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

In [2]:
```
dic_allsheets =pd.read_excel("/Users/zheguan/CWR_fig/Customer_Churn_Data_Large.xlsx",sheet_name= None)
```

1. Explore data set structure

In [3]:
```
dic_allsheets.keys()
```

Out[3]:
```
dict_keys(['Customer_Demographics', 'Transaction_History', 'Customer_Service', 'Online_Activity', 'Churn_Status
'])
```

In [4]:
```
dic_allsheets['Customer_Demographics'].columns
```

Out[4]:
```
Index(['CustomerID', 'Age', 'Gender', 'MaritalStatus', 'IncomeLevel'], dtype='object')
```

In [3]:
```
df_sheet1 = dic_allsheets['Customer_Demographics']
df_sheet2 = dic_allsheets['Transaction_History']
df_sheet3 = dic_allsheets['Customer_Service']
df_sheet4 = dic_allsheets['Online_Activity']
df_sheet5 = dic_allsheets['Churn_Status']
```

Descriptive statistics

In [11]:
```
df_sheet1.describe()
```

Out[11]:

	CustomerID	Age
count	1000.000000	1000.000000
mean	500.500000	43.267000
std	288.819436	15.242311
min	1.000000	18.000000
25%	250.750000	30.000000
50%	500.500000	43.000000
75%	750.250000	56.000000
max	1000.000000	69.000000

In [12]:
```
#we can find that the primary key is not CustomerID
df_sheet2.describe()
```

Out[12]:

	CustomerID	TransactionID	TransactionDate	AmountSpent
count	5054.000000	5054.000000	5054	5054.000000
mean	501.424218	5510.538979	2022-07-01 19:25:37.158686208	250.707351
min	1.000000	1000.000000	2022-01-01 00:00:00	5.180000
25%	251.000000	3242.000000	2022-04-03 00:00:00	127.105000
50%	506.000000	5530.000000	2022-07-01 00:00:00	250.525000
75%	749.000000	7680.750000	2022-09-29 00:00:00	373.412500
max	1000.000000	9997.000000	2022-12-31 00:00:00	499.860000
std	285.172780	2582.088012	NaN	142.250838

In [13]:
```
#Similar to sheet2, Primary key here is interactionID
df_sheet3.describe()
```

Out[13]:

	CustomerID	InteractionID	InteractionDate
count	1002.000000	1002.000000	1002
mean	485.209581	5952.887226	2022-07-02 19:28:22.994011904
min	1.000000	2015.000000	2022-01-01 00:00:00
25%	238.250000	3991.500000	2022-04-07 00:00:00
50%	474.500000	5911.500000	2022-07-02 12:00:00
75%	735.750000	7908.250000	2022-09-30 00:00:00
max	995.000000	9997.000000	2022-12-30 00:00:00
std	287.030259	2305.819681	NaN

In [15]:
```
df_sheet4.describe()
```

Out[15]:

	CustomerID	LastLoginDate	LoginFrequency
count	1000.000000	1000	1000.000000
mean	500.500000	2023-07-05 21:28:48	25.912000
min	1.000000	2023-01-01 00:00:00	1.000000
25%	250.750000	2023-04-08 00:00:00	13.750000
50%	500.500000	2023-07-10 12:00:00	27.000000
75%	750.250000	2023-10-01 06:00:00	38.000000
max	1000.000000	2023-12-31 00:00:00	49.000000
std	288.819436	NaN	14.055953

In [14]:
```
#the ratio of churn to non-churn is about 1/5
df_sheet5.describe()
```

Out[14]:

	CustomerID	ChurnStatus
count	1000.000000	1000.000000
mean	500.500000	0.204000
std	288.819436	0.403171
min	1.000000	0.000000
25%	250.750000	0.000000
50%	500.500000	0.000000
75%	750.250000	0.000000
max	1000.000000	1.000000

2.Data Cleaning and transforming

check missing values

In [16]:
```
print (df_sheet1.isna().sum(),df_sheet2.isna().sum(),df_sheet3.isna().sum(),df_sheet4.isna().sum(),df_sheet5.isn
```

```
CustomerID       0
Age              0
Gender           0
MaritalStatus    0
IncomeLevel      0
dtype: int64 CustomerID         0
TransactionID      0
TransactionDate    0
AmountSpent        0
ProductCategory    0
dtype: int64 CustomerID          0
InteractionID       0
InteractionDate     0
InteractionType     0
ResolutionStatus    0
dtype: int64 CustomerID        0
LastLoginDate     0
LoginFrequency    0
ServiceUsage      0
dtype: int64 CustomerID     0
ChurnStatus    0
dtype: int64
```

check duplicated records

In [17]:
```
print(df_sheet1.duplicated().sum(),df_sheet2.duplicated().sum(),df_sheet3.duplicated().sum(),df_sheet4.duplicate
```

```
0 0 0 0 0
```

explore table 1, encoding related categorical variables

here we apply *one-hot encoding* for MaritalStatus and for other label varibles *label encoding* is used

In [18]:
```
#check table 1 variable
print(df_sheet1.dtypes)
print(df_sheet1["Gender"].unique())
print(df_sheet1["MaritalStatus"].unique())
print(df_sheet1["IncomeLevel"].unique())
```

```
CustomerID        int64
Age               int64
Gender           object
MaritalStatus    object
IncomeLevel      object
dtype: object
['M' 'F']
['Single' 'Married' 'Widowed' 'Divorced']
['Low' 'Medium' 'High']
```

In [20]:
```python
from sklearn.preprocessing import LabelEncoder

#There are two methods to transform object variable, One-hot encoding and Label encoding
#we choose Label encoding for Gender and IncomeLevel, as for MaritalStatus, we apply One-hot encoding
le = LabelEncoder()

df_sheet1_encoded = df_sheet1

df_sheet1_encoded['Gender'] = le.fit_transform(df_sheet1_encoded['Gender'])
df_sheet1_encoded['IncomeLevel'] = le.fit_transform(df_sheet1_encoded['IncomeLevel'])

df_sheet1_encoded = pd.get_dummies(df_sheet1_encoded, columns=['MaritalStatus'], prefix='MaritalStatus')
df_sheet1_encoded[df_sheet1_encoded.select_dtypes(include="bool").columns] = df_sheet1_encoded.select_dtypes(in
```

In [21]:
```python
df_sheet1_encoded
```

Out[21]:

	CustomerID	Age	Gender	IncomeLevel	MaritalStatus_Divorced	MaritalStatus_Married	MaritalStatus_Single	MaritalStatus_Wid
0	1	62	1	1	0	0	1	
1	2	65	1	1	0	1	0	
2	3	18	1	1	0	0	1	
3	4	21	1	1	0	0	0	
4	5	21	1	2	1	0	0	
...	...	...	...	...	...	...	...	
995	996	54	0	1	0	0	1	
996	997	19	1	0	0	0	0	
997	998	47	1	1	0	1	0	
998	999	23	1	0	0	0	0	
999	1000	34	1	1	0	0	0	

1000 rows × 8 columns

explore table 2, encoding related categorical variables

In [23]:
```python
print(df_sheet2.dtypes)
print(df_sheet2["ProductCategory"].unique())
```

```
CustomerID                  int64
TransactionID               int64
TransactionDate    datetime64[ns]
AmountSpent               float64
ProductCategory            object
dtype: object
['Electronics' 'Clothing' 'Furniture' 'Groceries' 'Books']
```

In [24]:
```python
# For transaction records, we can summarize statistical variables for each customer.
# Considering that we only have 1,000 customers in the dataset, we can count by year.
# If we had more records, we could use a more precise period for counting.

df_sheet2_product = df_sheet2.groupby(["CustomerID","ProductCategory"]).size().unstack()
df_sheet2_product = df_sheet2_product.fillna(0)
#rename columns
df_sheet2_product.columns = ["Count_{0}".format(col) for col in df_sheet2_product]

df_sheet2_cost = df_sheet2.groupby(["CustomerID","ProductCategory"])['AmountSpent'].sum().unstack()
df_sheet2_cost = df_sheet2_cost.fillna(0)
df_sheet2_cost.columns = ["Cost_{0}".format(col) for col in df_sheet2_cost]

merged_sheet2 = pd.merge(df_sheet2_product,df_sheet2_cost, on= "CustomerID", how = "inner")
merged_sheet2
```

Out[24]:

CustomerID	Count_Books	Count_Clothing	Count_Electronics	Count_Furniture	Count_Groceries	Cost_Books	Cost_Clothing	Cos
1	0.0	0.0	1.0	0.0	0.0	0.00	0.00	
2	0.0	2.0	3.0	1.0	1.0	0.00	452.33	
3	1.0	1.0	0.0	2.0	2.0	241.06	51.07	
4	0.0	1.0	2.0	1.0	1.0	0.00	44.22	
5	0.0	0.0	3.0	2.0	3.0	0.00	0.00	
...	...	...	...	...	...	...	...	
996	1.0	0.0	0.0	0.0	0.0	227.25	0.00	
997	0.0	0.0	1.0	1.0	0.0	0.00	0.00	
998	1.0	0.0	0.0	0.0	0.0	252.15	0.00	
999	1.0	0.0	2.0	4.0	2.0	80.28	0.00	
1000	2.0	0.0	0.0	2.0	2.0	491.71	0.00	

1000 rows × 10 columns

Similarly, we can reshape the sheet3

In [25]:
```
print(df_sheet3.dtypes)
print(df_sheet3["InteractionType"].unique())
print(df_sheet3["ResolutionStatus"].unique())
```

```
CustomerID                  int64
InteractionID               int64
InteractionDate     datetime64[ns]
InteractionType             object
ResolutionStatus            object
dtype: object
['Inquiry' 'Feedback' 'Complaint']
['Resolved' 'Unresolved']
```

In [26]:
```
df_sheet3_CountInteraction = df_sheet3.groupby(["CustomerID","InteractionType"]).size().unstack()
df_sheet3_CountInteraction = df_sheet3_CountInteraction.fillna(0)
df_sheet3_CountInteraction.columns = ["Count_{0}".format(col) for col in df_sheet3_CountInteraction.columns]

df_sheet3_CountResolution = df_sheet3.groupby(["CustomerID","ResolutionStatus"]).size().unstack()
df_sheet3_CountResolution =  df_sheet3_CountResolution.fillna(0)
df_sheet3_CountResolution.columns = ["Count_{0}".format(col) for col in df_sheet3_CountResolution.columns]

merged_sheet3 = pd.merge(df_sheet3_CountInteraction,df_sheet3_CountResolution, on = "CustomerID", how = "inner"
merged_sheet3

#notice the number of columns that indicates not all customer have interaction,
#we need to use left/outter join with other sheets in case of losing data
```

Out[26]:

CustomerID	Count_Complaint	Count_Feedback	Count_Inquiry	Count_Resolved	Count_Unresolved
1	0.0	0.0	1.0	1.0	0.0
2	0.0	0.0	1.0	1.0	0.0
3	0.0	0.0	1.0	1.0	0.0
4	0.0	0.0	2.0	1.0	1.0
6	0.0	1.0	0.0	1.0	0.0
...	...	...	...	...	...
989	2.0	0.0	0.0	0.0	2.0
990	1.0	1.0	0.0	1.0	1.0
992	0.0	1.0	0.0	0.0	1.0
994	2.0	0.0	0.0	0.0	2.0
995	0.0	0.0	1.0	1.0	0.0

668 rows × 5 columns

Explore table 4

We can convert the datetime 'lastlogindate' to an integer representing the number of days since the last login.

For 'ServiceUsage', we can apply the one-hot encoding method.

In [28]:
```
print(df_sheet4.dtypes)
print(df_sheet4["ServiceUsage"].unique())
```

```
CustomerID                int64
LastLoginDate     datetime64[ns]
LoginFrequency             int64
ServiceUsage              object
dtype: object
['Mobile App' 'Website' 'Online Banking']
```

In [29]:
```
df_sheet4_encoded = df_sheet4
df_sheet4_encoded['DaysSinceLastLogin'] = (pd.Timestamp.now() - df_sheet4_encoded['LastLoginDate']).dt.days

df_sheet4_encoded = pd.get_dummies(df_sheet4_encoded,columns=["ServiceUsage"],prefix="ServiceUsage")
df_sheet4_encoded[df_sheet4_encoded.select_dtypes("bool").columns] = df_sheet4_encoded.select_dtypes("bool").as

#last login date is not neccessary here
df_sheet4_encoded = df_sheet4_encoded.drop(columns="LastLoginDate",axis=1)
```

In [30]:
```
df_sheet4_encoded
```

Out[30]:

	CustomerID	LoginFrequency	DaysSinceLastLogin	ServiceUsage_Mobile App	ServiceUsage_Online Banking	ServiceUsage_Website
0	1	34	383	1	0	0
1	2	5	338	0	0	1
2	3	3	358	0	0	1
3	4	2	440	0	0	1
4	5	41	377	0	0	1
...	...	...	...	...	...	...
995	996	38	648	1	0	0
996	997	5	586	1	0	0
997	998	47	486	0	0	1
998	999	23	669	0	0	1
999	1000	22	452	1	0	0

1000 rows × 6 columns

merge all tables where the "left join" is used in case missing data

In [31]:
```
merged_df = pd.merge(df_sheet1_encoded,merged_sheet2,on = "CustomerID", how = "left")
merged_df = pd.merge(merged_df,merged_sheet3,on = "CustomerID", how = "left")
merged_df = pd.merge(merged_df,df_sheet4_encoded, on = "CustomerID", how = "left")
merged_df = pd.merge(merged_df,df_sheet5, on = "CustomerID", how = "left")
merged_df = merged_df.fillna(0)
```

In [32]:
```
merged_df
```

Out[32]:

	CustomerID	Age	Gender	IncomeLevel	MaritalStatus_Divorced	MaritalStatus_Married	MaritalStatus_Single	MaritalStatus_Wid
0	1	62	1	1	0	0	1	
1	2	65	1	1	0	1	0	
2	3	18	1	1	0	0	1	
3	4	21	1	1	0	0	0	
4	5	21	1	2	1	0	0	
...	...	...	...	...	...	...	...	
995	996	54	0	1	0	0	1	
996	997	19	1	0	0	0	0	
997	998	47	1	1	0	1	0	
998	999	23	1	0	0	0	0	
999	1000	34	1	1	0	0	0	

1000 rows × 29 columns

In [33]:
```
#Now, we can use one table to do more exploretory analysis
```

```
#before that, check again the data type for each columns
merged_df.dtypes
```

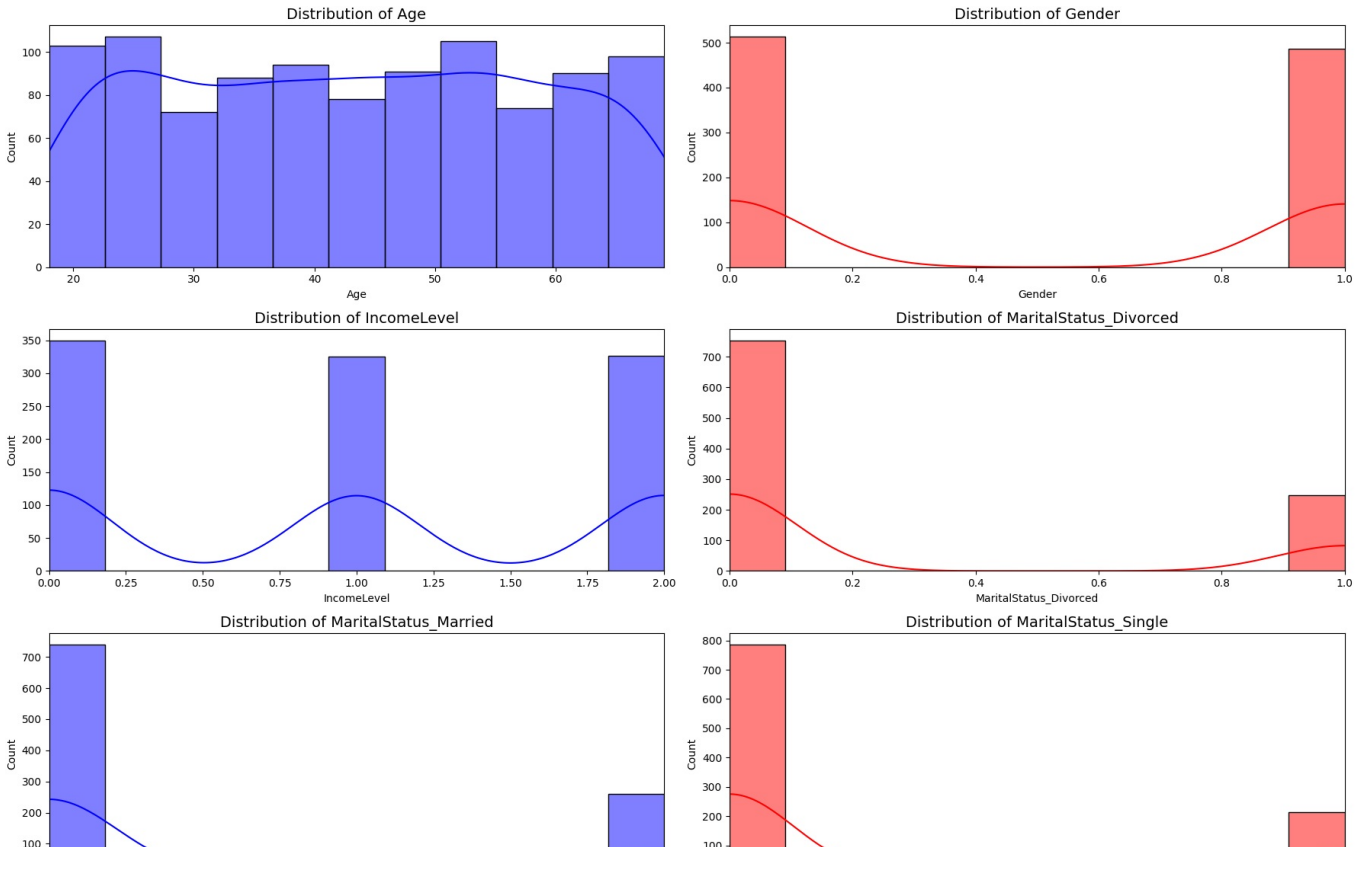
Out[33]:

```
CustomerID                      int64
Age                             int64
Gender                          int64
IncomeLevel                     int64
MaritalStatus_Divorced          int64
MaritalStatus_Married           int64
MaritalStatus_Single            int64
MaritalStatus_Widowed           int64
Count_Books                   float64
Count_Clothing                float64
Count_Electronics             float64
Count_Furniture               float64
Count_Groceries               float64
Cost_Books                    float64
Cost_Clothing                 float64
Cost_Electronics              float64
Cost_Furniture                float64
Cost_Groceries                float64
Count_Complaint               float64
Count_Feedback                float64
Count_Inquiry                 float64
Count_Resolved                float64
Count_Unresolved              float64
LoginFrequency                  int64
DaysSinceLastLogin              int64
ServiceUsage_Mobile App         int64
ServiceUsage_Online Banking     int64
ServiceUsage_Website            int64
ChurnStatus                     int64
dtype: object
```

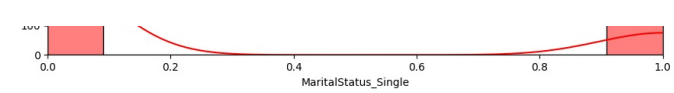
Quick Data visualisation

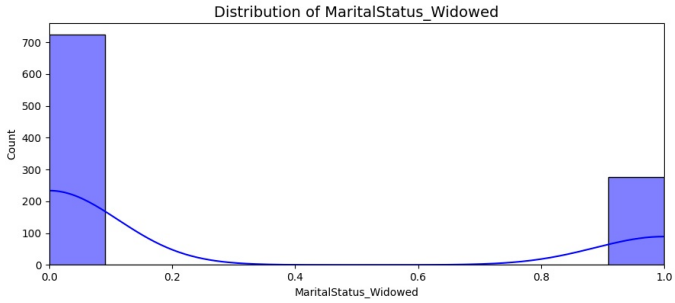
In [34]:

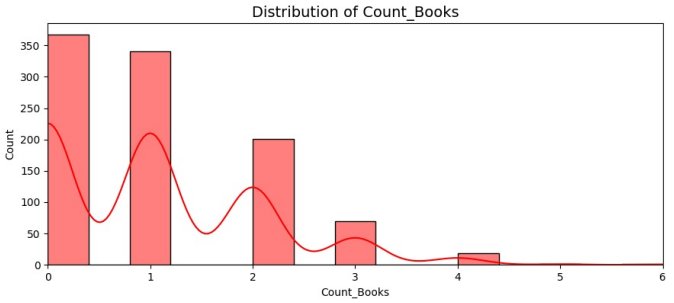
```
import matplotlib.pyplot as plt
import seaborn as sns
num_cols = merged_df.drop("CustomerID",axis=1).columns

fig, axes = plt.subplots(len(num_cols)//2, 2, figsize=(18, 4 * (len(num_cols)//2)))
axes = axes.flatten()

for i, col in enumerate(num_cols):
    sns.histplot(merged_df[col], ax=axes[i], color='b' if i % 2 == 0 else 'r', kde= True)
    axes[i].set_title(f'Distribution of {col}', fontsize=14)
    axes[i].set_xlim([merged_df[col].min(), merged_df[col].max()])

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

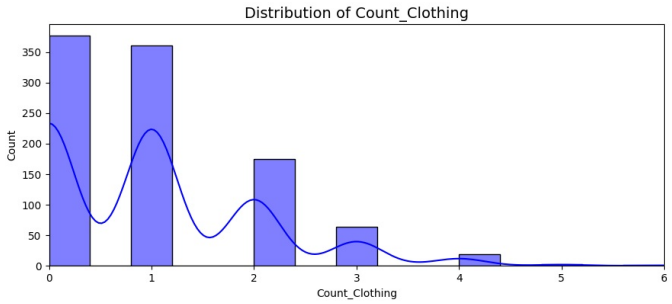


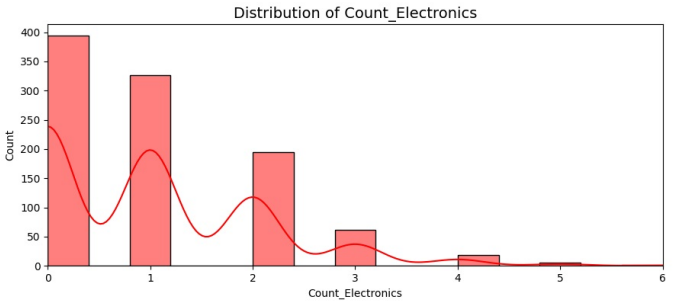

MaritalStatus_Married

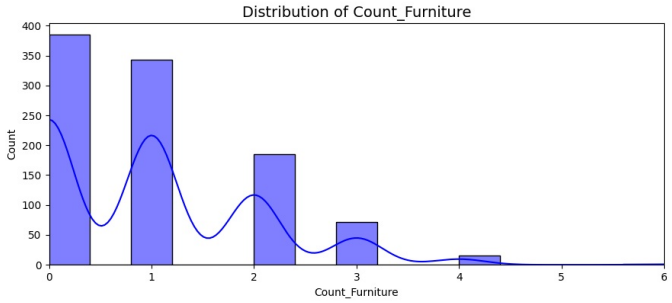

MaritalStatus_Single

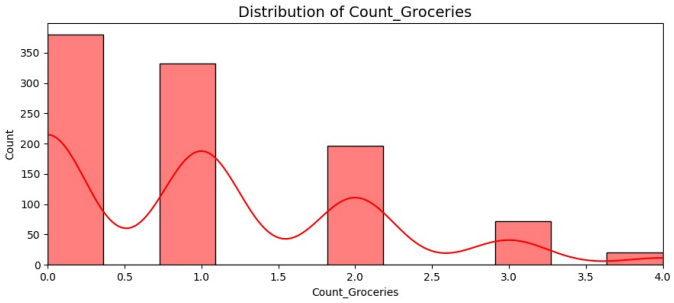

Distribution of MaritalStatus_Widowed
Count
MaritalStatus_Widowed

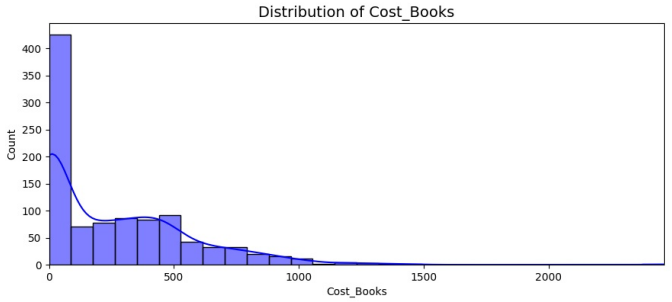

Distribution of Count_Books
Count
Count_Books


Distribution of Count_Clothing
Count
Count_Clothing

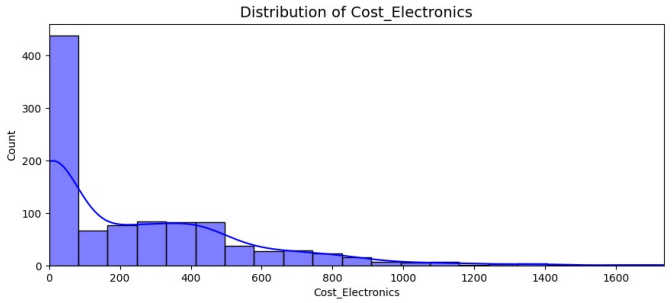

Distribution of Count_Electronics
Count
Count_Electronics

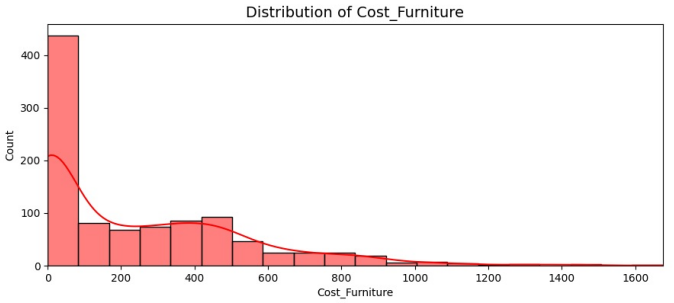

Distribution of Count_Furniture
Count
Count_Furniture

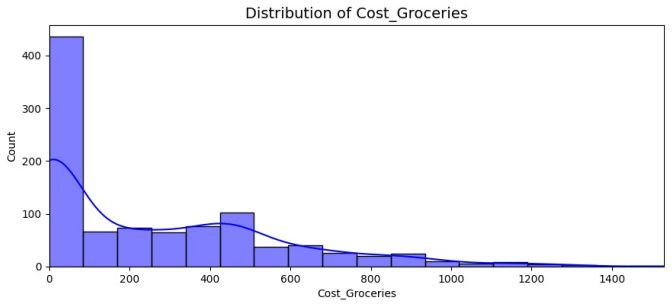

Distribution of Count_Groceries
Count
Count_Groceries

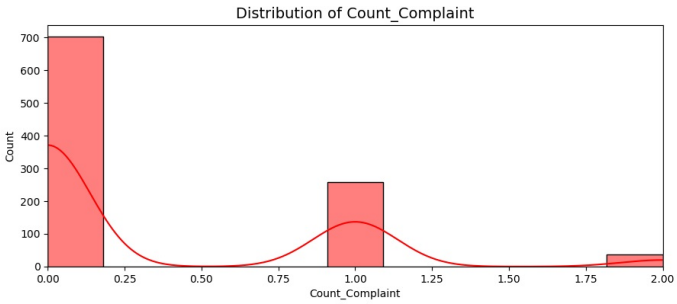

Distribution of Cost_Books
Count
Cost_Books

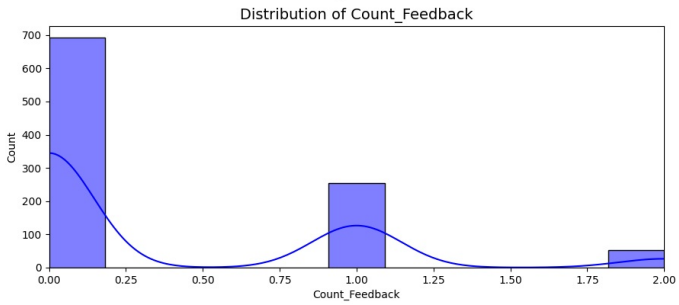

Distribution of Cost_Clothing
Count
Cost_Clothing

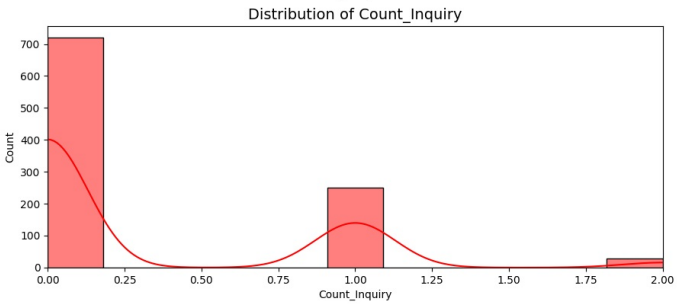

Distribution of Cost_Electronics
Count
Cost_Electronics

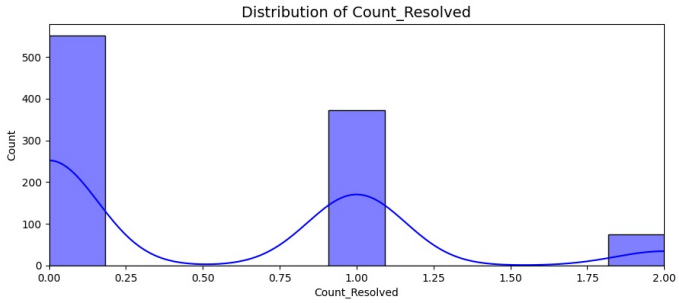

Distribution of Cost_Furniture
Count
Cost_Furniture

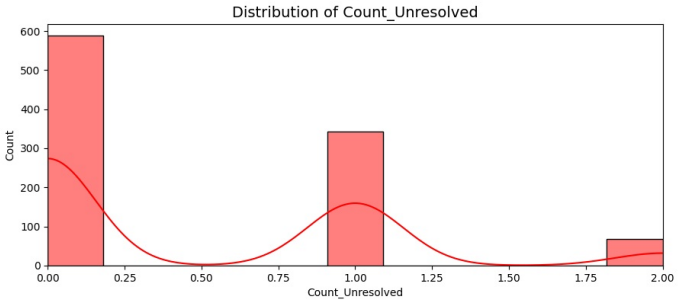

Distribution of Cost_Groceries
Count
Cost_Groceries

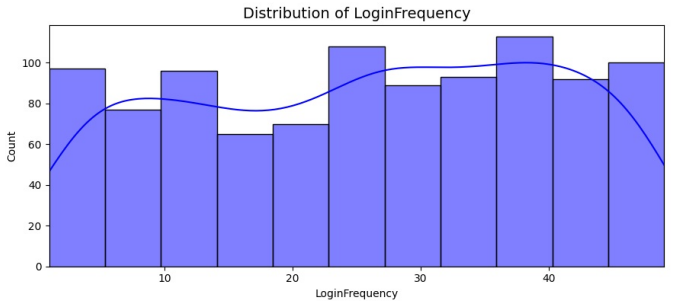

Distribution of Count_Complaint
Count
Count_Complaint

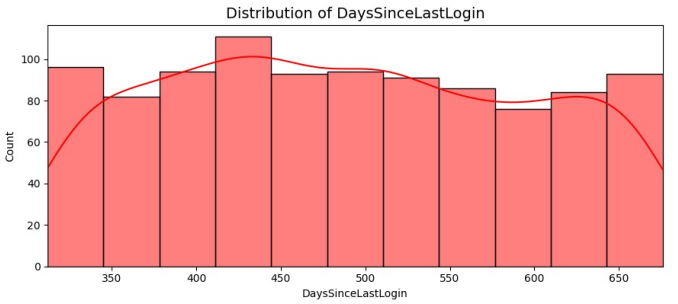

Distribution of Count_Feedback
Count
Count_Feedback

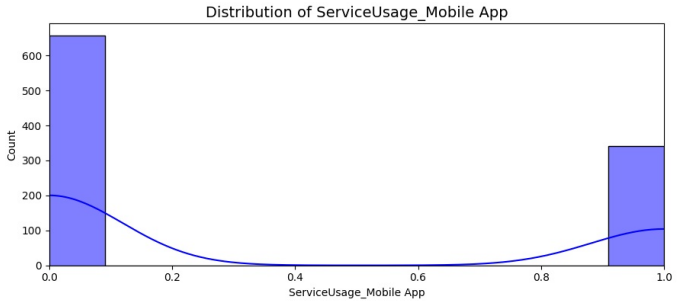

Distribution of Count_Inquiry
Count
Count_Inquiry

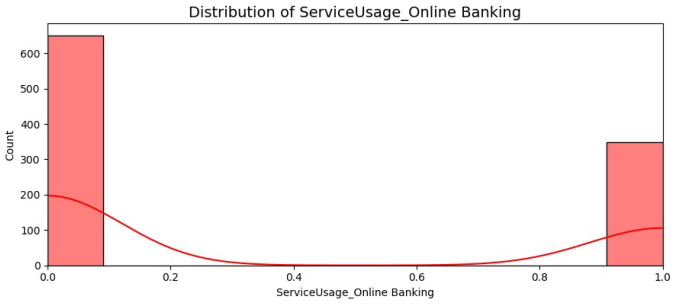


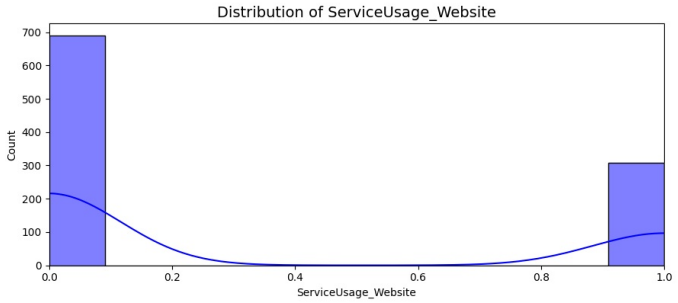


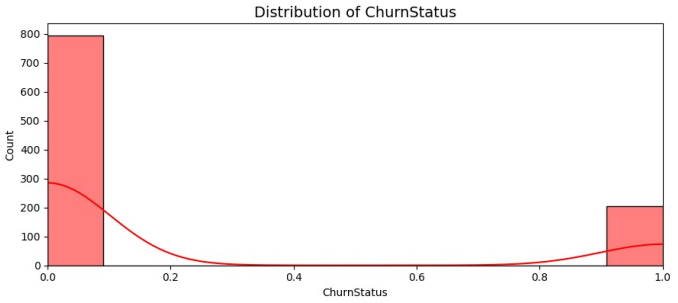




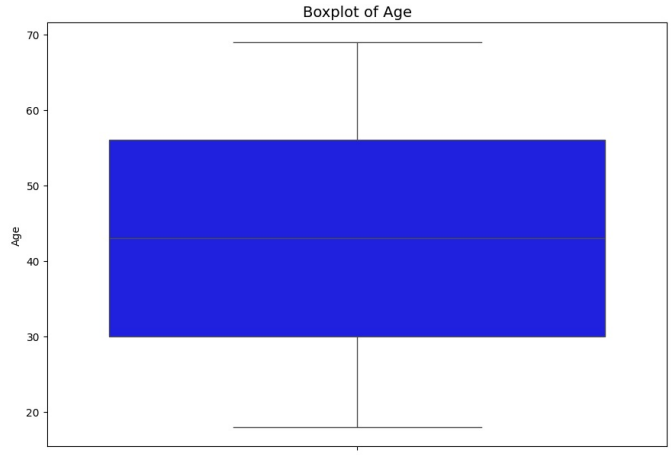


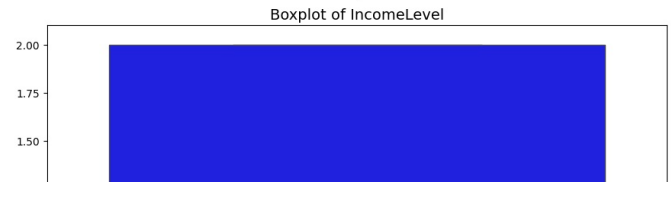


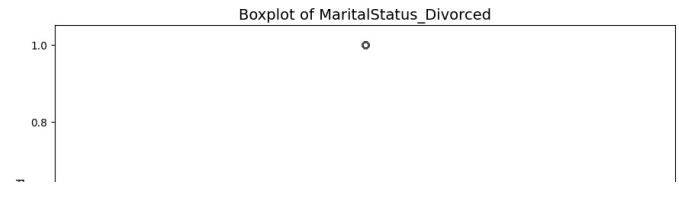




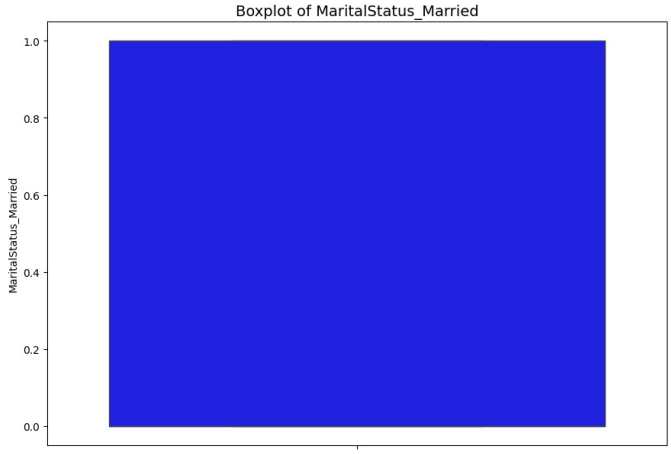
In [35]:
```
num_cols = merged_df.drop("CustomerID", axis=1).columns  # Assuming "CustomerID" is not needed for plotting

fig, axes = plt.subplots(len(num_cols)//2, 2, figsize=(18, 6 * (len(num_cols)//2)))
axes = axes.flatten()

for i, col in enumerate(num_cols):
    sns.boxplot(data=merged_df, y=col, ax=axes[i], color='b' if i % 2 == 0 else 'r')
    axes[i].set_title(f'Boxplot of {col}', fontsize=14)

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



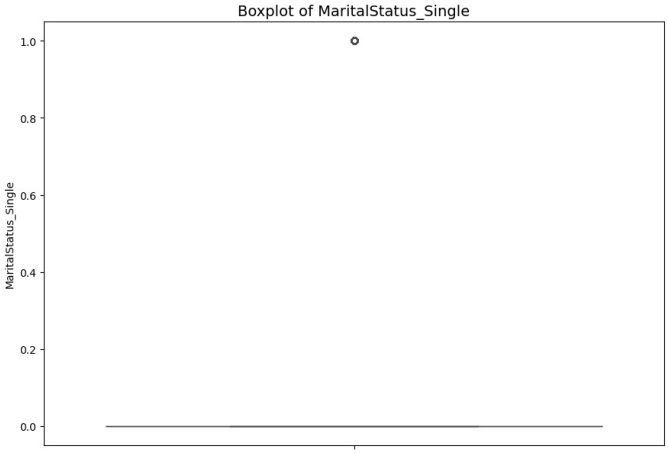




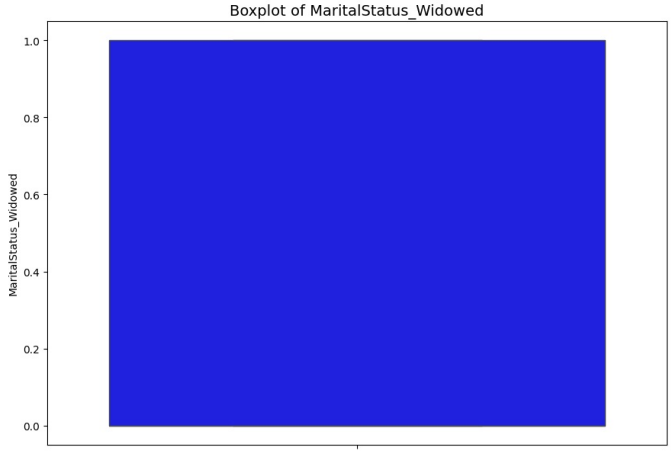
Boxplot of MaritalStatus_Married



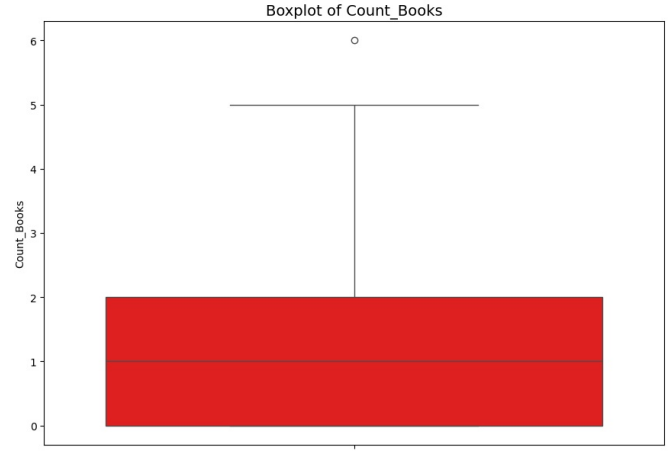
Boxplot of MaritalStatus_Single



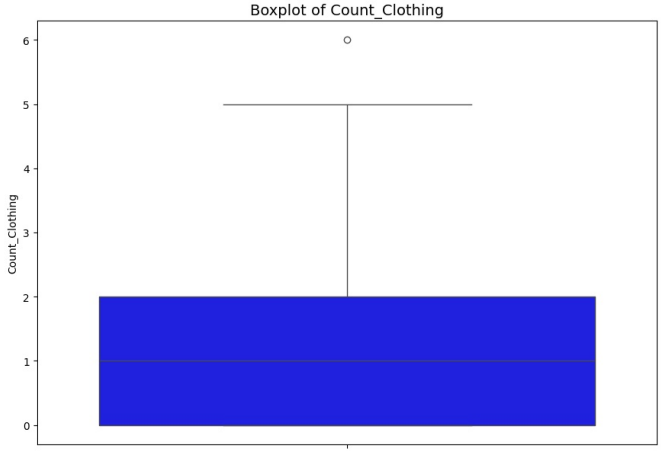
Boxplot of MaritalStatus_Widowed



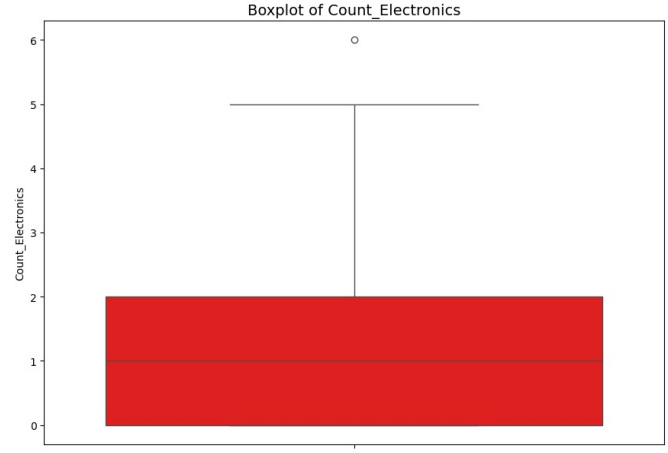
Boxplot of Count_Books



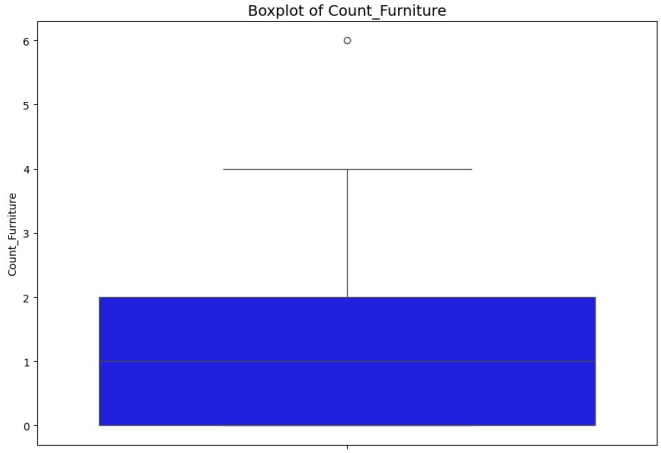
Boxplot of Count_Clothing



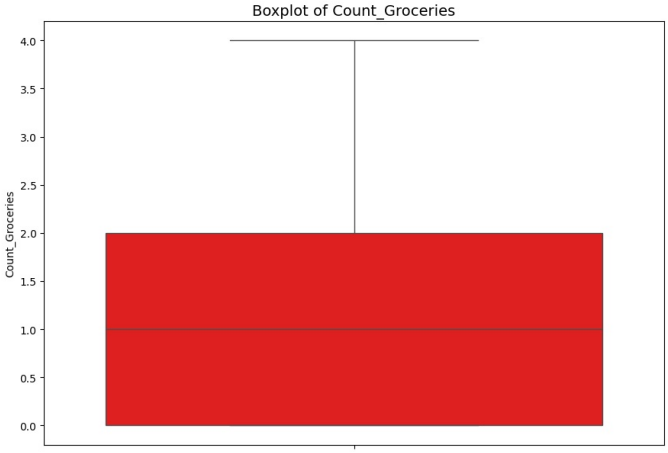
Boxplot of Count_Electronics



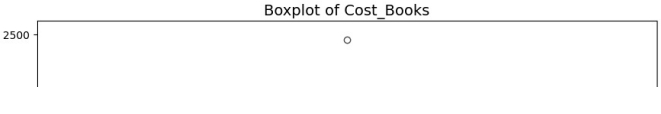
Boxplot of Count_Furniture



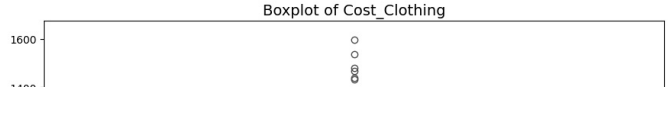
Boxplot of Count_Groceries

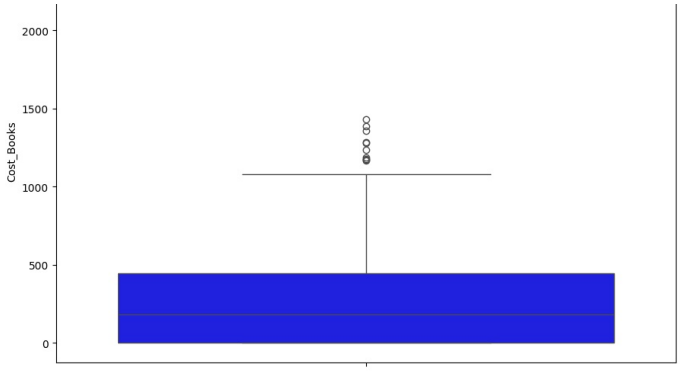


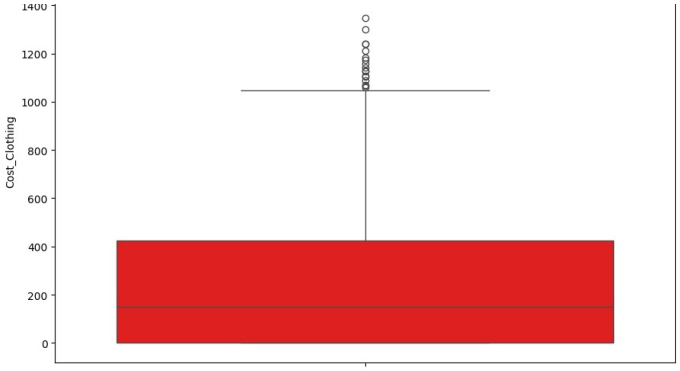
Boxplot of Cost_Books



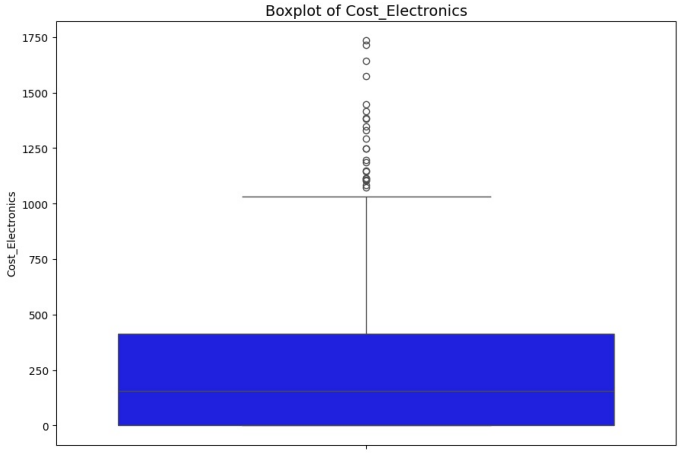
Boxplot of Cost_Clothing



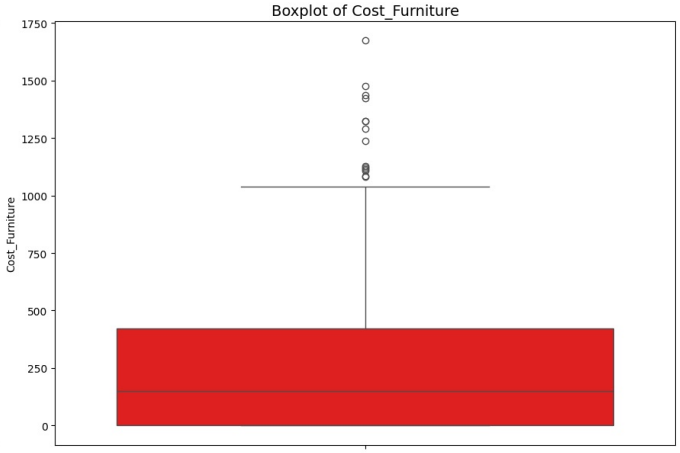




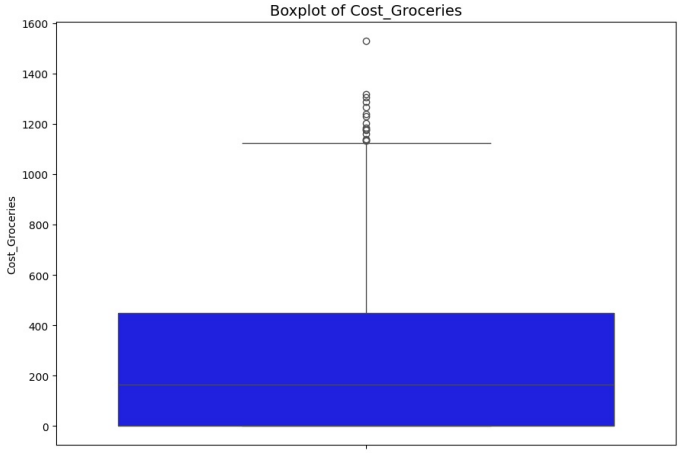
Boxplot of Cost_Electronics



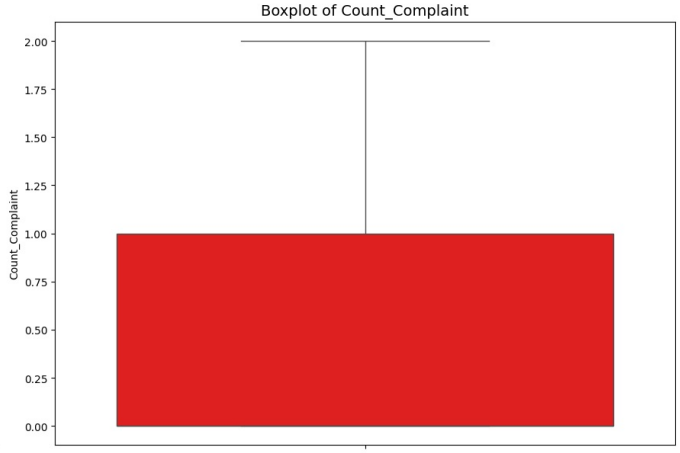
Boxplot of Cost_Furniture



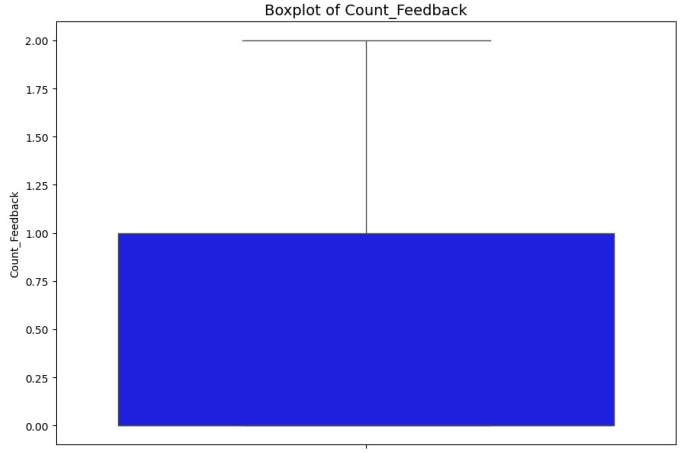
Boxplot of Cost_Groceries



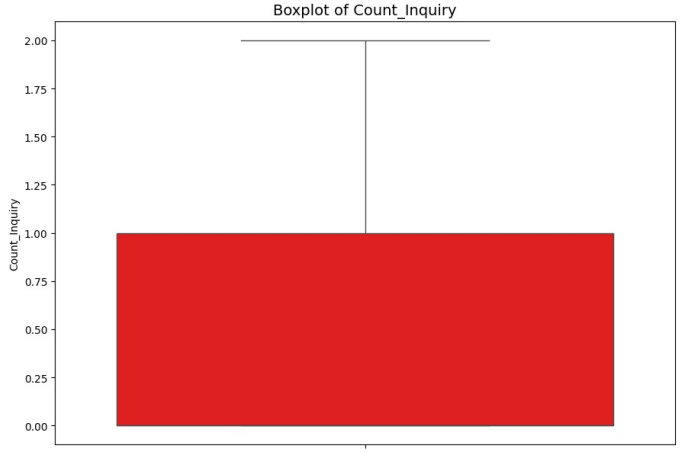
Boxplot of Count_Complaint



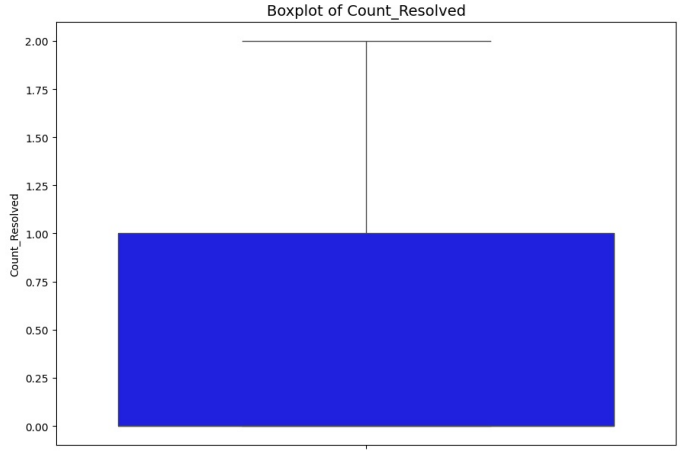
Boxplot of Count_Feedback



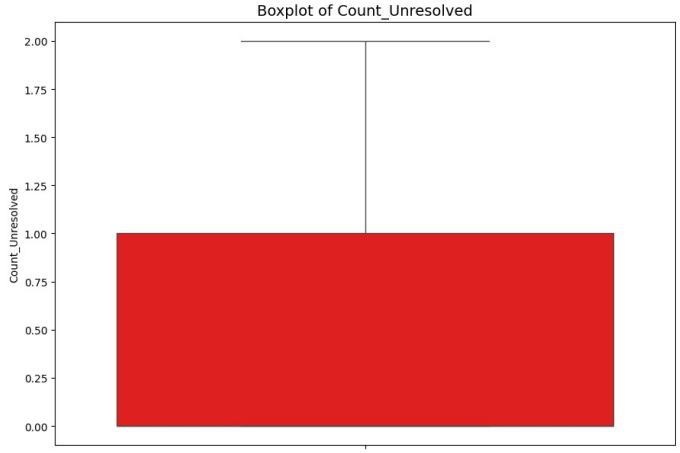
Boxplot of Count_Inquiry

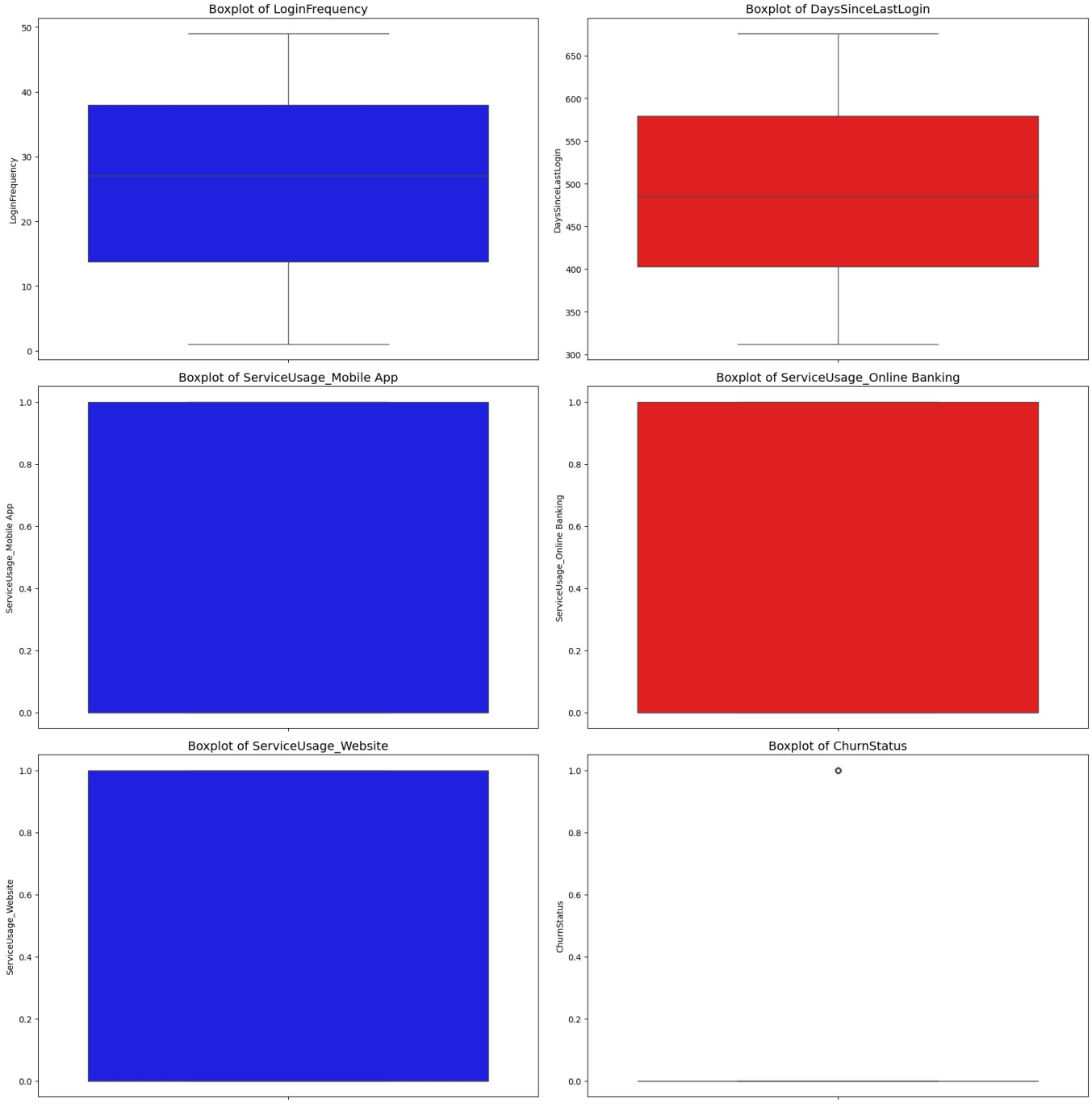


Boxplot of Count_Resolved



Boxplot of Count_Unresolved

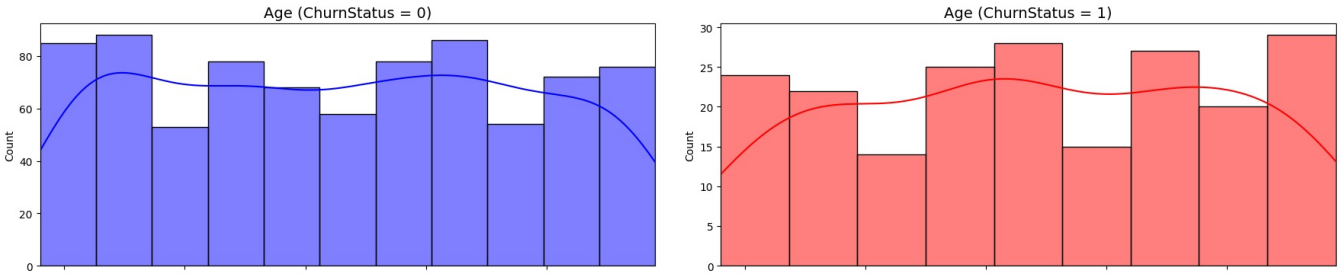


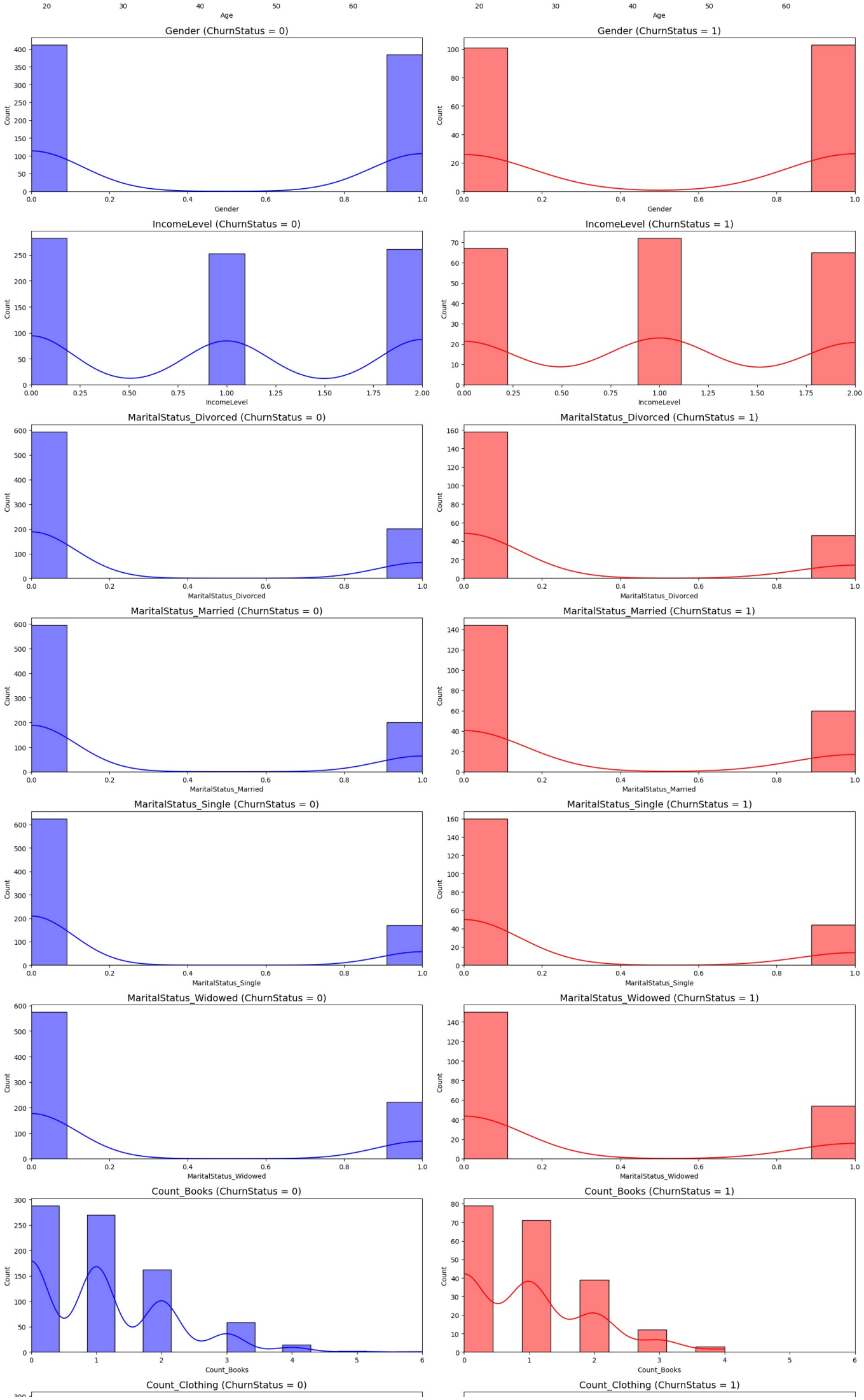


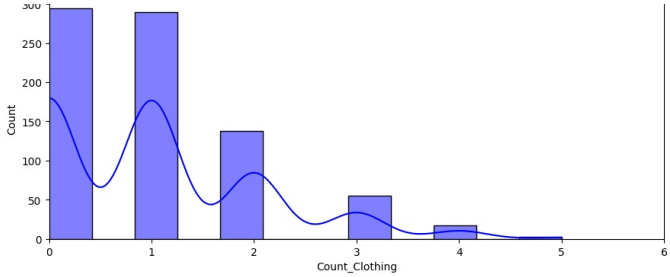
Considering the size of data, we would not delete outliers curretly, but it is needed in larger data set.

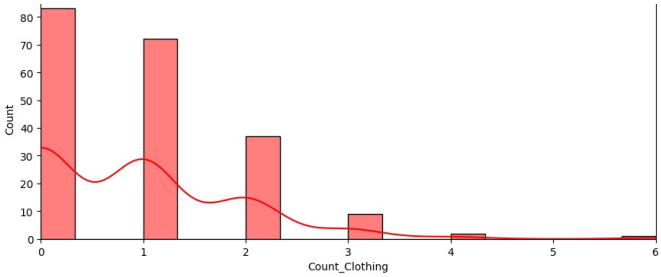
we can do a quick comparison between "ChurnStatus == 0 and 1"

In [36]:
```
fig, axes = plt.subplots(len(num_cols), 2, figsize=(18, 4 * len(num_cols)))  # Two columns for side-by-side plo

for i, col in enumerate(num_cols):
    # Plot for ChurnStatus == 0
    sns.histplot(merged_df[merged_df['ChurnStatus'] == 0][col], ax=axes[i, 0], color='b', kde=True)
    axes[i, 0].set_title(f'{col} (ChurnStatus = 0)', fontsize=14)
    axes[i, 0].set_xlim([merged_df[col].min(), merged_df[col].max()])

    # Plot for ChurnStatus == 1
    sns.histplot(merged_df[merged_df['ChurnStatus'] == 1][col], ax=axes[i, 1], color='r', kde=True)
    axes[i, 1].set_title(f'{col} (ChurnStatus = 1)', fontsize=14)
    axes[i, 1].set_xlim([merged_df[col].min(), merged_df[col].max()])

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

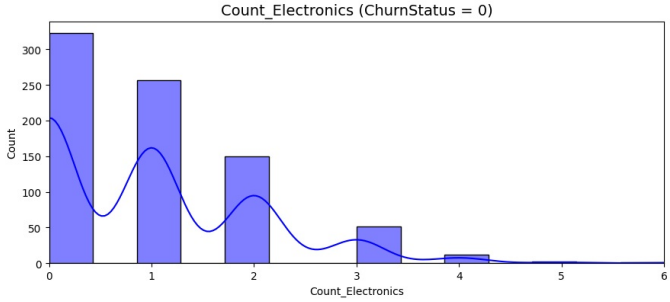



Age
Gender (ChurnStatus = 0)
Gender (ChurnStatus = 1)
Gender
IncomeLevel (ChurnStatus = 0)
IncomeLevel (ChurnStatus = 1)
IncomeLevel
MaritalStatus_Divorced (ChurnStatus = 0)
MaritalStatus_Divorced (ChurnStatus = 1)
MaritalStatus_Divorced
MaritalStatus_Married (ChurnStatus = 0)
MaritalStatus_Married (ChurnStatus = 1)
MaritalStatus_Married
MaritalStatus_Single (ChurnStatus = 0)
MaritalStatus_Single (ChurnStatus = 1)
MaritalStatus_Single
MaritalStatus_Widowed (ChurnStatus = 0)
MaritalStatus_Widowed (ChurnStatus = 1)
MaritalStatus_Widowed
Count_Books (ChurnStatus = 0)
Count_Books (ChurnStatus = 1)
Count_Books
Count_Clothing (ChurnStatus = 0)
Count_Clothing (ChurnStatus = 1)
Count

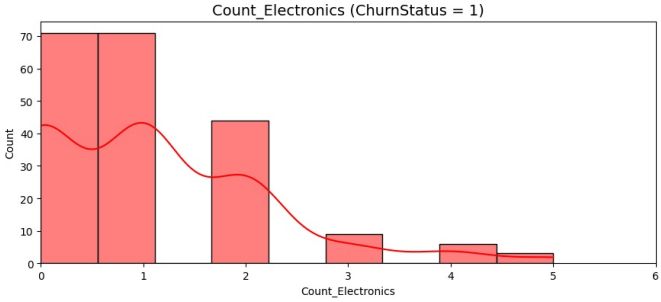




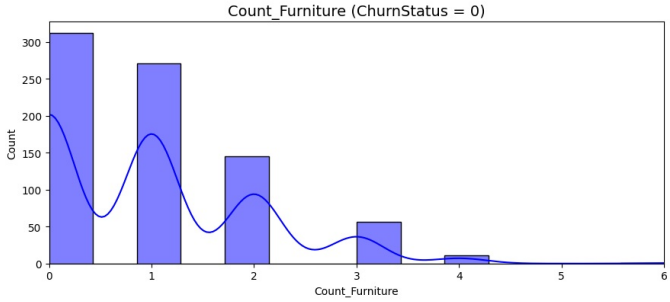
Count_Electronics (ChurnStatus = 0)



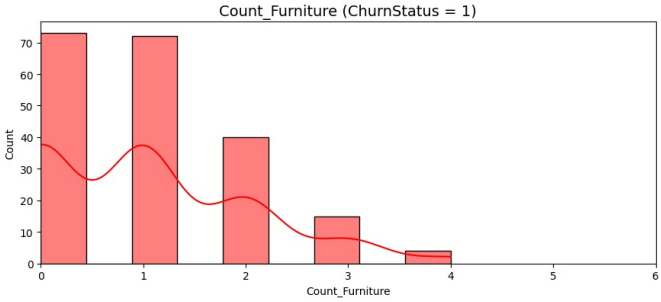
Count_Electronics (ChurnStatus = 1)



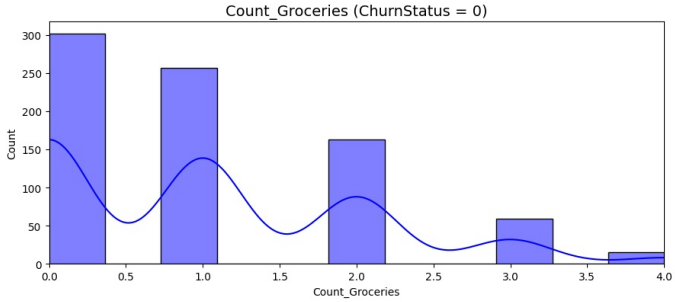
Count_Furniture (ChurnStatus = 0)



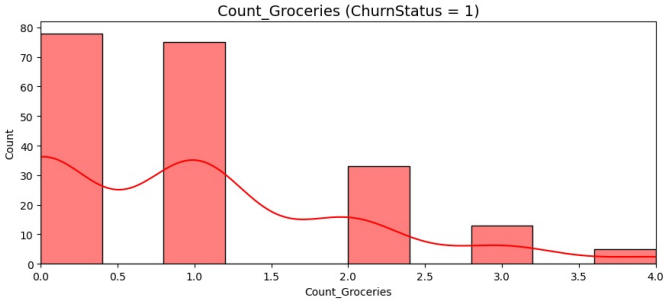
Count_Furniture (ChurnStatus = 1)



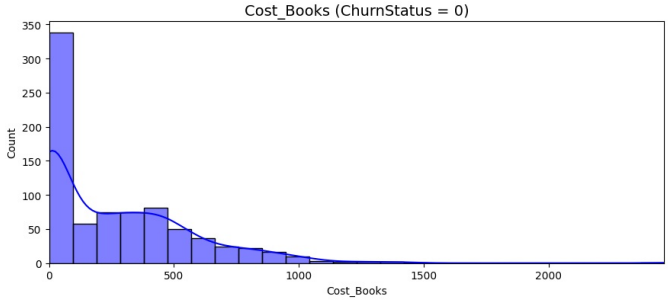
Count_Groceries (ChurnStatus = 0)



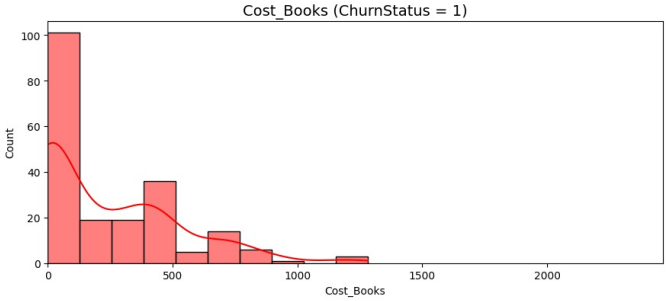
Count_Groceries (ChurnStatus = 1)



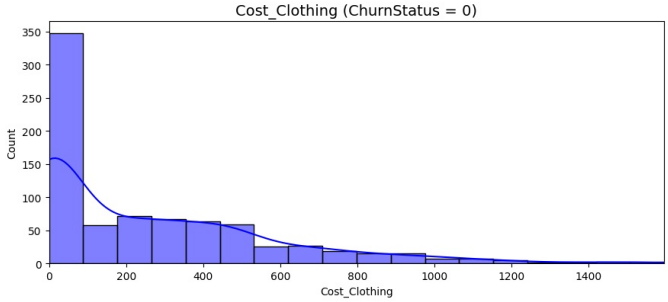
Cost_Books (ChurnStatus = 0)



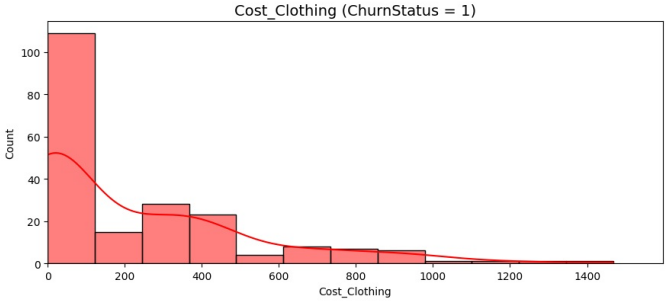
Cost_Books (ChurnStatus = 1)



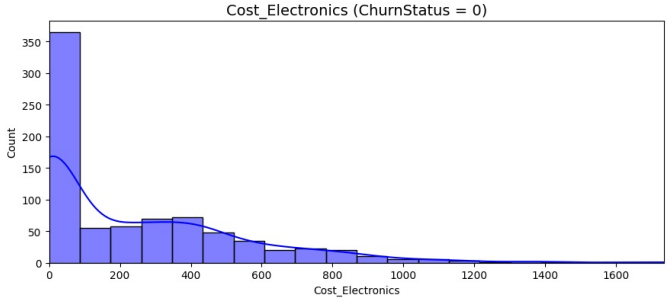
Cost_Clothing (ChurnStatus = 0)



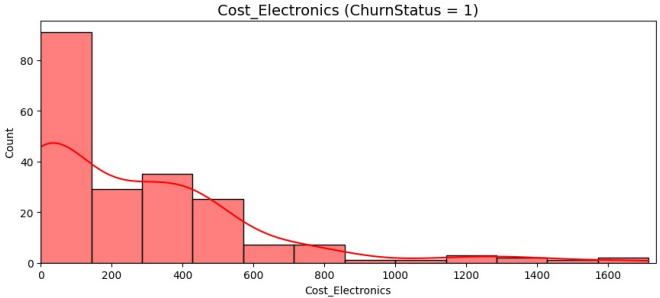
Cost_Clothing (ChurnStatus = 1)



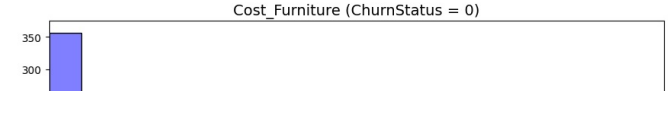
Cost_Electronics (ChurnStatus = 0)



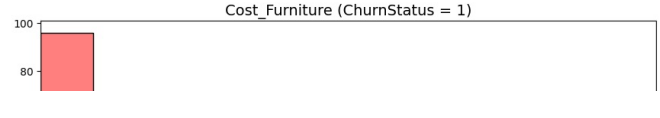
Cost_Electronics (ChurnStatus = 1)

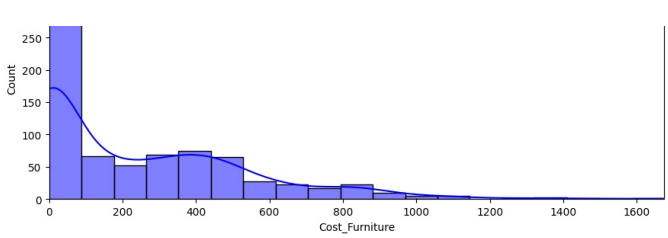


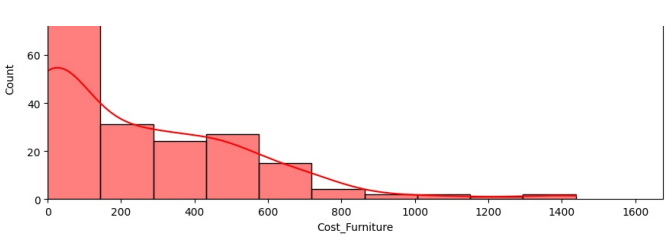
Cost_Furniture (ChurnStatus = 0)



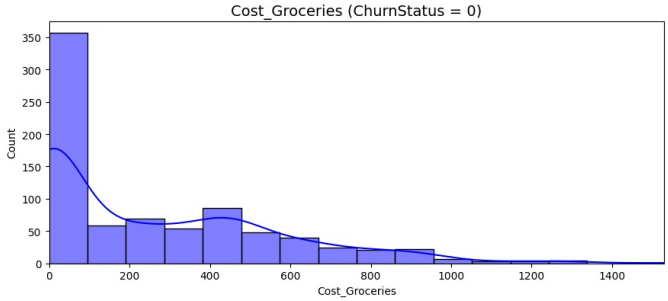
Cost_Furniture (ChurnStatus = 1)



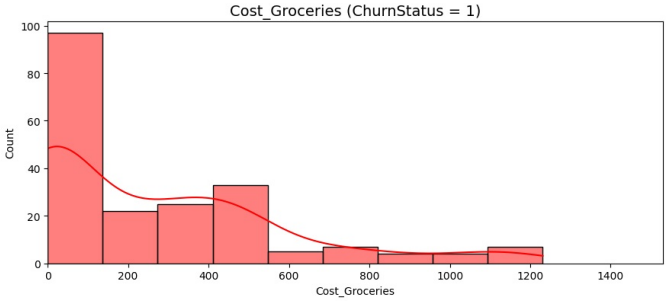




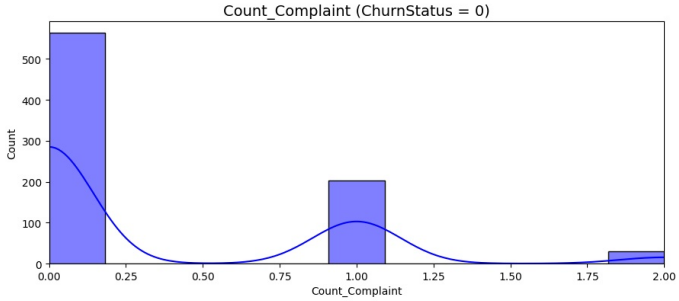
Cost_Groceries (ChurnStatus = 0)



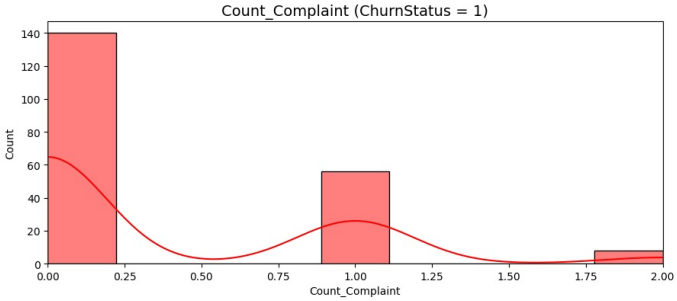
Cost_Groceries (ChurnStatus = 1)



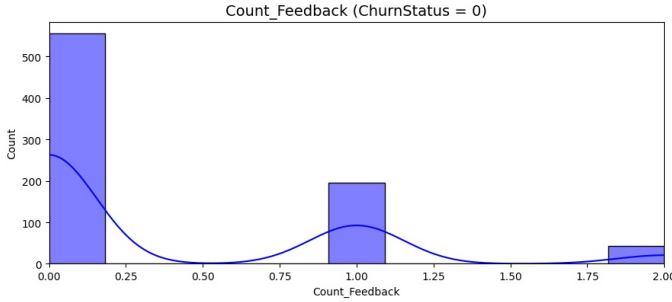
Count_Complaint (ChurnStatus = 0)



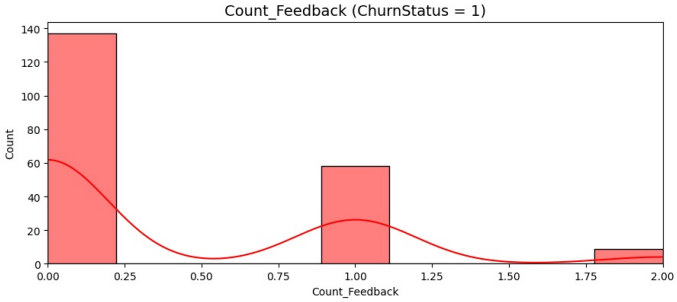
Count_Complaint (ChurnStatus = 1)



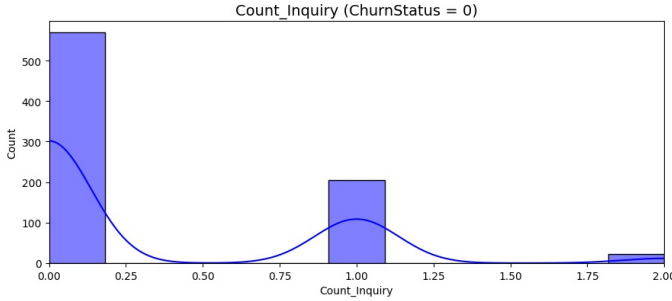
Count_Feedback (ChurnStatus = 0)



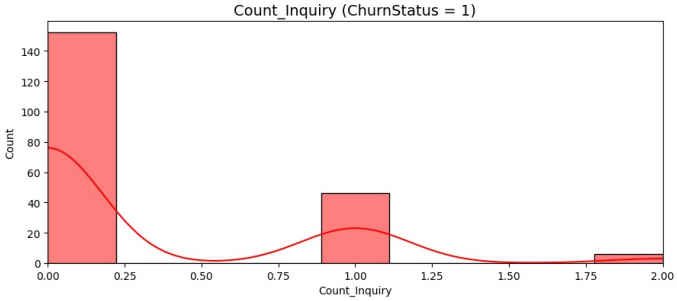
Count_Feedback (ChurnStatus = 1)



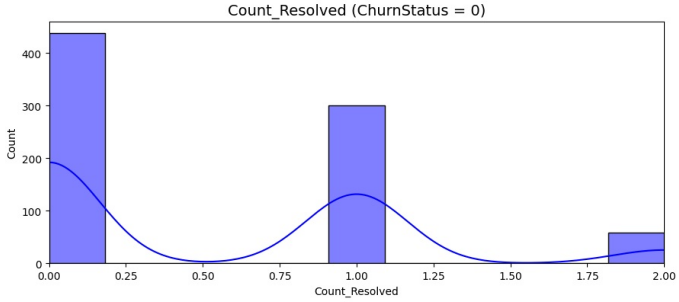
Count_Inquiry (ChurnStatus = 0)



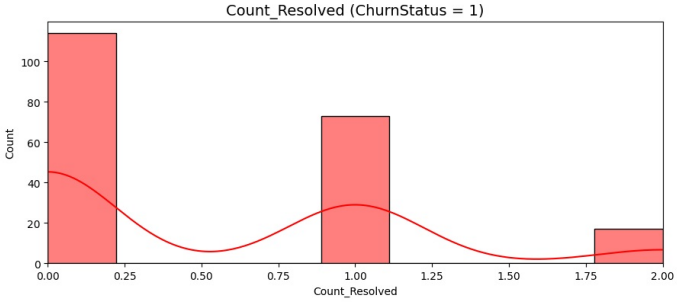
Count_Inquiry (ChurnStatus = 1)



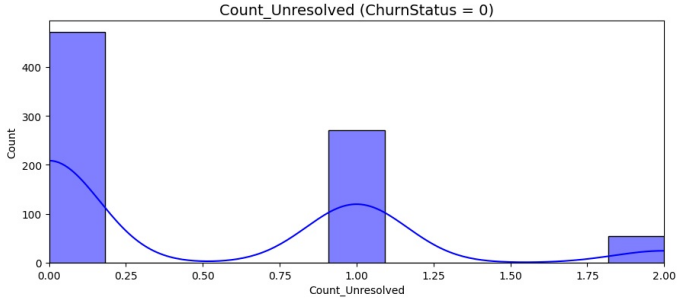
Count_Resolved (ChurnStatus = 0)



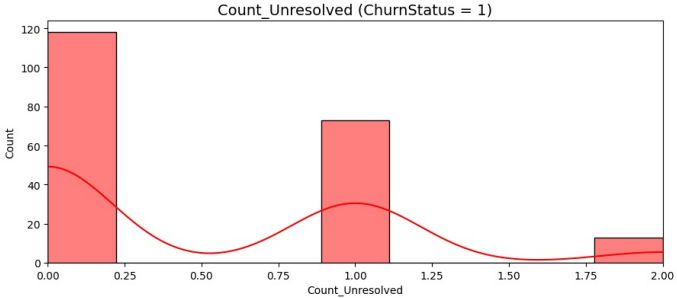
Count_Resolved (ChurnStatus = 1)



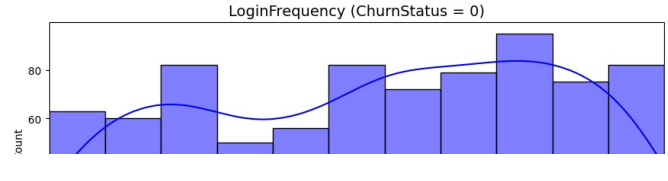
Count_Unresolved (ChurnStatus = 0)



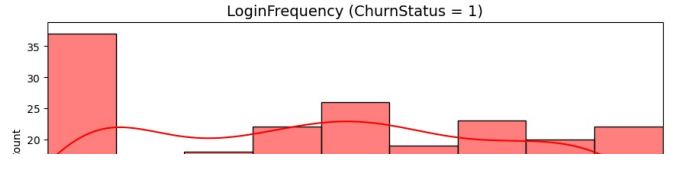
Count_Unresolved (ChurnStatus = 1)

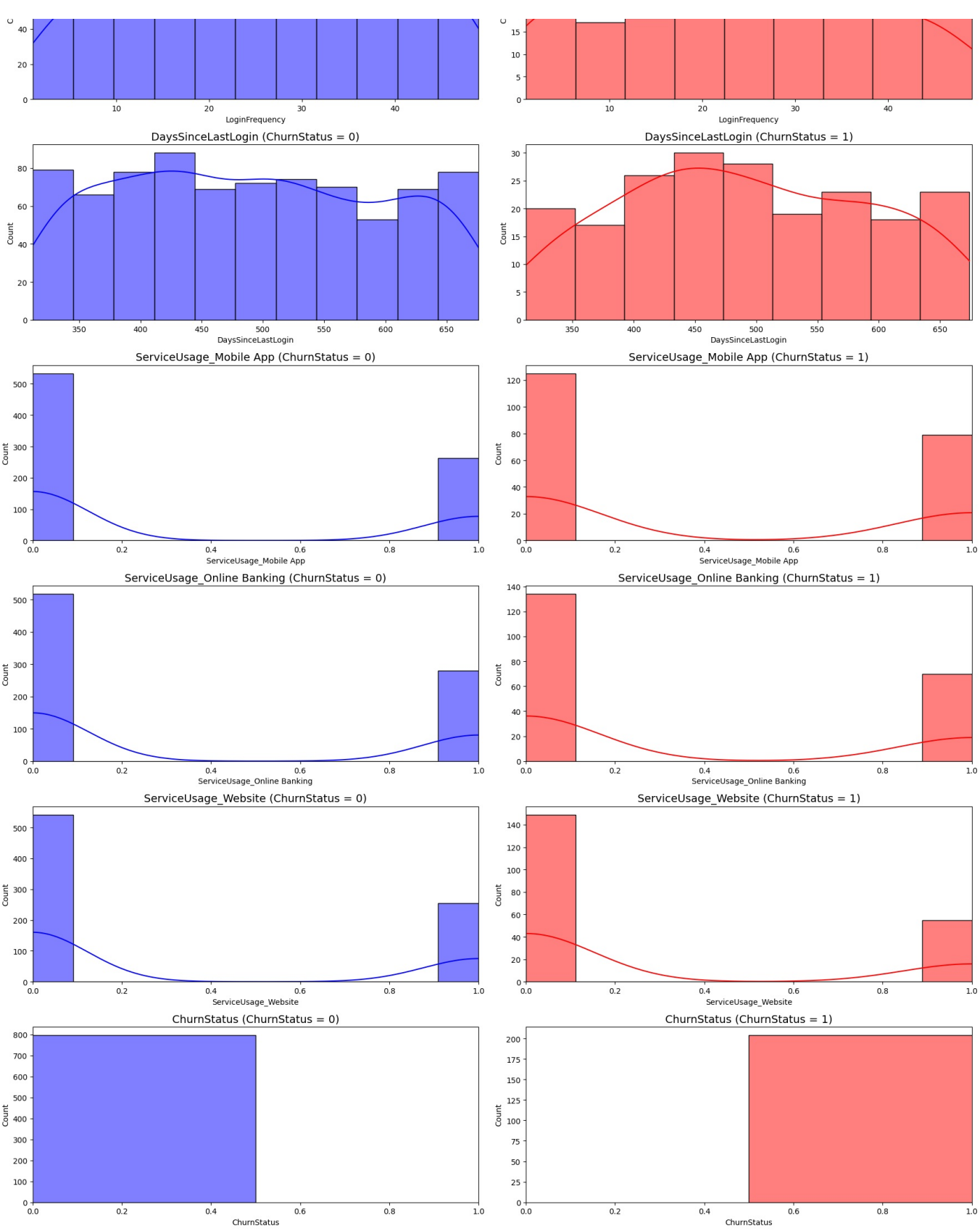


LoginFrequency (ChurnStatus = 0)



LoginFrequency (ChurnStatus = 1)

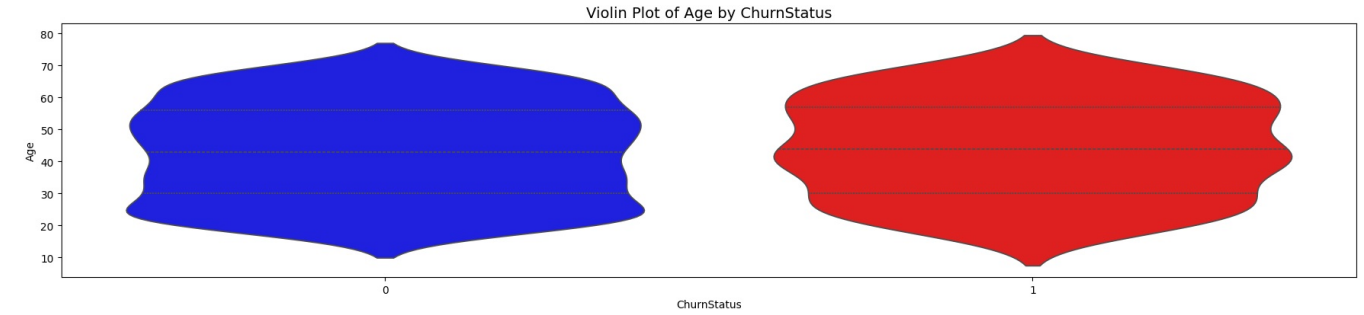


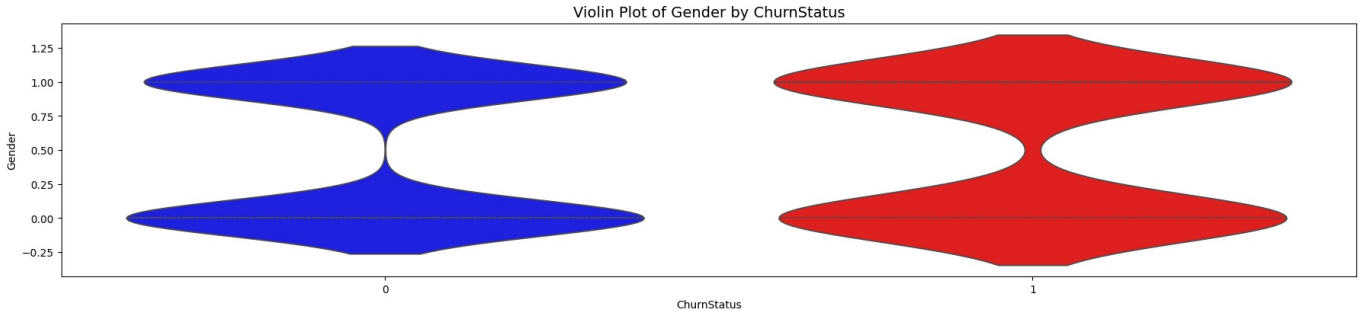


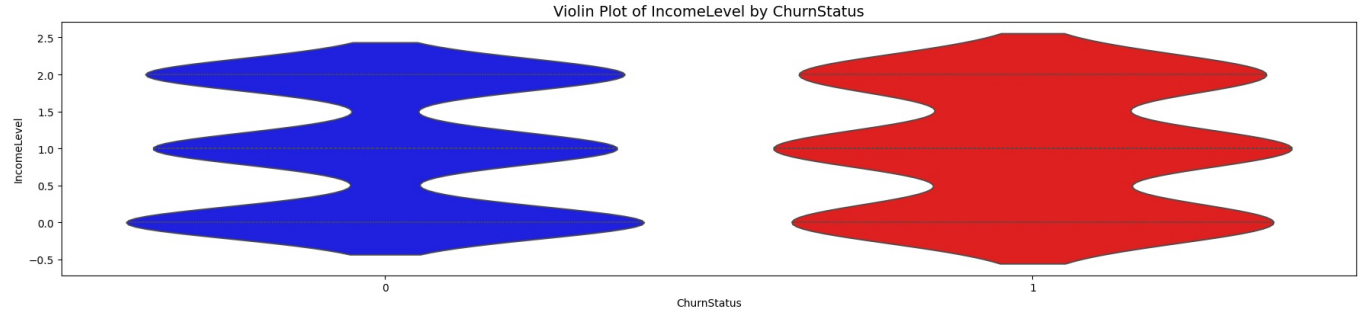
by violin plots, we can make comparisons more easily.

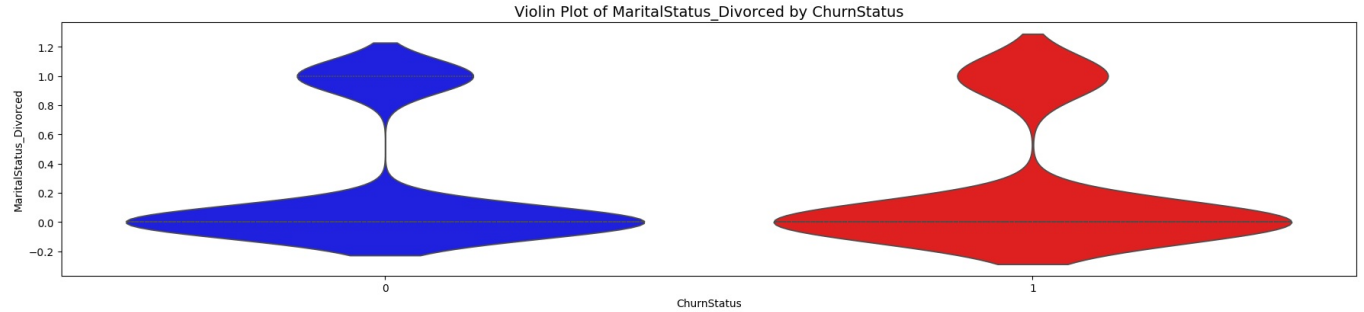
In [39]:
```
fig, axes = plt.subplots(len(num_cols)-1, 1, figsize=(18, 4 * len(num_cols)))  # Single column of plots

for i, col in enumerate(num_cols):

    if col =="ChurnStatus" : continue
    data = merged_df[[col, 'ChurnStatus']].dropna()
    sns.violinplot(
        data=data,
        x='ChurnStatus',
        y=col,
        ax=axes[i],
        hue='ChurnStatus',
        palette=['b', 'r'],
        inner='quartile',
        legend=False
    )
```

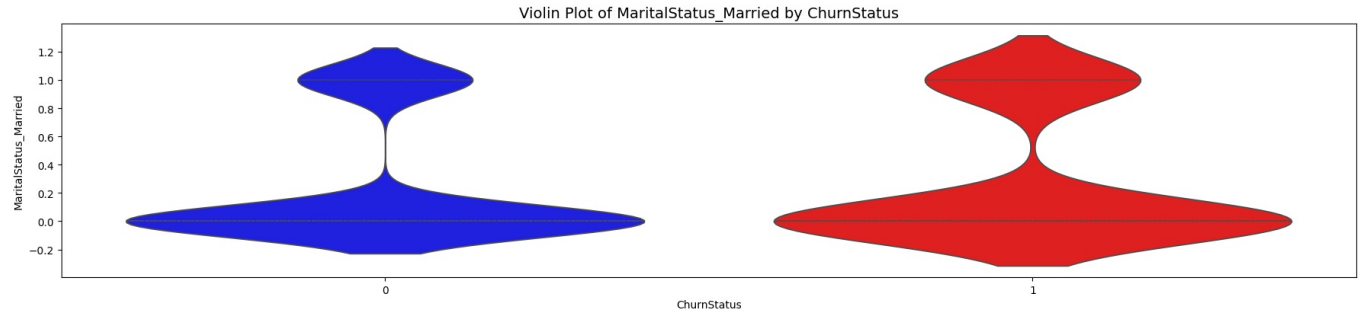
```
    axes[i].set_title(f'Violin Plot of {col} by ChurnStatus', fontsize=14)

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

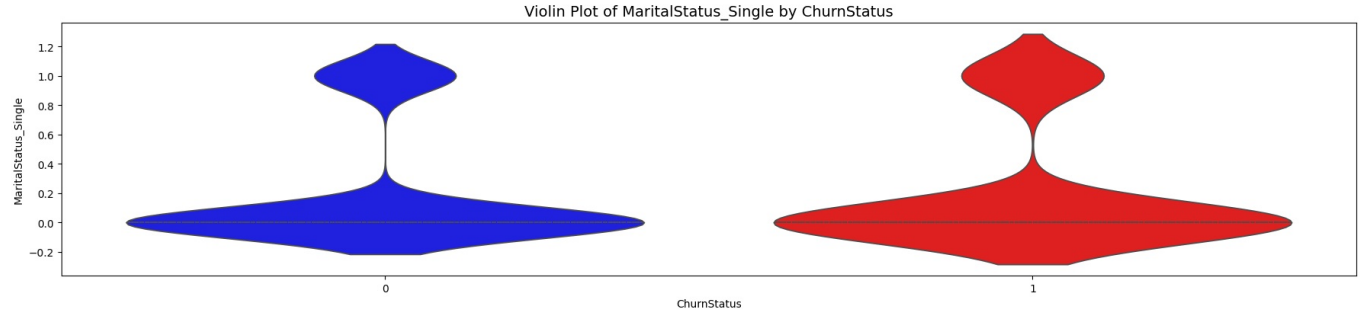


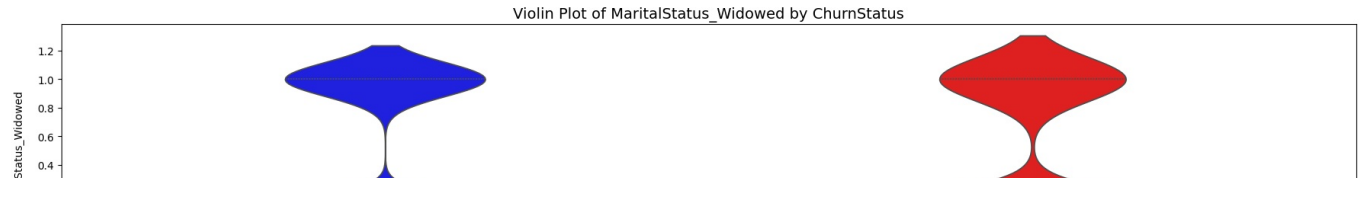


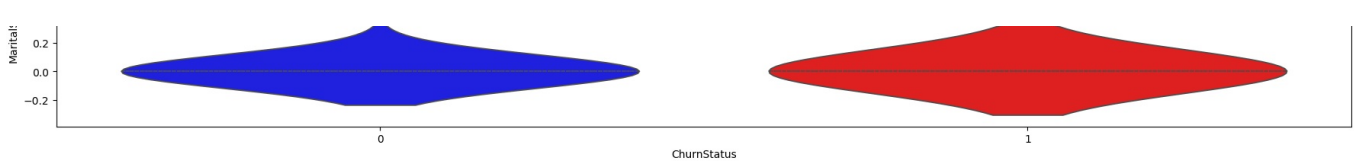


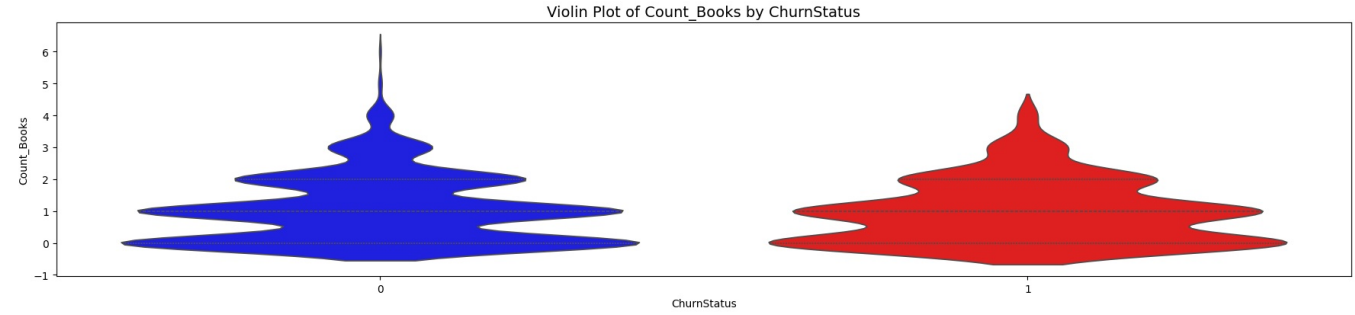


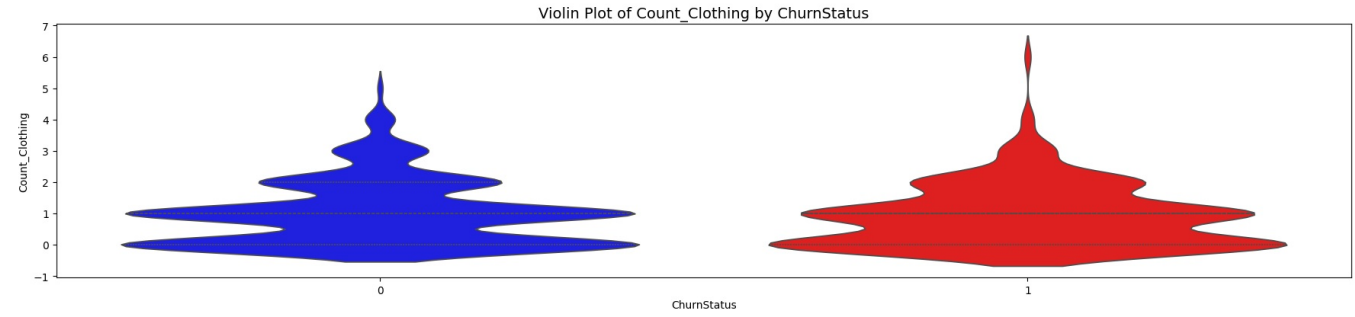


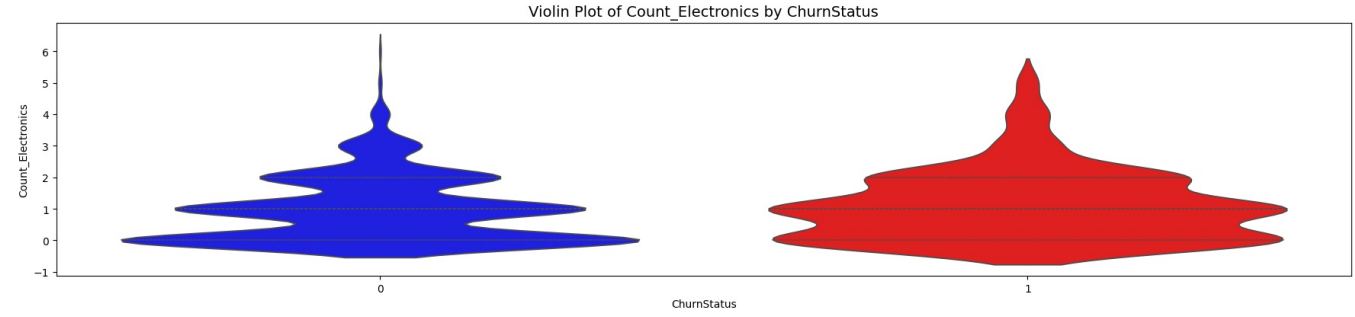


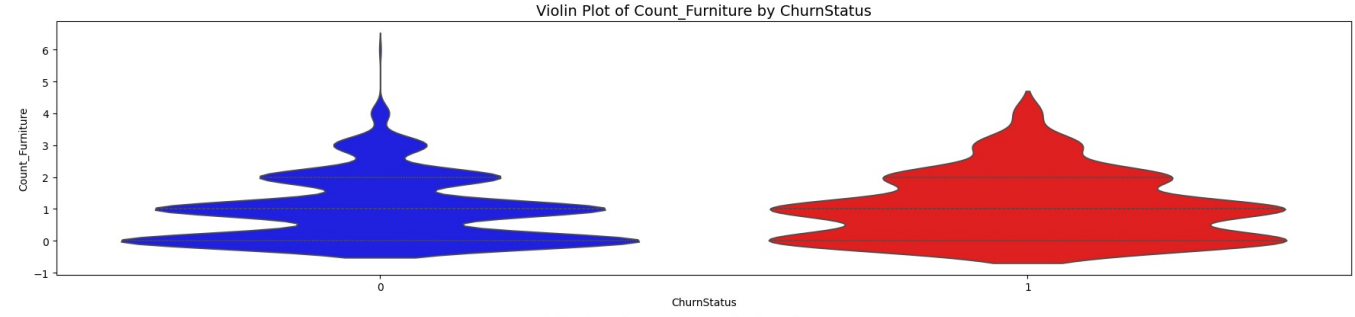


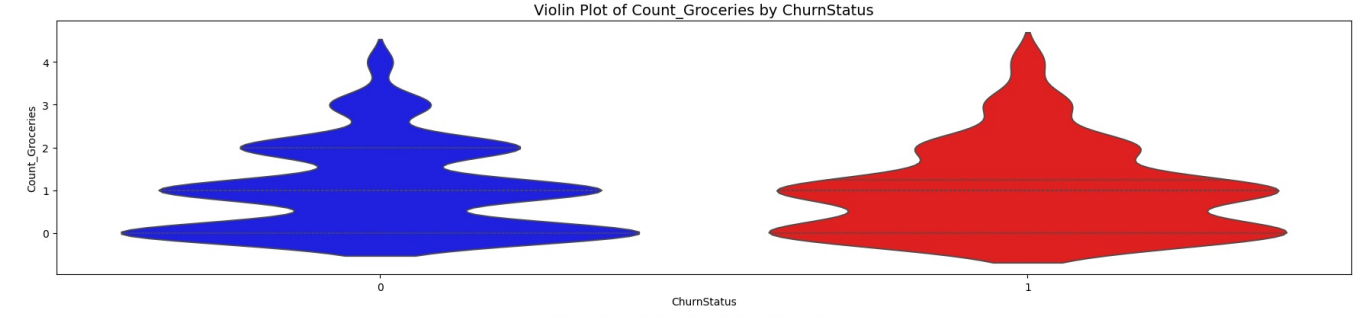

Marital
0.2
0.0
−0.2
0
1
ChurnStatus

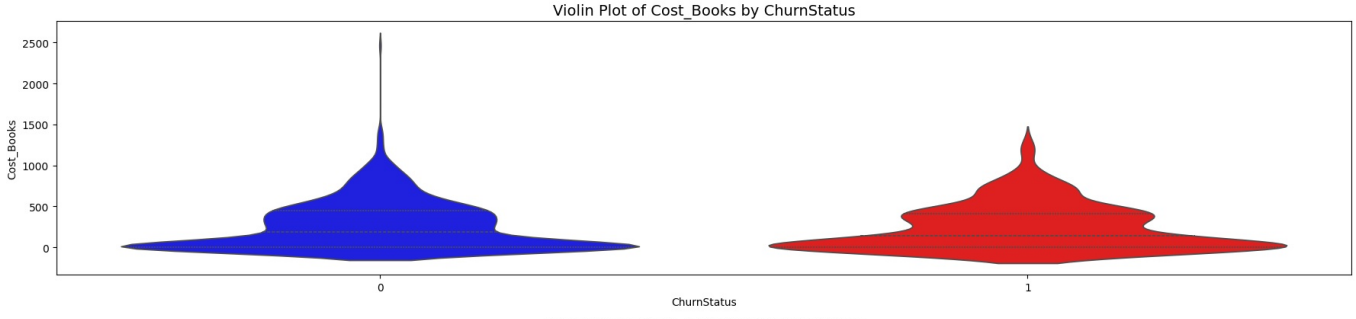

Violin Plot of Count_Books by ChurnStatus
Count_Books
6
5
4
3
2
1
0
−1
0
1
ChurnStatus

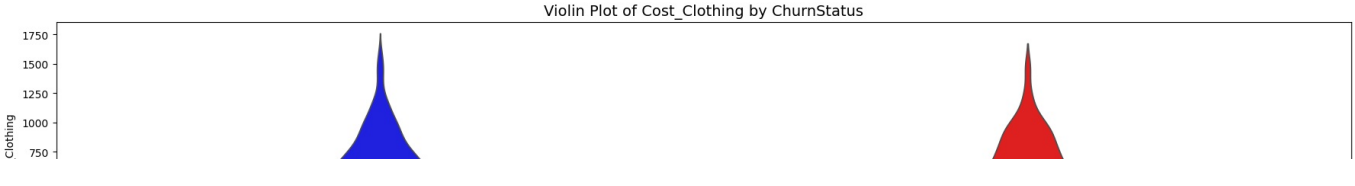

Violin Plot of Count_Clothing by ChurnStatus
Count_Clothing
7
6
5
4
3
2
1
0
−1
0
1
ChurnStatus

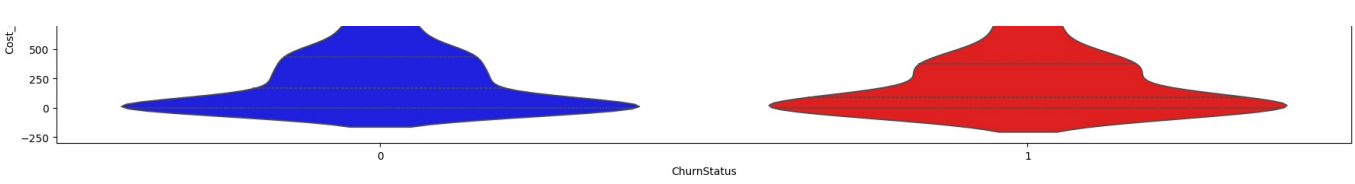

Violin Plot of Count_Electronics by ChurnStatus
Count_Electronics
6
5
4
3
2
1
0
−1
0
1
ChurnStatus


Violin Plot of Count_Furniture by ChurnStatus
Count_Furniture
6
5
4
3
2
1
0
−1
0
1
ChurnStatus

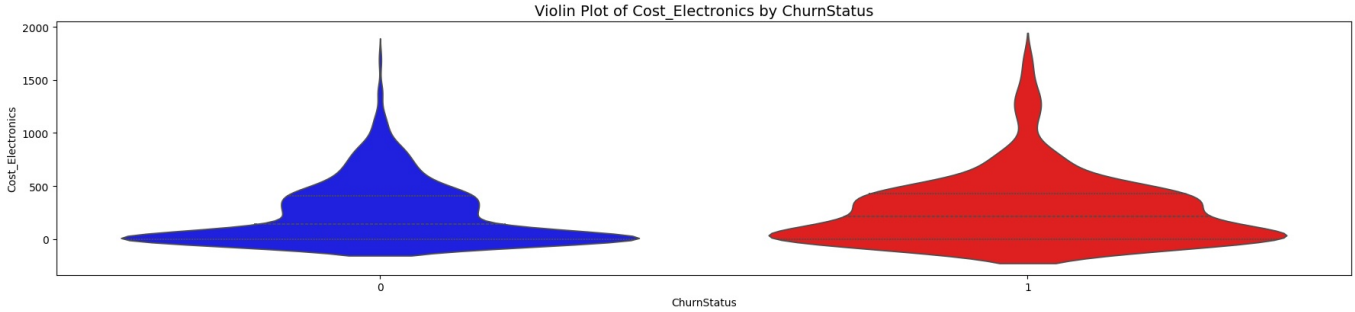

Violin Plot of Count_Groceries by ChurnStatus
Count_Groceries
4
3
2
1
0
0
1
ChurnStatus


Violin Plot of Cost_Books by ChurnStatus
Cost_Books
2500
2000
1500
1000
500
0
0
1
ChurnStatus

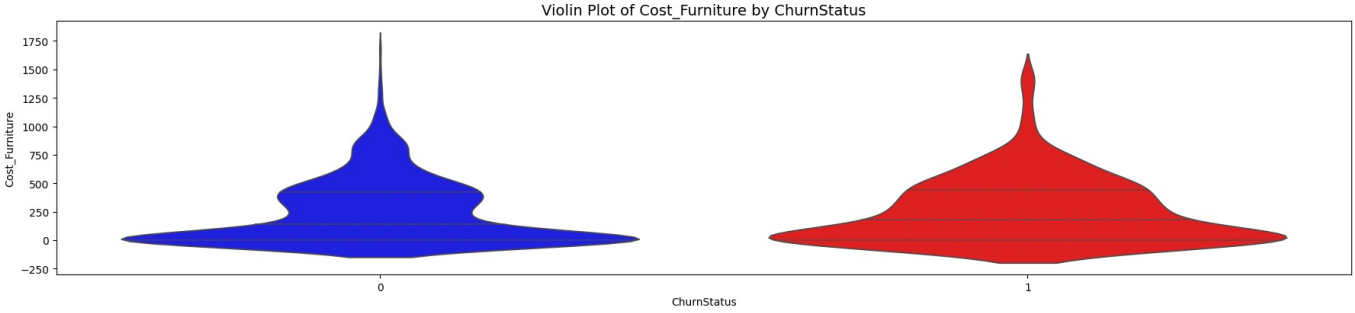

Violin Plot of Cost_Clothing by ChurnStatus
Clothing
1750
1500
1250
1000
750



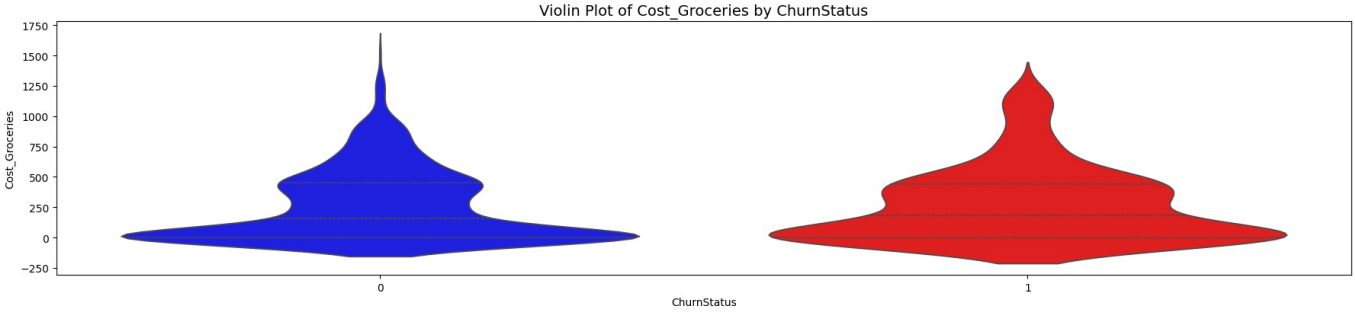
Violin Plot of Cost_Electronics by ChurnStatus



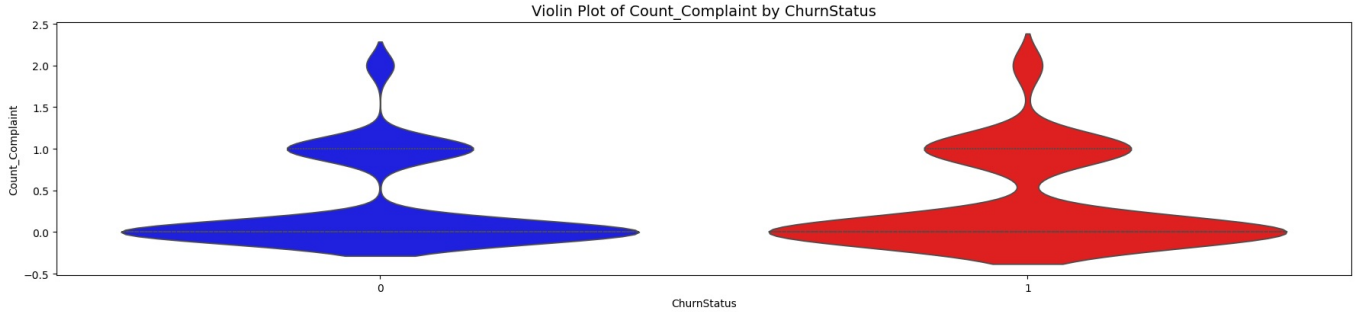
Violin Plot of Cost_Furniture by ChurnStatus



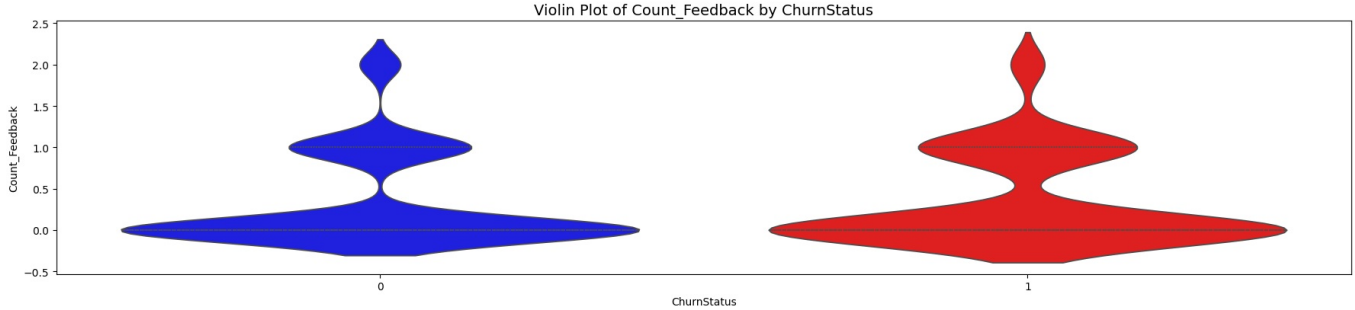
Violin Plot of Cost_Groceries by ChurnStatus



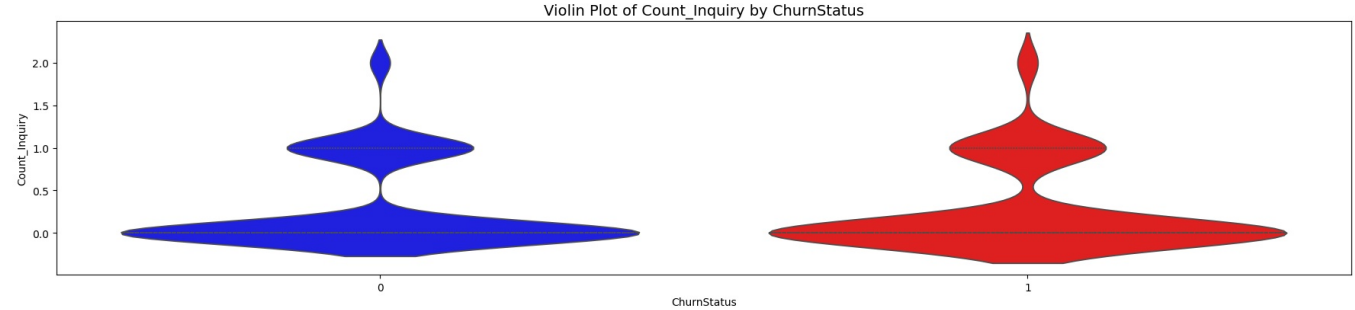
Violin Plot of Count_Complaint by ChurnStatus



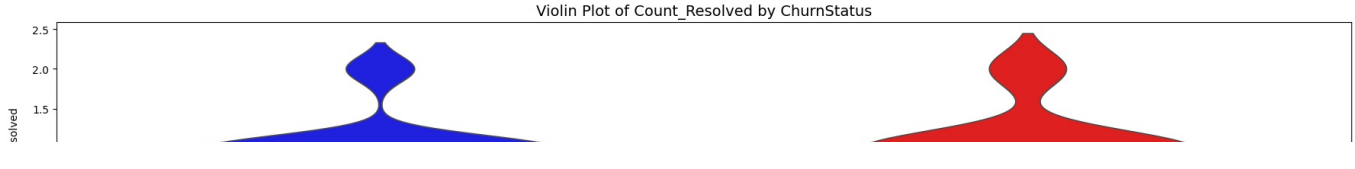
Violin Plot of Count_Feedback by ChurnStatus

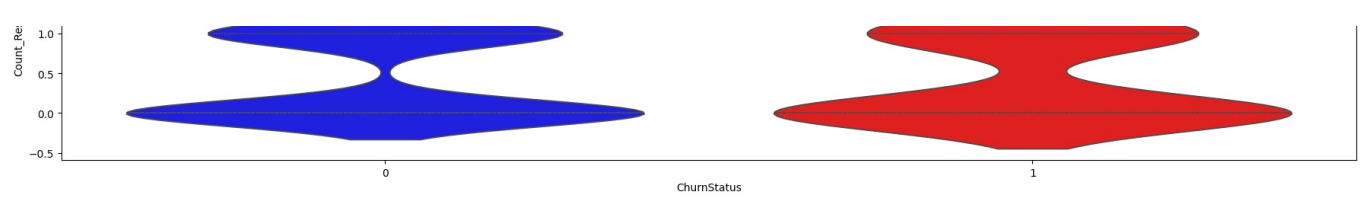


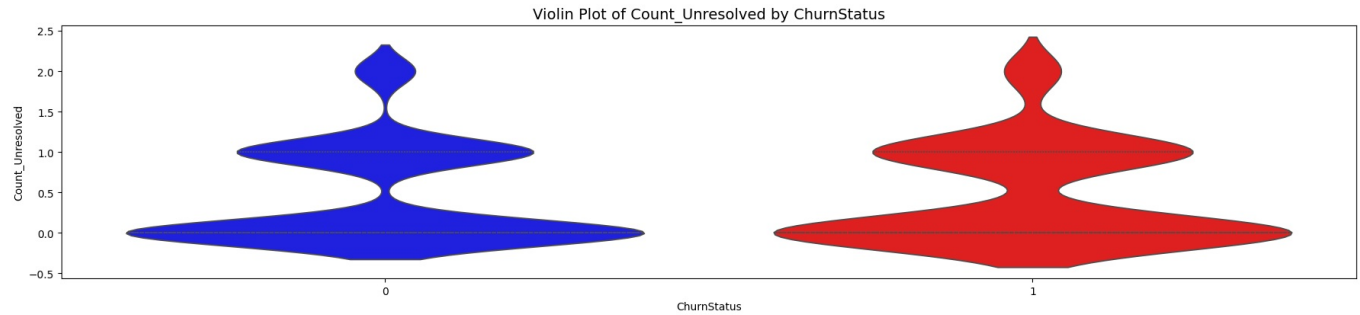
Violin Plot of Count_Inquiry by ChurnStatus

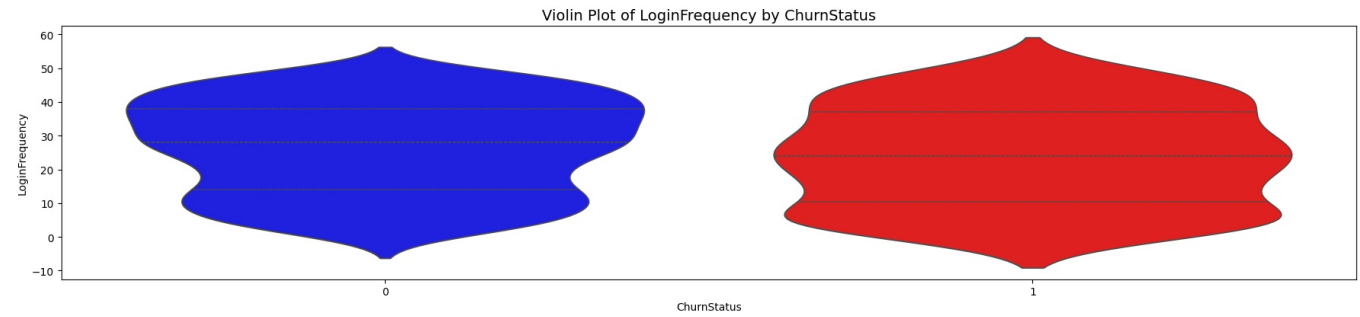


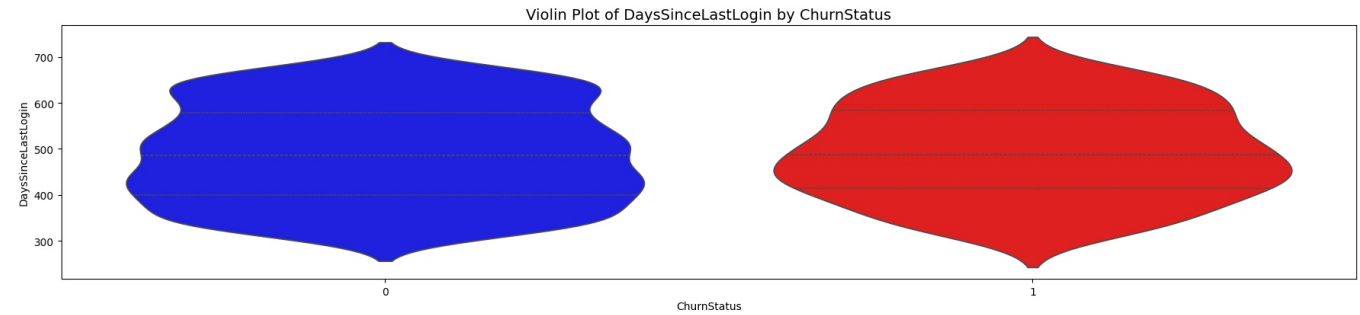
Violin Plot of Count_Resolved by ChurnStatus

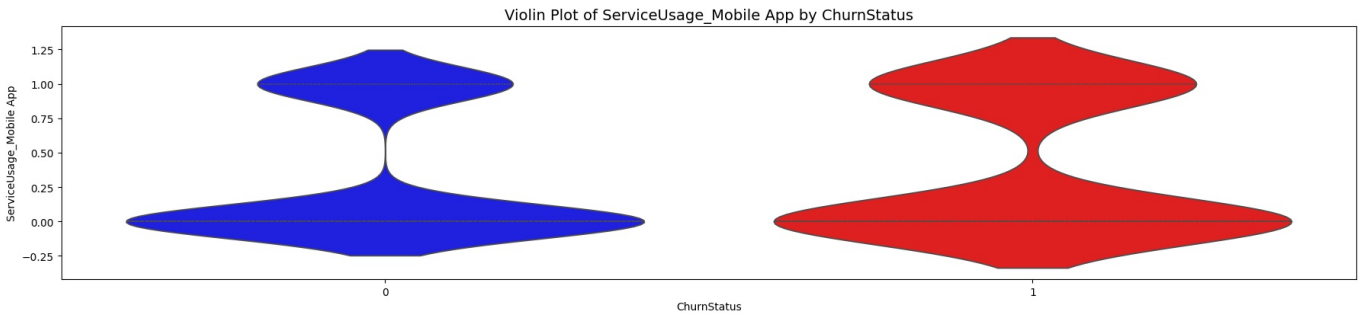


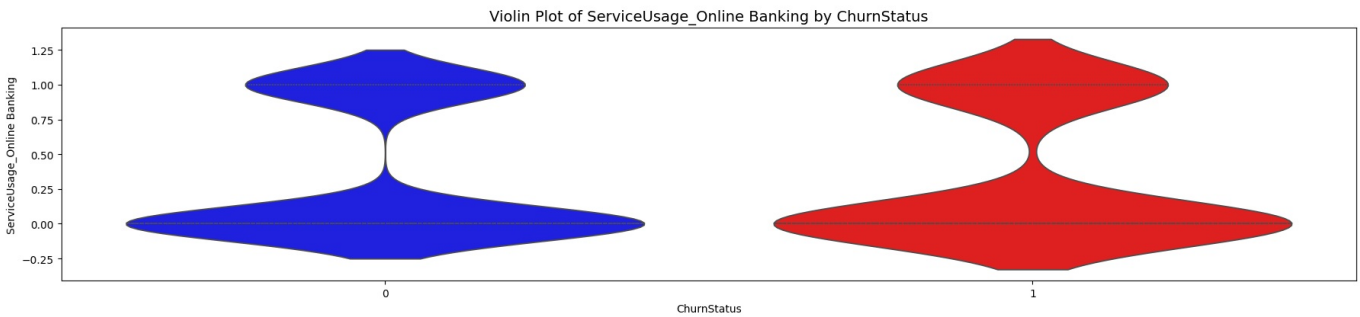


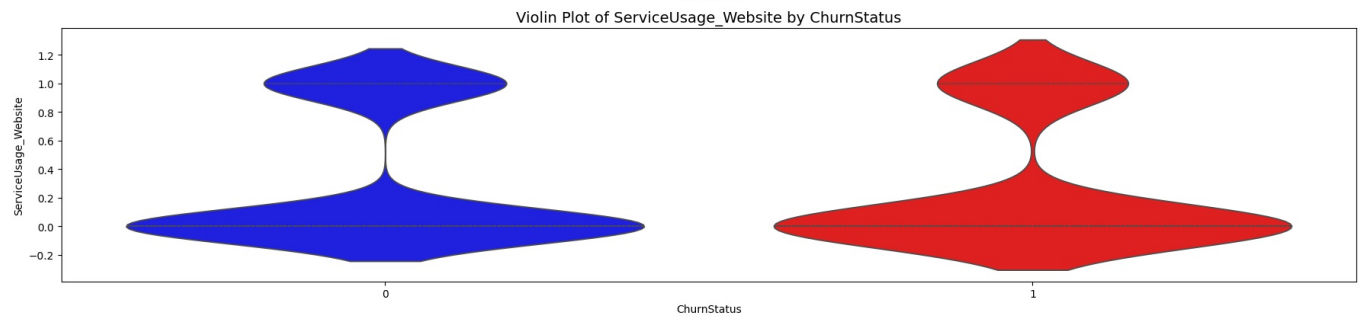












Normalisation and standardisation

because different variable' scales are possiblely far different, we can scale into (0,1)

there are two methods std/rob, we choose rob here.

In [40]:
```
from sklearn.preprocessing import StandardScaler, RobustScaler

std_scaler = StandardScaler()
rob_scaler = RobustScaler()

num_cols = merged_df.drop(["CustomerID"],axis = 1).columns

scaled_df = merged_df.copy()

for i, col in enumerate(num_cols):
    #if col == "CustomerID": continue
    scaled_df['scaled_'+col] = rob_scaler.fit_transform(scaled_df[col].values.reshape(-1,1))
    scaled_df.drop([col], axis=1, inplace=True)
```

In [41]:
```
scaled_df
```

Out[41]:

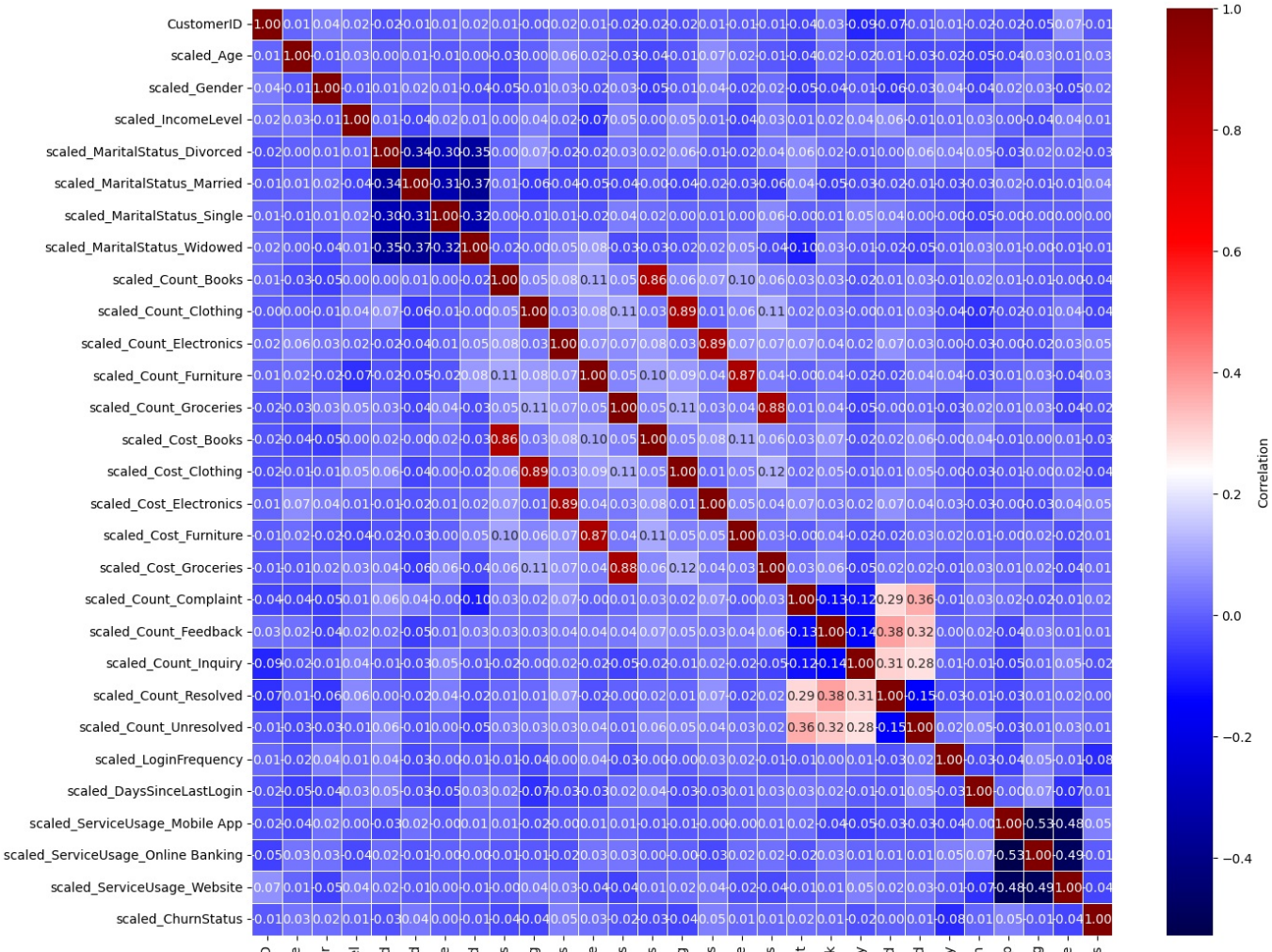
	CustomerID	scaled_Age	scaled_Gender	scaled_IncomeLevel	scaled_MaritalStatus_Divorced	scaled_MaritalStatus_Married	sc
0	1	0.730769	1.0	0.0	0.0	0.0	
1	2	0.846154	1.0	0.0	0.0	1.0	
2	3	-0.961538	1.0	0.0	0.0	0.0	
3	4	-0.846154	1.0	0.0	0.0	0.0	
4	5	-0.846154	1.0	0.5	1.0	0.0	
...	...	...	...	...	...	...	
995	996	0.423077	0.0	0.0	0.0	0.0	
996	997	-0.923077	1.0	-0.5	0.0	0.0	
997	998	0.153846	1.0	0.0	0.0	1.0	
998	999	-0.769231	1.0	-0.5	0.0	0.0	
999	1000	-0.346154	1.0	0.0	0.0	0.0	

1000 rows × 29 columns

we can also plot the correlations between variables

In [42]:
```
plt.figure(figsize=(15, 13))
sns.heatmap(data=scaled_df.corr(),cmap="seismic", annot=True,fmt=".2f",
            cbar_kws={'label': 'Correlation'}, linewidths=0.5)
```

Out[42]:
```
<Axes: >
```



From this correlation plot, we can observe:

(1) Some variables have strong correlations, such as MaritalStatus, Cost/Count for different products, and ServiceUsage. This could lead to overfitting in our future model development.

(2) ChurnStatus shows an extremely weak relationship with other variables, suggesting that additional transformations or operations on the variables might be necessary.

Prepare data

We need to prepare the training data now because the test data should remain blinded until the final step.

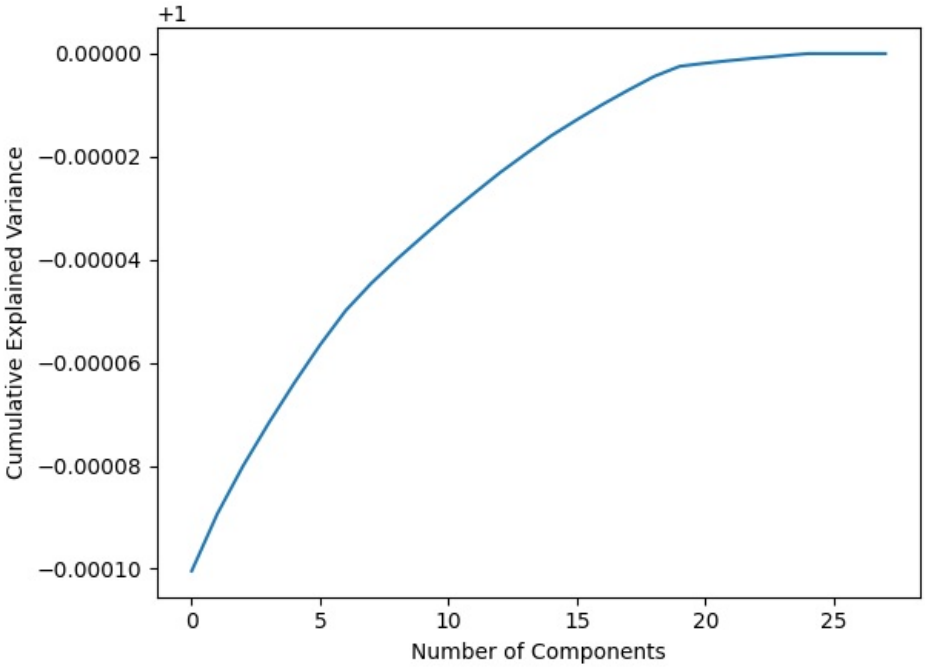
In [44]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
X = scaled_df.drop('scaled_ChurnStatus', axis=1)
y = scaled_df['scaled_ChurnStatus']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify= y, random_state=42) # random
train_data = pd.concat([X_train, y_train], axis=1)
test_data = pd.concat([X_test, y_test], axis=1)
```

In [45]:
```
X = train_data.drop('scaled_ChurnStatus', axis=1)
y = train_data['scaled_ChurnStatus']
```

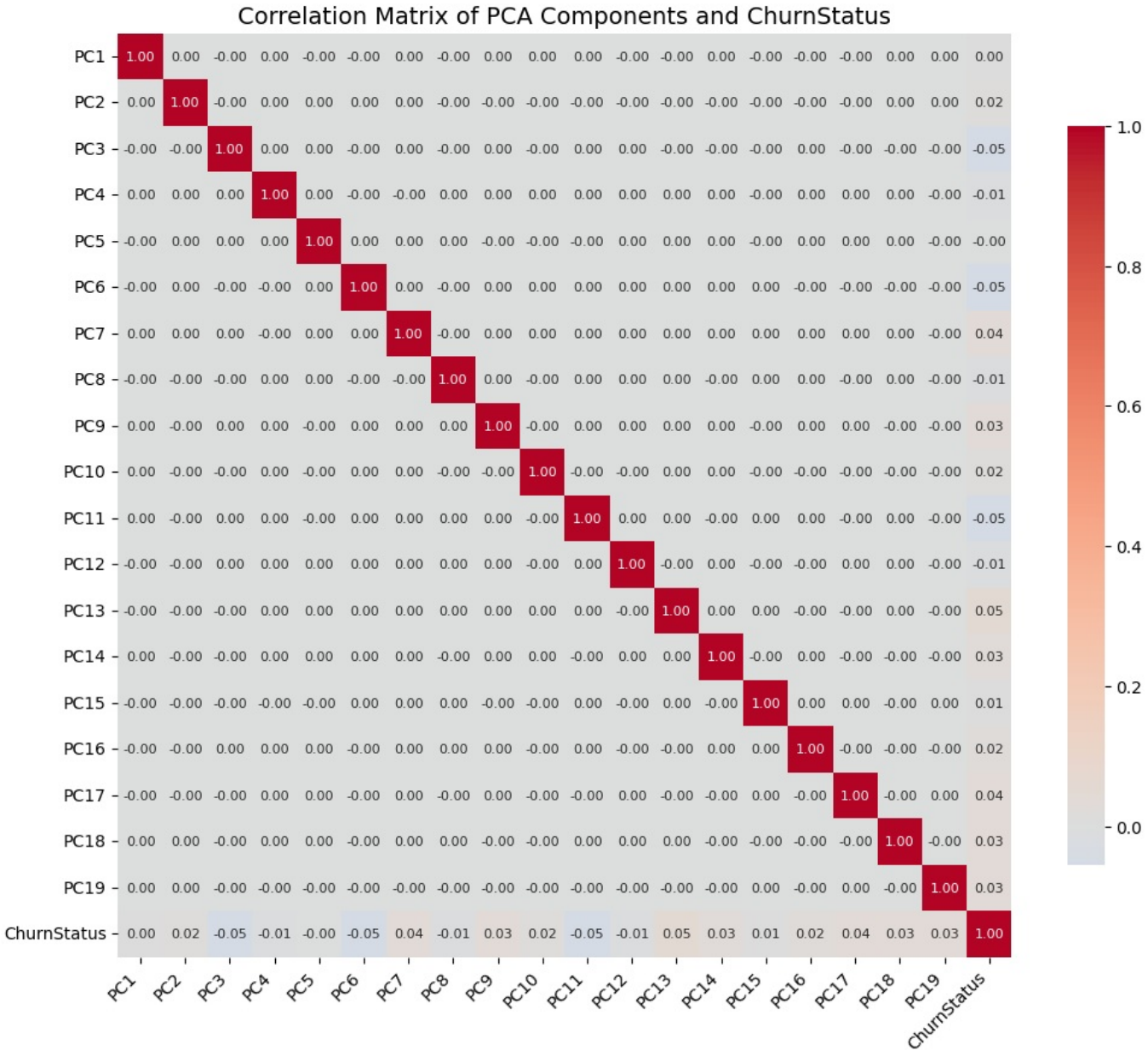
PCA techniques are used here to reduce dimensions, and we can see that almost all variance can be explained when the number of components exceeds 19.

In [49]:
```
from sklearn.decomposition import PCA
imp
X_reduced_pca = PCA(n_components=19, random_state=42).fit_transform(X.values)
```

In [50]:
```
pca = PCA().fit(X)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
```

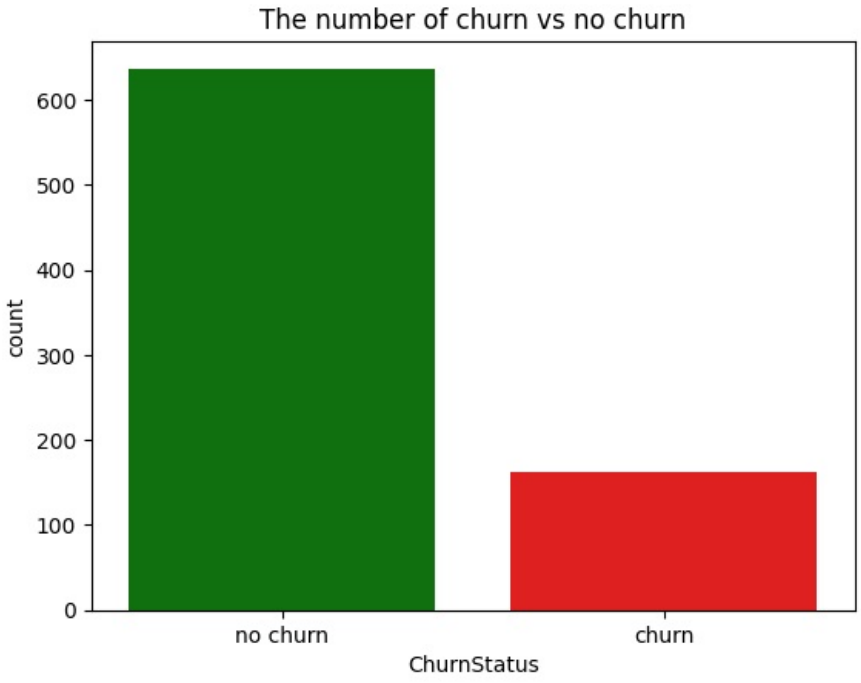


In [51]:
```
pca_df = pd.DataFrame(X_reduced_pca, columns=[f'PC{i+1}' for i in range(X_reduced_pca.shape[1])])
pca_df['ChurnStatus'] = y.values

correlation_matrix = pca_df.corr()

plt.figure(figsize=(12, 10))
sns.heatmap(
    correlation_matrix,
    annot=True,
    fmt=".2f",
    cmap='coolwarm',
    center=0,
    cbar_kws={'shrink': 0.8},
    annot_kws={"size": 8}
)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)
plt.title('Correlation Matrix of PCA Components and ChurnStatus', fontsize=14)
plt.show()
```



From the correlation plot, it appears that the correlations with other variables are still small. Perhaps we can use undersampling techniques to prepare the training data.

In [52]:
```
sns.countplot(x=pca_df['ChurnStatus'].map({0: 'no churn', 1: 'churn'}), hue=pca_df['ChurnStatus'].map({0: 'no cl
plt.title("The number of churn vs no churn")
plt.show()
```



Resampling Techniques

To better recognize the characteristics of the churn sample, we use the undersampling method to create balanced samples.

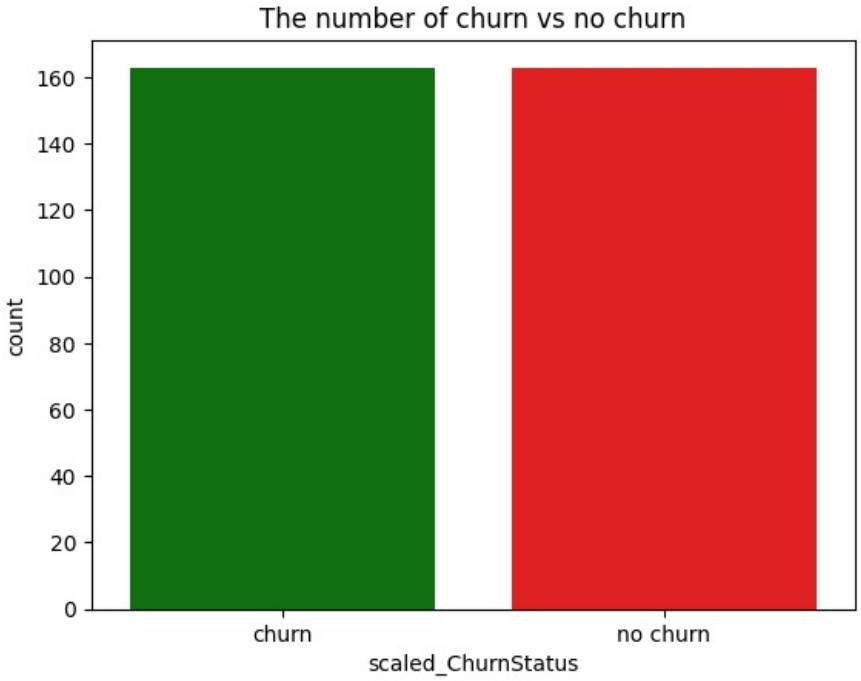
In [54]:
```
train_data = train_data.sample(frac=1)

# amount of fraud classes rows.
train_data_churn_df = train_data.loc[train_data['scaled_ChurnStatus'] == 1]
train_data_non_churn_df = train_data.loc[train_data['scaled_ChurnStatus'] == 0][:train_data['scaled_ChurnStatus

train_data_balanced = pd.concat([train_data_churn_df, train_data_non_churn_df])

# Shuffle dataframe rows
train_data_df_balanced = train_data_balanced.sample(frac=1, random_state=42)

train_data_df_balanced
```

Out[54]:

	CustomerID	scaled_Age	scaled_Gender	scaled_IncomeLevel	scaled_MaritalStatus_Divorced	scaled_MaritalStatus_Married	sc
871	872	-0.153846	1.0	-0.5	0.0	0.0	
60	61	0.423077	1.0	0.0	0.0	0.0	
304	305	0.961538	1.0	-0.5	0.0	1.0	
281	282	-0.038462	1.0	-0.5	0.0	0.0	
477	478	0.884615	1.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	
318	319	-0.884615	0.0	0.5	1.0	0.0	
805	806	0.615385	1.0	-0.5	0.0	0.0	
505	506	0.346154	0.0	0.0	0.0	0.0	
248	249	0.076923	0.0	0.0	0.0	0.0	
561	562	0.807692	1.0	0.0	0.0	1.0	

326 rows × 29 columns

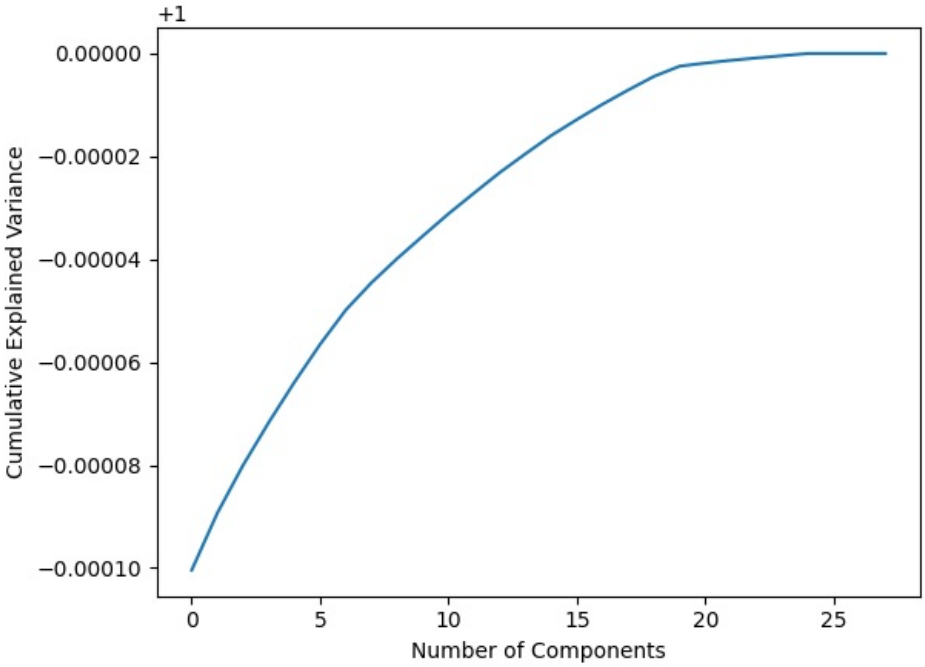
In [58]:
```
sns.countplot(x=train_data_balanced['scaled_ChurnStatus'].map({0: 'no churn', 1: 'churn'}), hue=train_data_balan
plt.title("The number of churn vs no churn")
plt.show()
```



In [59]:
```
X_balanced = train_data_balanced.drop('scaled_ChurnStatus', axis=1)
y_balanced = train_data_balanced['scaled_ChurnStatus']
```

In [60]:
```
ca = PCA().fit(X_balanced)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
```

```
plt.show()
```



In [62]:
```
pca = PCA(n_components=19, random_state=42)
Xb_reduced_pca = pca.fit_transform(X_balanced.values)
```

(19, 28)

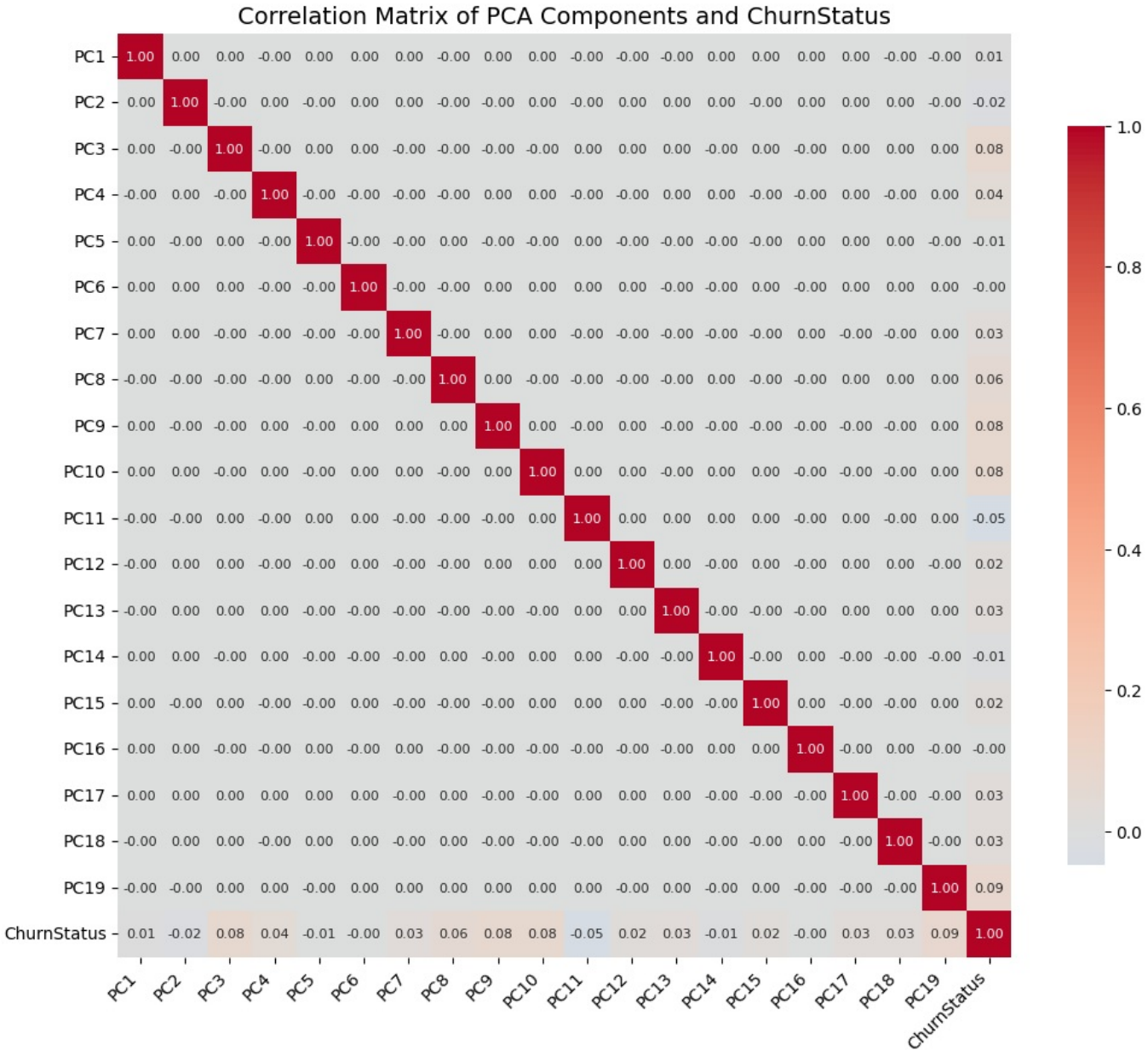
In [64]:
```
pca_df_balanced = pd.DataFrame(Xb_reduced_pca, columns=[f'PC{i+1}' for i in range(Xb_reduced_pca.shape[1])])
pca_df_balanced['ChurnStatus'] = y_balanced.values
pca_df_balanced.sample(frac=1, random_state=42)
```

Out[64]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	P
234	368.286083	1.820903	2.314741	-1.118311	-0.816468	0.012268	-0.489836	-1.321363	-0.445955	-0.028330	-0.114339	-0.444
110	-442.714653	-0.562897	1.422026	0.079733	0.396563	-0.681496	-0.256787	0.729890	0.492817	-0.192466	0.642643	-0.801
249	-198.714581	-1.575127	0.089765	-0.048746	-0.988521	-0.512616	-0.261532	-0.562096	0.419140	0.872340	-0.447279	0.011
9	-221.714612	0.937770	1.324516	0.127355	-0.512729	-0.354595	-0.127648	0.277680	-0.672376	0.587234	-0.966952	-0.130
93	-25.714401	-0.174221	-0.450639	0.572094	1.644786	-1.225202	-0.095261	-0.636833	0.109632	0.520321	0.324443	0.547
...	...	...	...	...	...	...	...	...	...	...	...	
188	-184.715177	-1.320225	0.097349	0.018833	-0.266051	1.123860	-0.607500	-0.562081	-0.835230	-0.299538	-0.042674	0.687
71	302.284907	-1.009477	0.408383	-0.305248	0.247043	0.327415	-0.618765	0.595964	0.036636	-0.567483	-0.230474	0.197
106	2.285040	-0.436316	-0.965179	-0.730000	-0.555381	0.054865	1.325756	-0.700621	0.446788	-0.604096	-1.198569	-0.650
270	-254.714723	-1.263265	0.119833	0.099997	-0.081802	1.391070	0.410745	-0.356467	0.092851	-0.350235	0.580466	0.127
102	58.285224	0.818832	-0.841914	1.172032	-0.945193	-0.465264	-0.649896	-0.436040	-0.452893	-0.420522	-0.354393	0.292

326 rows × 20 columns

In [65]:
```
correlation_matrix_balanced = pca_df_balanced.corr()


plt.figure(figsize=(12, 10))
sns.heatmap(
    correlation_matrix_balanced,
    annot=True,
    fmt=".2f",
    cmap='coolwarm',
    center=0,
    cbar_kws={'shrink': 0.8},
    annot_kws={"size": 8}
)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)
plt.title('Correlation Matrix of PCA Components and ChurnStatus', fontsize=14)
plt.show()
```



From correlation matrix, correlation between PC3, PC8, PC9, PC11, PC19 and ChurnStatus >=0.05

Extract the components (or loadings) from a fitted PCA model, and mainly focus on these varibales with large correlation

In [66]:
```
loadings = pca.components_
print(loadings.shape)
feature_names = X_balanced.columns
loading_matrix = pd.DataFrame(loadings.T, index=feature_names, columns=[f'PC{i+1}' for i in range(19)])
print(loading_matrix)
```

```
(19, 28)
                                     PC1       PC2       PC3       PC4  \
CustomerID                      1.000000 -0.000137 -0.000114 -0.000013
scaled_Age                      0.000098  0.060505  0.075789  0.065558
scaled_Gender                   0.000052  0.001993  0.040571  0.073400
scaled_IncomeLevel             -0.000038  0.057591  0.014269 -0.022360
scaled_MaritalStatus_Divorced  -0.000028 -0.005401 -0.047052 -0.065677
scaled_MaritalStatus_Married   -0.000036 -0.060669  0.042699 -0.022387
scaled_MaritalStatus_Single    -0.000020  0.079098  0.025355 -0.015990
scaled_MaritalStatus_Widowed    0.000084 -0.013028 -0.021003  0.104054
scaled_Count_Books              0.000017  0.291482 -0.008267  0.047161
scaled_Count_Clothing           0.000072  0.164731 -0.276250 -0.092733
scaled_Count_Electronics        0.000105  0.290390  0.378896 -0.097904
scaled_Count_Furniture          0.000064  0.212584 -0.070892  0.490572
scaled_Count_Groceries         -0.000008  0.248646 -0.254594 -0.166050
scaled_Cost_Books               0.000009  0.393061  0.009651  0.056667
scaled_Cost_Clothing            0.000023  0.256011 -0.384351 -0.150497
scaled_Cost_Electronics         0.000168  0.432987  0.624948 -0.197520
scaled_Cost_Furniture          -0.000016  0.311069 -0.059390  0.685167
scaled_Cost_Groceries           0.000058  0.337920 -0.351827 -0.277045
scaled_Count_Complaint         -0.000013 -0.000142  0.021741 -0.065316
scaled_Count_Feedback           0.000011  0.141898 -0.062483 -0.126074
scaled_Count_Inquiry           -0.000136  0.087763 -0.064017 -0.043202
scaled_Count_Resolved          -0.000113  0.148529  0.009185 -0.210207
```

```
scaled_Count_Unresolved            -0.000026  0.080990 -0.113944 -0.024386
scaled_LoginFrequency              -0.000056 -0.016955  0.049433 -0.073818
scaled_DaysSinceLastLogin           0.000060  0.037783 -0.045753  0.041835
scaled_ServiceUsage_Mobile App      0.000122 -0.027346  0.061389  0.009845
scaled_ServiceUsage_Online Banking -0.000209 -0.015271 -0.024530 -0.000537
scaled_ServiceUsage_Website         0.000087  0.042617 -0.036859 -0.009307

                                          PC5       PC6       PC7  \
CustomerID                        -1.968464e-07  0.000095 -0.000062
scaled_Age                        -1.708096e-01 -0.026466  0.070656
scaled_Gender                     -4.539478e-02 -0.112209 -0.003945
scaled_IncomeLevel                 4.443116e-03  0.030022 -0.060390
scaled_MaritalStatus_Divorced     -5.286797e-02  0.055926  0.116458
scaled_MaritalStatus_Married      -2.926639e-02 -0.041073 -0.066528
scaled_MaritalStatus_Single        2.894114e-02 -0.053416 -0.032420
scaled_MaritalStatus_Widowed       5.319322e-02  0.038564 -0.017510
scaled_Count_Books                -3.204219e-02  0.406081 -0.199512
scaled_Count_Clothing              4.046243e-01 -0.118516  0.000695
scaled_Count_Electronics           5.013876e-02 -0.134924  0.050593
scaled_Count_Furniture             3.118444e-02 -0.151427  0.041786
scaled_Count_Groceries            -3.661153e-01 -0.205084 -0.009218
scaled_Cost_Books                 -8.719271e-02  0.591611 -0.300914
scaled_Cost_Clothing               5.748955e-01 -0.147422  0.005827
scaled_Cost_Electronics            1.031947e-01 -0.207337  0.140291
scaled_Cost_Furniture             -7.140898e-03 -0.194597  0.072285
scaled_Cost_Groceries             -4.937484e-01 -0.270358 -0.041752
scaled_Count_Complaint             8.757405e-03  0.091972  0.230484
scaled_Count_Feedback              1.041760e-02  0.115008  0.359690
scaled_Count_Inquiry               1.005093e-01  0.127419 -0.053995
scaled_Count_Resolved              1.307213e-01  0.042219 -0.204092
scaled_Count_Unresolved           -1.103706e-02  0.292180  0.740271
scaled_LoginFrequency              1.026445e-01 -0.040848 -0.056467
scaled_DaysSinceLastLogin         -9.917557e-02  0.196956  0.176234
scaled_ServiceUsage_Mobile App    -6.102621e-02 -0.109581  0.028596
scaled_ServiceUsage_Online Banking -5.042064e-02  0.003420 -0.017932
scaled_ServiceUsage_Website        1.114468e-01  0.106161 -0.010664

                                       PC8       PC9      PC10      PC11  \
CustomerID                        0.000185 -0.000076 -0.000206 -0.000096
scaled_Age                        0.001233 -0.010016  0.259530 -0.123011
scaled_Gender                     0.002775 -0.109954 -0.014575  0.055245
scaled_IncomeLevel                0.037925 -0.055629 -0.006417  0.187502
scaled_MaritalStatus_Divorced    -0.008092 -0.119324  0.016573  0.128887
scaled_MaritalStatus_Married      0.014885  0.137077 -0.117767 -0.177033
scaled_MaritalStatus_Single       0.065542  0.020563  0.011645 -0.015213
scaled_MaritalStatus_Widowed     -0.072335 -0.038316  0.089549  0.063359
scaled_Count_Books               -0.117143  0.093166 -0.000243  0.063234
scaled_Count_Clothing            -0.129706  0.029439  0.040451 -0.048924
scaled_Count_Electronics         -0.074935 -0.029030 -0.005740 -0.067769
scaled_Count_Furniture            0.107579  0.046922 -0.026834  0.019156
scaled_Count_Groceries           -0.031987  0.014397 -0.059225  0.033090
scaled_Cost_Books                -0.132892  0.059798  0.019643  0.074340
scaled_Cost_Clothing             -0.203406 -0.004877  0.016762 -0.035799
scaled_Cost_Electronics          -0.132570 -0.055546 -0.068399 -0.020471
scaled_Cost_Furniture             0.175617 -0.000788 -0.057752  0.000641
scaled_Cost_Groceries            -0.025251  0.036393 -0.072658  0.039425
scaled_Count_Complaint            0.155017  0.556030 -0.298077 -0.145306
scaled_Count_Feedback             0.347114 -0.304685  0.482479  0.145022
scaled_Count_Inquiry              0.208116 -0.008401 -0.266476 -0.186677
scaled_Count_Resolved             0.756645  0.101963  0.103279 -0.219647
scaled_Count_Unresolved          -0.046397  0.140981 -0.185353  0.032685
scaled_LoginFrequency             0.260989 -0.137413 -0.468184  0.726489
scaled_DaysSinceLastLogin         0.006057 -0.368005 -0.058614 -0.175589
scaled_ServiceUsage_Mobile App    0.001113  0.440192  0.278836  0.212138
scaled_ServiceUsage_Online Banking  0.024377 -0.380844 -0.377445 -0.351838
scaled_ServiceUsage_Website      -0.025490 -0.059348  0.098609  0.139700

                                      PC12      PC13      PC14      PC15  \
CustomerID                       -0.000052  0.000019  0.000100 -0.000088
scaled_Age                       -0.095146  0.771708 -0.307046  0.372924
scaled_Gender                     0.075936  0.157113  0.490573  0.168710
scaled_IncomeLevel                0.123961 -0.107294 -0.013230  0.168037
scaled_MaritalStatus_Divorced     0.071269 -0.007446 -0.021978 -0.073362
scaled_MaritalStatus_Married     -0.018730  0.179070  0.089331 -0.121510
scaled_MaritalStatus_Single      -0.096768  0.031477  0.091020  0.135016
scaled_MaritalStatus_Widowed      0.044230 -0.203101 -0.158373  0.059856
scaled_Count_Books                0.023018  0.102913  0.083692 -0.033669
scaled_Count_Clothing             0.053554  0.066266 -0.018558  0.011664
scaled_Count_Electronics          0.023004 -0.044516 -0.030144 -0.030195
scaled_Count_Furniture            0.005530 -0.002111  0.001959 -0.089358
scaled_Count_Groceries           -0.034836 -0.078429  0.008010 -0.017461
scaled_Cost_Books                 0.076694  0.076307  0.072952 -0.121830
scaled_Cost_Clothing              0.161453  0.151389  0.002240  0.041184
```

```
scaled_Cost_Electronics            -0.020959 -0.086050 -0.014369  0.013947
scaled_Cost_Furniture              -0.020384 -0.069994 -0.048288 -0.032371
scaled_Cost_Groceries              -0.045360 -0.062701 -0.073001  0.014532
scaled_Count_Complaint              0.235456  0.052415 -0.312277 -0.152957
scaled_Count_Feedback               0.009074  0.053358  0.141124 -0.302045
scaled_Count_Inquiry               -0.376585 -0.129203  0.293901  0.522509
scaled_Count_Resolved               0.060621 -0.047561 -0.051443  0.011443
scaled_Count_Unresolved            -0.192676  0.024131  0.174191  0.056065
scaled_LoginFrequency               0.089355  0.257363 -0.103096  0.052546
scaled_DaysSinceLastLogin           0.565552 -0.224019 -0.254644  0.440003
scaled_ServiceUsage_Mobile App      0.343415 -0.041562  0.299536  0.229489
scaled_ServiceUsage_Online Banking  0.119978  0.222856  0.128335 -0.298493
scaled_ServiceUsage_Website        -0.463392 -0.181293 -0.427871  0.069004

                                       PC16      PC17      PC18      PC19
CustomerID                          0.000073 -0.000023  0.000060 -0.000053
scaled_Age                         -0.009071 -0.091853 -0.026917 -0.127168
scaled_Gender                      -0.318157 -0.133861 -0.505822  0.530021
scaled_IncomeLevel                 -0.018879 -0.004228  0.068846 -0.109768
scaled_MaritalStatus_Divorced       0.647407 -0.195659 -0.463496 -0.081472
scaled_MaritalStatus_Married       -0.144703  0.760486 -0.076207 -0.041746
scaled_MaritalStatus_Single         0.110429 -0.223312  0.630017  0.484731
scaled_MaritalStatus_Widowed       -0.613134 -0.341515 -0.090313 -0.361513
scaled_Count_Books                 -0.006875  0.003727 -0.034729 -0.031449
scaled_Count_Clothing               0.050545 -0.017981 -0.033626 -0.030632
scaled_Count_Electronics           -0.014828 -0.048687 -0.041355 -0.005686
scaled_Count_Furniture              0.023432  0.014107 -0.008784 -0.014822
scaled_Count_Groceries             -0.053162 -0.005755 -0.001947 -0.022844
scaled_Cost_Books                  -0.003472 -0.000153  0.021723  0.058487
scaled_Cost_Clothing               -0.021888  0.047505 -0.010263  0.019487
scaled_Cost_Electronics             0.010045  0.043356 -0.027848 -0.062455
scaled_Cost_Furniture               0.095205  0.062734 -0.008935 -0.019640
scaled_Cost_Groceries              -0.057691  0.016083 -0.011306  0.001320
scaled_Count_Complaint             -0.081299 -0.180476 -0.120833  0.216204
scaled_Count_Feedback              -0.111429  0.115597  0.093529 -0.018617
scaled_Count_Inquiry                0.089146  0.002732 -0.040858 -0.262991
scaled_Count_Resolved              -0.024802 -0.066364 -0.123776 -0.008101
scaled_Count_Unresolved            -0.078781  0.004217  0.055614 -0.057303
scaled_LoginFrequency              -0.064691  0.076806  0.073689 -0.051257
scaled_DaysSinceLastLogin           0.009725  0.241538  0.029123  0.166141
scaled_ServiceUsage_Mobile App      0.091661  0.050751  0.087038 -0.179973
scaled_ServiceUsage_Online Banking -0.034090 -0.199686  0.108475 -0.141974
scaled_ServiceUsage_Website        -0.057572  0.148934 -0.195513  0.321947
```

In [71]:

```python
feature_names = X_balanced.columns

loading_matrix = pd.DataFrame(loadings.T, index=feature_names, columns=[f'PC{i+1}' for i in range(loadings.shape

pcs_to_check = ['PC3', 'PC8', 'PC9', 'PC10', 'PC11', 'PC19']

top_features_per_pc = {}
for pc in pcs_to_check:
    top_features = loading_matrix[pc].abs().sort_values(ascending=False).head()
    top_features_per_pc[pc] = top_features

top_features_df = pd.DataFrame(top_features_per_pc)
print(top_features_df)
```

```
                                     PC3       PC8       PC9      PC10  \
scaled_Cost_Clothing            0.384351  0.203406       NaN       NaN
scaled_Cost_Electronics         0.624948       NaN       NaN       NaN
scaled_Cost_Groceries           0.351827       NaN       NaN       NaN
scaled_Count_Clothing           0.276250       NaN       NaN       NaN
scaled_Count_Complaint               NaN       NaN  0.556030  0.298077
scaled_Count_Electronics        0.378896       NaN       NaN       NaN
scaled_Count_Feedback                NaN  0.347114  0.304685  0.482479
scaled_Count_Inquiry                 NaN  0.208116       NaN       NaN
scaled_Count_Resolved                NaN  0.756645       NaN       NaN
scaled_DaysSinceLastLogin            NaN       NaN  0.368005       NaN
scaled_Gender                        NaN       NaN       NaN       NaN
scaled_IncomeLevel                   NaN       NaN       NaN       NaN
scaled_LoginFrequency                NaN  0.260989       NaN  0.468184
scaled_MaritalStatus_Single          NaN       NaN       NaN       NaN
scaled_MaritalStatus_Widowed         NaN       NaN       NaN       NaN
scaled_ServiceUsage_Mobile App       NaN       NaN  0.440192  0.278836
scaled_ServiceUsage_Online Banking   NaN       NaN  0.380844  0.377445
scaled_ServiceUsage_Website          NaN       NaN       NaN       NaN

                                    PC11      PC19
scaled_Cost_Clothing                 NaN       NaN
scaled_Cost_Electronics              NaN       NaN
scaled_Cost_Groceries                NaN       NaN
scaled_Count_Clothing                NaN       NaN
scaled_Count_Complaint               NaN       NaN
scaled_Count_Electronics             NaN       NaN
scaled_Count_Feedback                NaN       NaN
scaled_Count_Inquiry                 NaN  0.262991
scaled_Count_Resolved           0.219647       NaN
scaled_DaysSinceLastLogin            NaN       NaN
scaled_Gender                        NaN  0.530021
scaled_IncomeLevel              0.187502       NaN
scaled_LoginFrequency           0.726489       NaN
scaled_MaritalStatus_Single          NaN  0.484731
scaled_MaritalStatus_Widowed         NaN  0.361513
scaled_ServiceUsage_Mobile App  0.212138       NaN
scaled_ServiceUsage_Online Banking  0.351838       NaN
scaled_ServiceUsage_Website          NaN  0.321947
```

The number of times the main variables appear in the loading matrix

In [72]:
```
feature_counts = top_features_df.notna().sum(axis=1)
feature_counts = feature_counts[feature_counts > 0]
print(feature_counts)
```

```
scaled_Cost_Clothing                2
scaled_Cost_Electronics             1
scaled_Cost_Groceries               1
scaled_Count_Clothing               1
scaled_Count_Complaint              2
scaled_Count_Electronics            1
scaled_Count_Feedback               3
scaled_Count_Inquiry                2
scaled_Count_Resolved               2
scaled_DaysSinceLastLogin           1
scaled_Gender                       1
scaled_IncomeLevel                  1
scaled_LoginFrequency               3
scaled_MaritalStatus_Single         1
scaled_MaritalStatus_Widowed        1
scaled_ServiceUsage_Mobile App      3
scaled_ServiceUsage_Online Banking  3
scaled_ServiceUsage_Website         1
dtype: int64
```

Summary

(1) Preliminary Statistics and Visualization

Preliminary statistics and visualizations were depicted to provide an overview of the dataset and the relationships among key variables.

Certain principal components showed meaningful correlations with `ChurnStatus`, indicating important underlying patterns that could influence churn prediction.

Notable variables contributing to these components include `Count_Feedback`, `LoginFrequency`, `ServiceUsage_Mobile App`, and `ServiceUsage_Online`, among others.

(2) Training and Test Data Sets Preparation

The training dataset was carefully constructed using a balanced sample to address class imbalance issues, ensuring that both churned and non-churned samples were adequately represented.

Additionally, a test dataset comprising 20% of the data was created using a fixed random seed to guarantee reproducibility and consistency in model evaluation.

This approach provides a robust framework for training and testing predictive models.