```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import altair as alt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

df_customer_churn_data = pd.read_csv('Customer+Churn+Data.csv')

print(df_customer_churn_data.head().to_markdown(index=False,
numalign="left", stralign="left"))
```

```
| AccountID   | Churn   | Tenure   | City_Tier   | CC_Contacted_LY   |
Payment     | Gender   | Service_Score   | Account_user_count   |
account_segment   | CC_Agent_Score   | Marital_Status   |
rev_per_month   | Complain_ly   | rev_growth_yoy   |
coupon_used_for_payment   | Day_Since_CC_connect   | cashback   |
Login_device   |
|:------------|:--------|:---------|:------------|:------------------|
:------------|:---------|:----------------|:---------------------|:---
---------------|:-----------------|:-----------------|:---------------
-|:--------------|:-----------------|:-------------------------|:-----
-------------------|:-----------|:---------------|
| 20000       | 1       | 4        | 3           | 6                 |
Debit Card  | Female   | 3               | 3                    |
Super             | 2                | Single           | 9
| 1             | 11               | 1                          | 5
| 160        | Mobile          |
| 20001       | 1       | 0        | 1           | 8                 |
UPI         | Male     | 3               | 4                    |
Regular Plus      | 3                | Single           | 7
| 1             | 15               | 0                          | 0
| 121        | Mobile          |
| 20002       | 1       | 0        | 1           | 30                |
Debit Card  | Male     | 2               | 4                    |
Regular Plus      | 3                | Single           | 6
| 1             | 14               | 0                          | 3
| nan        | Mobile          |
| 20003       | 1       | 0        | 3           | 15                |
Debit Card  | Male     | 2               | 4                    |
Super             | 5                | Single           | 8
| 0             | 23               | 0                          | 3
| 134        | Mobile          |
| 20004       | 1       | 0        | 1           | 12                |
Credit Card | Male     | 2               | 3                    |
Regular Plus      | 5                | Single           | 3
| 0             | 11               | 1                          | 3
| 130        | Mobile          |
```

```
print(df_customer_churn_data.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11260 entries, 0 to 11259
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   AccountID                11260 non-null  int64  
 1   Churn                    11260 non-null  int64  
 2   Tenure                   11158 non-null  object 
 3   City_Tier                11148 non-null  float64
 4   CC_Contacted_LY          11158 non-null  float64
 5   Payment                  11151 non-null  object 
 6   Gender                   11152 non-null  object 
 7   Service_Score            11162 non-null  float64
 8   Account_user_count       11148 non-null  object 
 9   account_segment          11163 non-null  object 
 10  CC_Agent_Score           11144 non-null  float64
 11  Marital_Status           11048 non-null  object 
 12  rev_per_month            11158 non-null  object 
 13  Complain_ly              10903 non-null  float64
 14  rev_growth_yoy           11260 non-null  object 
 15  coupon_used_for_payment  11260 non-null  object 
 16  Day_Since_CC_connect     10903 non-null  object 
 17  cashback                 10789 non-null  object 
 18  Login_device             11039 non-null  object 
dtypes: float64(5), int64(2), object(12)
memory usage: 1.6+ MB
None

df_customer_churn_data.columns = 
df_customer_churn_data.columns.str.strip()

print(df_customer_churn_data.columns)

Index(['AccountID', 'Churn', 'Tenure', 'City_Tier', 'CC_Contacted_LY',
       'Payment', 'Gender', 'Service_Score', 'Account_user_count',
       'account_segment', 'CC_Agent_Score', 'Marital_Status', 
'rev_per_month',
       'Complain_ly', 'rev_growth_yoy', 'coupon_used_for_payment',
       'Day_Since_CC_connect', 'cashback', 'Login_device'],
      dtype='object')

df_customer_churn_data.columns = 
df_customer_churn_data.columns.str.strip()

# List of columns to check for non-numeric values
columns_to_check = ['Tenure', 'Account_user_count', 'rev_per_month', 
'rev_growth_yoy',
                    'coupon_used_for_payment', 'Day_Since_CC_connect', 
'cashback']
```

```
# Iterate through each column and filter non-numeric values
for column in columns_to_check:
    if column in df_customer_churn_data.columns:
        # Get all unique non-numeric values from the column (Corrected
indentation)
        non_numeric_values =
df_customer_churn_data[pd.to_numeric(df_customer_churn_data[column],
errors='coerce').isna()][column].unique()

        if len(non_numeric_values) > 0:  # Check if there are any non-
numeric values
            if len(non_numeric_values) > 20:
                # Sample 20 of them if there are too many unique
values
                print(f"Non-numeric values in column '{column}':
{np.random.choice(non_numeric_values, 20, replace=False)}")
            else:
                # Otherwise print all unique non-numeric values from
the column
                print(f"Non-numeric values in column '{column}':
{non_numeric_values}")
    else:
        print(f"Column '{column}' not found in DataFrame")
```

```
Non-numeric values in column 'Tenure': ['#' nan]
Non-numeric values in column 'Account_user_count': [nan '@']
Non-numeric values in column 'rev_per_month': ['+' nan]
Non-numeric values in column 'rev_growth_yoy': ['$']
Non-numeric values in column 'coupon_used_for_payment': ['#' '$' '*']
Non-numeric values in column 'Day_Since_CC_connect': [nan '$']
Non-numeric values in column 'cashback': [nan '$']
```

```
# List of columns to clean and convert to numeric
columns_to_convert = ['Tenure', 'Account_user_count', 'rev_per_month',
'rev_growth_yoy',
                      'coupon_used_for_payment',
'Day_Since_CC_connect', 'cashback']

# Characters to remove from the specified columns
characters_to_remove = ['#', '@', '+', '$', '*']

# Remove the specified characters from the columns
for column in columns_to_convert:
    for char in characters_to_remove:
        df_customer_churn_data[column] =
df_customer_churn_data[column].astype(str).str.replace(char, '',
regex=False)

    # Convert the column to numeric, setting failed conversions to NaN
```

```
    df_customer_churn_data[column] =
pd.to_numeric(df_customer_churn_data[column], errors='coerce')

# Count the number of missing values in each column
missing_value_counts = df_customer_churn_data.isnull().sum()

# Calculate the proportion of missing values in each column
total_rows = len(df_customer_churn_data)
missing_value_proportions = missing_value_counts / total_rows

# Calculate the proportion of missing values in each column
total_rows = len(df_customer_churn_data)
missing_value_proportions = missing_value_counts / total_rows

# Report the columns and their proportion of missing values
for column, proportion in missing_value_proportions.items():
    print(f"Column '{column}': {proportion:.2%}")
```

```
Column 'AccountID': 0.00%
Column 'Churn': 0.00%
Column 'Tenure': 1.94%
Column 'City_Tier': 0.99%
Column 'CC_Contacted_LY': 0.91%
Column 'Payment': 0.97%
Column 'Gender': 0.96%
Column 'Service_Score': 0.87%
Column 'Account_user_count': 3.94%
Column 'account_segment': 0.86%
Column 'CC_Agent_Score': 1.03%
Column 'Marital_Status': 1.88%
Column 'rev_per_month': 7.02%
Column 'Complain_ly': 3.17%
Column 'rev_growth_yoy': 0.03%
Column 'coupon_used_for_payment': 0.03%
Column 'Day_Since_CC_connect': 3.18%
Column 'cashback': 4.20%
Column 'Login_device': 1.96%
```

```
# Impute missing values
numerical_columns =
df_customer_churn_data.select_dtypes(include=np.number)
categorical_columns =
df_customer_churn_data.select_dtypes(include=object)

# Impute numerical columns with mean
num_imputer = SimpleImputer(strategy='mean')
df_customer_churn_data[numerical_columns.columns] =
num_imputer.fit_transform(numerical_columns)

# Impute categorical columns with mode
cat_imputer = SimpleImputer(strategy='most_frequent')
```

```python
df_customer_churn_data[categorical_columns.columns] =
cat_imputer.fit_transform(categorical_columns)

# Select numerical columns
numerical_columns =
df_customer_churn_data.select_dtypes(include='number')

# Calculate descriptive statistics for numerical columns
descriptive_stats = numerical_columns.describe()

# Display descriptive statistics
print("Descriptive statistics for numerical columns:\n")
print(descriptive_stats.to_markdown())
```

```
Descriptive statistics for numerical columns:

|       |   AccountID |        Churn |     Tenure |    City_Tier |
CC_Contacted_LY |   Service_Score |   Account_user_count |
CC_Agent_Score |   rev_per_month |   Complain_ly |   rev_growth_yoy |
coupon_used_for_payment |   Day_Since_CC_connect |   cashback |
|:------|------------:|-------------:|-----------:|------------:|----
-------------:|----------------:|---------------------:|-------------
----:|----------------:|-------------:|-----------------:|-----------
---------------:|---------------------:|-----------:|
| count |    11260    | 11260        | 11260      | 11260        |
11260       |    11260         |          11260        |      11260
|     11260       |  11260        |      11260        |
11260       |            11260       |  11260     |
| mean  |    25629.5  |     0.168384 |    11.0251 |     1.65393  |
17.8671  |        2.90253  |              3.69286 |          3.06649 |
6.36259 |      0.285334 |         16.1934  |                  1.79062
|                4.63319 |    196.235 |
| std   |     3250.63 |     0.374223 |    12.7545 |     0.910453 |
8.81308 |        0.722419 |             1.0026  |          1.37265 |
11.4837  |      0.444377 |          3.75722 |
1.96929 |                3.63838 |    174.864 |
| min   |    20000    |     0        |     0      |     1        |
4       |        0        |             1       |          1       |
1       |      0        |          4       |                  0
|                0       |      0     |
| 25%   |    22814.8  |     0        |     2      |     1        |
11       |        2        |              3       |          2        |
3       |      0        |         13       |                  1
|                2       |    148     |
| 50%   |    25629.5  |     0        |     9      |     1        |
16       |        3        |              4       |          3        |
5       |      0        |         15       |                  1
|                4       |    167     |
| 75%   |    28444.2  |     0        |    16      |     3        |
23       |        3        |              4       |          4        |
```

```
7        |      1         |          19       |                   2
|                7        |     197      |
| max   |    31259    |      1         |    99       |      3          |
132        |        5         |              6        |          5
|       140       |      1         |          28        |
16        |               47        |   1997      |
```

```python
# Select categorical columns (object type)
categorical_columns =
df_customer_churn_data.select_dtypes(include='object')

# Compute and display value counts for each categorical column
for column in categorical_columns:
    value_counts = categorical_columns[column].value_counts()
    print(f"\nValue counts for '{column}':\n")
    print(value_counts.to_markdown())
```

```
Value counts for 'Payment':

| Payment          |   count |
|:-----------------|--------:|
| Debit Card       |    4696 |
| Credit Card      |    3511 |
| E wallet         |    1217 |
| Cash on Delivery |    1014 |
| UPI              |     822 |

Value counts for 'Gender':

| Gender   |   count |
|:---------|--------:|
| Male     |    6436 |
| Female   |    4178 |
| M        |     376 |
| F        |     270 |

Value counts for 'account_segment':

| account_segment   |   count |
|:------------------|--------:|
| Super             |    4159 |
| Regular Plus      |    3862 |
| HNI               |    1639 |
| Super Plus        |     771 |
| Regular           |     520 |
| Regular +         |     262 |
| Super +           |      47 |

Value counts for 'Marital_Status':
```

```
| Marital_Status   |   count |
|:-----------------|--------:|
| Married          |    6072 |
| Single           |    3520 |
| Divorced         |    1668 |

Value counts for 'Login_device':

| Login_device   |   count |
|:---------------|--------:|
| Mobile         |    7703 |
| Computer       |    3018 |
| &&&&           |     539 |
```

Exploratory Data Analysis (EDA)

K-Means clustering

```
# Standardize the numerical columns
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numerical_columns)

# Apply K-Means clustering with the optimal 'k' (let's assume k = 3
based on the Elbow method)
kmeans = KMeans(n_clusters=3, random_state=42)
df_customer_churn_data['Cluster'] = kmeans.fit_predict(scaled_data)

# Analyze the clusters
for cluster_id in range(3):
    print(f"\nCluster {cluster_id} statistics:\n")

    # Filter data for the current cluster
    cluster_data =
df_customer_churn_data[df_customer_churn_data['Cluster'] ==
cluster_id]

    # Calculate and display descriptive statistics for numerical
columns in the cluster
    cluster_stats = cluster_data[numerical_columns.columns].describe()
    print(cluster_stats.to_markdown())

    # Calculate and display value counts for categorical columns in
the cluster
    for column in categorical_columns:
        value_counts = cluster_data[column].value_counts()
        print(f"\nValue counts for '{column}' in Cluster
{cluster_id}:\n")
        print(value_counts.to_markdown())
```

Cluster 0 statistics:

	AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Service_Score	Account_user_count	CC_Agent_Score	rev_per_month	Complain_ly	rev_growth_yoy	coupon_used_for_payment	Day_Since_CC_connect	cashback
count	1847	1847	1847	1847	1847	1847	1847	1847	1847	1847	1847	1847	1847	1847
mean	25545.6	1	3.24643	1.82519	19.3344	2.90711	3.91756	3.39437	6.94107	0.539241	16.1159	1.64808	3.31431	181.63
std	3247.21	0	6.08847	0.954147	8.8682	0.6991	0.999369	1.33497	13.0953	0.49235	3.88145	1.7577	3.0744	182.78
min	20000	1	0	1	4	2	1	1	1	0	4	0	0	110
25%	22788	1	0	1	12	2	3	3	3	0	13	1	1	136
50%	25632	1	1	1	17	3	4	3	5	1	15	1	2	153
75%	28359.5	1	3	3	26	3	5	5	7	1	18	2	5	184
max	31251	1	99	3	43	4	6	5	138	1	28	16	15	1997

Value counts for 'Payment' in Cluster 0:

Payment	count
Debit Card	708
Credit Card	483

E wallet	268
Cash on Delivery	245
UPI	143

Value counts for 'Gender' in Cluster 0:

Gender	count
Male	1101
Female	631
M	77
F	38

Value counts for 'account_segment' in Cluster 0:

account_segment	count
Regular Plus	1046
Super	422
HNI	240
Regular +	67
Super Plus	37
Regular	34
Super +	1

Value counts for 'Marital_Status' in Cluster 0:

Marital_Status	count
Single	929
Married	683
Divorced	235

Value counts for 'Login_device' in Cluster 0:

Login_device	count
Mobile	1174
Computer	588
&&&&	85

Cluster 1 statistics:

	AccountID	Churn	Tenure	City_Tier	CC_Contacted_LY	Service_Score	Account_user_count	CC_Agent_Score	rev_per_month	Complain_ly	rev_growth_yoy	coupon_used_for_payment	Day_Since_CC_connect	cashback

```
| count |     4583    | 4583          | 4583      | 4583         |
4583      |     4583        |          4583         |        4583
|      4583       |   4583        |       4583        |
4583        |             4583       |   4583     |
| mean  |    27357.8  |    0.00894611 |   13.5005 |    1.64536  |
19.1848 |        3.37392  |             4.17226  |          3.01718 |
7.04442 |      0.23106  |         17.0268  |                  2.61505
|                5.95503 |    217.461 |
| std   |     2824.22 |    0.0941701  |   13.4828 |    0.909909 |
9.1873 |         0.562648 |             0.811764 |          1.38465 |
12.2241  |      0.414101 |          3.82381 |
2.33066 |                3.89792 |    194.392 |
| min   |    20069    |    0          |    0      |    1         |
6      |        2        |             1        |          1        |
1       |      0         |          4        |                   0
|                0       |      0      |
| 25%   |    24661.5  |    0          |    6      |    1         |
13      |        3        |             4         |          2        |
3       |      0         |         14        |                   1
|                3       |    161      |
| 50%   |    28474    |    0          |   11      |    1         |
17      |        3        |             4         |          3        |
6       |      0         |         16        |                   2
|                5       |    183      |
| 75%   |    29877.5  |    0          |   18      |    3         |
24      |        4        |             5         |          4        |
7       |      0.285334 |         20        |                   3
|                8       |    222      |
| max   |    31258    |    1          |   99      |    3         |
132      |        5        |             6         |          5        |
140       |      1         |          28        |                   16
|               47       |   1992      |

Value counts for 'Payment' in Cluster 1:

| Payment          |   count |
|:-----------------|--------:|
| Debit Card       |    1949 |
| Credit Card      |    1444 |
| E wallet         |     498 |
| Cash on Delivery |     378 |
| UPI              |     314 |

Value counts for 'Gender' in Cluster 1:

| Gender   |   count |
|:---------|--------:|
| Male     |    2561 |
| Female   |    1783 |
| M        |     138 |
```

```
| F         |     101 |

Value counts for 'account_segment' in Cluster 1:

| account_segment   |   count |
|:------------------|--------:|
| Super             |    1810 |
| Regular Plus      |    1185 |
| HNI               |     787 |
| Super Plus        |     423 |
| Regular           |     271 |
| Regular +         |      86 |
| Super +           |      21 |

Value counts for 'Marital_Status' in Cluster 1:

| Marital_Status   |   count |
|:-----------------|--------:|
| Married          |    2725 |
| Single           |    1194 |
| Divorced         |     664 |

Value counts for 'Login_device' in Cluster 1:

| Login_device   |   count |
|:---------------|--------:|
| Mobile         |    3169 |
| Computer       |    1199 |
| &&&&           |     215 |

Cluster 2 statistics:

|       |   AccountID |         Churn |    Tenure |   City_Tier |
CC_Contacted_LY |   Service_Score |   Account_user_count |
CC_Agent_Score |   rev_per_month |   Complain_ly |   rev_growth_yoy |
coupon_used_for_payment |   Day_Since_CC_connect |   cashback |
|:------|------------:|--------------:|----------:|------------:|-----
------------:|----------------:|---------------------:|--------------
---:|----------------:|--------------:|-----------------:|------------
--------------:|---------------------:|-----------:|
| count |     4830    | 4830          | 4830      | 4830        |
4830       |     4830        |          4830        |       4830
|      4830       |   4830        |       4830       |
4830       |             4830       |   4830     |
| mean  |    24021.6  |    0.00165631 |   11.6508 |    1.59657  |
16.0557  |        2.45348  |             3.15205  |          2.98791 |
5.49443 |      0.239739 |        15.4322  |                  1.06285
|                  3.88328 |    181.681 |
| std   |     2759.31 |    0.0406683  |   12.778  |    0.885711 |
8.07372 |        0.563034 |             0.898154 |          1.35755 |
```

```
9.94024 |      0.418931 |          3.46984 |                     1.23457
|                 3.15412 |    148.137 |
| min   |    20017    |    0           |    0      |    1         |
5       |        0        |             1         |          1       |
1       |      0        |          4         |                     0
|                 0       |      0     |
| 25%   |    21512.2  |    0           |    4      |    1         |
10       |        2        |             3         |          2       |
2       |      0        |         13         |                     0
|                 2       |    138     |
| 50%   |    23385    |    0           |    9      |    1         |
14       |        2        |             3         |          3       |
4       |      0        |         14         |                     1
|                 3       |    156     |
| 75%   |    26636.5  |    0           |   16      |    3         |
20       |        3        |             4         |          4       |
6.36259 |      0.285334 |         18         |                     1
|                 7       |    194     |
| max   |    31259    |    1           |   99      |    3         |
126       |        4        |             5         |          5
|       140       |      1         |         28        |
11       |                30       |   1985      |

Value counts for 'Payment' in Cluster 2:

| Payment          |   count |
|:-----------------|--------:|
| Debit Card       |    2039 |
| Credit Card      |    1584 |
| E wallet         |     451 |
| Cash on Delivery |     391 |
| UPI              |     365 |

Value counts for 'Gender' in Cluster 2:

| Gender   |   count |
|:---------|--------:|
| Male     |    2774 |
| Female   |    1764 |
| M        |     161 |
| F        |     131 |

Value counts for 'account_segment' in Cluster 2:

| account_segment   |   count |
|:------------------|--------:|
| Super             |    1927 |
| Regular Plus      |    1631 |
| HNI               |     612 |
| Super Plus        |     311 |
```

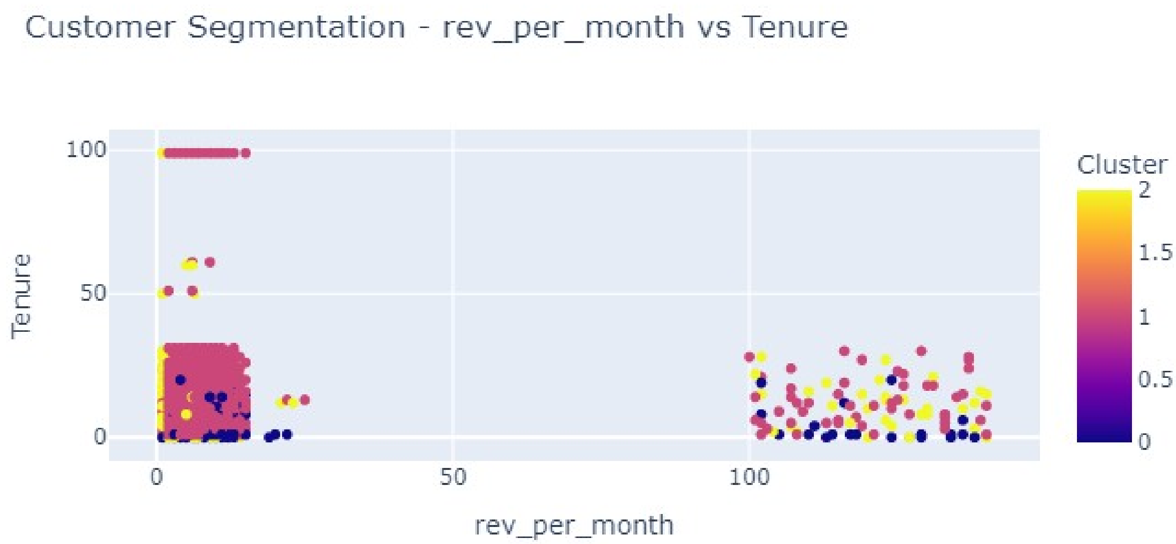
```
| Regular              |     215 |
| Regular +            |     109 |
| Super +              |      25 |

Value counts for 'Marital_Status' in Cluster 2:

| Marital_Status    |   count |
|:------------------|--------:|
| Married           |    2664 |
| Single            |    1397 |
| Divorced          |     769 |

Value counts for 'Login_device' in Cluster 2:

| Login_device    |   count |
|:----------------|--------:|
| Mobile          |    3360 |
| Computer        |    1231 |
| &&&&            |     239 |

# Visualize the clusters (example using two numerical columns, adjust
as needed)
chart = alt.Chart(df_customer_churn_data).mark_circle().encode(
    x='rev_per_month',
    y='Tenure',
    color='Cluster:N',
    tooltip = ['rev_per_month', 'Tenure', 'Cluster']
).properties(
    title='Customer Segmentation'
).interactive()

# Save the plot
chart.save('customer_segmentation_plot.json')
```

Can not use Altair to visualize the chart in notebook as it has more than 5000 rows.

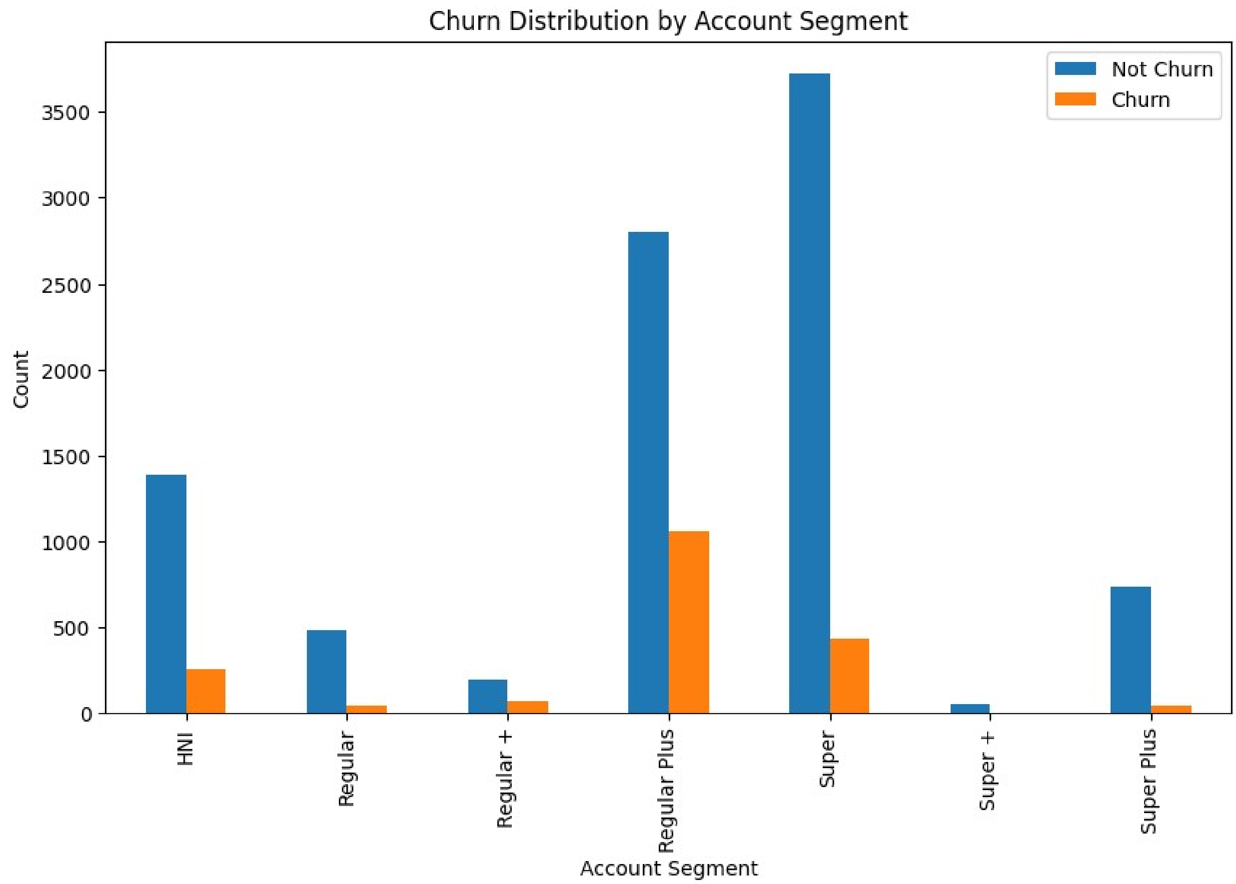
Now i will Try to visualize the Chart by **Importing Plotly!**

```
import plotly.express as px
import plotly.graph_objects as go

# Create the scatter plot
fig = px.scatter(df_customer_churn_data,
                 x='rev_per_month',
                 y='Tenure',
                 color='Cluster',
                 title='Customer Segmentation - rev_per_month vs
Tenure')
```

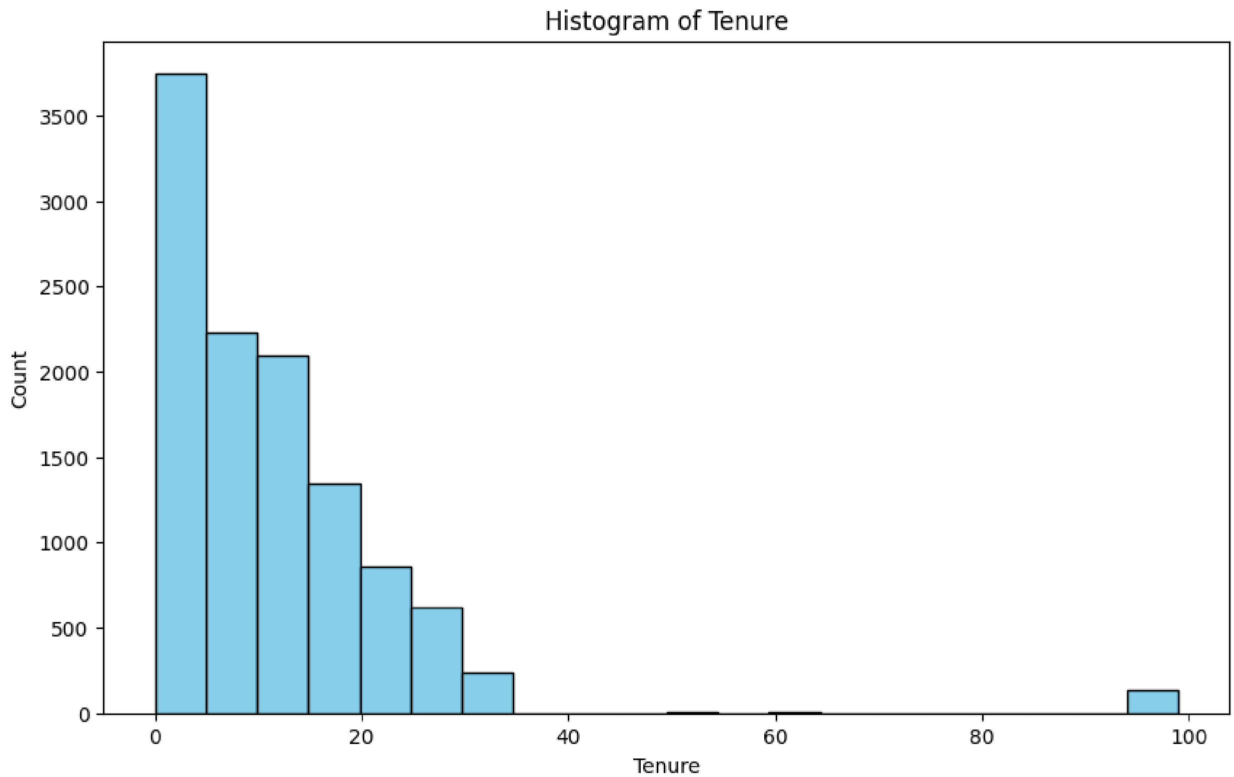
```
# Display the plot
fig.show()
```



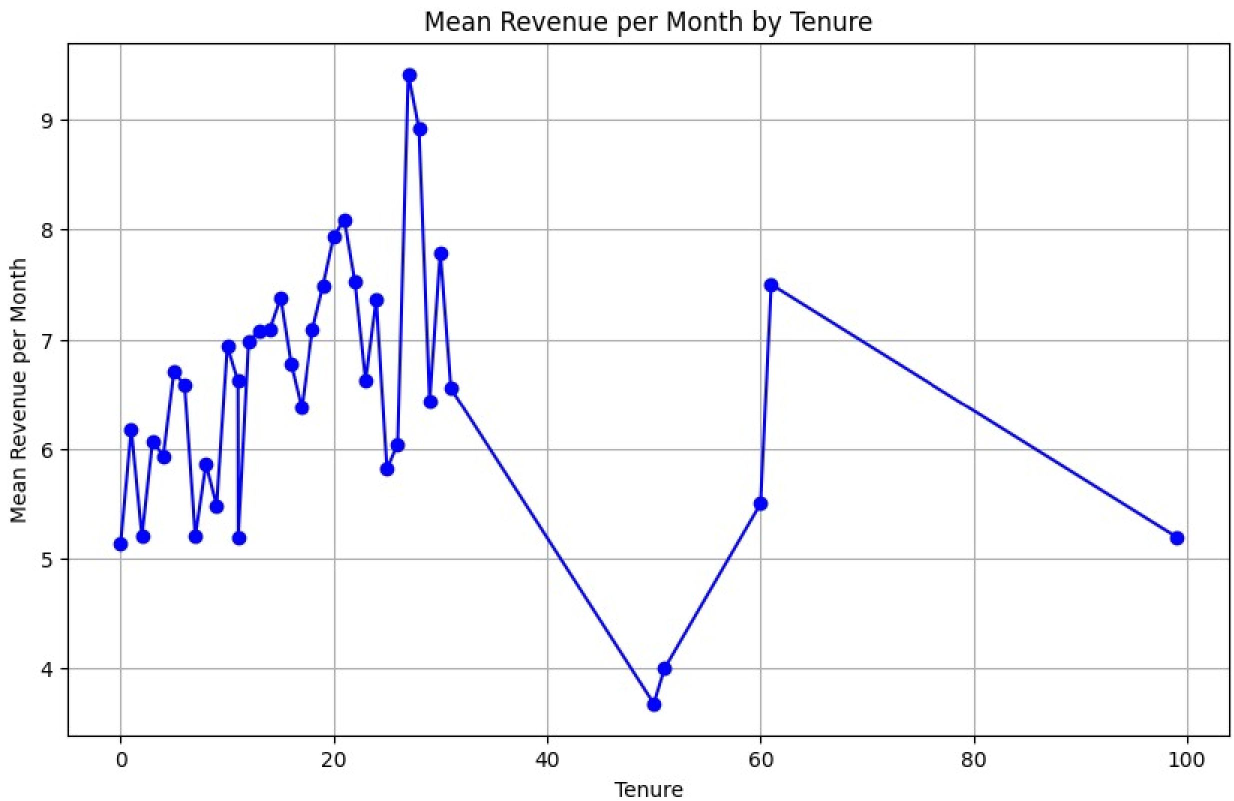

```
import matplotlib.pyplot as plt

# 1. Bar Chart: Churn Distribution by Account Segment

# Count the occurrences of each `account_segment` for both churned and
non-churned customers
churn_counts = df_customer_churn_data.groupby(['account_segment',
'Churn']).size().reset_index(name='Count')

# Pivot the data for plotting
churn_pivot = churn_counts.pivot(index='account_segment',
columns='Churn', values='Count')

# Plot the bar chart
churn_pivot.plot(kind='bar', figsize=(10, 6))
plt.title('Churn Distribution by Account Segment')
plt.xlabel('Account Segment')
plt.ylabel('Count')
plt.legend(['Not Churn', 'Churn'])
plt.show()
```




```
# 2. Histogram: Histogram of Tenure

# Plot the histogram
plt.figure(figsize=(10, 6))
plt.hist(df_customer_churn_data['Tenure'], bins=20, color='skyblue',
edgecolor='black')
plt.title('Histogram of Tenure')
plt.xlabel('Tenure')
plt.ylabel('Count')
plt.show()
```

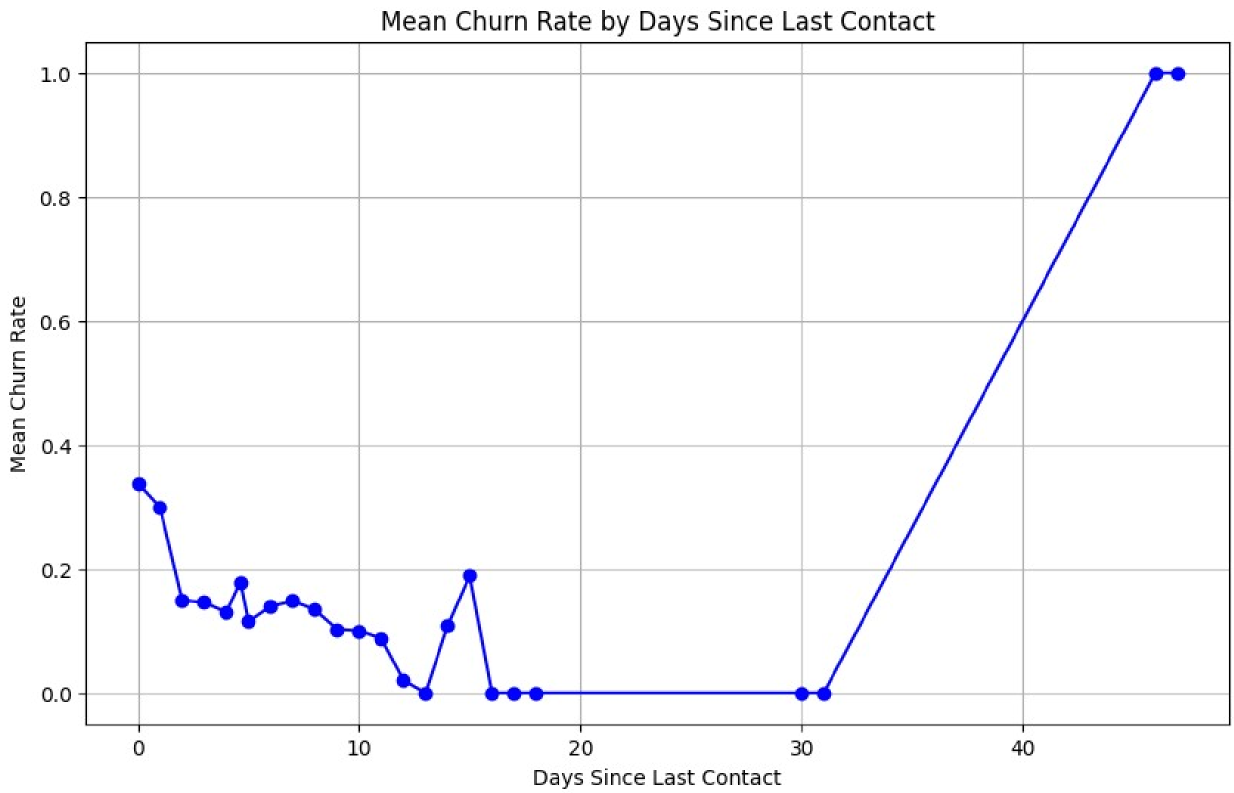



```
# 3. Line Chart: Mean Revenue per Month by Tenure

# Group the data by `Tenure` and calculate the mean of `rev_per_month`
tenure_revenue = df_customer_churn_data.groupby('Tenure')
['rev_per_month'].mean().reset_index()

# Plot the line chart
plt.figure(figsize=(10, 6))
plt.plot(tenure_revenue['Tenure'], tenure_revenue['rev_per_month'],
marker='o', linestyle='-', color='b')
plt.title('Mean Revenue per Month by Tenure')
plt.xlabel('Tenure')
plt.ylabel('Mean Revenue per Month')
plt.grid(True)
plt.show()
```

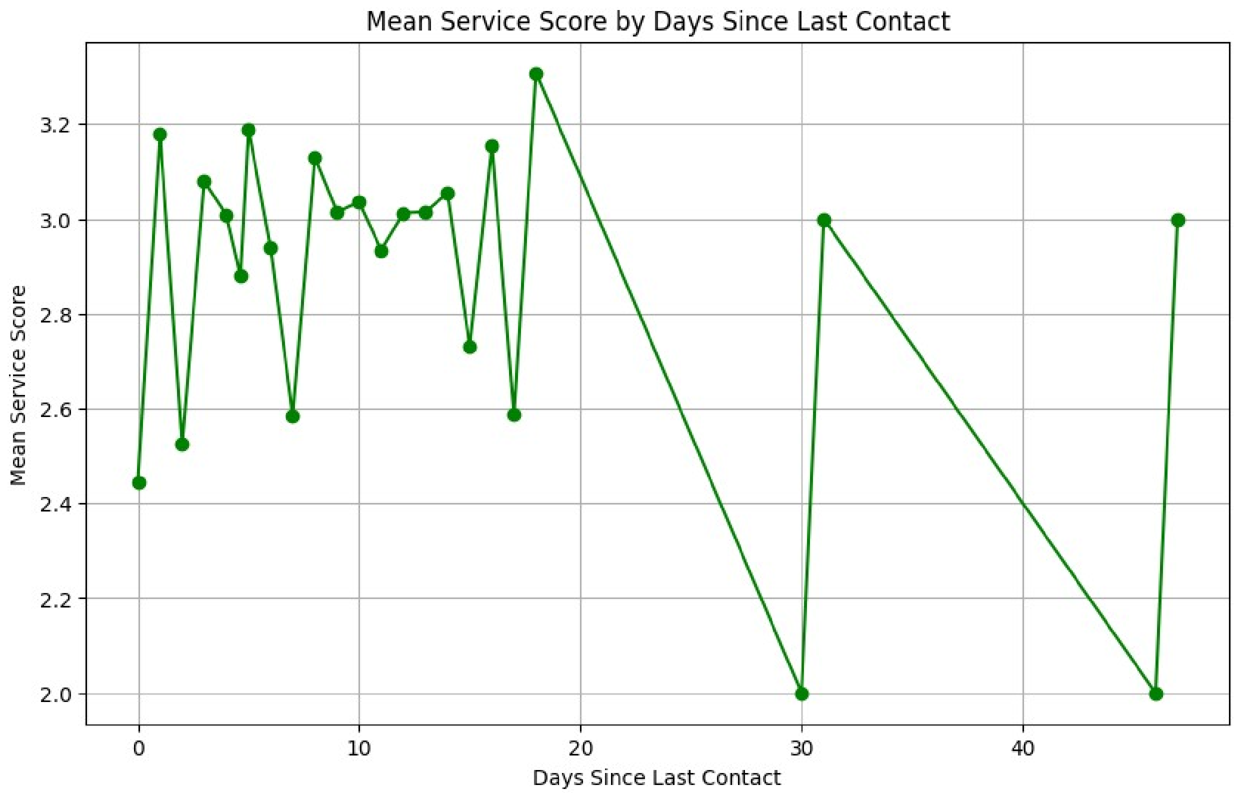


Customer Journey Maps

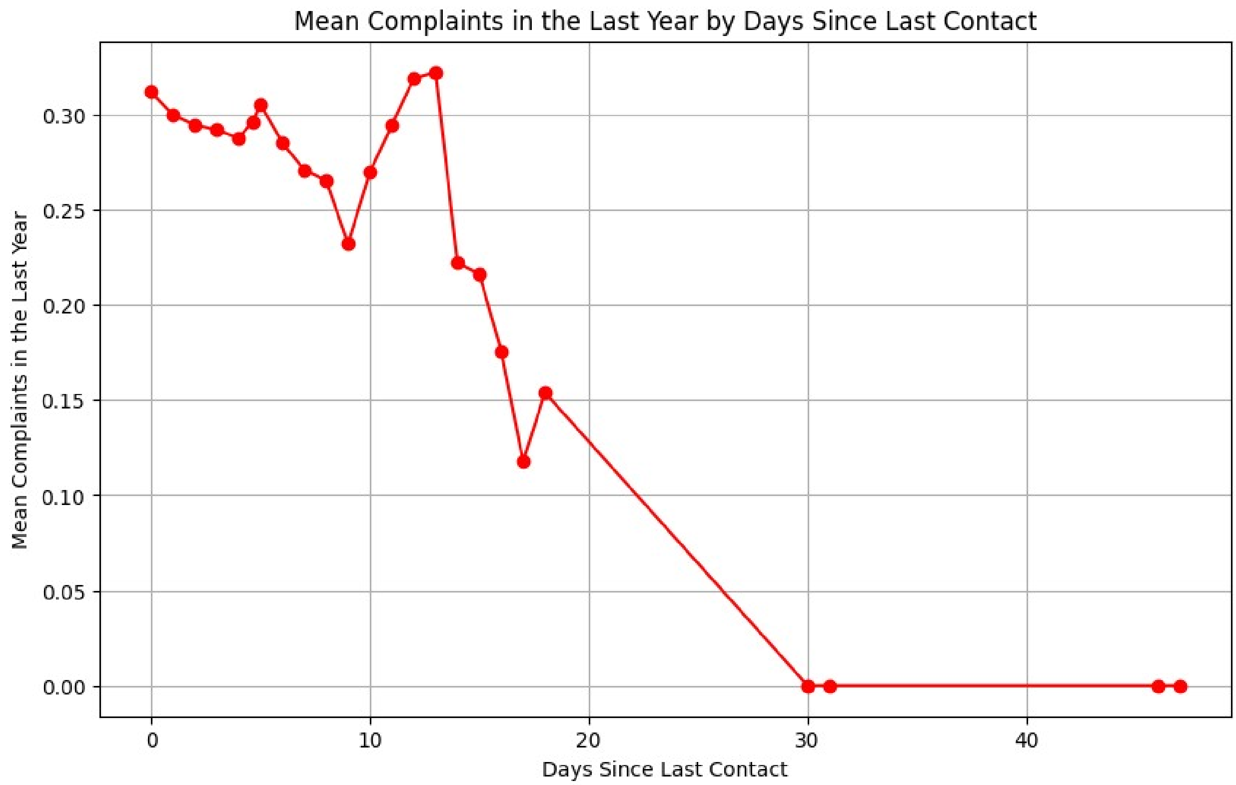
```
# 1. Group the data by `Day_Since_CC_connect` and calculate the mean
of `Churn`, `Service_Score`, and `Complain_ly`. Reset the index and
store the result in `journey_data`.
journey_data = df_customer_churn_data.groupby('Day_Since_CC_connect')
[['Churn', 'Service_Score', 'Complain_ly']].mean().reset_index()

# 2. Create a line chart using matplotlib with `Day_Since_CC_connect`
on the x-axis and the mean of `Churn` on the y-axis. Add appropriate
title and labels.
plt.figure(figsize=(10, 6))
plt.plot(journey_data['Day_Since_CC_connect'], journey_data['Churn'],
marker='o', linestyle='-', color='b')
plt.title('Mean Churn Rate by Days Since Last Contact')
plt.xlabel('Days Since Last Contact')
plt.ylabel('Mean Churn Rate')
plt.grid(True)
```




```python
# 3. Create another line chart with `Day_Since_CC_connect` on the x-axis and the mean of `Service_Score` on the y-axis. Add appropriate title and labels.
plt.figure(figsize=(10, 6))
plt.plot(journey_data['Day_Since_CC_connect'],
journey_data['Service_Score'], marker='o', linestyle='-', color='g')
plt.title('Mean Service Score by Days Since Last Contact')
plt.xlabel('Days Since Last Contact')
plt.ylabel('Mean Service Score')
plt.grid(True)
```




```python
# 4. Create a third line chart with `Day_Since_CC_connect` on the x-axis and the mean of `Complain_ly` on the y-axis. Add appropriate title and labels.
plt.figure(figsize=(10, 6))
plt.plot(journey_data['Day_Since_CC_connect'], journey_data['Complain_ly'], marker='o', linestyle='-', color='r')
plt.title('Mean Complaints in the Last Year by Days Since Last Contact')
plt.xlabel('Days Since Last Contact')
plt.ylabel('Mean Complaints in the Last Year')
plt.grid(True)
```



We will proceed with model training using the Random Forest Classifier.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder
```

Preparing Data for Model development!

```
# Encode categorical variables
le = LabelEncoder()
for col in
df_customer_churn_data.select_dtypes(include='object').columns:
    df_customer_churn_data[col] =
le.fit_transform(df_customer_churn_data[col])

# Split data into features (X) and target (y)
X = df_customer_churn_data.drop('Churn', axis=1)
y = df_customer_churn_data['Churn']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
# Print the shapes of the resulting datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (9008, 19)
X_test shape: (2252, 19)
y_train shape: (9008,)
y_test shape: (2252,)
```

Initialize and train the Random Forest model!

```
# 1. Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# 2. Train the model
rf_classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=42)
```

```
# 3. Predict on the training set
y_train_pred = rf_classifier.predict(X_train)

# 4. Print the model's accuracy on the training set
print("Training Accuracy:", accuracy_score(y_train, y_train_pred))
```

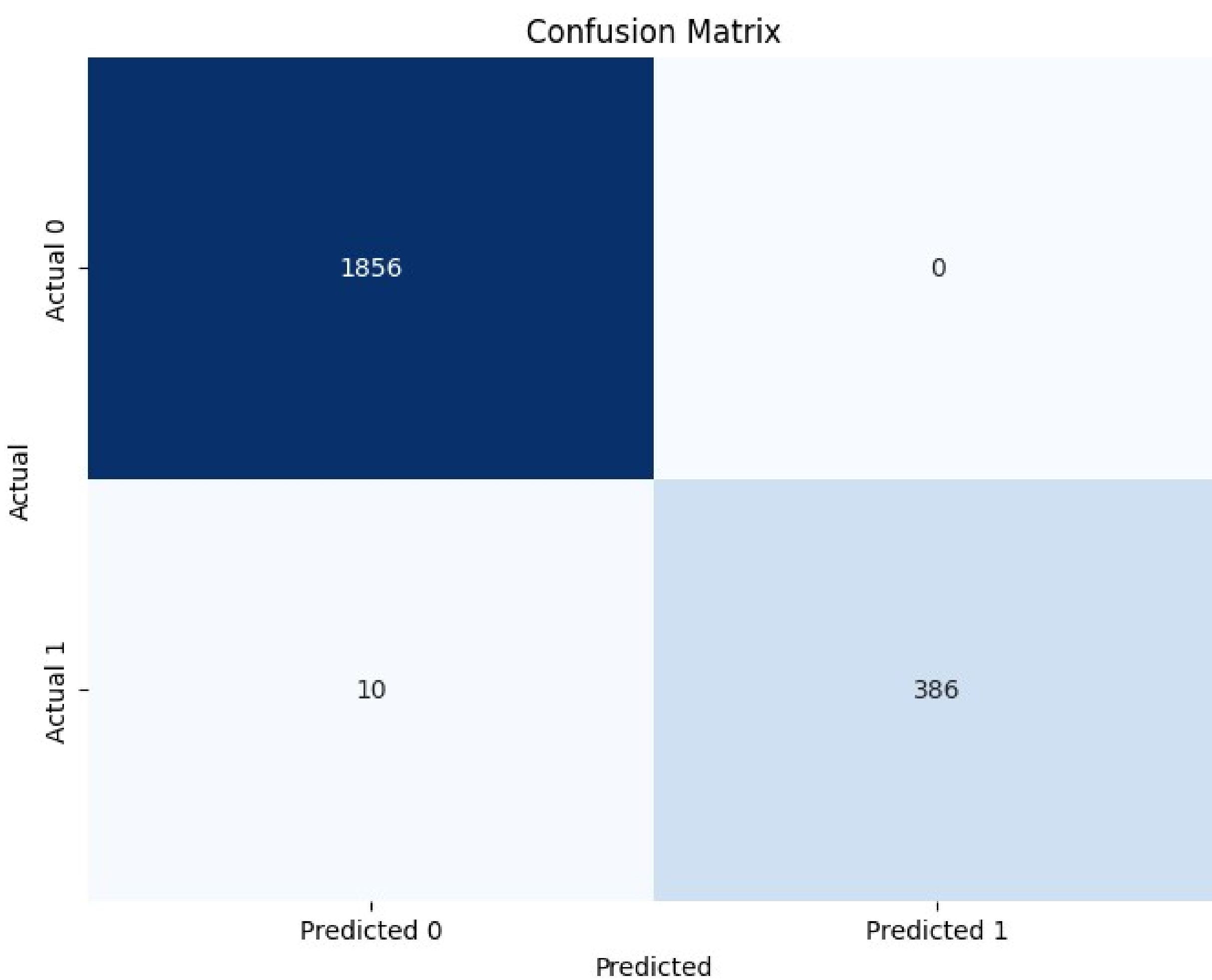
```
Training Accuracy: 1.0
```

Make predictions on the test set

```
y_pred = rf_classifier.predict(X_test)

#Evaluate the model
test_accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

#Print the results
print("Test Accuracy:", test_accuracy)
print("Classification Report:\n", classification_rep)
print("Confusion Matrix:\n", conf_matrix)
```

```
Test Accuracy: 0.9955595026642984
Classification Report:
               precision    recall  f1-score   support

         0.0       0.99      1.00      1.00      1856
         1.0       1.00      0.97      0.99       396

    accuracy                           1.00      2252
```

```
   macro avg       1.00      0.99      0.99      2252
weighted avg       1.00      1.00      1.00      2252

Confusion Matrix:
 [[1856    0]
 [  10  386]]

#Create a dataframe for the confusion matrix
cm_df = pd.DataFrame(conf_matrix, index=['Actual 0', 'Actual 1'],
columns=['Predicted 0', 'Predicted 1'])

#Create a heatmap using the confusion matrix dataframe
plt.figure(figsize=(8, 6))
sns.heatmap(cm_df, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



```
#Extract metrics from classification report and create a dataframe
report_data = classification_report(y_test, y_pred, output_dict=True)
metrics_df = pd.DataFrame({
    'Metric': ['Precision', 'Recall', 'F1-score'],
    'Class 0': [report_data['0.0']['precision'], report_data['0.0']
['recall'], report_data['0.0']['f1-score']],
    'Class 1': [report_data['1.0']['precision'], report_data['1.0']
['recall'], report_data['1.0']['f1-score']]
})

#Create a grouped bar chart
metrics_df.plot(x='Metric', kind='bar', figsize=(10, 6))
plt.title('Classification Metrics by Class')
plt.ylabel('Score')
plt.xticks(rotation=0)
plt.legend(title='Class')
plt.show()
```