
Married                          -1.4031   0.1056 -13.2850 0.0000 -1.6101 -1.1961
Number of Dependents              6.2976   0.5564  11.3175 0.0000  5.2070  7.3882
Number of Referrals               5.9307   0.3722  15.9355 0.0000  5.2013  6.6602
Avg Monthly Long Distance Charges 0.0518   0.1300   0.3983 0.6904 -0.2031  0.3067
Avg Monthly GB Download          -0.0047   0.0025  -1.9219 0.0546 -0.0095  0.0001
Online Security                   0.5956   0.0909   6.5535 0.0000  0.4174  0.7737
Online Backup                     0.2666   0.0814   3.2754 0.0011  0.1071  0.4261
Device Protection Plan           -0.0436   0.0838  -0.5205 0.6027 -0.2079  0.1206
Premium Tech Support              0.4536   0.0911   4.9818 0.0000  0.2752  0.6321
Streaming TV                     -0.1720   0.0849  -2.0261 0.0428 -0.3383 -0.0056
Streaming Movies                  0.7710   0.1100   7.0106 0.0000  0.5554  0.9865
Streaming Music                  -1.0547   0.1081  -9.7557 0.0000 -1.2666 -0.8428
Paperless Billing                -0.4579   0.0792  -5.7805 0.0000 -0.6131 -0.3026
Total Refunds                     0.0095   0.0048   1.9988 0.0456  0.0002  0.0189
Contract_Month-to-Month           0.6522   0.1538   4.2414 0.0000  0.3508  0.9536
Contract_One Year                 2.6115   0.1885  13.8548 0.0000  2.2420  2.9809
Contract_Two Year                 3.8988   0.2299  16.9601 0.0000  3.4482  4.3493
Payment Method_Mailed Check      -0.7775   0.1495  -5.2020 0.0000 -1.0705 -0.4846
================================================================================
```

In [58]:
```
#Removing features with P values higher than 0.05
# List of columns to drop
columns_to_drop = ['Avg Monthly Long Distance Charges','Avg Monthly GB Download', '
# Drop the specified columns from the DataFrame X
X = X.drop(columns_to_drop, axis=1)
```

In [59]:
```
logit_model=sm.Logit(Y,X)
result2=logit_model.fit()
print(result2.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.365699
         Iterations 8
                            Results: Logit
================================================================================
Model:                  Logit               Method:              MLE
Dependent Variable:     Customer Status     Pseudo R-squared:    0.387
Date:                   2024-05-05 21:37    AIC:                 4851.1792
No. Observations:       6589                BIC:                 4959.8697
Df Model:               15                  Log-Likelihood:      -2409.6
Df Residuals:           6573                LL-Null:             -3929.5
Converged:              1.0000              LLR p-value:         0.0000
No. Iterations:         8.0000              Scale:               1.0000
--------------------------------------------------------------------------------
                             Coef.  Std.Err.    z     P>|z|   [0.025  0.975]
--------------------------------------------------------------------------------
Age                         -1.3355   0.1384  -9.6518 0.0000 -1.6067 -1.0643
Married                     -1.4060   0.1055 -13.3273 0.0000 -1.6128 -1.1992
Number of Dependents         6.1496   0.5511  11.1589 0.0000  5.0695  7.2298
Number of Referrals          5.9200   0.3718  15.9218 0.0000  5.1913  6.6488
Online Security              0.5932   0.0908   6.5345 0.0000  0.4153  0.7711
Online Backup                0.2601   0.0810   3.2106 0.0013  0.1013  0.4189
Premium Tech Support         0.4500   0.0905   4.9719 0.0000  0.2726  0.6274
Streaming TV                -0.1733   0.0846  -2.0491 0.0405 -0.3390 -0.0075
Streaming Movies             0.7908   0.1091   7.2481 0.0000  0.5769  1.0046
Streaming Music             -1.0849   0.1056 -10.2775 0.0000 -1.2918 -0.8780
Paperless Billing           -0.4611   0.0791  -5.8289 0.0000 -0.6162 -0.3061
Total Refunds                0.0095   0.0048   1.9952 0.0460  0.0002  0.0189
Contract_Month-to-Month      0.4972   0.1044   4.7645 0.0000  0.2927  0.7018
Contract_One Year            2.4482   0.1481  16.5328 0.0000  2.1580  2.7384
Contract_Two Year            3.7308   0.1980  18.8387 0.0000  3.3426  4.1189
Payment Method_Mailed Check -0.7758   0.1493  -5.1955 0.0000 -1.0685 -0.4832
================================================================================
```

Data Splitting

In [60]:
```
# Splitting the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_sta

# Checking the shape of the resulting sets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of Y_train:", Y_train.shape)
print("Shape of Y_test:", Y_test.shape)
```

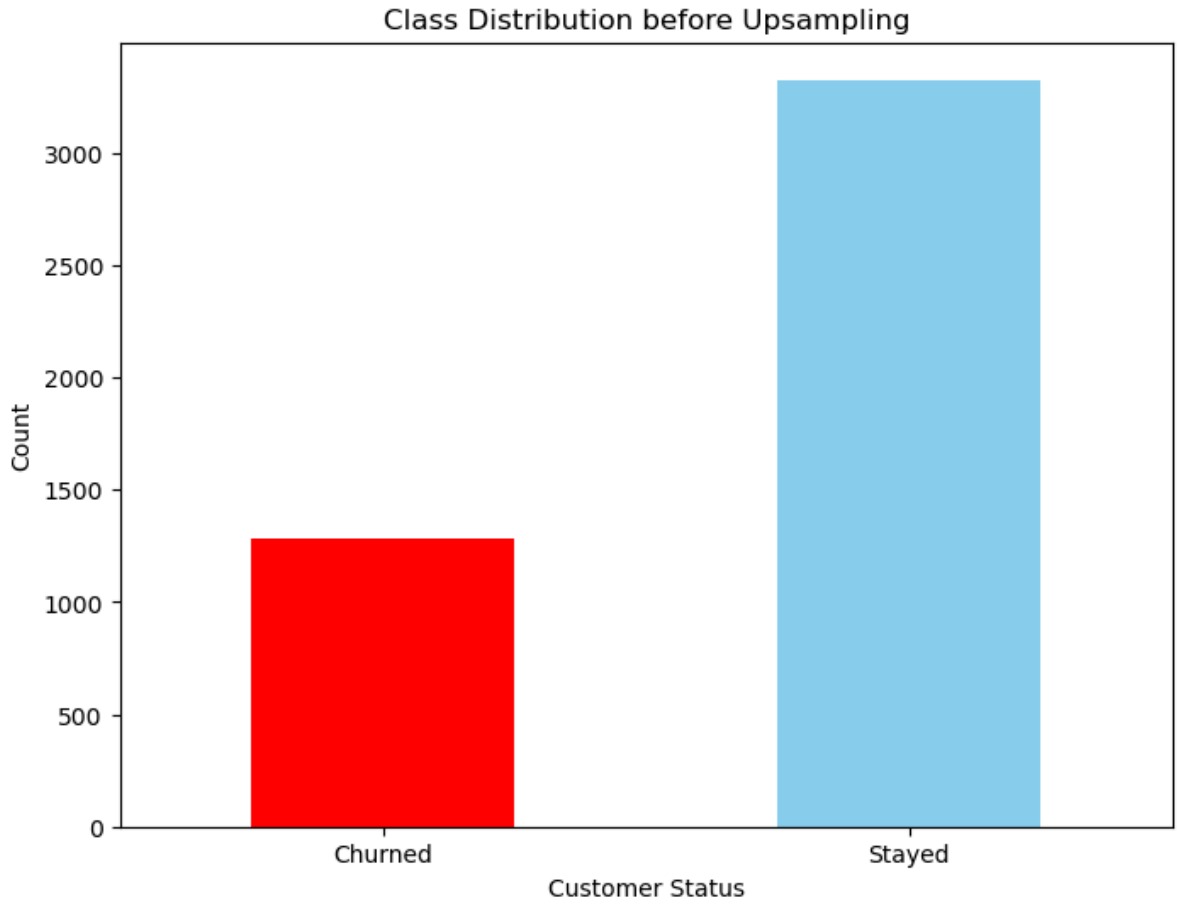
```
Shape of X_train: (4612, 16)
Shape of X_test: (1977, 16)
Shape of Y_train: (4612,)
Shape of Y_test: (1977,)
```

Target Variable Distribution & Upsampling

Plotting Customer Status before upsampling

In [61]:
```
# Count the occurrences of each class
class_counts = Y_train.value_counts()

# Reorder the class counts so that Churned (0) is displayed before Stayed (1)
```

```
class_counts_reordered = class_counts.reindex([0, 1])

# Plot the class distribution
plt.figure(figsize=(8, 6))
class_counts_reordered.plot(kind='bar', color=['red', 'skyblue',])
plt.xlabel('Customer Status')
plt.ylabel('Count')
plt.title('Class Distribution before Upsampling')
plt.xticks([0, 1], ['Churned', 'Stayed'], rotation=0)  # Set x-axis ticks to displa
plt.savefig("Target variable distribution.jpg")
plt.show()
```



Upsampling using SMOTE

In [62]:
```
# Initialize SMOTE with a desired sampling strategy
smote = SMOTE(sampling_strategy='auto', random_state=42)
# Upsample the minority class in the training data
X_train, Y_train = smote.fit_resample(X_train, Y_train)
X_test, Y_test = smote.fit_resample(X_test, Y_test)
```

In [63]:
```
# Count the occurrences of each class
class_counts = Y_train.value_counts()
# Reorder the class counts so that Churned (0) is displayed before Stayed (1)
class_counts_reordered = class_counts.reindex([0, 1])
# Plot the class distribution
plt.figure(figsize=(8, 6))
class_counts_reordered.plot(kind='bar', color=['red', 'skyblue',])
plt.xlabel('Customer Status')
plt.ylabel('Count')
plt.title('Class Distribution after Upsampling')
plt.xticks([0, 1], ['Churned', 'Stayed'], rotation=0)
plt.savefig("Distribution after SMOTE.jpg")
plt.show()
```

In [64]:
```
Y_train.value_counts()
```

Out[64]:
```
Customer Status
1    3327
0    3327
Name: count, dtype: int64
```

In [65]:
```
Y_test.value_counts()
```

Out[65]:
```
Customer Status
0    1393
1    1393
Name: count, dtype: int64
```

Machine Learning Algorithms

Logistic Regression

In [66]:
```
# Train a Logistic Regression Model (using the default parameters)
log_model = LogisticRegression(random_state=16, max_iter=1000)
# fit the model with data
log_model.fit(X_train, Y_train)
```

Out[66]:
```
LogisticRegression(max_iter=1000, random_state=16)
```

In [67]:
```
y_pred_log = log_model.predict(X_test)
```

In [68]:
```
#printing accuracy
accuracy_log = accuracy_score(Y_test, y_pred_log)
precision_log = precision_score(Y_test, y_pred_log)
recall_log = recall_score(Y_test, y_pred_log)
f1_log = f1_score(Y_test, y_pred_log)
print("Accuracy:", accuracy_log)
print("Precision:", precision_log)
print("Recall:", recall_log)
print("F1 Score:", f1_log)
# Print classification report
print("Logistic Regression Classification Report before optimisation:")
print(classification_report(Y_test, y_pred_log))
```

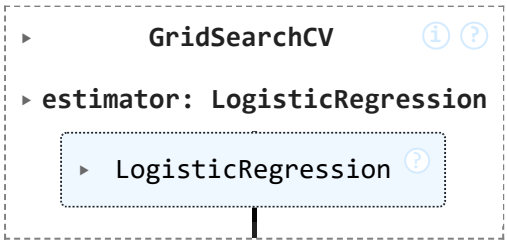
```
Accuracy: 0.8201722900215362
Precision: 0.8534072900158478
Recall: 0.7731514716439339
F1 Score: 0.8112994350282486
Logistic Regression Classification Report before optimisation:
              precision    recall  f1-score   support

           0       0.79      0.87      0.83      1393
           1       0.85      0.77      0.81      1393

    accuracy                           0.82      2786
   macro avg       0.82      0.82      0.82      2786
weighted avg       0.82      0.82      0.82      2786
```

Optimisation of Logistic Regression Model

In [69]:
```
# Define the parameter grid
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga']
}
```

In [70]:
```
warnings.filterwarnings("ignore", category=ConvergenceWarning)
# Creating a logistic regression model
logistic_regression = LogisticRegression(max_iter=1000, random_state=16)
# The GridSearchCV object
grid_search = GridSearchCV(logistic_regression, param_grid, cv=5, scoring='accuracy
# Performing grid search to find the best hyperparameters
grid_search.fit(X_train, Y_train)
```

Out[70]:



In [71]:
```
# Printing the best hyperparameters found
print("Best hyperparameters:", grid_search.best_params_)
```

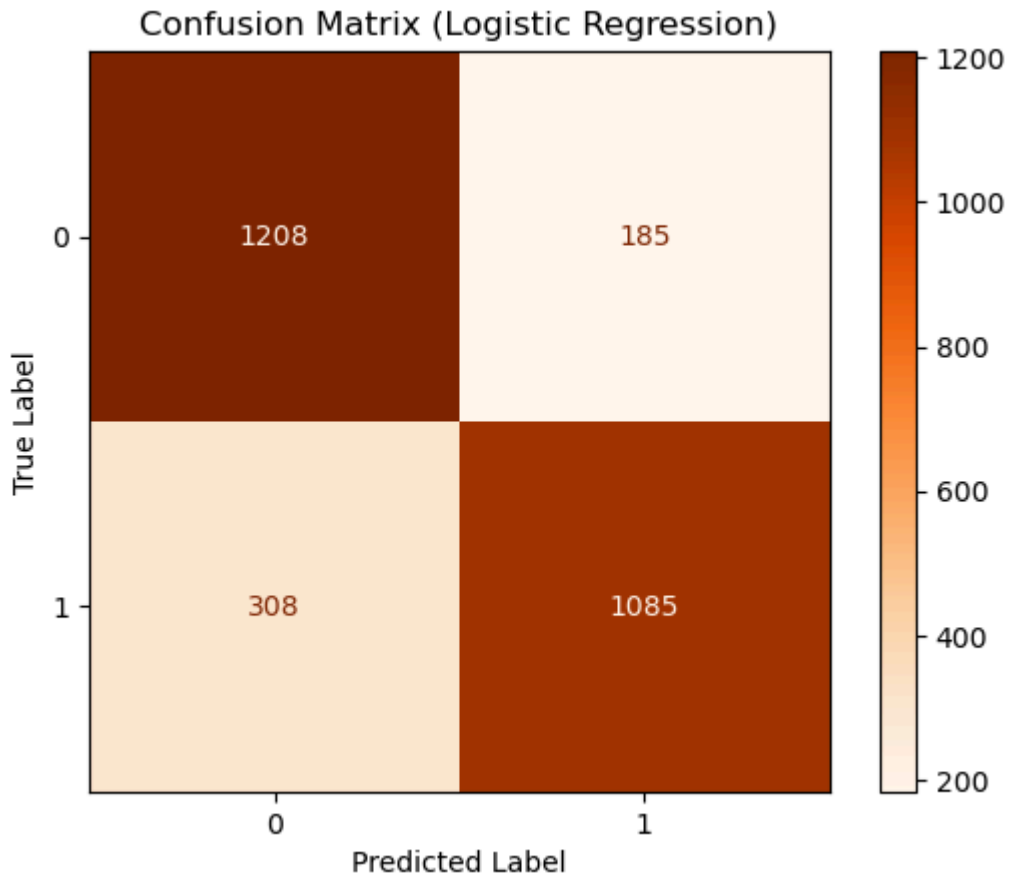
```
Best hyperparameters: {'C': 100, 'penalty': 'l1', 'solver': 'liblinear'}
```

In [72]:
```
# Get the best model
best_model = grid_search.best_estimator_
# Evaluate the best model on the test set
y_pred_best = best_model.predict(X_test)
```

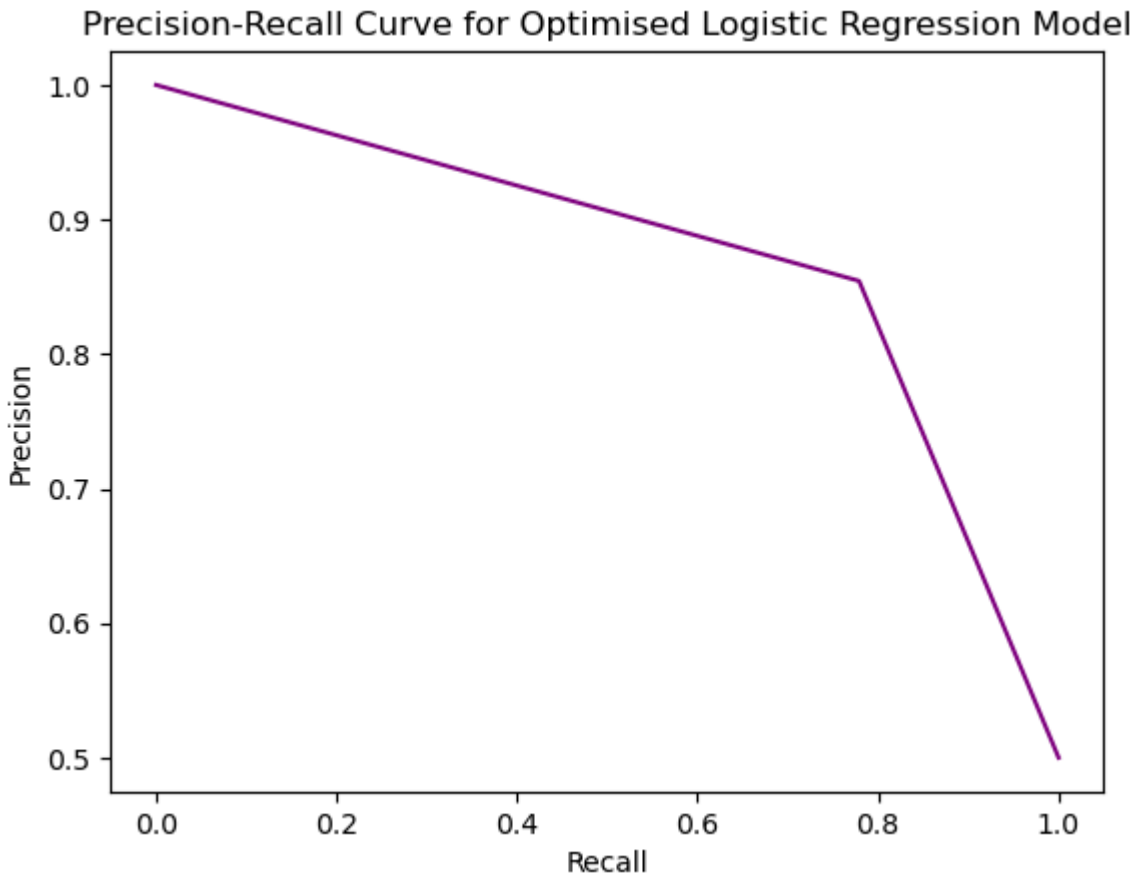
In [73]:
```
# Calculate accuracy
accuracy_logreg = accuracy_score(Y_test, y_pred_best)
precision_logreg = precision_score(Y_test, y_pred_best)
recall_logreg = recall_score(Y_test, y_pred_best)
f1_logreg = f1_score(Y_test, y_pred_best)
print("Accuracy of best model:", accuracy_logreg)
print("Precision of best model:", precision_logreg)
print("Recall of best model:", recall_logreg)
print("F1 of best model:", f1_logreg)
# Print classification report
print("Logistic Regression Classification Report:")
print(classification_report(Y_test, y_pred_best))
```

```
Accuracy of best model: 0.8230437903804738
Precision of best model: 0.8543307086614174
Recall of best model: 0.7788944723618091
F1 of best model: 0.8148704468644387
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.87      0.83      1393
           1       0.85      0.78      0.81      1393

    accuracy                           0.82      2786
   macro avg       0.83      0.82      0.82      2786
weighted avg       0.83      0.82      0.82      2786
```

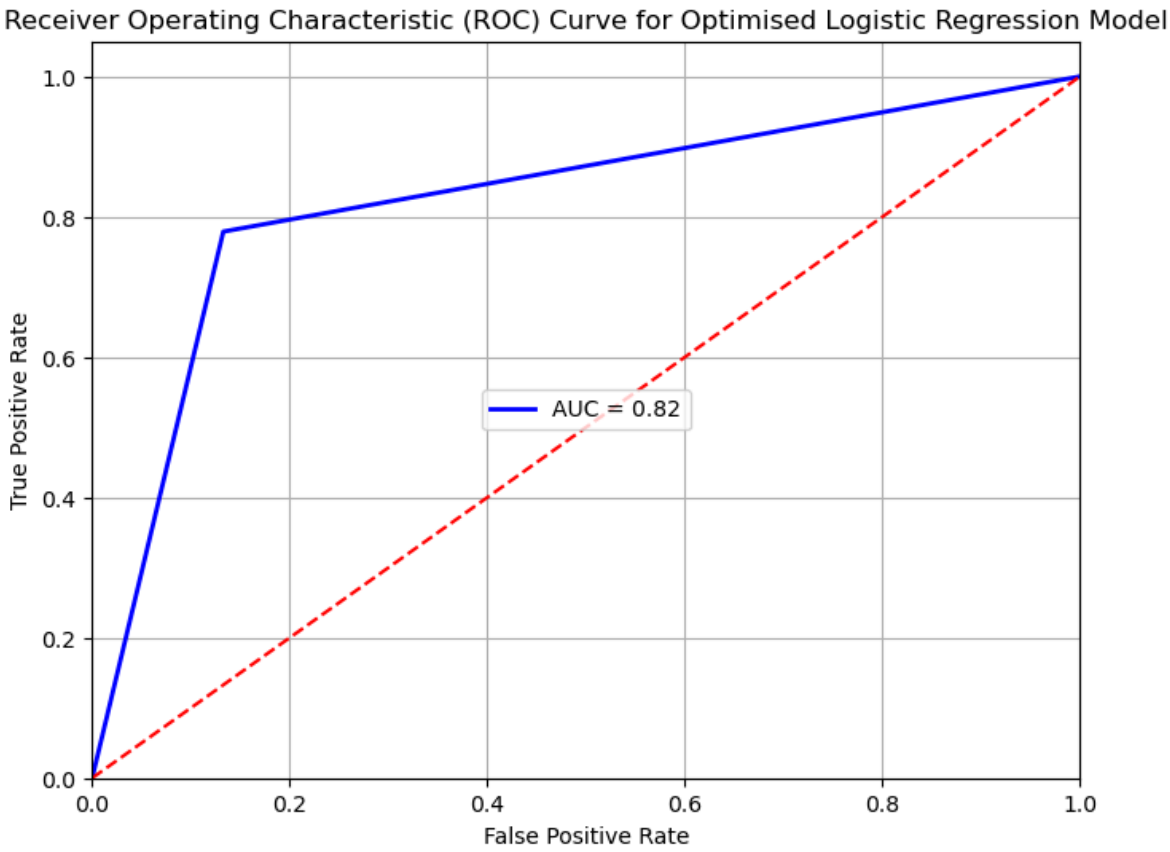
In [74]:
```
# Plot the confusion matrix for the Logistic Regression Model
cm_log = confusion_matrix(Y_test, y_pred_best)
disp_log = metrics.ConfusionMatrixDisplay(confusion_matrix=cm_log);
disp_log.plot(cmap='Oranges')
plt.title('Confusion Matrix (Logistic Regression)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('Confusion Matrix for Logistic Regression.jpg')
plt.show()
```



In [75]:
```
# Calculate Precision and Recall for the optimized model
precision_best, recall_best, thresholds_best = precision_recall_curve(Y_test, y_pre
# Create Precision-Recall Curve
fig, ax = plt.subplots()
ax.plot(recall_best, precision_best, color='purple')
# Adding axis labels and title to the plot
ax.set_title('Precision-Recall Curve for Optimised Logistic Regression Model')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
# Save the plot as an image file and display
plt.savefig('Precision-Recall Curve for Optimised Logistic Regression Model.jpg')
plt.show()
```



In [76]:
```
# Calculate False Positive Rate, True Positive Rate, and Thresholds for the optimiz
fpr_best, tpr_best, thresholds_best = roc_curve(Y_test, y_pred_best)
# Calculate the AUC score for the optimized model
auc_score_best = roc_auc_score(Y_test, y_pred_best)
# Plot ROC curve for the optimized model
plt.figure(figsize=(8, 6))
plt.plot(fpr_best, tpr_best, color='blue', lw=2, label=f'AUC = {auc_score_best:.2f}
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Optimised Logistic Reg
plt.legend(loc='center')
plt.grid(True)
plt.savefig('Receiver Operating Characteristic (ROC) Curve for Optimised Logistic R
plt.show()
```



Random Forest Classifier

In [77]:
```
# Train a Random Forest model
rf_model = RandomForestClassifier()
rf_model.fit(X_train, Y_train)
```

Out[77]:
```
RandomForestClassifier()
```

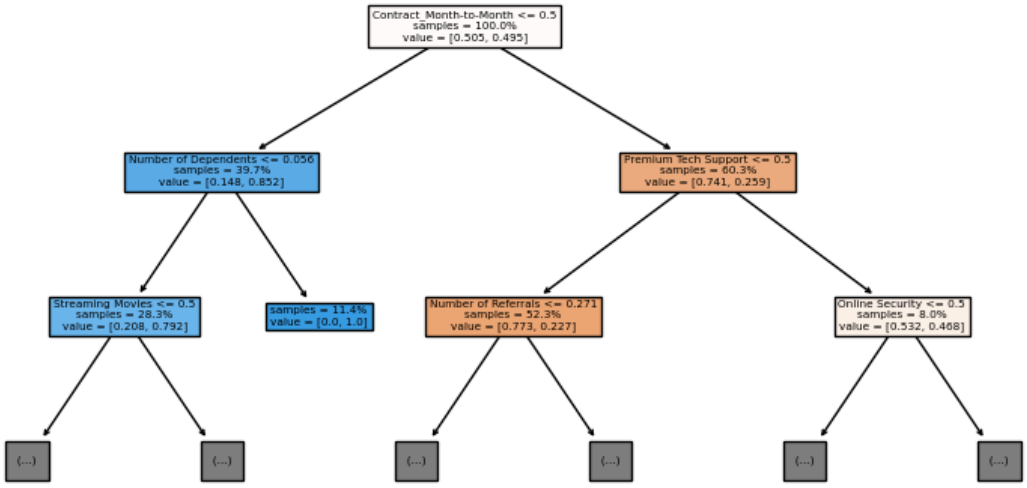
In [78]:
```
y_pred_rf = rf_model.predict(X_test)
```

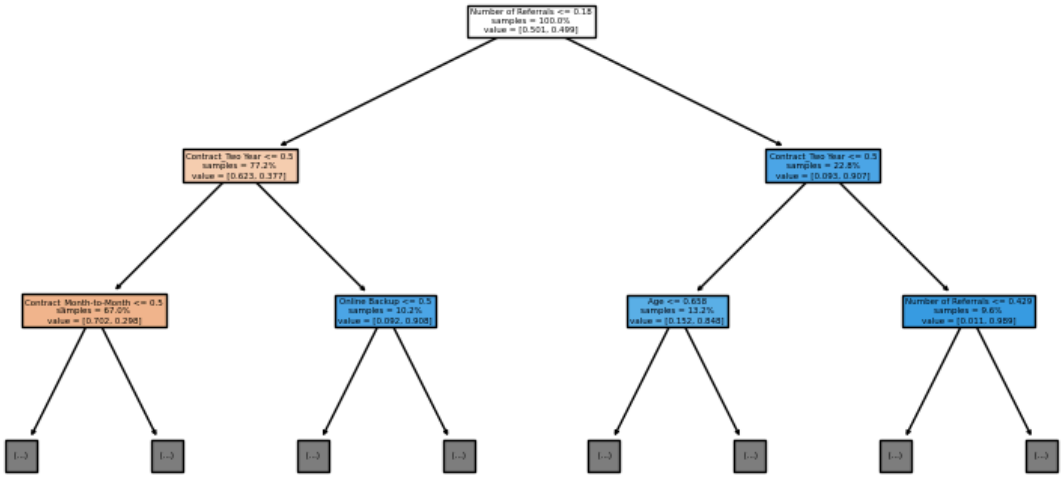
In [79]:
```
#printing accuracy
accuracy = accuracy_score(Y_test, y_pred_rf)
print("Accuracy:", accuracy)
# Print classification report
print("Random Forest Classification Report before optimisation:")
print(classification_report(Y_test, y_pred_rf))
```

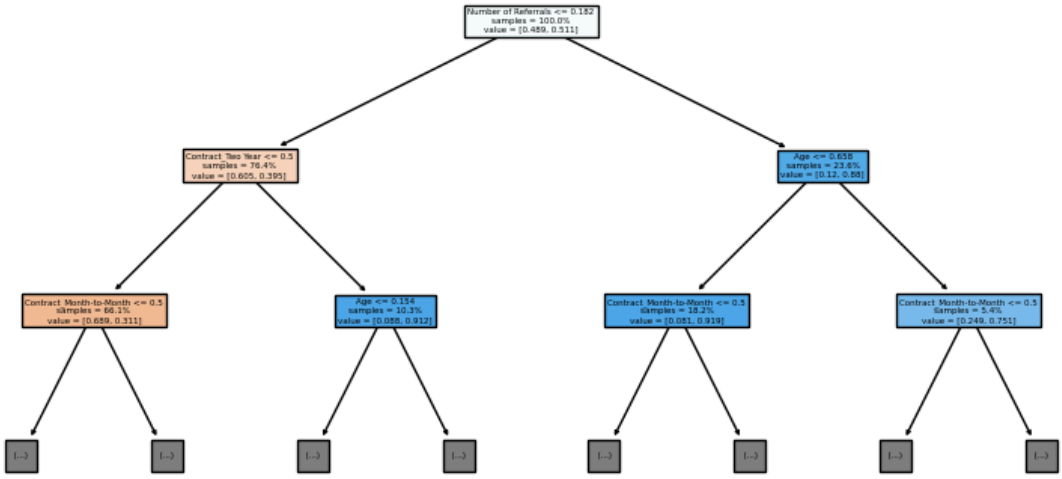
```
Accuracy: 0.7954055994256999
Random Forest Classification Report before optimisation:
              precision    recall  f1-score   support

           0       0.83      0.74      0.78      1393
           1       0.76      0.85      0.81      1393

    accuracy                           0.80      2786
   macro avg       0.80      0.80      0.79      2786
weighted avg       0.80      0.80      0.79      2786
```

In [80]:
```
# Convert X_train to a pandas DataFrame
X_train_df = pd.DataFrame(X_train)
```

```
from sklearn.tree import plot_tree

# Export the first three decision trees from the forest
for i in range(3):
    plt.figure(figsize=(10, 5))
    tree = rf_model.estimators_[i]
    plot_tree(tree,
              feature_names=X_train_df.columns,
              filled=True,
              max_depth=2,
              impurity=False,
              proportion=True)
    plt.show()
```



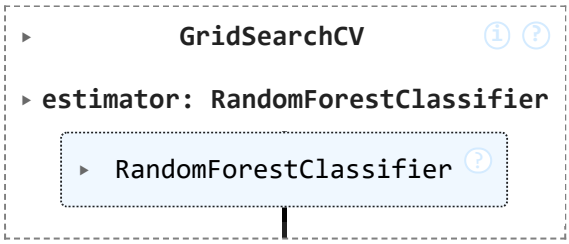




Optimising the Random Forest Model

In [81]:
```
# Define the parameter grid for RandomForestClassifier
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
# The grid search
grid_search = GridSearchCV(estimator=RandomForestClassifier(), param_grid=param_gri
# Fit the grid search to the data
grid_search.fit(X_train, Y_train)
```

Out[81]:



In [82]:
```
# Get the best parameters and the best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_
# Print the best parameters
print("Best Parameters:", best_params)
```

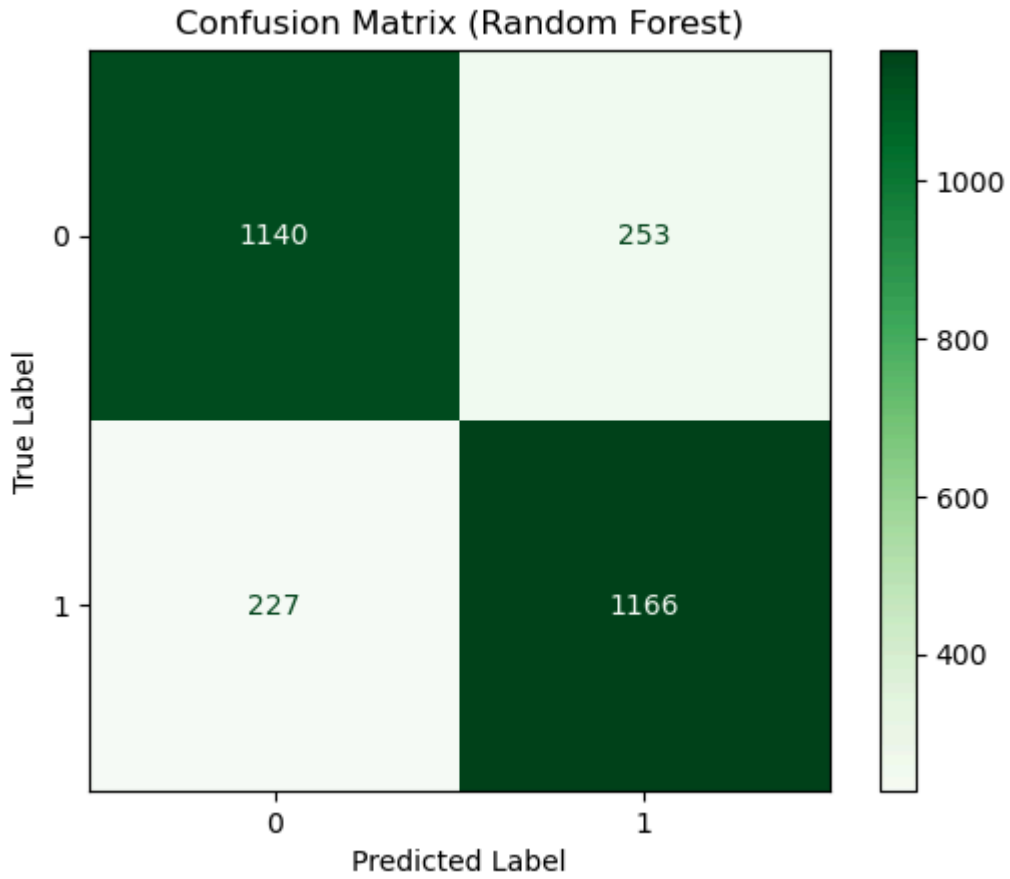
```
Best Parameters: {'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 5,
'n_estimators': 100}
```

In [83]:
```
# Generate predictions with the best model
y_pred_best_rf = best_estimator.predict(X_test)
accuracy_rf = accuracy_score(Y_test, y_pred_best_rf)
precision_rf = precision_score(Y_test, y_pred_best_rf)
recall_rf = recall_score(Y_test, y_pred_best_rf)
f1_rf = f1_score(Y_test, y_pred_best_rf)

print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1 Score:", f1_rf)
# Print classification report
```

```
print("Random Forest Classification Report:")
print(classification_report(Y_test, y_pred_best_rf))
```

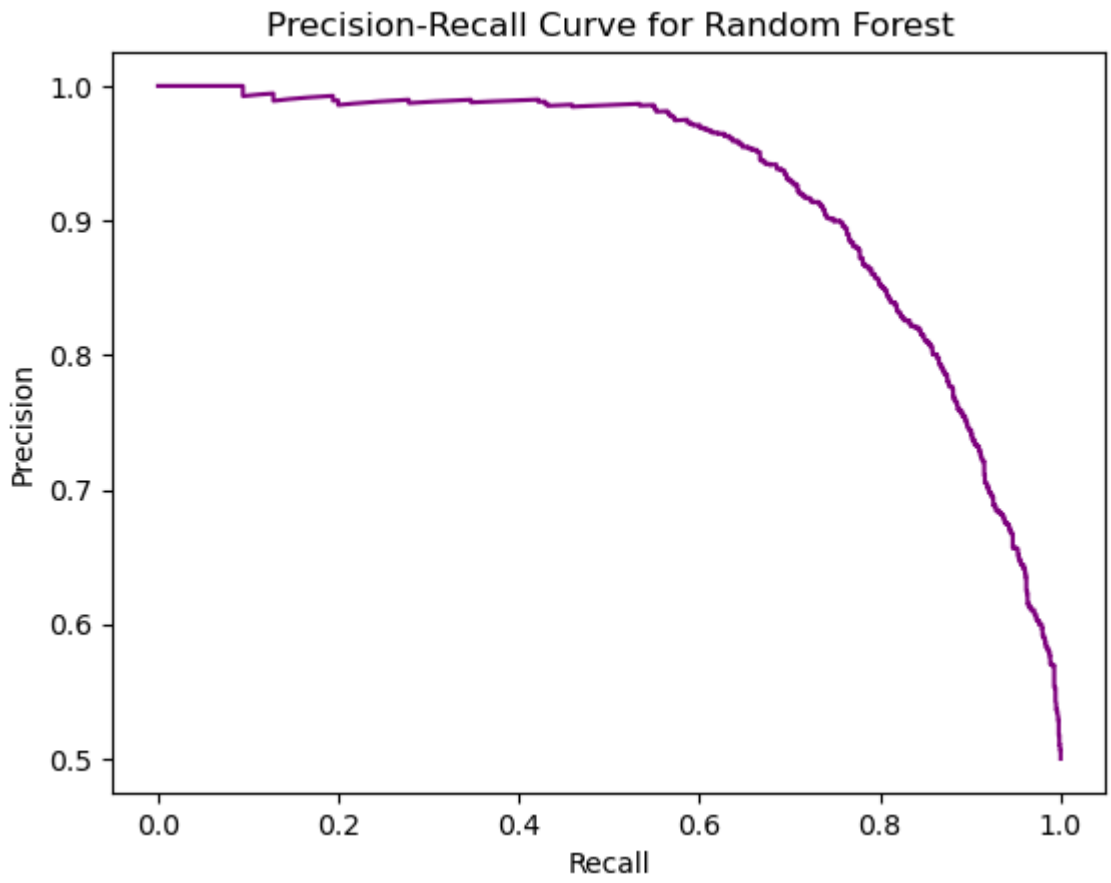
```
Accuracy: 0.8277099784637473
Precision: 0.8217054263565892
Recall: 0.8370423546302943
F1 Score: 0.829302987197724
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.82      0.83      1393
           1       0.82      0.84      0.83      1393

    accuracy                           0.83      2786
   macro avg       0.83      0.83      0.83      2786
weighted avg       0.83      0.83      0.83      2786
```

In [84]:
```
# Plot the confusion matrix for the Random Forest Model
cm = confusion_matrix(Y_test, y_pred_best_rf)
disp_rf = metrics.ConfusionMatrixDisplay(confusion_matrix=cm);
disp_rf.plot(cmap='Greens')
plt.title('Confusion Matrix (Random Forest)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig("Confusion Matrix for Random Forest")
plt.show()
```

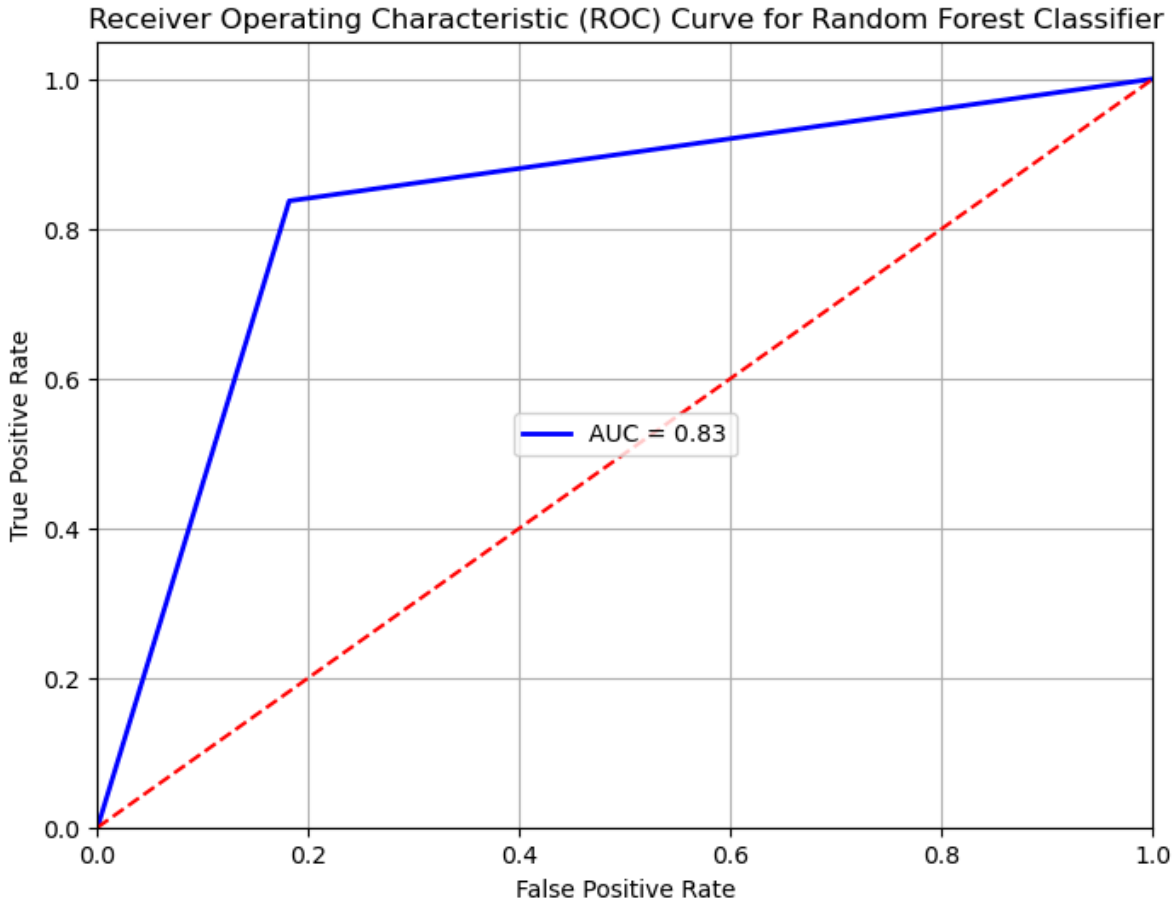


In [85]:
```
# Calculate Precision and Recall
precision, recall, thresholds = precision_recall_curve(Y_test, best_estimator.predi
# Create Precision Recall Curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')
# Adding axis labels and title to the plot
ax.set_title('Precision-Recall Curve for Random Forest')
ax.set_ylabel('Precision')
```

```
ax.set_xlabel('Recall')
plt.savefig('Precision-Recall Curve for Random Forest.jpg')
plt.show()
```



In [86]:
```
# Calculate the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(Y_test, y_pred_best_rf)
# Calculate the AUC score
auc_score = roc_auc_score(Y_test, y_pred_best_rf)
# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC = {auc_score:.2f}')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Random Forest Classifi
plt.legend(loc='center')
plt.grid(True)
plt.savefig('Receiver Operating Characteristic (ROC) Curve for Random Forest Classi
plt.show()
```



K Nearest Neighbour

In [87]:
```
# Initializing the KNN classifier with a specified number of neighbors (e.g., 3)
knn = KNeighborsClassifier(n_neighbors=3)
```

In [88]:
```
# Training the classifier on the training data
knn.fit(X_train, Y_train)
```

Out[88]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [89]:
```
#Evaluating the performance of the model using the test set
knn_Y_predict = knn.predict(X_test)
```

In [90]:
```
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(Y_test, knn_Y_predict))
```

```
Accuracy: 0.7613065326633166
```

In [91]:
```
# Print classification report
print("KNN Classification Report:")
print(classification_report(Y_test,knn_Y_predict))
```

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

           0       0.77      0.74      0.76      1393
           1       0.75      0.78      0.77      1393

    accuracy                           0.76      2786
   macro avg       0.76      0.76      0.76      2786
weighted avg       0.76      0.76      0.76      2786
```

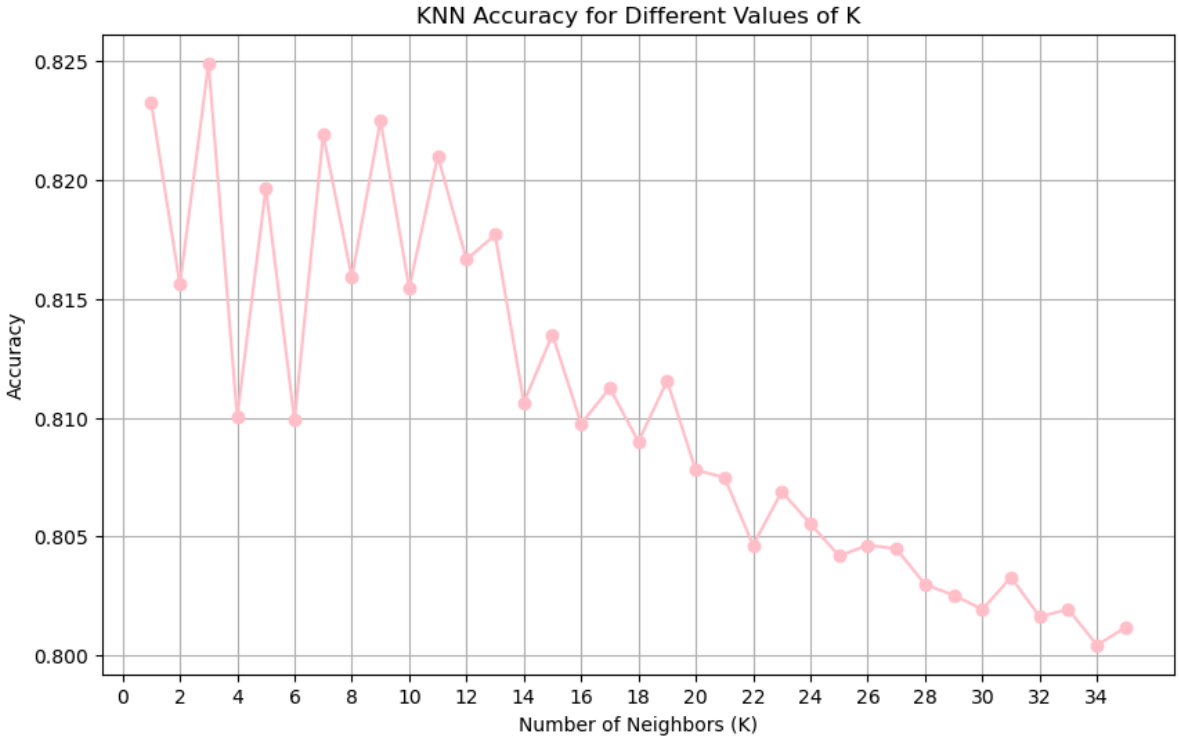
Optimising the KNN Model

In [92]:
```
# Defining the range of values for n_neighbors
neighbors = np.arange(1, 36)
```

In [93]:
```
# Create an empty list to store the accuracy at different values of K
KNN_accuracy = []
```

In [94]:
```
# Loop through each value of n_neighbors and perform grid search
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    grid_search = GridSearchCV(knn, {'n_neighbors': [k]}, cv=5)
    grid_search.fit(X_train, Y_train)
    KNN_accuracy.append(grid_search.best_score_)
```

In [95]:
```
# Plot the mean test scores
plt.figure(figsize=(10, 6))
plt.plot(neighbors, KNN_accuracy, marker='o')
plt.title('KNN Accuracy for Different Values of K')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.xticks(np.arange(0, 36, step=2))
plt.savefig('KNN Accuracy for Different Values of K.jpg')
plt.show()
```



In [96]:
```
best_index = np.argmax(KNN_accuracy)
best_k = neighbors[best_index]

print(f"The best value of n_neighbors (K) is: {best_k}")
```

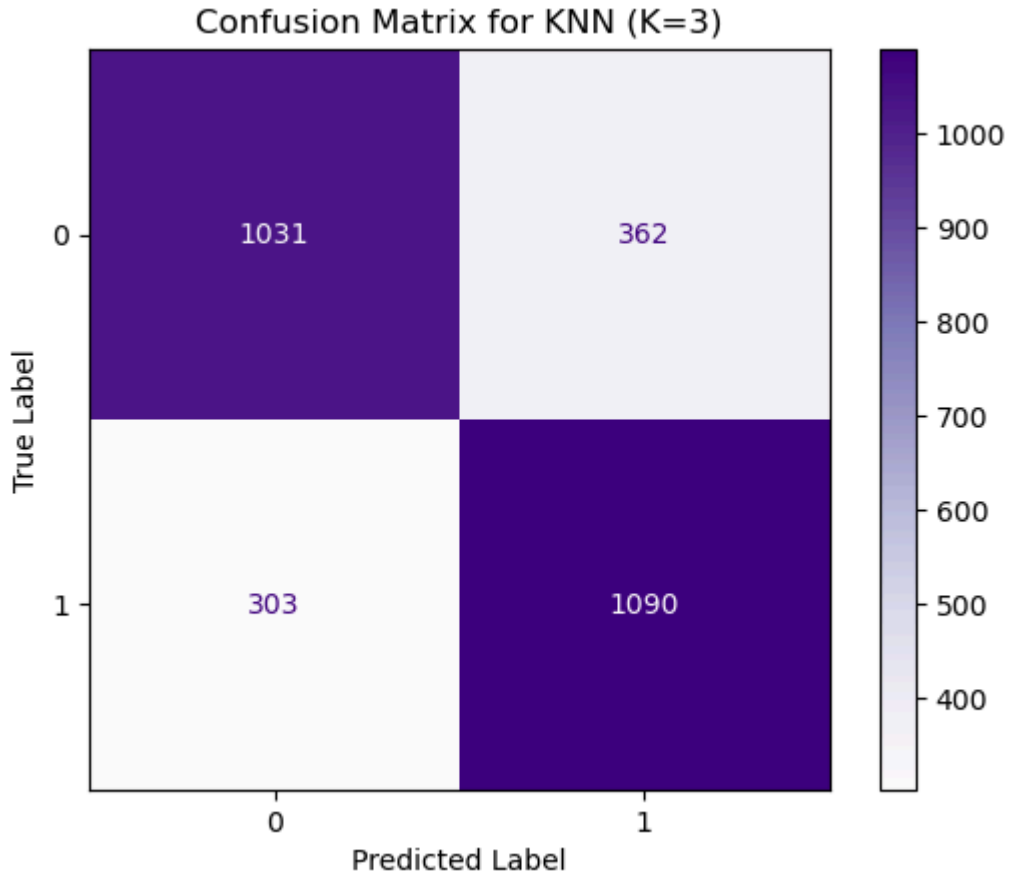
```
The best value of n_neighbors (K) is: 3
```

In [97]:
```
knn= KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train,Y_train)
best_k_predictions = knn.predict(X_test)
```

In [98]:
```
# Calculate precision and recall for the best K value
accuracy_knn = accuracy_score(Y_test, best_k_predictions)
precision_knn = precision_score(Y_test, best_k_predictions)
recall_knn = recall_score(Y_test, best_k_predictions)
f1_knn = f1_score(Y_test, best_k_predictions)
# Print the accuracy, precision, and recall for the best K value
print(f"Best K value: {best_k}")
print(f"Accuracy for best K value: {accuracy_knn}")
print(f"Precision for best K value: {precision_knn}")
print(f"Recall for best K value: {recall_knn}")
print(f"F1 Score for best K value: {f1_knn}")

# Print classification report
print("KNN Classification Report:")
print(classification_report(Y_test ,best_k_predictions))
```

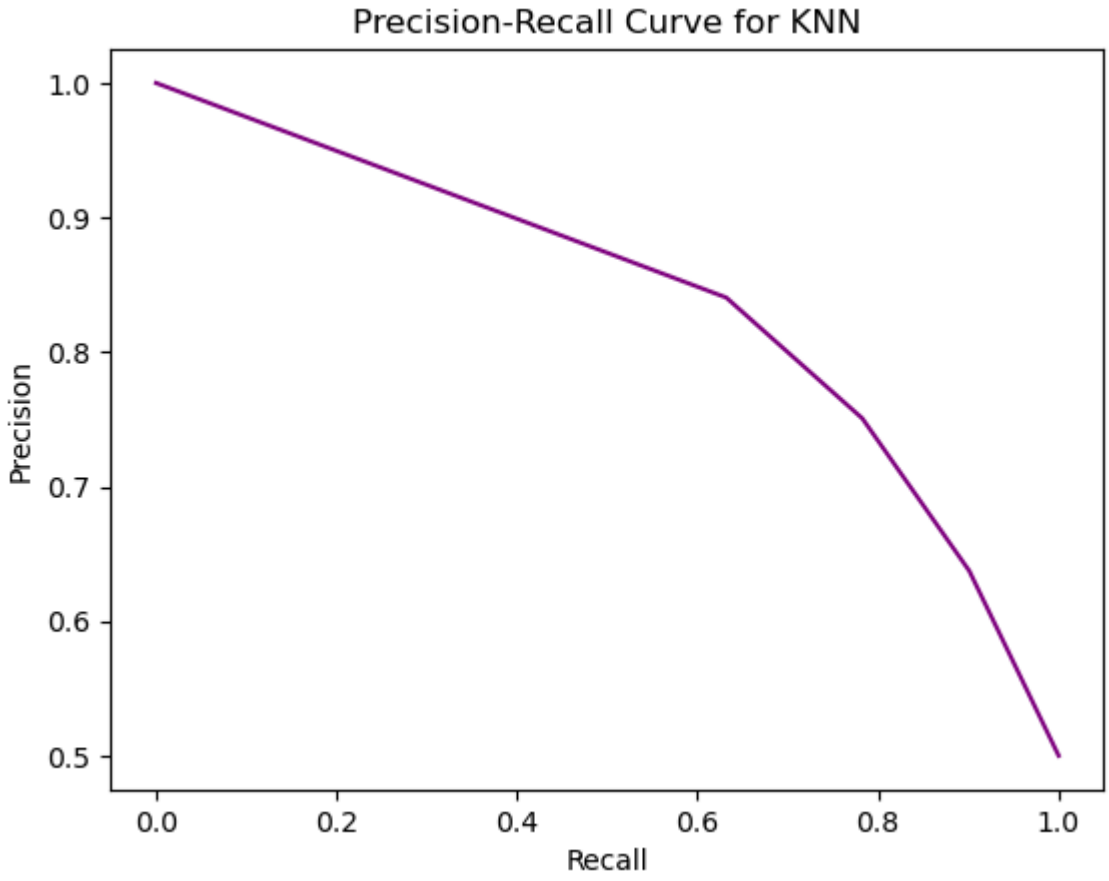
```
Best K value: 3
Accuracy for best K value: 0.7613065326633166
Precision for best K value: 0.7506887052341598
Recall for best K value: 0.782483847810481
F1 Score for best K value: 0.7662565905096661
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.74      0.76      1393
           1       0.75      0.78      0.77      1393

    accuracy                           0.76      2786
   macro avg       0.76      0.76      0.76      2786
weighted avg       0.76      0.76      0.76      2786
```

In [99]:
```
# Plot the confusion matrix for the KNN Model
cm_knn = confusion_matrix(Y_test, best_k_predictions)
disp_knn = metrics.ConfusionMatrixDisplay(confusion_matrix=cm_knn);
disp_knn.plot(cmap='Purples')
plt.title(f'Confusion Matrix for KNN (K={best_k})')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('Confusion Matrix for KNN.jpg')
plt.show()
```



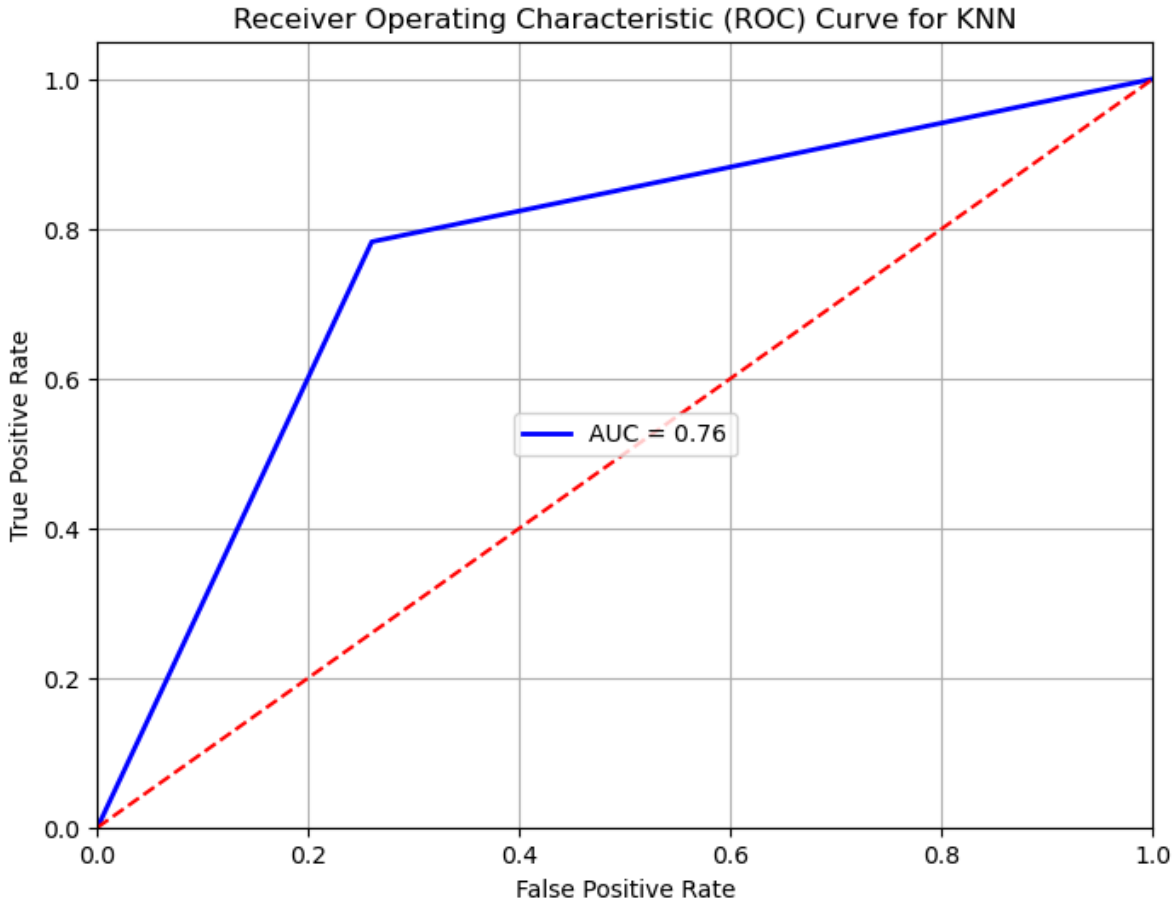
In [100…

```
# Calculate Precision and Recall
precision, recall, thresholds = precision_recall_curve(Y_test, knn.predict_proba(X_
# Create Precision Recall Curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')
# Adding axis labels and title to the plot
ax.set_title('Precision-Recall Curve for KNN')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
plt.savefig('Precision-Recall Curve for KNN.jpg')
plt.show()
```



In [101…

```
# Calculate the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(Y_test, best_k_predictions)
# Calculate the AUC score
auc_score = roc_auc_score(Y_test, best_k_predictions)
# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC = {auc_score:.2f}')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for KNN')
plt.legend(loc='center')
plt.grid(True)
plt.savefig('Receiver Operating Characteristic (ROC) Curve for KNN.jpg')
plt.show()
```



Guassian Naives Bayes (GNB) Model

In [102…
```
# Initializing the GNB classifier
GNB = GaussianNB()
```

In [103…
```
# Fitting the classifier to the training data
GNB.fit(X_train, Y_train)
```

Out[103]:
```
GaussianNB
GaussianNB()
```

In [104…
```
# Predicting the target variable for the test data
gnb_Y_predict = GNB.predict(X_test)
```

In [105…
```
# Calculate precision and recall for the GNB Model
accuracy_gnb = accuracy_score(Y_test, gnb_Y_predict)
precision_gnb = precision_score(Y_test, gnb_Y_predict)
recall_gnb= recall_score(Y_test, gnb_Y_predict)
f1_gnb= f1_score(Y_test, gnb_Y_predict)
# Print the accuracy, precision, and recall for GNB Model
print(f"Accuracy for GNB Model: {accuracy_gnb} ")
print(f"Precision for GNB Model: {precision_gnb}")
print(f"Recall for GNB Model: {recall_gnb}")
print(f"F1 Score for GNB Model: {f1_gnb}")
```

```
Accuracy for GNB Model: 0.7989949748743719
Precision for GNB Model: 0.8078344419807835
Recall for GNB Model: 0.7846374730796841
F1 Score for GNB Model: 0.7960670065549891
```

In [106…
```
# Print classification report
print("GNB Classification Report:")
print(classification_report(Y_test, gnb_Y_predict))
```

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

           0       0.79      0.81      0.80      1393
           1       0.81      0.78      0.80      1393

    accuracy                           0.80      2786
   macro avg       0.80      0.80      0.80      2786
weighted avg       0.80      0.80      0.80      2786
```

Optimising the GNB Model

In [107…
```
# Define the parameter grid
param_grid = {'var_smoothing': np.logspace(0,-9, num=100)}
# Initialize GridSearchCV
grid_search = GridSearchCV(GNB, param_grid, cv=5, scoring='accuracy')
```

In [108…
```
# Fit the grid search to the data
grid_search.fit(X_train, Y_train)
# Get the best parameters
best_params = grid_search.best_params_
```

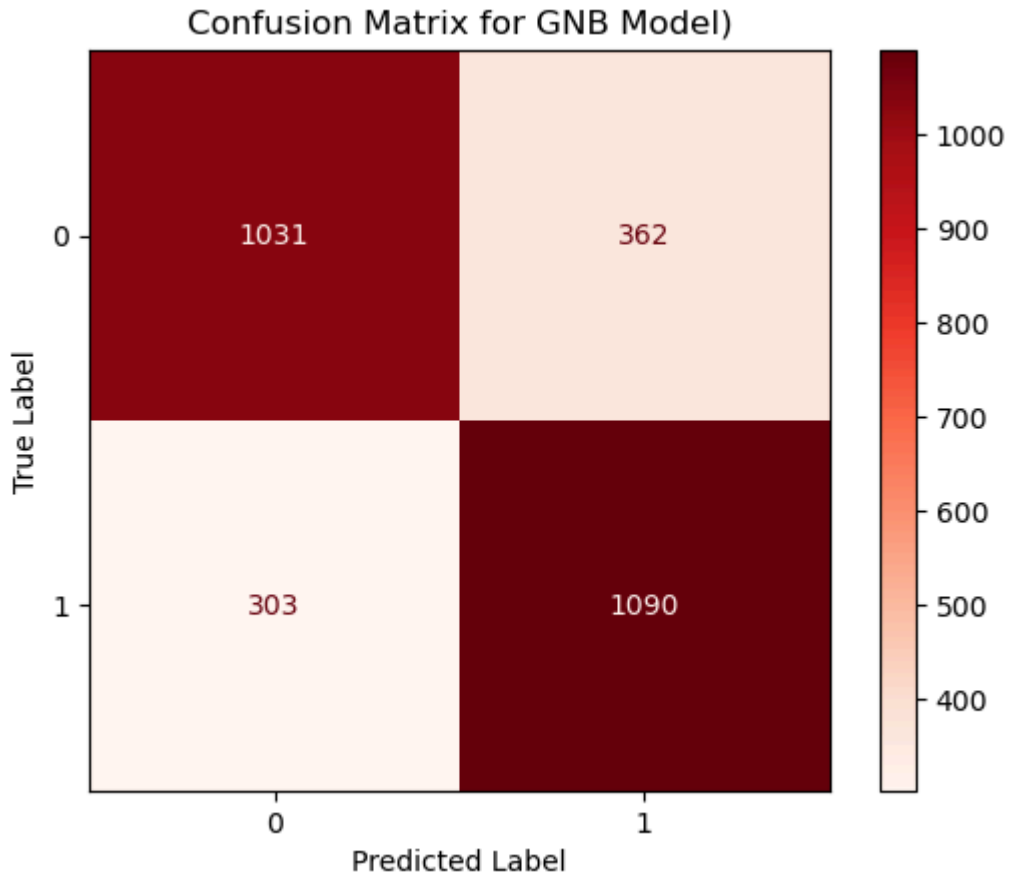
In [109…
```
# Initialize GNB with the best parameters
optimised_GNB = GaussianNB(var_smoothing=best_params['var_smoothing'])
# Fit the optimized model to the training data
optimised_GNB.fit(X_train, Y_train)
# Predict the target variable for the test data
optimised_gnb_Y_predict = optimised_GNB.predict(X_test)
```

In [110…
```
# Calculate and print accuracy, precision, and recall for the optimized GNB model
accuracy_gnb = accuracy_score(Y_test, optimised_gnb_Y_predict)
precision_gnb = precision_score(Y_test, optimised_gnb_Y_predict)
recall_gnb = recall_score(Y_test, optimised_gnb_Y_predict)
f1_gnb = f1_score(Y_test, optimised_gnb_Y_predict)

print(f"Accuracy for Optimised GNB Model: {accuracy_gnb}")
print(f"Precision for Optimised GNB Model: {precision_gnb}")
print(f"Recall for Optimised GNB Model: {recall_gnb}")
print(f"F1 Score for Optimised GNB Model: {f1_gnb}")
# Print classification report
print("GNB Classification Report:")
print(classification_report(Y_test,  optimised_gnb_Y_predict))
```

```
Accuracy for Optimised GNB Model: 0.7982770997846375
Precision for Optimised GNB Model: 0.8070953436807096
Recall for Optimised GNB Model: 0.7839195979899497
F1 Score for Optimised GNB Model: 0.7953386744355426
GNB Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.81      0.80      1393
           1       0.81      0.78      0.80      1393

    accuracy                           0.80      2786
   macro avg       0.80      0.80      0.80      2786
weighted avg       0.80      0.80      0.80      2786
```

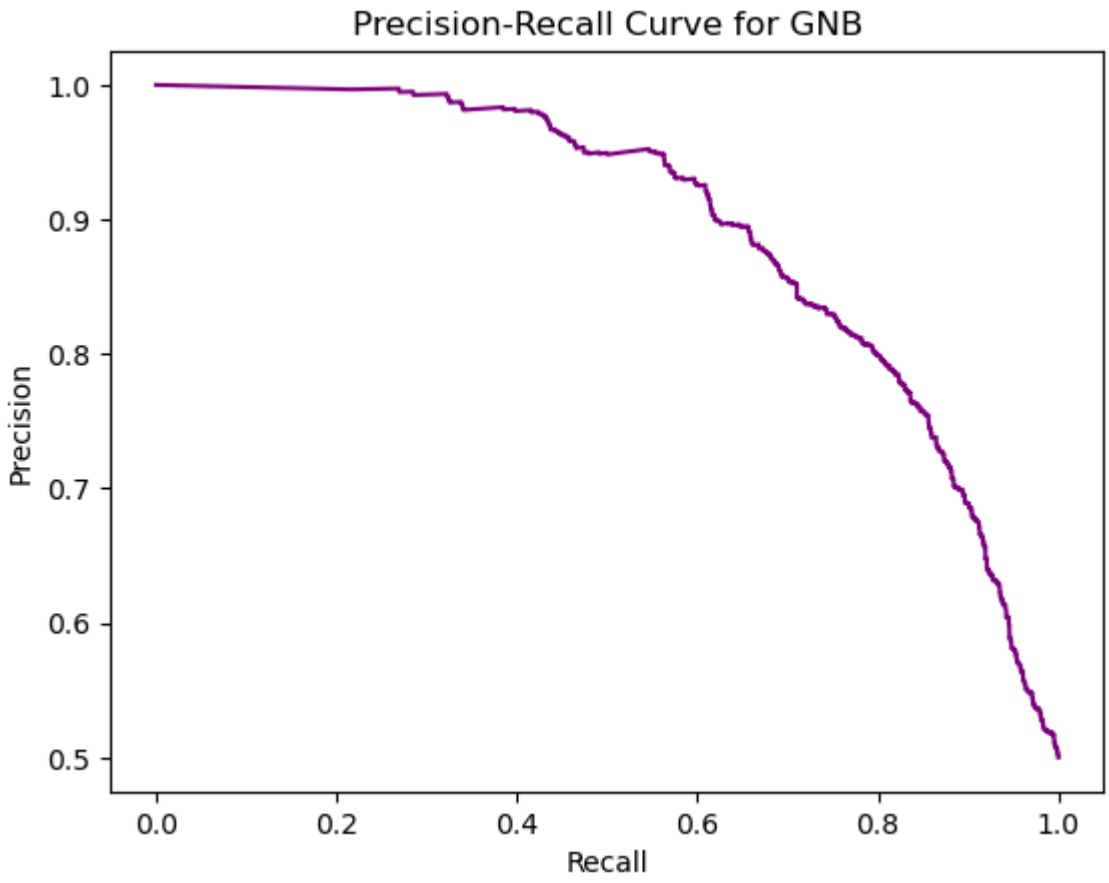
In [111…

```
# Plot the confusion matrix for the GNB Model
cm_gnb = confusion_matrix(Y_test,  optimised_gnb_Y_predict)
disp_gnb = metrics.ConfusionMatrixDisplay(confusion_matrix=cm_knn);
disp_gnb.plot(cmap='Reds')
plt.title(f'Confusion Matrix for GNB Model)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig('Confusion Matrix for GNB.jpg')
plt.show()
```



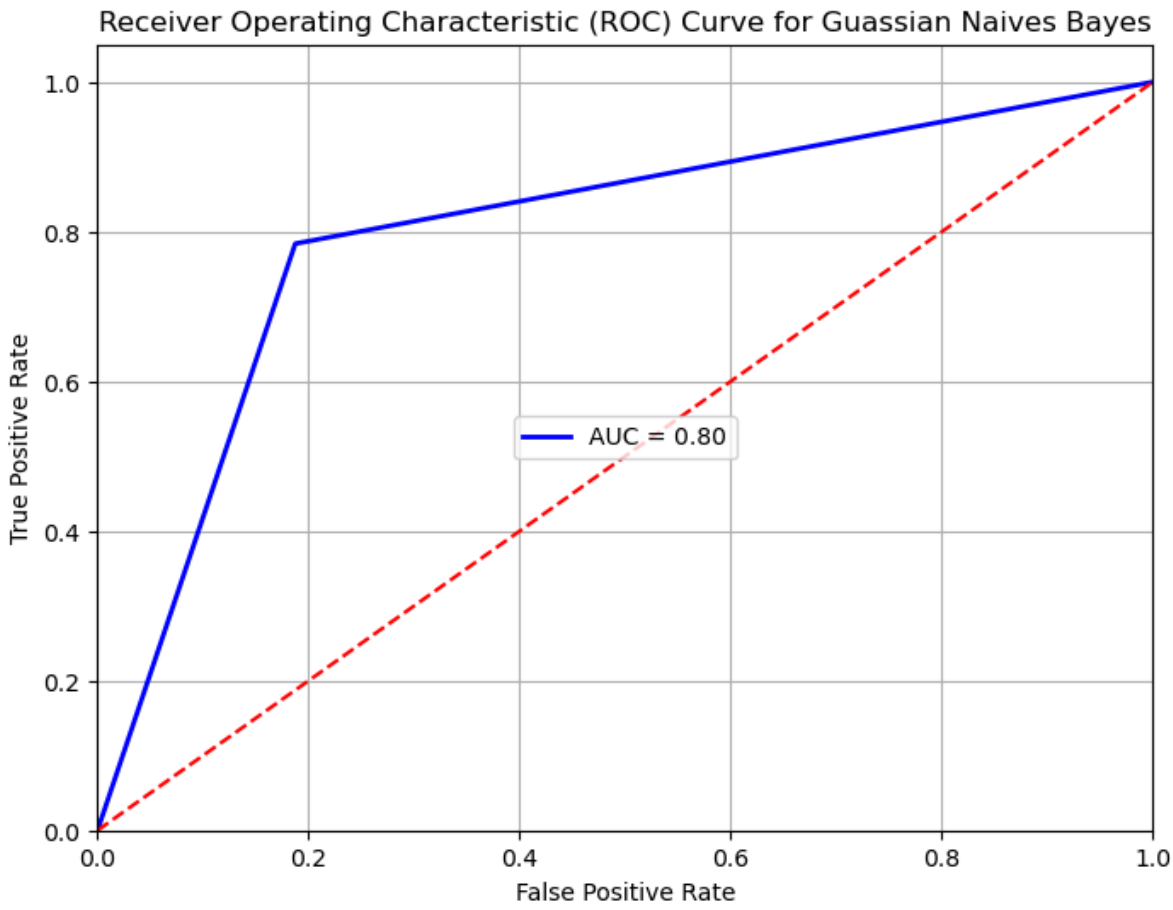
In [112…

```
# Calculate Precision and Recall
precision, recall, thresholds = precision_recall_curve(Y_test, optimised_GNB.predic
# Create Precision Recall Curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')
# Adding axis labels and title to the plot
ax.set_title('Precision-Recall Curve for GNB')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
plt.savefig('Precision-Recall Curve for GNB.jpg')
plt.show()
```



In [113…

```
# Calculate the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(Y_test, optimised_gnb_Y_predict)
# Calculate the AUC score
auc_score = roc_auc_score(Y_test, optimised_gnb_Y_predict)
# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC = {auc_score:.2f}')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Guassian Naives Bayes'
plt.legend(loc='center')
plt.grid(True)
plt.savefig('Receiver Operating Characteristic (ROC) Curve for GNB.jpg')
plt.show()
```

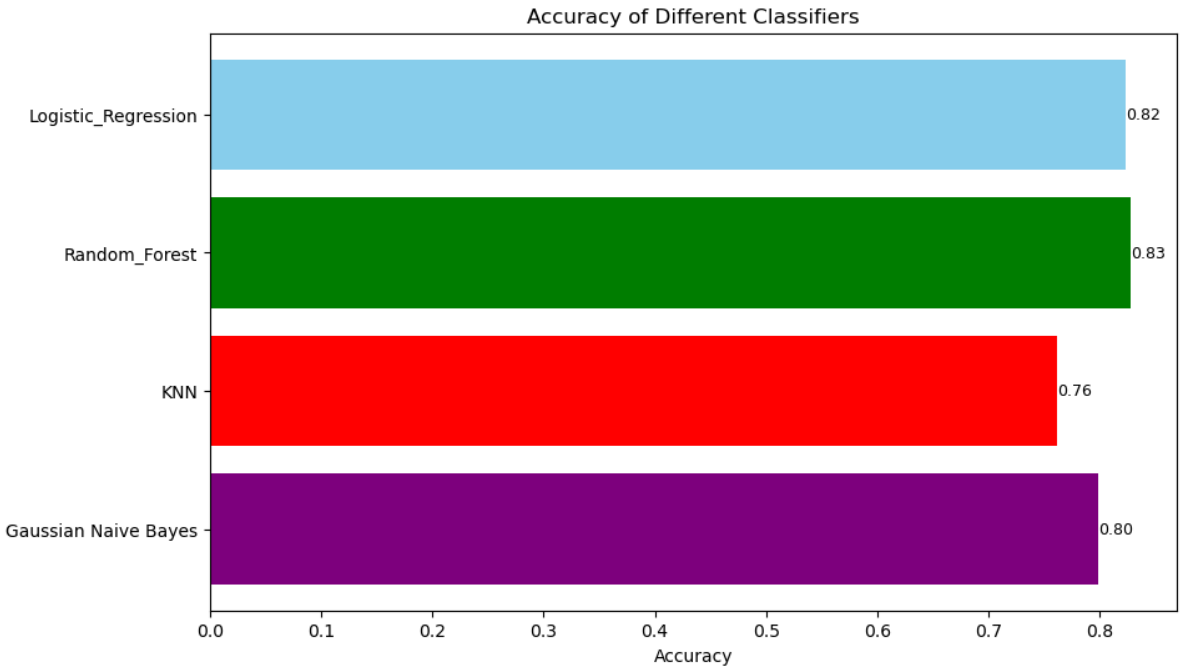


Machine Learning Models Comparison

Accuracy Comparison

In [114…

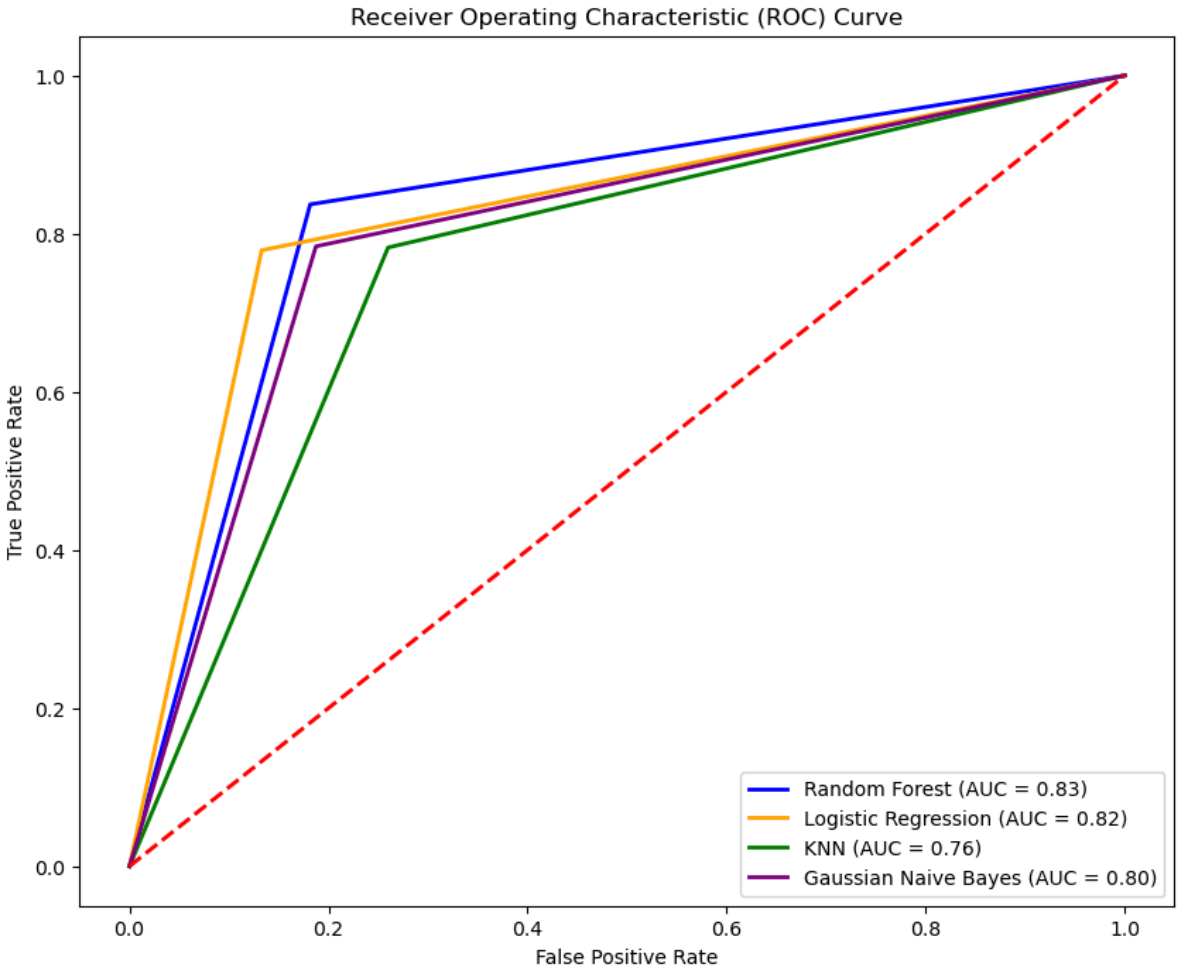
```
# Classifier names
classifiers = ['Logistic_Regression', 'Random_Forest', 'KNN', 'Gaussian Naive Bayes
# Accuracy scores obtained from cross-validation or testing
accuracy_scores = [accuracy_logreg, accuracy_rf, accuracy_knn, accuracy_gnb]
# Define colors for each classifier
colors = ['skyblue', 'green', 'red', 'purple']
# Create a bar plot
plt.figure(figsize=(10, 6))
bars = plt.barh(classifiers, accuracy_scores, color=colors)
plt.xlabel('Accuracy')
plt.title('Accuracy of Different Classifiers')
plt.gca().invert_yaxis()
# Add labels on bars
for bar, score in zip(bars, accuracy_scores):
    plt.text(bar.get_width(), bar.get_y() + bar.get_height()/2, f'{score:.2f}',
             va='center', ha='left', fontsize=9)
plt.savefig('Model Comparision.jpg')
plt.show()
```



Receiver Operating Characteristic (ROC) - Comparison of Models

In [115…

```
# Calculate ROC curves and AUC scores for each model
models = {
    'Random Forest': (y_pred_best_rf, 'blue'),
    'Logistic Regression': (y_pred_best, 'orange'),
    'KNN': (best_k_predictions, 'green'),
    'Gaussian Naive Bayes': (optimised_gnb_Y_predict, 'purple')
}
plt.figure(figsize=(10, 8))
# Plot ROC curves for each model
for model_name, (y_pred, color) in models.items():
    fpr, tpr, _ = roc_curve(Y_test, y_pred)
    auc_score = roc_auc_score(Y_test, y_pred)
    plt.plot(fpr, tpr, lw=2, label=f'{model_name} (AUC = {auc_score:.2f})', color=c
# Plot the diagonal line (no-skill line)
plt.plot([0, 1], [0, 1], linestyle='--', color='red', lw=2)
# Add legend, labels, and title
plt.legend(loc='lower right')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.savefig('Combined_ROC_Curve.png')
plt.show()
```



Precision-Recall & F1 Comparison

In [116…

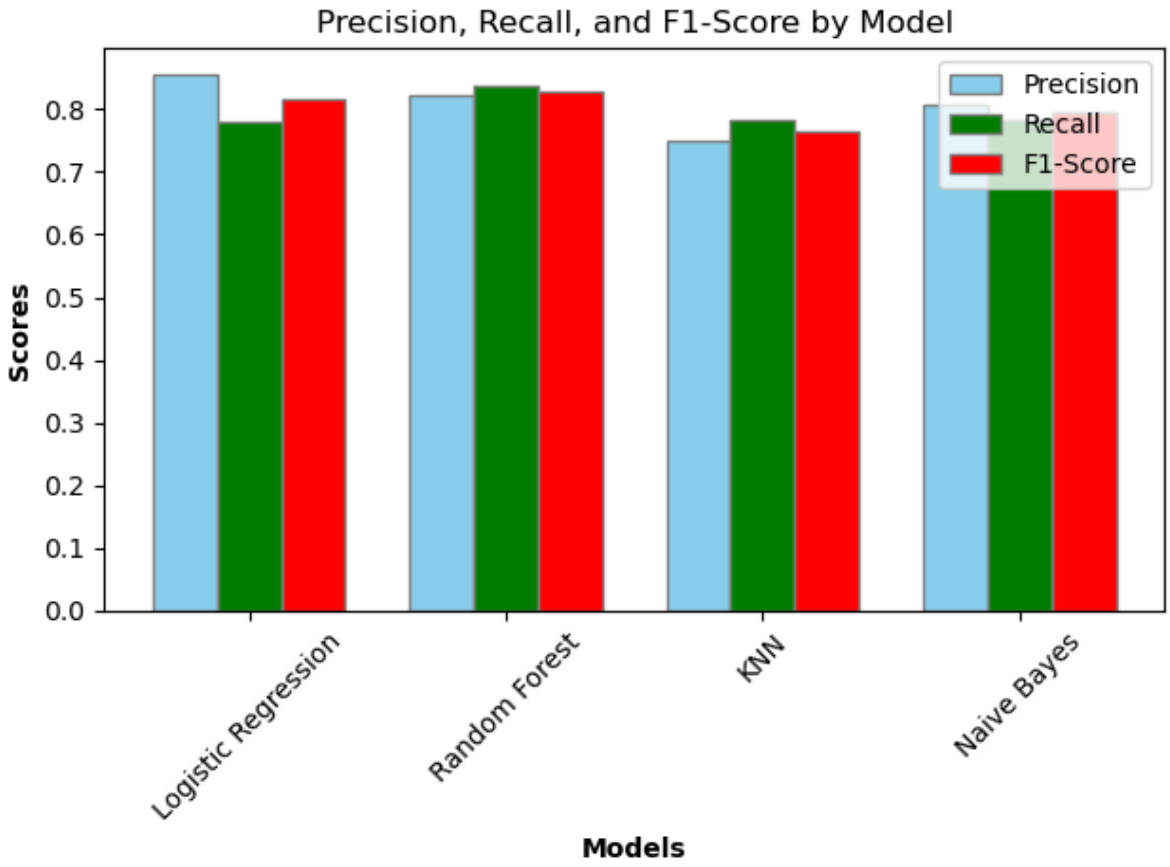
```
# Define the results dictionary
results = {
    'Model': ['Logistic Regression', 'Random Forest', 'KNN', 'Naive Bayes'],
    'Accuracy %': [accuracy_score(Y_test, y_pred_best), accuracy_score(Y_test, y_pr
    'Precision %': [precision_score(Y_test, y_pred_best), precision_score(Y_test, y
    'Recall %': [recall_score(Y_test, y_pred_best), recall_score(Y_test, y_pred_bes
    'F1-Score': [f"{f1_score(Y_test, y_pred_best):.2f}", f"{f1_score(Y_test, y_prec
}
# Convert accuracy, precision, recall scores to percentages and F1 scores to string
results['Accuracy %'] = [f"{score * 100:.2f}%" for score in results['Accuracy %']]
results['Precision %'] = [f"{score * 100:.2f}%" for score in results['Precision %']
results['Recall %'] = [f"{score * 100:.2f}%" for score in results['Recall %']]
# Create a DataFrame
results_df = pd.DataFrame(results)
# Display the DataFrame
print(results_df)
# Plot the DataFrame as a table with increased font size
fig, ax = plt.subplots(figsize=(14, 8))
ax.axis('tight')
ax.axis('off')
table = ax.table(cellText=results_df.values, colLabels=results_df.columns, loc='cer
# Adjust font size
table.auto_set_font_size(False)
table.set_fontsize(10)
# Save the table as an image
plt.savefig('performance_metrics_table.jpg')
plt.show()
```

```
                 Model Accuracy % Precision % Recall % F1-Score
0  Logistic Regression      82.30%       85.43%   77.89%     0.81
1        Random Forest      82.77%       82.17%   83.70%     0.83
2                  KNN      76.13%       75.07%   78.25%     0.77
3          Naive Bayes      79.83%       80.71%   78.39%     0.80
```

Model	Accuracy %	Precision %	Recall %	F1-Score
Logistic Regression	82.30%	85.43%	77.89%	0.81
Random Forest	82.77%	82.17%	83.70%	0.83
KNN	76.13%	75.07%	78.25%	0.77
Naive Bayes	79.83%	80.71%	78.39%	0.80

In [117…

```
# Define data
models = ['Logistic Regression', 'Random Forest', 'KNN', 'Naive Bayes']
y_true = [Y_test, Y_test, Y_test, Y_test]  # Replace with your true labels
y_pred = [y_pred_best, y_pred_best_rf, best_k_predictions, optimised_gnb_Y_predict]
# Calculate precision, recall, and F1-score
precision_scores = [precision_score(true, pred) for true, pred in zip(y_true, y_pre
recall_scores = [recall_score(true, pred) for true, pred in zip(y_true, y_pred)]
f1_scores = [f1_score(true, pred) for true, pred in zip(y_true, y_pred)]
# Set the width of the bars
bar_width = 0.25
# Set the positions of the bars on the x-axis
r1 = np.arange(len(models))
r2 = [x + bar_width for x in r1]
r3 = [x + bar_width for x in r2]
# Plot grouped bar chart with custom colors
plt.bar(r1, precision_scores, color='skyblue', width=bar_width, edgecolor='grey', l
plt.bar(r2, recall_scores, color='green', width=bar_width, edgecolor='grey', label=
plt.bar(r3, f1_scores, color='red', width=bar_width, edgecolor='grey', label='F1-Sc
# Add xticks on the middle of the group bars
plt.xlabel('Models', fontweight='bold')
plt.xticks([r + bar_width for r in range(len(models))], models, rotation=45)
# Add labels and title
plt.ylabel('Scores', fontweight='bold')
plt.title('Precision, Recall, and F1-Score by Model')
# Add legend
plt.legend()
# Show plot
plt.tight_layout()
plt.savefig('PRF.jpg')
plt.show()
```


Precision, Recall, and F1-Score by Model
Precision
Recall
F1-Score
Scores
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Logistic Regression
Random Forest
KNN
Naive Bayes
Models