

MATH6006 Statistical Methods(24-25)

Optimizing Consumer Behavior Predictions Through Statistical Feature Engineering

Zhe Guan¹

¹The University of Southampton, Southampton, United Kingdom E-mail: zg2u24@soton.ac.uk.

Keywords: Customer Churn Prediction, Statistical Methods, Markov Chains, Machine Learning

Abstract

Consumer behavior prediction is a critical challenge in data-driven marketing and customer relationship management. This project conducts a feature engineering framework utilizing various statistical methods to enhance the predictive performance of machine learning models in churn analysis. A combination of feature transformations and binning techniques enhances feature-target correlations between features and the target variable. Markov Chain state transition modeling captures dynamic customer interaction patterns over time. Hypothesis testing and correlation analysis help identify the most influential variables. The XGBoost algorithm is applied to compare model performance between datasets with statistically enhanced features and those without feature engineering.

Data Source

The consumer data used in this project is sourced from the Lloyds Bank Simulation on the Forage platform¹. This dataset is utilised only for academic and research purposes, with no commercial intent.

Contents

1	Introduction	2
1.1	Project Introduction	2
1.2	Data Introduction	2
2	Feature Engineering	2
2.1	Customer Demographics	2
2.2	Customer Interaction	3
2.3	Transaction History	5
2.4	Login Activity	6
3	Classifier Training	7
4	Conclusion	8
5	Appendix	9
5.1	Codes and additional plots	9
5.2	Simulation Certificate	81

¹<https://www.theforge.com/simulations/lloyds-banking-group/data-science-fpey>

1. Introduction

1.1. Project Introduction

Customer churn occurs when a customer discontinues their relationship with a company [1]. As a key indicator of business sustainability, the churn rate is now regarded as equally significant as financial profitability in assessing growth, and businesses try various methods to minimise churn to maintain customer retention [2]. Traditional churn prediction models often overlook the impact of statistical feature engineering. This study aims to validate the impact of statistical transformations on machine learning models.

This project applies various statistical methods to enhance feature engineering and consumer behaviour prediction. Techniques include Z-score transformations [3] to normalise and highlight feature significance, Markov chains [4] for modelling customer state transitions, Kernel Density Estimation (KDE) [5] distributions and bin separations to improve feature-target correlations, rolling-window aggregation techniques [6] to capture short-term trends in customer behaviour. Student's t-test [7] is used to compare the mean values of continuous features between churned and retained customers, while the χ^2 test [8] evaluates the association between categorical features and customer churn. Additionally, correlation analysis [9] is conducted to identify key predictive variables.

Two classifiers are designed for comparative analysis to evaluate the impact of statistical feature engineering and machine learning techniques. The first model is a baseline XGBoost classifier [10] trained on raw data without any feature enhancements, serving as a benchmark. The second model integrates enhanced features to optimise predictive performance. Additionally, Conditional Tabular GAN (CTGAN) [11] deep learning techniques are used to generate more minority synthetic tabular data.

1.2. Data Introduction

The original dataset consists of four sheets containing information on 1,000 distinct customers, including Customer Demographics, Interaction, Transaction, and Login Activity History.

- The Customer Demographics sheet provides basic customer information, including age, gender, marital status, and income level.
- The Interaction sheet records customer service interactions, including InteractionID, InteractionDate, and ResolutionStatus, with three different interaction types: inquiry, feedback, and complaint, categorised under the InteractionType column.
- The Transaction History sheet documents customer transactions from 2022, containing TransactionID, TransactionDate, AmountSpent, and five different product categories: electronics, groceries, books, clothing, and furniture.
- The Login Activity History is stored in the Online Activity Sheet, which tracks LastLoginDate, LoginFrequency, and ServiceUsage, specifying whether the customer accessed the service via a website or mobile app.

All five sheets are linked through a common identifier, CustomerID. The churn status is stored separately in the ChurnStatus sheet.

2. Feature Engineering

2.1. Customer Demographics

Customer demographics can be categorised into two types: continuous and discrete variables. By analysing the only continuous variable, Age distribution, we can apply appropriate binning techniques to reduce variability and capture meaningful trends within specific age groups.

Sheet Name	Original Features
Customer Demographics	Age, Gender, MaritalStatus, and IncomeLevel.
Interaction	InteractionDate, ResolutionStatus, and InteractionType
Transaction History	TransactionDate, AmountSpent, and ProductCategories
Login Activity History	LastLoginDate, LoginFrequency, and ServiceUsage
Churn Status	Churn Status(1/0)

Table 1. Summary of Data Sheets and Their Original Features.

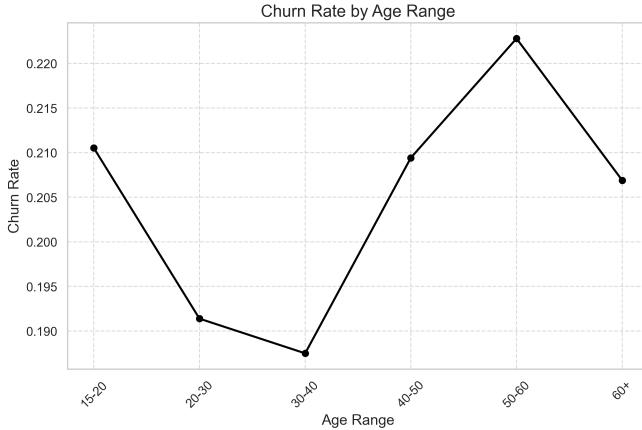


Figure 1. Churn Rate by Age Range with binning.

As shown in Fig 1, it indicates groups in 20-30 and 30-40 have a relatively low churn rate compared to other groups. This trend may be attributed to their stable income levels, which contribute to higher customer retention.

To emphasise this trend among different groups for later analysis, the Z-score is calculated for each group with:

$$Z = \frac{X - \mu}{\sigma} \quad (2.1)$$

where X is the average churn rate in each group, μ and σ are the mean value and standard variance of churn rate.

For effective transformation from categorical to numerical data, one-hot encoding [12] is applied to variables such as Gender, MaritalStatus, and IncomeLevel, which convert each category into a binary vector representation without introducing unintended ordinal relationships.

2.2. Customer Interaction

In the Interaction sheet, three different interaction categories are recorded: feedback, inquiry, and complaint. Each interaction has a corresponding ResolutionStatus, which is either solved or unsolved.

Although the inquiry shows a lower churn rate as shown in Fig 2, it may not be a real situation without considering the ResolutionStatus. For example, successfully resolved inquiries are likely to increase customer retention. To more accurately model this process, a Markov Chain is applied. To avoid the complexity of higher-order chains while preserving key interaction patterns, a combined feature of InteractionType and ResolutionStatus is introduced.

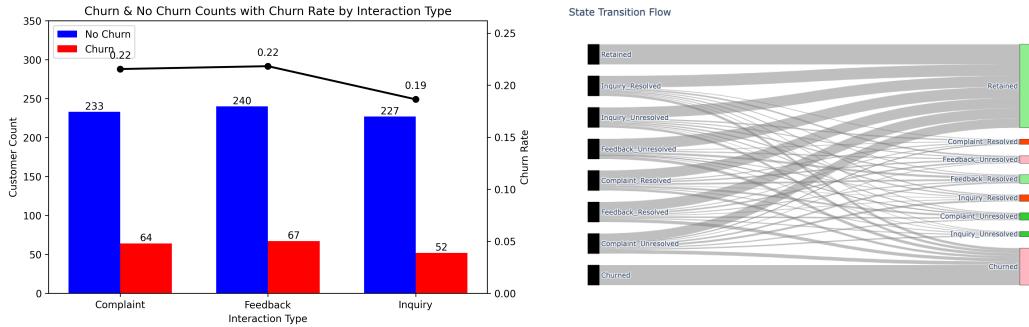


Figure 2. Interaction Type Counts with Churn Rates (left) and state transition Flow (right).

The customer journey begins with a specific type of interaction and progresses through various states based on their interaction types and resolution outcomes. Finally, the customer transitions into one of two absorbing states, either churned or retained, where they remain indefinitely, as shown in Fig 3.

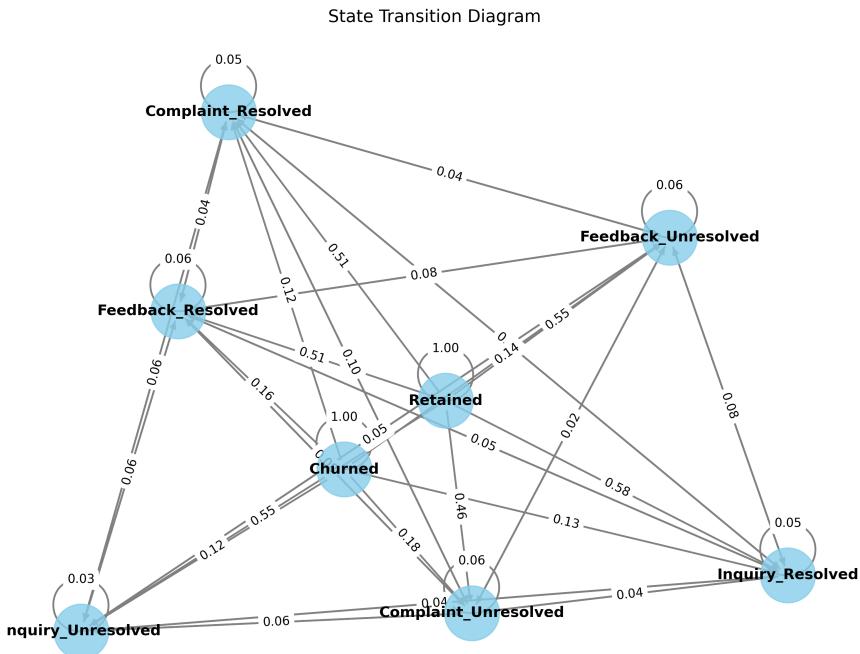


Figure 3. State Trasition Chain with combined features.

The transition matrix can also be calculated as shown in Fig 4, and we can calculate absorption probabilities for the recurrent class with formula 2.2:

$$\begin{aligned} F &= (I - T)^{-1} S \\ &= US \end{aligned} \tag{2.2}$$

where I is the identity matrix, T represents the transition matrix of the transient classes, and S denotes the cells that describe transitions from transient classes to recurrent classes.



Figure 4. Trasition Matrix with combined features.

State	Retention Probability	Churn Probability
Inquiry Resolved	0.8070	0.1930
Inquiry Unresolved	0.8068	0.1932
Feedback Resolved	0.7708	0.2292
Feedback Unresolved	0.7906	0.2094
Complaint Resolved	0.8025	0.1975
Complaint Unresolved	0.7448	0.2552

Table 2. Absorption Probability Matrix (F).

Among all states, Complaint Unresolved has the highest churn probability (25.52%), suggesting that unresolved complaints are a significant risk factor for customer attrition. Conversely, Inquiry Resolved and Inquiry Unresolved show relatively lower churn probabilities (19.30% and 19.32%, respectively). To account for this, a new feature is introduced that assigns a churn probability based on each customer's final interaction type as shown in Tab 2.

Furthermore, churn status is influenced not only by the most recent interaction but also by prior interactions. To capture this relationship, additional features are engineered, such as the frequency of each interaction type, resolution status, and duration between interactions.

2.3. Transaction History

The Transaction Sheet provides insights into hidden consumption behaviours. To analyse customer activity levels, key metrics such as minimum, maximum, total spending, average transaction value, and transaction count are calculated. These metrics help quantify purchase frequency, spending patterns, and overall engagement.

Moreover, analysing customer spending behaviour across different product categories can help uncover valuable insights for churn prediction. A natural approach is to analyse consumer spending patterns across different product categories within a short time window while considering their churn status.

As shown in Fig 5, churned customers have a smaller total spent, and Fig. 6 illustrates the differences in short-time purchasing behaviour between retained and churned customers, showing that churned customers tend to spend less on electronics and clothing. To better capture these patterns, proper binning is applied to categorise customers based on their spending range.

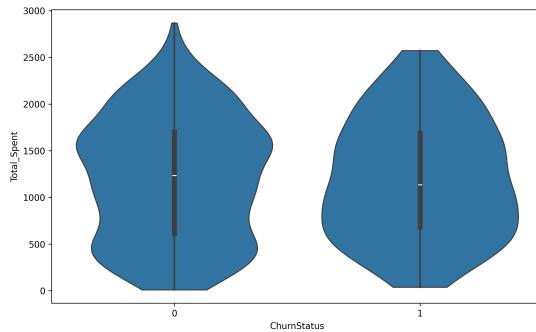


Figure 5. Distribution of total transaction for churned and retained customer.

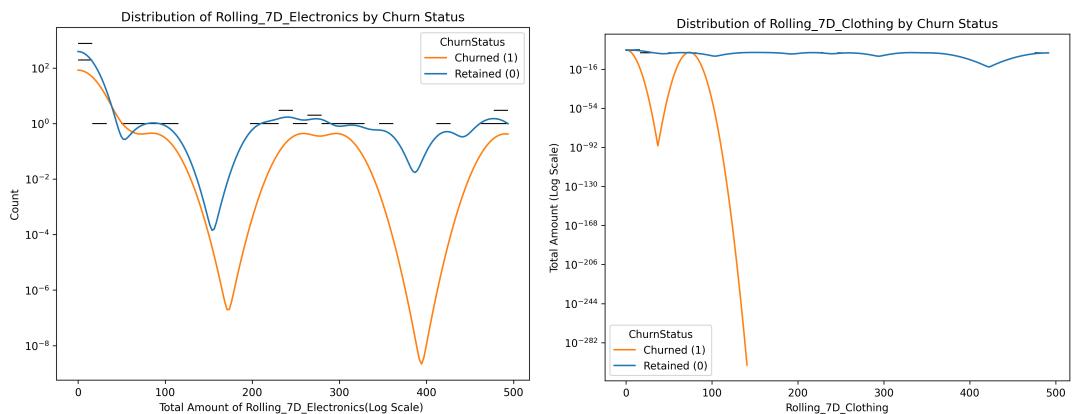


Figure 6. KDE plots for short-term spending patterns in electronics (left) and clothing (right). .

The binning settings for five different products are shown in Tab 3.

2.4. Login Activity

The Login Activity sheet contains three key features: LastLoginDate, LoginFrequency, and ServiceUsage. To transform LastLoginDate into numerical data, we can calculate the difference between each entry and the maximum LastLoginDate in the dataset. Once transformed, we can apply a t-test to determine whether there are significant differences in feature mean values between users with different ChurnStatus.

For category variable ServiceUsage, χ^2 statistics can be calculated to indicate whether there is a relationship with ChurnStatus.

Product Category	bining settings
Furniture	[0, 100, 210, ∞]
Clothing	[0, 10, 40, 150, ∞]
Electronics	[0, 160, 200, 310, 360, 420, 440, ∞]
Books	[0, 80, 220, ∞]
Groceries	[0, 80, 200, 280, 320, 380, 410, ∞]

Table 3. Summary of binning settings for each product. Binning was determined using KDE to optimise distribution separations..

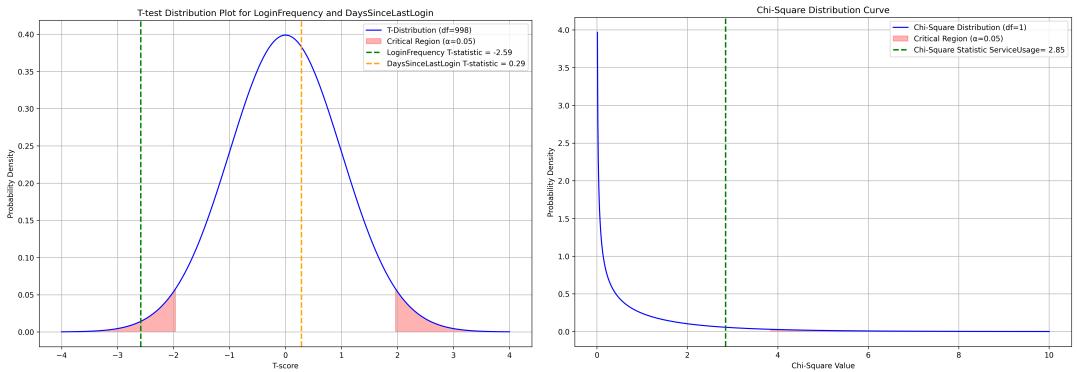


Figure 7. T-test distribution plot for LoginFrequency and DaysSinceLastLogin (left), and χ^2 test result for ServiceUsage (right).

As shown in Fig 7, As shown in Fig. 7, The t-test statistic for login frequency falls within the critical region ($\alpha = 0.05$), leading to the rejection of the null hypothesis that churned and retained customers have the same mean login frequency, indicating that login frequency may be a useful predictor of customer churn. However, DaysSinceLastLogin does not show a significant difference between churned and retained users, indicating that the time since the last login may not be a strong individual indicator of churn. Similarly, the χ^2 test for ServiceUsage suggests that there is no strong relationship between service usage and churn status.

Moreover, LoginFrequency is normalised by dividing each value by the maximum LoginFrequency, providing a relative measure of customer engagement. Additionally, ServiceUsage is transformed using one-hot encoding to effectively represent categorical information.

3. Classifier Training

After preparing the enhanced features introduced above, we can analyse the correlation among the variables, and explore the impact of these features on churn prediction by comparing a baseline XGBoost model with an enhanced model trained on CTGAN-generated synthetic data. To address data imbalance, we generate additional churned user samples at 5x the original size using CTGAN, then merge them with the original dataset. Both models are trained using the same XGBoost hyperparameters.

From Tab 4 and AUC-ROC [13] Fig 8, the enhanced features significantly improve the accuracy of identifying churned customers by slightly sacrificing the accuracy in predicting retained customers,

Fig 9 shows that recent grocery spending (Rolling_7D_Groceries) is the most influential factor in predictions, followed by customer service interactions (Unresolved issues, Complaints) and user engagement with online services (Online Banking, Website Usage). Demographic factors like age and marital status also play a role.

	Churned Customer	Retained Customer
Baseline Model Precision	0.27	0.80
Baseline Recall Rate	0.10	0.93
Feature Enhanced Model Precision	0.94	0.76
Feature Enhanced Model Recall	0.81	0.92

Table 4. Comparison of precision and recall between the baseline and feature-enhanced models for churned and retained customers.

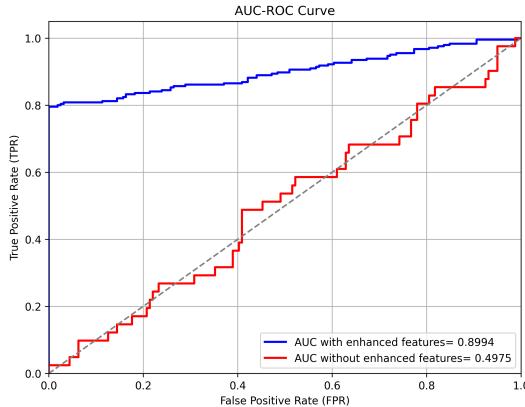


Figure 8. AUC-ROC plot for the baseline and feature-enhanced models.

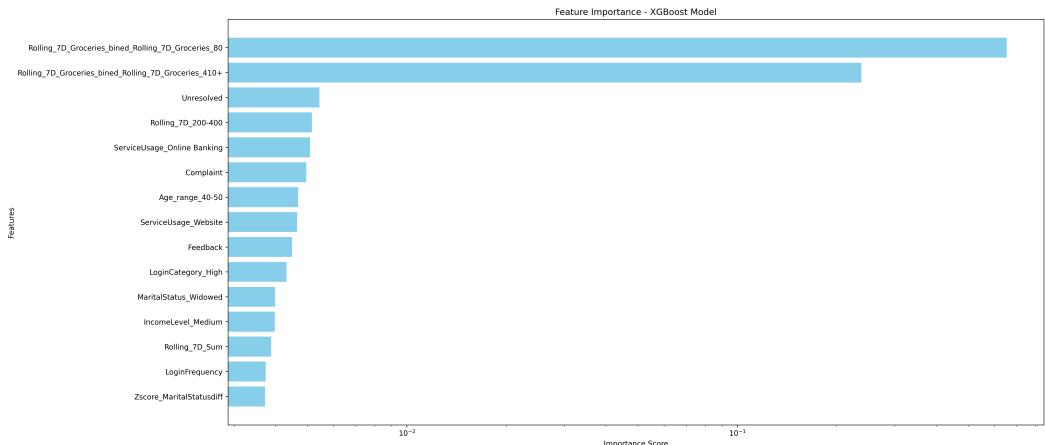


Figure 9. importance rank of top 15 variables in the feature-enhanced model.

4. Conclusion

This study shows the impact of statistical feature engineering in improving customer churn prediction. By applying techniques such as Z-score normalisation, Markov Chains, and so on, we extracted key predictive variables that enhanced model performance. The feature-enhanced XGBoost model, trained with CTGAN-generated synthetic data, showed significant improvements in identifying churned customers compared to the baseline model.

For business aspects, the findings indicate that factors like unresolved complaints, recent spending behaviour, and online engagement play crucial roles in churn prediction. These insights can help

businesses develop targeted retention strategies, reducing customer attrition and improving long-term engagement.

References

- [1] Sanket Agrawal et al. “Customer Churn Prediction Modelling Based on Behavioural Patterns Analysis using Deep Learning”. In: *2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE)*. 2018, pp. 1–6. doi: [10.1109/ICSCEE.2018.8538420](https://doi.org/10.1109/ICSCEE.2018.8538420).
- [2] Muhammad Raza Khan et al. “Behavioral Modeling for Churn Prediction: Early Indicators and Accurate Predictors of Custom Defection and Loyalty”. In: *2015 IEEE International Congress on Big Data*. IEEE, June 2015, pp. 677–680. doi: [10.1109/bigdatacongress.2015.107](https://doi.org/10.1109/bigdatacongress.2015.107). URL: <http://dx.doi.org/10.1109/BigDataCongress.2015.107>.
- [3] Saralees Nadarajah. “A generalized normal distribution”. In: *Journal of Applied Statistics* 32.7 (2005), pp. 685–694. doi: [10.1080/02664760500079464](https://doi.org/10.1080/02664760500079464). eprint: <https://doi.org/10.1080/02664760500079464>. URL: <https://doi.org/10.1080/02664760500079464>.
- [4] Ka Chan, C. Lenard, and Terence Mills. “An Introduction to Markov Chains”. In: Dec. 2012. doi: [10.13140/2.1.1833.8248](https://doi.org/10.13140/2.1.1833.8248).
- [5] Stanislaw Weglarczyk. “Kernel density estimation and its application”. In: *ITM Web of Conferences* 23 (Nov. 2018), p. 00037. doi: [10.1051/itmconf/20182300037](https://doi.org/10.1051/itmconf/20182300037).
- [6] Kanat Tangwongsan, Martin Hirzel, and Scott Schneider. “Sliding-Window Aggregation Algorithms”. In: *Encyclopedia of Big Data Technologies*. Ed. by Sherif Sakr and Albert Zomaya. Cham: Springer International Publishing, 2018, pp. 1–6. ISBN: 978-3-319-63962-8. doi: [10.1007/978-3-319-63962-8_157-1](https://doi.org/10.1007/978-3-319-63962-8_157-1). URL: https://doi.org/10.1007/978-3-319-63962-8_157-1.
- [7] Student. “The probable error of a mean”. In: *Biometrika* (1908), pp. 1–25.
- [8] Karl Pearson. “X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50.302 (July 1900), pp. 157–175. doi: [10.1080/14786440009463897](https://doi.org/10.1080/14786440009463897). URL: <https://doi.org/10.1080/14786440009463897>.
- [9] Samithamby Senthilnathan. “Usefulness of Correlation Analysis”. In: *SSRN Electronic Journal* (July 2019). doi: [10.2139/ssrn.3416918](https://doi.org/10.2139/ssrn.3416918).
- [10] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. ACM, Aug. 2016, pp. 785–794. doi: [10.1145/2939672.2939785](https://doi.org/10.1145/2939672.2939785). URL: <http://dx.doi.org/10.1145/2939672.2939785>.
- [11] Lei Xu et al. “Modeling Tabular data using Conditional GAN”. In: *Advances in Neural Information Processing Systems*. 2019.
- [12] Ekaterina Poslavskaya and Alexey Korolev. *Encoding categorical data: Is there yet anything ‘hotter’ than one-hot encoding?* 2023. arXiv: [2312.16930 \[cs.LG\]](https://arxiv.org/abs/2312.16930). URL: <https://arxiv.org/abs/2312.16930>.
- [13] Andrew P. Bradley. “The use of the area under the ROC curve in the evaluation of machine learning algorithms”. In: *Pattern Recognition* 30.7 (1997), pp. 1145–1159. ISSN: 0031-3203. doi: [https://doi.org/10.1016/S0031-3203\(96\)00142-2](https://doi.org/10.1016/S0031-3203(96)00142-2). URL: <https://www.sciencedirect.com/science/article/pii/S0031320396001422>.

5. Appendix

5.1. Codes and additional plots

```
In [4]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import networkx as nx
from collections import defaultdict
import plotly.graph_objects as go
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from xgboost import XGBClassifier
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

```
In [5]: data = pd.read_excel("/Users/zheguan/CWR_fig/Customer_Churn_Data_Large.xlsx")
```

```
In [6]: data
```

```

Out[6]: {'Customer_Demographics':      CustomerID  Age Gender MaritalStatus IncomeLevel
          0           1   62     M       Single        Low
          1           2   65     M     Married        Low
          2           3   18     M       Single        Low
          3           4   21     M     Widowed        Low
          4           5   21     M     Divorced       Medium
          ..
          ..
          995         996  54     F       Single        Low
          996         997  19     M     Widowed       High
          997         998  47     M     Married        Low
          998         999  23     M     Widowed       High
          999        1000  34     M     Widowed        Low

[1000 rows x 5 columns],      CustomerID  TransactionID TransactionDate  Amount
'Transaction_History':      countSpent ProductCategory
          0           1
          cs
          1           2
          ng
          2           2
          cs
          3           2
          re
          4           2
          ng
          ..
          ..
          5049        1000
          es
          5050        1000
          ks
          5051        1000
          es
          5052        1000
          ks
          5053        1000
          re

[5054 rows x 5 columns],      CustomerID  InteractionID InteractionDate InteractionType \
'Customer_Service':      \_
          0           1
          1           2
          2           3
          3           4
          4           4
          ..
          ..
          997         990
          998         992
          999         994
          1000        994
          1001        995

```

```

0           Resolved
1           Resolved
2           Resolved
3           Resolved
4      Unresolved
...
997      Unresolved
998      Unresolved
999      Unresolved
1000     Unresolved
1001     Resolved

[1002 rows x 5 columns],
'Online_Activity':      CustomerID LastLoginDate  LoginFrequency ServiceUs
age
0           1   2023-10-21          34  Mobile App
1           2   2023-12-05          5   Website
2           3   2023-11-15          3   Website
3           4   2023-08-25          2   Website
4           5   2023-10-27         41   Website
...
995      996   2023-01-29          38  Mobile App
996      997   2023-04-01          5   Mobile App
997      998   2023-07-10          47   Website
998      999   2023-01-08          23   Website
999     1000   2023-08-13          22  Mobile App

[1000 rows x 4 columns],
'Churn_Status':      CustomerID ChurnStatus
0           1           0
1           2           1
2           3           0
3           4           0
4           5           0
...
995      996           0
996      997           0
997      998           0
998      999           0
999     1000           0

[1000 rows x 2 columns]}

```

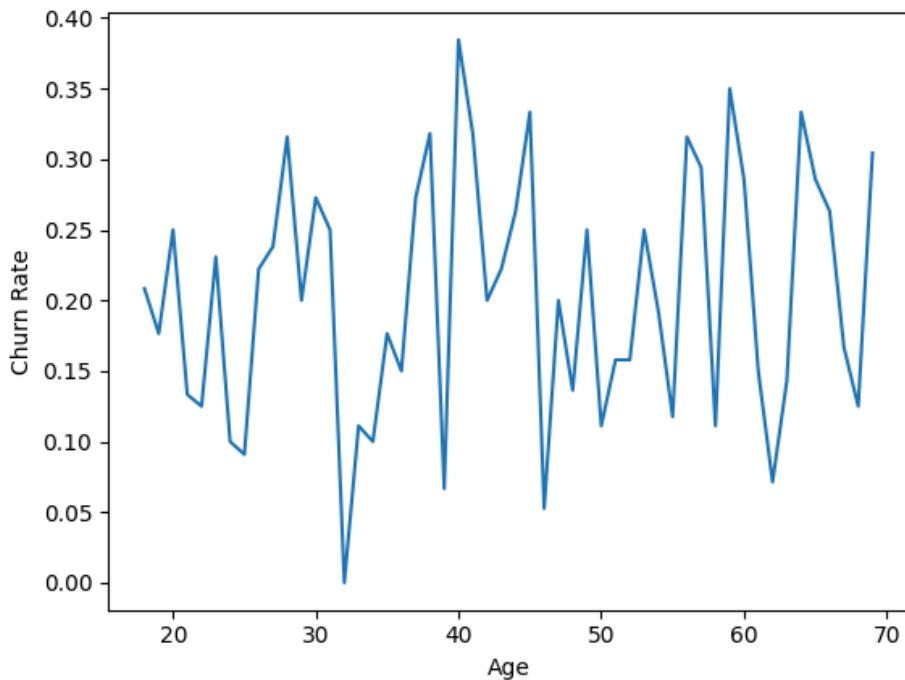
Customer_Demographics Sheet

```
In [7]: Customer_Demographics = pd.merge(data['Customer_Demographics'], data['Churn_#train_data, test_data = train_test_split(Customer_Demographics, test_size=0.2)
```

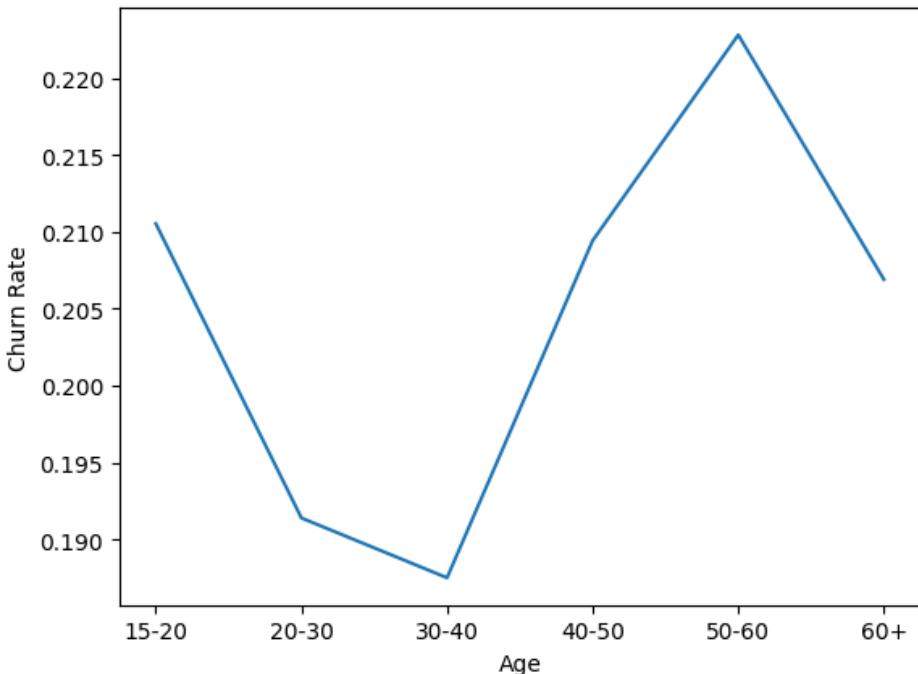
```
In [8]: Customer_Demographics["Age_range"] = pd.cut(Customer_Demographics["Age"], bins = [15, 20, 30, 40, 50, 60, np.inf], labels = ["15-20", "20-30", "30-40", "40-50", "50+"])
```

```
In [9]: Age_distribution = pd.crosstab(Customer_Demographics["Age"], Customer_Demogra
```

```
In [16]: plt.plot(Age_distribution["Age"],Age_distribution[1])
plt.xlabel("Age")
plt.ylabel("Churn Rate")
plt.savefig("Plots/age_churn_rate.png", dpi=300, bbox_inches='tight')
plt.show()
```



```
In [15]: Agerange_distribution = pd.crosstab(Customer_Demographics["Age_range"],Customer_Demographics["Churn"])
plt.plot(Agerange_distribution["Age_range"],Agerange_distribution[1])
plt.xlabel("Age")
plt.ylabel("Churn Rate")
plt.savefig("Plots/agerange_churn_rate.png", dpi=300, bbox_inches='tight')
plt.show()
```



```
In [18]: average_churn_rate = sum(Customer_Demographics['ChurnStatus'])/Customer_Demographic
average_churn_rate
```

```
Out[18]: 0.204
```

```
In [19]: def cal_Zscore(Customer_Demographics, input_col, target_col):
    average_churn_rate = sum(Customer_Demographics[target_col])/Customer_Demographic

    Distribution = pd.crosstab(Customer_Demographics[input_col],Customer_Demographic)
    Distribution['number'] = Distribution[input_col].map(Customer_Demographic
    Distribution["sigma"] = Distribution[0]*Distribution[1]/Distribution['nu
    Distribution["Zscore_"+input_col+"diff"] = (Distribution[1]-average_chur

    df = pd.merge(Customer_Demographics, Distribution[[input_col, "Zscore_"+
    return df,Distribution,average_churn_rate
```

```
In [20]: Gender_Distribution = cal_Zscore(Customer_Demographics, "Gender", "ChurnStat
Age_rangeDistribution = cal_Zscore(Customer_Demographics, "Age_range", "Chur
MaritalStatus_Distribution = cal_Zscore(Customer_Demographics, "MaritalStatus
IncomeLevel_Distribution = cal_Zscore(Customer_Demographics, "IncomeLevel",
```

/var/folders/pt/0nf2m3pj1f3b373wsyfc6glh0000gn/T/ipykernel_52421/1435638261.py:5: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

```
Distribution['number'] = Distribution[input_col].map(Customer_Demographic
input_col).size().astype(float))
```

```
In [21]: distributions = {
    "Gender": Gender_Distribution,
    "Age_range": Age_rangeDistribution,
    "MaritalStatus": MaritalStatus_Distribution,
    "IncomeLevel": IncomeLevel_Distribution
}

def merge_distribution(ori_df, dic):
    data = ori_df

    for key, distribution in dic.items():
        data = pd.merge(data, distribution[[key, "Zscore_" + key + "diff"]], how="left")

    return data

enhanced_Customer_Demographics = merge_distribution(Customer_Demographics, distributions)
```

```
In [22]: enhanced_Customer_Demographics = pd.get_dummies(enhanced_Customer_Demographics)
enhanced_Customer_Demographics
```

Out[22]:

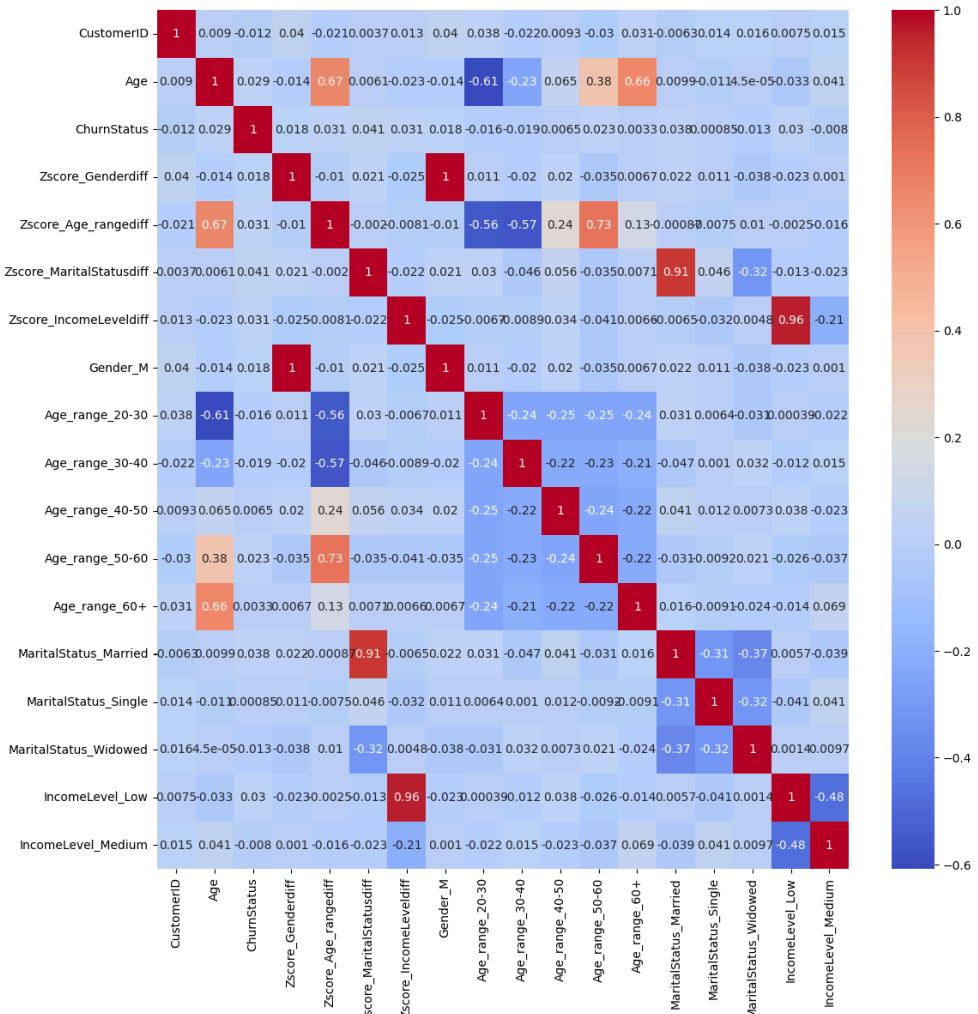
	CustomerID	Age	ChurnStatus	Zscore_Genderdiff	Zscore_Age_rangediff
0	1	62	0	21.898796	3.071478
1	2	65	1	21.898796	3.071478
2	3	18	0	21.898796	2.238200
3	4	21	0	21.898796	-17.033005
4	5	21	0	21.898796	-17.033005
...
995	996	54	0	-23.096539	20.951841
996	997	19	0	21.898796	2.238200
997	998	47	0	21.898796	6.257337
998	999	23	0	21.898796	-17.033005
999	1000	34	0	21.898796	-19.062154

1000 rows × 18 columns

```
In [24]: corr_df = enhanced_Customer_Demographics.corr()

plt.figure(figsize=(13, 13))
sns.heatmap(corr_df, annot=True, cmap='coolwarm')
```

Out[24]: <Axes: >



Cusomer Service sheet

```
In [25]: Customer_Service = data['Customer_Service']
Customer_Service = pd.merge(Customer_Service, data['Churn_Status'], how = "l

In [26]: prob_InteractionType = Customer_Service.groupby(["InteractionType", "ChurnStatus"])
prob_InteractionType["churn_rate"] = prob_InteractionType[1]/(prob_InteractionType[0].sum())

In [27]: prob_InteractionType["sigma"] = prob_InteractionType["churn_rate"]*(1-prob_InteractionType["churn_rate"])

In [28]: avg_prob_InteractionType = prob_InteractionType[1].sum() / (prob_InteractionType[0].sum())
prob_InteractionType["Zscore_InteractionType"] = (prob_InteractionType["churn_rate"] - avg_prob_InteractionType) / prob_InteractionType["sigma"]
Customer_Service[Customer_Service["ChurnStatus"] == 0].groupby(["InteractionType", "ChurnStatus"])
```

Out[28]:

	CustomerID	InteractionType	1	3	4	6	9	11	12	13	14
0		Complaint	NaN	1.0	1.0						
1		Feedback	NaN	NaN	NaN	1.0	NaN	2.0	1.0	NaN	1.0
2		Inquiry	1.0	1.0	2.0	NaN	1.0	NaN	1.0	NaN	NaN

3 rows × 526 columns

In [29]: prob_InteractionType

Out[29]:

	ChurnStatus	InteractionType	0	1	churn_rate	sigma	Zscore_Interactio
0		Complaint	233	64	0.215488	0.009809	0.8
1		Feedback	240	67	0.218241	0.009737	1.1
2		Inquiry	227	52	0.186380	0.009079	-2.2

In [36]:

```
import matplotlib.pyplot as plt
import numpy as np

interaction_types = prob_InteractionType["InteractionType"]
no_churn_counts = prob_InteractionType[0]
churn_counts = prob_InteractionType[1]
churn_rates = prob_InteractionType["churn_rate"]

x = np.arange(len(interaction_types))

fig, ax1 = plt.subplots(figsize=(8, 5))
plt.ylim(0,350)
# Bar Chart for Churn & No Churn
bar_width = 0.35
ax1.bar(x - bar_width/2, no_churn_counts, bar_width, label="No Churn", color='blue')
ax1.bar(x + bar_width/2, churn_counts, bar_width, label="Churn", color='red')

ax1.set_xlabel("Interaction Type")
ax1.set_ylabel("Customer Count")
ax1.set_xticks(x)
ax1.set_xticklabels(interaction_types)
ax1.legend(loc="upper left")

ax2 = ax1.twinx()
ax2.plot(x, churn_rates, color='black', marker='o', linestyle='--', linewidth=2)

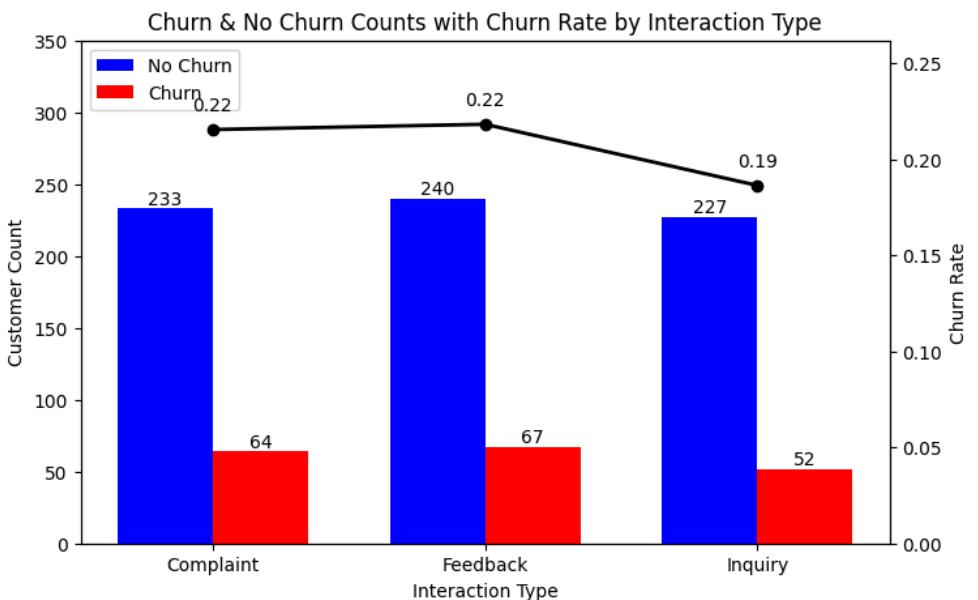
ax2.set_ylabel("Churn Rate")
ax2.set_ylim(0, max(churn_rates) * 1.2)

# Show values on bars
for i in range(len(x)):
    ax1.text(x[i] - bar_width/2, no_churn_counts.iloc[i] + 3, str(no_churn_c
    ax1.text(x[i] + bar_width/2, churn_counts.iloc[i] + 3, str(churn_counts.i
```

```

        ax2.text(x[i], churn_rates.iloc[i] + 0.01, f'{churn_rates.iloc[i]:.2f}', color='black')
    plt.title("Churn & No Churn Counts with Churn Rate by Interaction Type")
    plt.savefig("Plots/churn_rate_interactiontype.png", dpi=300, bbox_inches='tight')
    plt.show()

```



```
In [37]: Customer_Service["Status"] = Customer_Service["InteractionType"] + "_" + Customer_Service["CustomerID"]
```

```
In [38]: final_status = (Customer_Service.groupby("CustomerID", group_keys=False) [[ "Status" ]].apply(lambda x : x.iloc[-1] ) .reset_index() .rename(columns={"ChurnStatus": "Status"}))
```

```
In [40]: combined_Customer_Service = (pd.concat([Customer_Service,final_status]) .sort_values(by=["CustomerID","InteractionDate"] ) ) combined_Customer_Service["Status"] = combined_Customer_Service["Status"].replace(["Cust","Churn"],[0,1])
```

```
In [41]: Customer_Service_enhanced = Customer_Service.sort_values(by=["CustomerID", "InteractionDate"]) .groupby("CustomerID").agg( First_Interaction=('InteractionDate','min'), Last_Interaction=('InteractionDate','max'), Last_Status = ('Status','last') ) Customer_Service_enhanced['Duration_Interaction'] = \ (Customer_Service_enhanced['Last_Interaction'] - Customer_Service_enhanced['First_Interaction']) / 2
```

```
In [42]: Customer_Service["InteractionDate"] = pd.to_datetime(Customer_Service["InteractionDate"]) Customer_Service = Customer_Service.sort_values(by=["CustomerID", "InteractionDate"])
```

Loading [MathJax]/extensions/Safe.js

```

        Interaction_Count=("InteractionID", "count")
    ).reset_index()

daily_interactions["Rolling_Interaction_Count"] = daily_interactions.groupby(
    .rolling(window=7, min_periods=1).sum().reset_index(level=0, drop=True)

max_interaction_7d = daily_interactions.loc[daily_interactions.groupby("Cust
test = max_interaction_7d.merge(data["Churn_Status"], how = "left" ,on = "Cu
test[test['ChurnStatus'] == 1].head(10)

```

Out[42]:

	CustomerID	InteractionDate	Interaction_Count	Rolling_Interaction_Count
1	2	2022-03-17	1	1.0
5	8	2022-09-13	1	2.0
12	16	2022-09-08	1	2.0
14	18	2022-11-08	1	1.0
21	26	2022-12-12	1	1.0
27	33	2022-05-30	1	1.0
34	41	2022-02-26	1	1.0
43	61	2022-04-10	1	1.0
46	68	2022-06-06	1	2.0
48	71	2022-04-13	1	2.0

In [43]:

```

Customer_Service.groupby("CustomerID")['InteractionType'].value_counts().unstack()
Customer_Service.groupby("CustomerID")['ResolutionStatus'].value_counts().unstack()
Customer_Service.groupby("CustomerID")['Status'].value_counts().unstack()
interaction_statistics = pd.concat([Customer_Service.groupby("CustomerID")['

```

In [44]:

```

chains = combined_Customer_Service.groupby("CustomerID")["Status"].apply(list)
transform = defaultdict(lambda: defaultdict(int))

for i in chains:
    #print(i[0])
    for k in range(len(i)-1):
        src = i[k]
        des = i[k+1]
        transform[src][des] += 1
        #print(src,des)

transform_count = pd.DataFrame(transform).T
transform_prob = transform_count.div(transform_count.sum(axis=1), axis=0)
transform_prob
transform_prob.loc["Retained"] = [1,0,0,0,0,0,0,0]
transform_prob.loc["Churned"] = [0,1,0,0,0,0,0,0]

expected_order = ["Retained", "Churned", "Inquiry_Resolved", "Inquiry_Unres

```

```
transform_prob = transform_prob.reindex(index=expected_order, columns=expected_order)
transform_prob
```

Out[44]:

	Retained	Churned	Inquiry_Resolved	Inquiry_Unresolved
Retained	1.000000	0.000000	0.000000	0.000000
Churned	0.000000	1.000000	0.000000	0.000000
Inquiry_Resolved	0.577381	0.130952	0.053571	0.0357
Inquiry_Unresolved	0.546763	0.122302	0.064748	0.0287
Feedback_Resolved	0.507538	0.155779	0.035176	0.0502
Feedback_Unresolved	0.546584	0.142857	0.043478	0.0372
Complaint_Resolved	0.512821	0.115385	0.032051	0.0384
Complaint_Unresolved	0.463687	0.178771	0.094972	0.0614

In [45]: combined_Customer_Service

Out[45]:

	CustomerID	InteractionID	InteractionDate	InteractionType	Resolution
0	1	6363.0	2022-03-31	Inquiry	Resolved
0	1	NaN	NaT	NaN	Unresolved
1	2	3329.0	2022-03-17	Inquiry	Resolved
1	2	NaN	NaT	NaN	Unresolved
2	3	9976.0	2022-08-24	Inquiry	Resolved
...
1000	994	8508.0	2022-05-14	Complaint	Unresolved
999	994	3087.0	2022-07-02	Complaint	Unresolved
666	994	NaN	NaT	NaN	Unresolved
1001	995	9101.0	2022-05-02	Inquiry	Resolved
667	995	NaN	NaT	NaN	Unresolved

1670 rows × 7 columns

In [47]:

```
G = nx.DiGraph()

for src in transform_prob.columns:
    for dst in transform_prob.index:
        prob = transform_prob.loc[dst, src]
        if prob > 0:
            G.add_edge(src, dst, weight=prob)

pos = nx.spring_layout(G, seed=42)
```

Loading [MathJax]/extensions/Safe.js

```

#pos = nx.kamada_kawai_layout(G)
#pos = nx.shell_layout(G)
#pos = nx.circular_layout(G)
#pos = nx.spectral_layout(G)
#pos = nx.multipartite_layout(G, subset_key="layer")

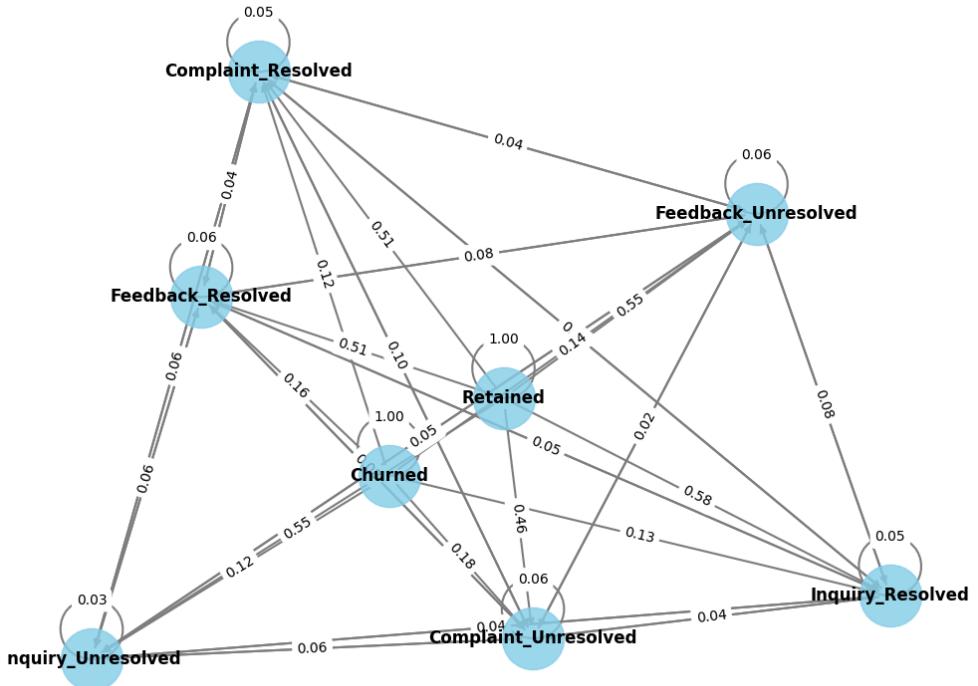
plt.figure(figsize=(10, 8))
nx.draw_networkx_nodes(G, pos, node_size=2000, node_color="skyblue", alpha=0.8)
nx.draw_networkx_edges(G, pos, width=1.5, edge_color="gray", arrows=True)

edge_labels = {(u, v): f"{d['weight']:.2f}" for u, v, d in G.edges(data=True)}
nx.draw_networkx_labels(G, pos, font_size=12, font_weight="bold")
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=10)

plt.title("State Transition Diagram", fontsize=14)
plt.axis("off")
plt.tight_layout()
plt.savefig("Plots/chain_interactiontype.png", dpi=300, bbox_inches='tight')
plt.show()

```

State Transition Diagram



In [48]:

```

labels = list(transform_prob.columns)
source = []
target = []
value = []

```

Loading [MathJax]/extensions/Safe.js

```

for src_idx, src in enumerate(transform_prob.columns):
    for dst_idx, dst in enumerate(transform_prob.index):
        prob = transform_prob.loc[dst, src]
        if prob > 0:
            target.append(src_idx)
            source.append(len(labels) + dst_idx)
            value.append(prob)

labels += list(transform_prob.index)

fig = go.Figure(go.Sankey(
    node=dict(
        pad=20,
        thickness=20,
        line=dict(color="black", width=0.5),
        label=labels,
        color=["#90EE90", "#FFB6C1", "#FF4500", "#32CD32"] * 2
    ),
    link=dict(
        source=source,
        target=target,
        value=value,
        color="rgba(128, 128, 128, 0.5)"
    )
))
fig.update_layout(
    title_text="State Transition Flow",
    font_size=12,
    height=600
)
fig.show()

```

```
In [49]: plt.figure(figsize=(12,8))
sns.heatmap(transform_prob.fillna(0), annot=True, cmap='Blues')
#plt.title("State Transition Probability Matrix")
plt.savefig("Plots/transition_matrix.png", dpi=300, bbox_inches='tight')
plt.show()
```



```
In [50]: import scipy.linalg
```

```
P = np.array(transform_prob)

absorbing_states = np.where(np.diag(P) == 1)[0]
transient_states = np.where(np.diag(P) != 1)[0]

Q = P[np.ix_(transient_states, transient_states)]
R = P[np.ix_(transient_states, absorbing_states)]

# N = (I - Q)^(-1)
I = np.eye(Q.shape[0])
N = np.linalg.inv(I - Q)

# B = N @ R
B = N @ R
print("Absorb Prob Matrix B :\n", B)
```

Absorb Prob Matrix B :

```
[[0.80700758 0.19299242]
 [0.80678326 0.19321674]
 [0.77081632 0.22918368]
 [0.79064724 0.20935276]
 [0.80247131 0.19752869]
 [0.74481898 0.25518102]]
```

```
Tn [51]: R
```

Loading [MathJax]/extensions/Safe.js

```
Out[51]: array([[0.80700758, 0.19299242],  
 [0.80678326, 0.19321674],  
 [0.77081632, 0.22918368],  
 [0.79064724, 0.20935276],  
 [0.80247131, 0.19752869],  
 [0.74481898, 0.25518102]])
```

```
In [52]: Absorb_Matrix = {  
     'Inquiry_Resolved': B[0,1],  
     'Inquiry_Unresolved': B[1,1],  
     'Feedback_Resolved': B[2,1],  
     'Feedback_Unresolved': B[3,1],  
     'Complaint_Resolved': B[4,1],  
     'Complaint_Unresolved': B[5,1]  
}  
  
interaction_statistics['Markov_Churn_prob'] = Customer_Service_enhanced['Last_Status']
```

```
In [54]: #interaction_statistics  
Customer_Service_enhanced
```

```
Out[54]:
```

CustomerID	First_Iteration	Last_Iteration	Last_Status	Duration_In
1	2022-03-31	2022-03-31	Inquiry_Resolved	
2	2022-03-17	2022-03-17	Inquiry_Resolved	
3	2022-08-24	2022-08-24	Inquiry_Resolved	
4	2022-07-03	2022-11-18	Inquiry_Resolved	
6	2022-05-05	2022-05-05	Feedback_Resolved	
...
989	2022-08-03	2022-10-07	Complaint_Unresolved	
990	2022-09-17	2022-10-25	Complaint_Unresolved	
992	2022-09-29	2022-09-29	Feedback_Unresolved	
994	2022-05-14	2022-07-02	Complaint_Unresolved	
995	2022-05-02	2022-05-02	Inquiry_Resolved	

668 rows × 4 columns

```
In [55]: enhanced_Interaction_Customer = enhanced_Customer_Demographics.merge(Customer_Service_enhanced)  
       .merge(interaction_statistics['Markov_Churn_prob'])
```

```
In [56]: enhanced_Interaction_Customer
```

Out[56]:

	CustomerID	Age	ChurnStatus	Zscore_Genderdiff	Zscore_Age_rangediff
0	1	62	0	21.898796	3.071478
1	2	65	1	21.898796	3.071478
2	3	18	0	21.898796	2.238200
3	4	21	0	21.898796	-17.033005
4	5	21	0	21.898796	-17.033005
...
995	996	54	0	-23.096539	20.951841
996	997	19	0	21.898796	2.238200
997	998	47	0	21.898796	6.257337
998	999	23	0	21.898796	-17.033005
999	1000	34	0	21.898796	-19.062154

1000 rows × 20 columns

```
In [68]: Customer_Service_count = Customer_Service.groupby(["CustomerID", "InteractionType"])[["Count"]].sum().reset_index()

In [69]: Customer_Service_resolutioncount = Customer_Service.groupby(["CustomerID", "ResolutionType"])[["Count"]].sum().reset_index()

In [57]: enhanced_Interaction_Customer['Duration_Interation'] = enhanced_Interaction_Customer['Duration'].apply(lambda x: x / 1000)
enhanced_Interaction_Customer['Markov_Churn_prob'] = enhanced_Interaction_Customer['Churn'].apply(lambda x: 1 if x == 1 else 0)

In [72]: enhanced_Interaction_Customer = enhanced_Interaction_Customer.merge(Customer)

In [74]: enhanced_Interaction_Customer
```

Out[74]:

	CustomerID	Age	ChurnStatus	Zscore_Genderdiff	Zscore_Age_rangediff
0	1	62	0	21.898796	3.071478
1	2	65	1	21.898796	3.071478
2	3	18	0	21.898796	2.238200
3	4	21	0	21.898796	-17.033005
4	5	21	0	21.898796	-17.033005
...
995	996	54	0	-23.096539	20.951841
996	997	19	0	21.898796	2.238200
997	998	47	0	21.898796	6.257337
998	999	23	0	21.898796	-17.033005
999	1000	34	0	21.898796	-19.062154

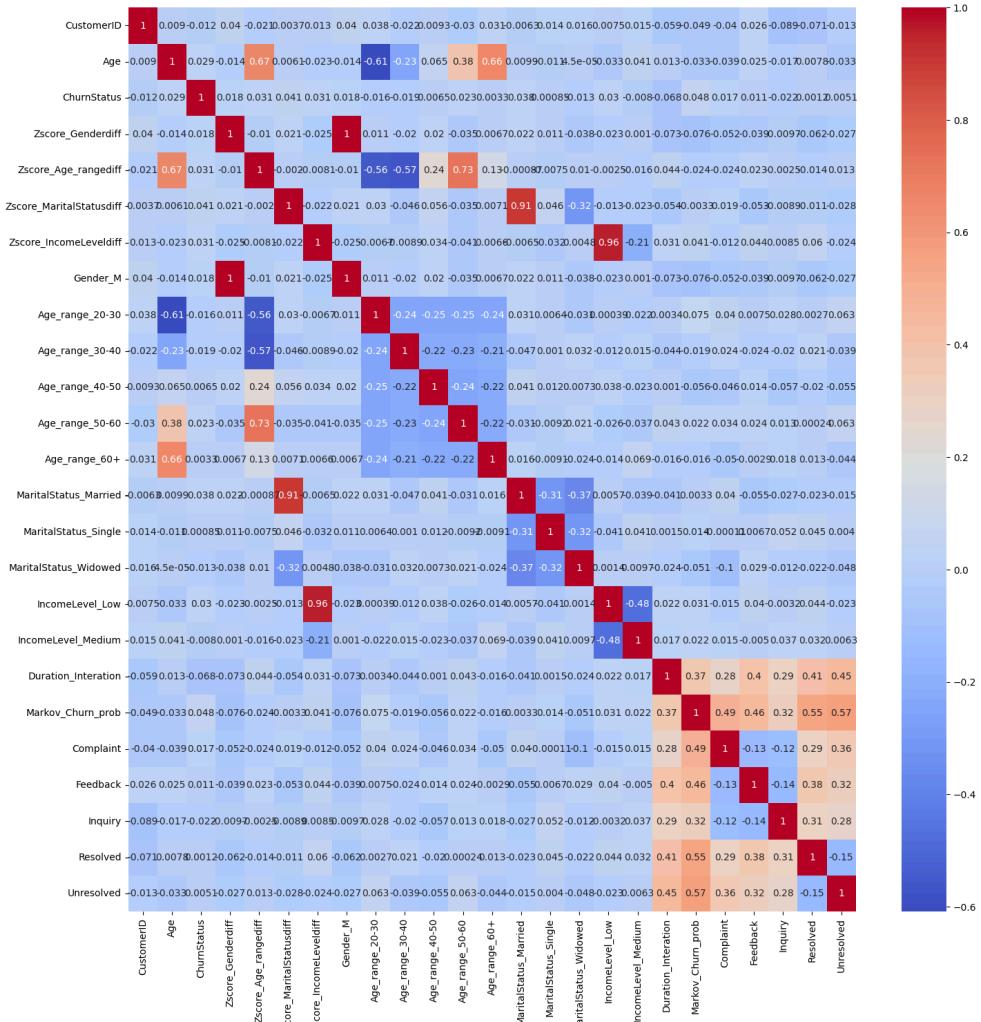
1000 rows × 25 columns

In [76]:

```
corr_df = enhanced_Interaction_Customer.corr()

plt.figure(figsize=(17, 17))
sns.heatmap(corr_df, annot=True, cmap='coolwarm')
```

Out[76]: <Axes: >



Transaction Sheet

```
In [77]: Transaction_History = data["Transaction_History"]
        Transaction_History = Transaction_History.merge(data['Churn_Status'], how = 'left')
```

Out[77]:

	CustomerID	TransactionID	TransactionDate	AmountSpent	ProductCategory
0	1	7194	2022-03-27	416.50	Electronics
1	2	7250	2022-08-08	54.96	Clothing
2	2	9660	2022-07-25	197.50	Electronics
3	2	2998	2022-01-25	101.31	Furniture
4	2	1228	2022-07-24	397.37	Clothing
...
5049	1000	2724	2022-09-08	232.06	Groceries
5050	1000	2917	2022-12-13	324.98	
5051	1000	2979	2022-06-15	375.34	Groceries
5052	1000	8594	2022-04-08	166.73	
5053	1000	5529	2022-11-23	93.73	Furniture

5054 rows × 6 columns

In [78]:

```
df_diff = Transaction_History.groupby([ "CustomerID", "ProductCategory"])[ "AmountSpent"].agg({  
    First_Transaction="min",  
    Last_Transaction="max",  
    total_Transaction = "sum",  
    mean_Transaction = "mean",  
    count_Transaction = "count"  
})  
  
df_diff[ "Days_Difference" ] = ( df_diff[ "Last_Transaction" ] - df_diff[ "First_Transaction" ] ) / df_diff[ "Days_Difference" ]  
df_diff.reset_index().head(50)
```

Out[78]:

	CustomerID	ProductCategory	First_Transaction	Last_Transaction	total_Tran
0	1	Electronics	416.50	416.50	
1	2	Clothing	54.96	397.37	
2	2	Electronics	197.50	311.34	
3	2	Furniture	101.31	101.31	
4	2	Groceries	199.73	199.73	
5	3	Books	241.06	241.06	
6	3	Clothing	51.07	51.07	
7	3	Furniture	403.32	419.95	
8	3	Groceries	287.72	299.86	
9	4	Clothing	44.22	44.22	
10	4	Electronics	125.64	241.45	
11	4	Furniture	382.39	382.39	
12	4	Groceries	123.59	123.59	
13	5	Electronics	155.76	308.80	
14	5	Furniture	178.91	219.98	
15	5	Groceries	69.86	475.69	
16	6	Books	365.57	365.57	
17	6	Clothing	448.86	448.86	
18	6	Electronics	204.16	204.16	
19	6	Furniture	41.70	104.00	
20	7	Books	86.73	86.73	
21	8	Books	240.92	263.24	
22	8	Clothing	346.24	483.91	
23	8	Electronics	161.56	161.56	
24	8	Furniture	253.68	297.33	
25	9	Clothing	325.96	472.21	
26	9	Electronics	96.67	96.67	
27	9	Groceries	44.26	378.14	
28	10	Clothing	35.29	35.29	
29	10	Electronics	178.78	178.78	
30	10	Furniture	5.86	471.18	
31	10	Groceries	179.00	303.40	
32	11	Books	163.78	426.06	

CustomerID	ProductCategory	First_Transaction	Last_Transaction	total_Tra
33	11	Groceries	347.37	462.00
34	12	Electronics	87.10	87.10
35	13	Books	164.72	164.72
36	13	Clothing	54.17	54.17
37	13	Electronics	173.13	173.13
38	14	Books	150.22	150.22
39	14	Clothing	335.44	335.44
40	14	Furniture	412.20	468.35
41	14	Groceries	156.21	478.59
42	15	Books	160.79	283.39
43	15	Clothing	185.60	185.60
44	15	Electronics	383.90	383.90
45	15	Furniture	151.88	273.54
46	15	Groceries	68.63	406.29
47	16	Books	52.67	352.99
48	16	Clothing	470.53	470.53
49	16	Electronics	97.06	97.06

```
In [79]: Transaction_History["TransactionDate"] = pd.to_datetime(Transaction_History["TransactionDate"])
Transaction_History = Transaction_History.sort_values(by=["CustomerID", "Transac
daily_spending = Transaction_History.groupby(["CustomerID", "TransactionDate"])
    Transaction_Count=("TransactionID", "count"),
    Total_Spent=("AmountSpent", "sum"))
).reset_index()

daily_spending["Rolling_Transaction_Count"] = daily_spending.groupby("CustomerID").
daily_spending["Rolling_Total_Spent"] = daily_spending.groupby("CustomerID").

max_transaction_7d = daily_spending.loc[daily_spending.groupby("CustomerID")["Roll
max_spent_7d = daily_spending.loc[daily_spending.groupby("CustomerID")]["Roll
print("How many times spent in 7 days at most? \n", max_transaction_7d.merge(data[["Churn
print("How much spent in 7 days at most? \n", max_spent_7d.merge(data[["Churn

```

How many times spent in 7 days at most?

	CustomerID	TransactionDate	Transaction_Count	Total_Spent	\
0	1	2022-03-27	1	416.50	
1	2	2022-01-25	1	101.31	
2	3	2022-03-02	1	299.86	
3	4	2022-07-31	1	125.64	
4	5	2022-05-25	1	69.86	
5	6	2022-08-09	1	41.70	
6	7	2022-10-24	1	86.73	
7	8	2022-06-10	1	346.24	
8	9	2022-05-26	1	325.96	
9	10	2022-06-05	1	179.00	
10	11	2022-06-06	1	334.62	
11	12	2022-11-02	1	87.10	
12	13	2022-08-10	1	164.72	
13	14	2022-02-21	1	213.22	
14	15	2022-01-26	1	406.29	
15	16	2022-08-07	1	97.06	
16	17	2022-06-27	1	124.02	
17	18	2022-02-08	1	154.09	
18	19	2022-07-18	1	19.80	
19	20	2022-04-04	1	462.10	

Rolling_Transaction_Count Rolling_Total_Spent ChurnStatus

0	1.0	416.50	0
1	2.0	386.52	1
2	2.0	703.18	0
3	2.0	508.03	0
4	2.0	545.55	0
5	2.0	145.70	0
6	1.0	86.73	0
7	2.0	643.57	1
8	2.0	370.22	0
9	2.0	650.18	1
10	2.0	796.62	0
11	1.0	87.10	0
12	2.0	218.89	0
13	2.0	681.57	0
14	2.0	679.83	0
15	2.0	450.05	1
16	2.0	191.22	0
17	2.0	408.31	1
18	2.0	497.63	0
19	2.0	787.09	0

How much spent in 7 days at most?

	CustomerID	TransactionDate	Transaction_Count	Total_Spent	\
0	1	2022-03-27	1	416.50	
1	2	2022-07-25	1	197.50	
2	3	2022-03-02	1	299.86	
3	4	2022-07-31	1	125.64	
4	5	2022-12-21	1	218.98	
5	6	2022-10-29	1	448.86	
6	7	2022-10-24	1	86.73	
7	8	2022-11-09	1	483.91	
8	9	2022-10-19	1	378.14	
9	10	2022-06-05	1	179.00	

10	11	2022-06-06	1	334.62
11	12	2022-11-02	1	87.10
12	13	2022-09-27	1	173.13
13	14	2022-06-17	1	442.99
14	15	2022-03-28	1	283.39
15	16	2022-09-01	1	147.82
16	17	2022-07-19	1	344.04
17	18	2022-05-27	1	164.11
18	19	2022-07-18	1	19.80
19	20	2022-09-14	1	485.84

	Rolling_Transaction_Count	Rolling_Total_Spent	ChurnStatus
0	1.0	416.50	0
1	2.0	594.87	1
2	2.0	703.18	0
3	2.0	508.03	0
4	2.0	592.49	0
5	2.0	653.02	0
6	1.0	86.73	0
7	2.0	747.15	1
8	2.0	850.35	0
9	2.0	650.18	1
10	2.0	796.62	0
11	1.0	87.10	0
12	2.0	337.85	0
13	2.0	921.58	0
14	2.0	689.68	0
15	2.0	618.35	1
16	2.0	788.10	0
17	2.0	445.84	1
18	2.0	497.63	0
19	2.0	831.59	0

```
In [80]: Transaction_History["TransactionDate"] = pd.to_datetime(Transaction_History["TransactionDate"])

Transaction_History = Transaction_History.sort_values(by=["CustomerID", "ProductCategory", "TransactionDate"])

daily_spending = Transaction_History.groupby(["CustomerID", "ProductCategory", "TransactionDate"])
    .groupby("CustomerID", "ProductCategory", "TransactionDate")
    .agg(Transaction_Count=("TransactionID", "count"),
         Total_Spent=("AmountSpent", "sum"))
    .reset_index()

daily_spending["Rolling_Transaction_Count"] = daily_spending.groupby(["CustomerID", "ProductCategory"])
    .rolling(window=7, min_periods=1).sum().reset_index(level=[0,1], drop=True)

daily_spending["Rolling_Total_Spent"] = daily_spending.groupby(["CustomerID", "ProductCategory"])
    .rolling(window=7, min_periods=1).sum().reset_index(level=[0,1], drop=True)

daily_spending = daily_spending.dropna(subset=["Rolling_Transaction_Count", "Rolling_Total_Spent"])

max_transaction_7d = daily_spending.loc[daily_spending.groupby(["CustomerID", "ProductCategory"])["Rolling_Total_Spent"].transform(max) == daily_spending["Rolling_Total_Spent"]]
```

```

max_spent_7d = daily_spending.loc[daily_spending.groupby(["CustomerID", "Pro

churn_data = data["Churn_Status"][[ "CustomerID", "ChurnStatus"]]
max_transaction_7d = max_transaction_7d.merge(churn_data, how="left", on="Cu
max_spent_7d = max_spent_7d.merge(churn_data, how="left", on="CustomerID")

In [81]: Transaction_History["TransactionDate"] = pd.to_datetime(Transaction_History[

Transaction_History = Transaction_History.sort_values(by=[ "CustomerID", "Pro

daily_spending = Transaction_History.groupby(["CustomerID", "ProductCategory",
    Transaction_Count=( "TransactionID", "count"),
    Total_Spent=( "AmountSpent", "sum")
).reset_index()

In [82]: test = daily_spending.groupby("CustomerID")['Total_Spent'].rolling(window=7,
    min_periods=1).mean()

In [83]: test_1 = test.merge(data["Churn_Status"], how="left", on="CustomerID")[test.

In [84]: test_0 = test.merge(data["Churn_Status"], how="left", on="CustomerID")[test.

In [85]: df_diff = df_diff.reset_index().merge(data["Churn_Status"], how="left", on="

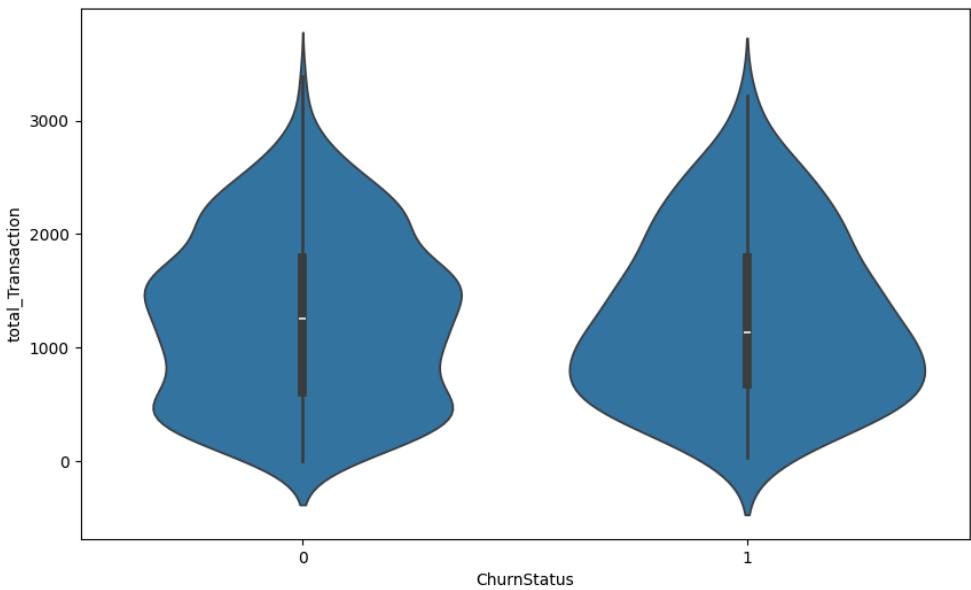
In [86]: all_total_spent = df_diff.groupby("CustomerID")["total_Transaction"].sum()
all_mean_spent = df_diff.groupby("CustomerID")["total_Transaction"].mean()
all_total_spent = all_total_spent.reset_index().merge(data["Churn_Status"],
all_mean_spent = all_mean_spent.reset_index().merge(data["Churn_Status"], ha

In [87]: import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.violinplot(x="ChurnStatus" , y= "total_Transaction"
                , data=all_total_spent)

Out[87]: <Axes: xlabel='ChurnStatus', ylabel='total_Transaction'>

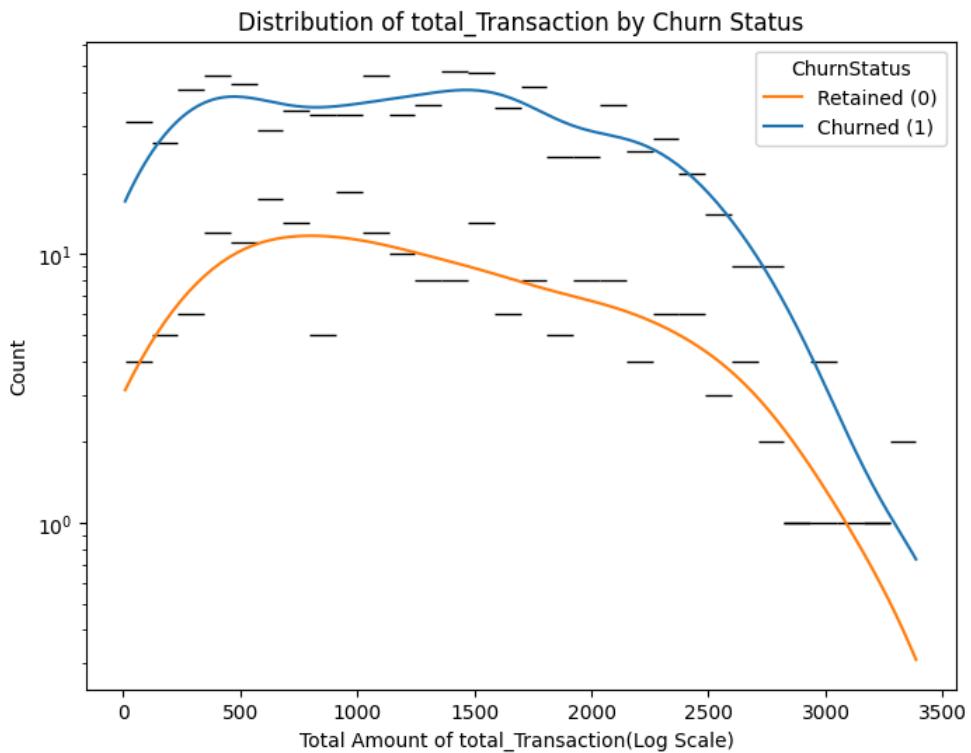
```



```
In [88]: plt.figure(figsize=(8,6))

sns.histplot(data=all_total_spent, x="total_Transaction", hue="ChurnStatus",
             bins=30, kde=True, log_scale=(False, True), alpha=0.5)

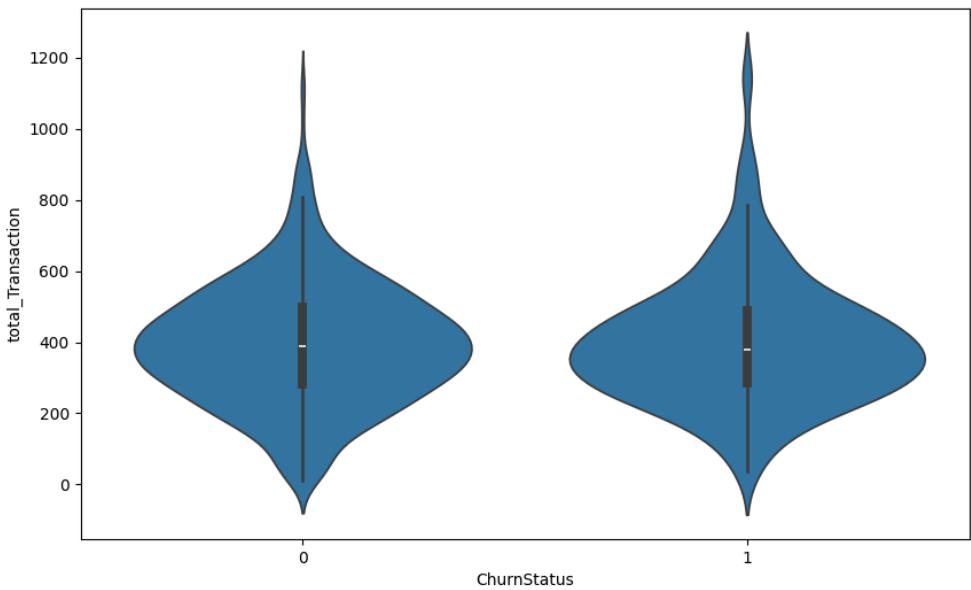
plt.title("Distribution of total_Transaction by Churn Status")
plt.xlabel("Total Amount of total_Transaction(Log Scale) ")
plt.ylabel("Count")
plt.legend(title="ChurnStatus", labels=["Retained (0)", "Churned (1)"]) # X
plt.show()
```



```
In [89]: import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.violinplot(x="ChurnStatus" , y= "total_Transaction"
                , data=all_mean_spent)
```

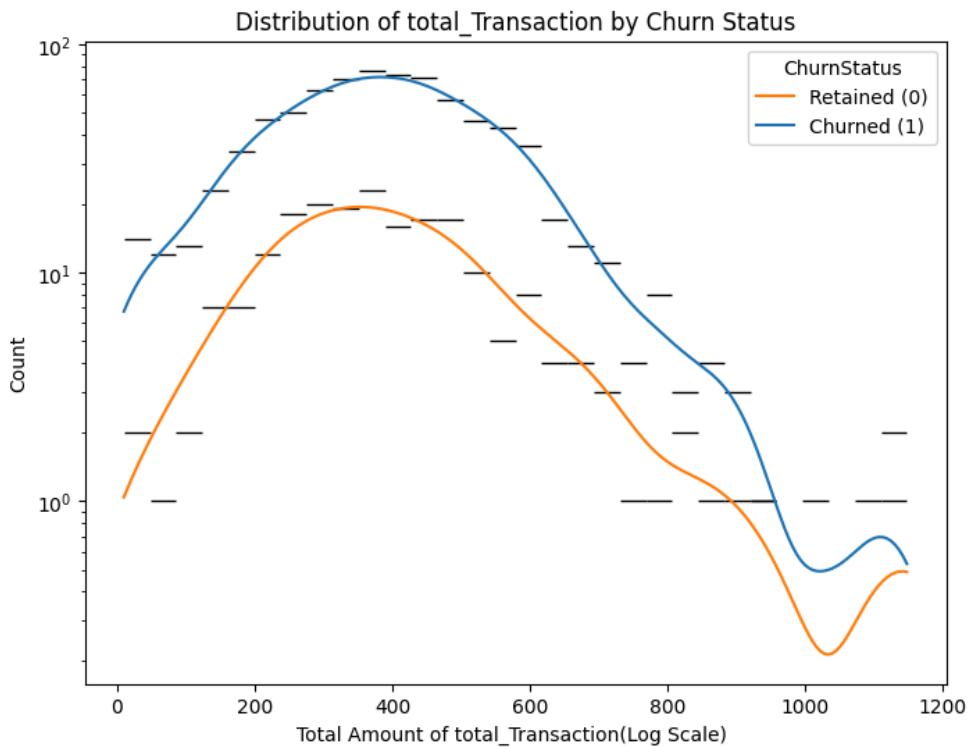
Out[89]: <Axes: xlabel='ChurnStatus', ylabel='total_Transaction'>



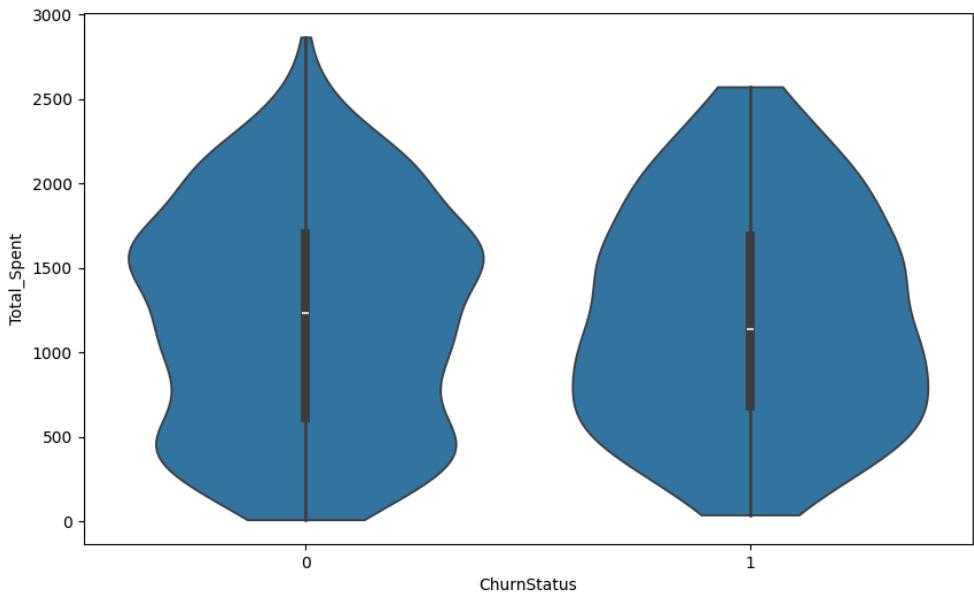
```
In [90]: plt.figure(figsize=(8,6))

sns.histplot(data=all_mean_spent, x="total_Transaction", hue="ChurnStatus",
             bins=30, kde=True, log_scale=(False, True), alpha=0.5)

plt.title("Distribution of total_Transaction by Churn Status")
plt.xlabel("Total Amount of total_Transaction(Log Scale) ")
plt.ylabel("Count")
plt.legend(title="ChurnStatus", labels=["Retained (0)", "Churned (1)"]) # X
plt.show()
```



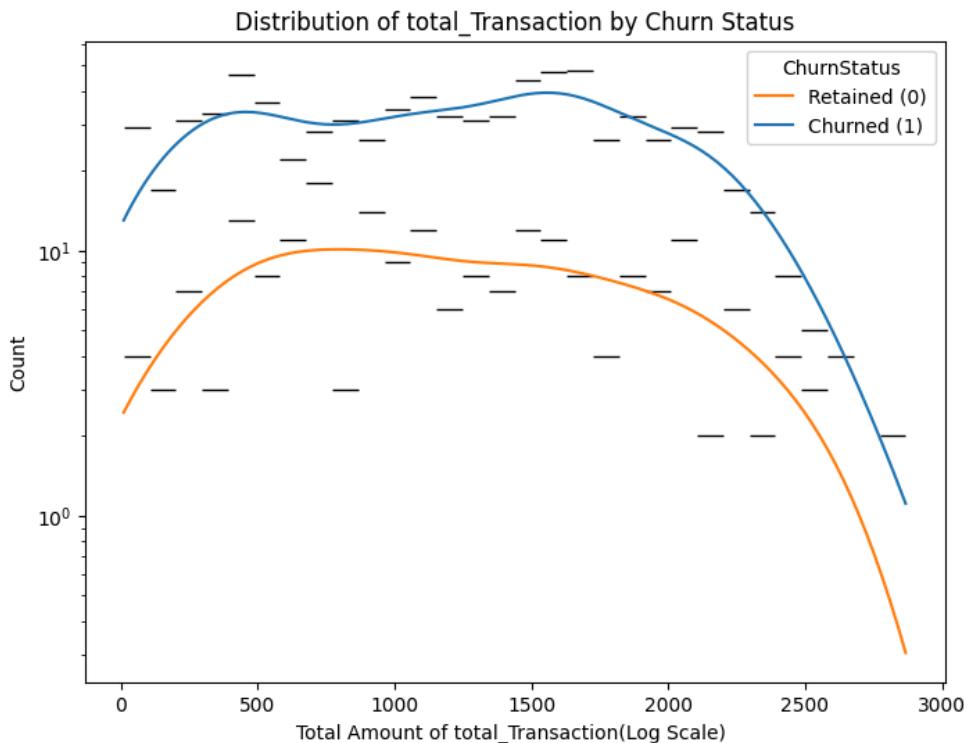
```
In [101]:  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(10, 6))  
sns.violinplot(x="ChurnStatus" , y= "Total_Spent"  
                , data=test.merge(data["Churn_Status"], how="left", on="CustomerID")  
plt.savefig("Plots/violin_total_spent.png", dpi=300, bbox_inches='tight')
```



```
In [92]: plt.figure(figsize=(8,6))

sns.histplot(data=test.merge(data["Churn_Status"], how="left", on="CustomerID")
             bins=30, kde=True, log_scale=(False, True), alpha=0.5)

plt.title("Distribution of total_Transaction by Churn Status")
plt.xlabel("Total Amount of total_Transaction(Log Scale) ")
plt.ylabel("Count")
plt.legend(title="ChurnStatus", labels=["Retained (0)", "Churned (1)"])
plt.show()
```



```
In [93]: test.merge(data["Churn_Status"], how="left", on="CustomerID")
```

```
Out[93]:
```

	CustomerID	Total_Spent	ChurnStatus
0	1	416.50	0
1	2	1547.42	1
2	3	1702.98	0
3	4	917.29	0
4	5	1692.69	0
...
995	996	227.25	0
996	997	419.82	0
997	998	252.15	0
998	999	2233.24	0
999	1000	1670.79	0

1000 rows × 3 columns

```
In [94]: mean_churned = all_total_spent[all_total_spent['ChurnStatus'] == 1]['total_T  
ined']  
mean_retained = all_total_spent[all_total_spent['ChurnStatus'] == 0]['total_
```

```

print(f"Churned means: {mean_churned:.2f}")
print(f"Retained means: {mean_retained:.2f}")

from scipy.stats import ttest_ind

churned_transactions = all_total_spent[all_total_spent['ChurnStatus'] == 1]
retained_transactions = all_total_spent[all_total_spent['ChurnStatus'] == 0]

t_stat, p_value = ttest_ind(churned_transactions, retained_transactions, equal_var=False)

print(f"T Statistics: {t_stat:.4f}, P Value: {p_value:.4f}")

if p_value < 0.05:
    print("Refuse H0")
else:
    print("Accept H0")

Churned means: 1269.01
Retained means: 1266.58
T Statistics: 0.0421, P Value: 0.9664
Accept H0

```

In [95]:

```

mean_churned = test.merge(data["Churn_Status"], how="left", on="CustomerID")
mean_retained = test.merge(data["Churn_Status"], how="left", on="CustomerID")

print(f"Churned means {mean_churned:.2f}")
print(f"Retained means: {mean_retained:.2f}")

from scipy.stats import ttest_ind

churned_transactions = test.merge(data["Churn_Status"], how="left", on="CustomerID")
retained_transactions = test.merge(data["Churn_Status"], how="left", on="CustomerID")

t_stat, p_value = ttest_ind(churned_transactions, retained_transactions, equal_var=False)

print(f"T Statistics: {t_stat:.4f}, P Value: {p_value:.4f}")

if p_value < 0.05:
    print("Refuse H0")
else:
    print("Accept H0")

```

```

Churned means 1202.08
Retained means: 1206.55
T Statistics: -0.0885, P Value: 0.9295
Accept H0

```

In [96]:

```
daily_spending.sort_values('TransactionDate').groupby('CustomerID').apply(lambda x: x[x['Category'] == 'Food'].sum())
```

```
/var/folders/pt/0nf2m3pj1f3b373wsyfc6glh0000gn/T/ipykernel_52421/4214387758.py:1: DeprecationWarning:
```

```
DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.
```

```
Out[96]: CustomerID
1      1.0
2      2.0
3      1.0
4      2.0
5      1.0
...
996     1.0
997     1.0
998     1.0
999     2.0
1000    1.0
Name: Transaction_Count, Length: 1000, dtype: float64
```

```
In [97]: test3 = (
    daily_spending
    .assign(TransactionDate=lambda df: pd.to_datetime(df['TransactionDate']))
    .sort_values(['CustomerID', 'TransactionDate'])
    .set_index('TransactionDate')
    .groupby('CustomerID')['Total_Spent']
    .rolling('7D', closed='left')
    .sum()
    .reset_index()
    .rename(columns={'Total_Spent': 'Rolling_7D_Sum'})
)
```

```
In [98]: test4 = (
    daily_spending
    .assign(TransactionDate=lambda df: pd.to_datetime(df['TransactionDate']))
    .sort_values(['CustomerID', 'TransactionDate'])
    .set_index('TransactionDate')
    .groupby(['CustomerID', 'ProductCategory'])['Total_Spent']
    .rolling('7D', closed='left')
    .sum()
    .reset_index()
)
```

```
In [99]: test4 = test4.fillna(0).groupby(["CustomerID", "ProductCategory"])["Total_Spent"]
test4 = test4.pivot_table(index="CustomerID", columns="ProductCategory", val
```

```
In [100]: test4
```

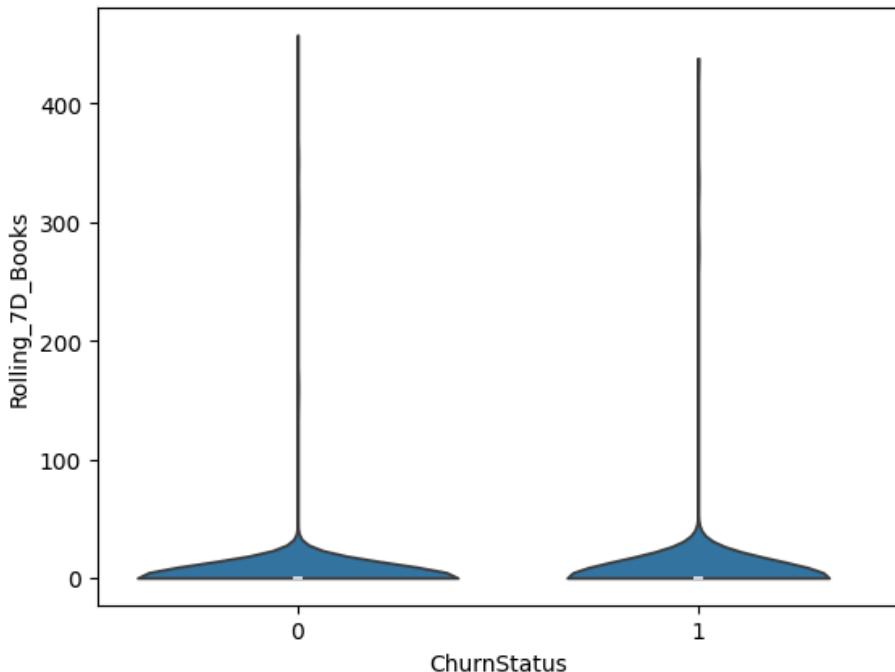
Out[100...]

	CustomerID	Rolling_7D_Books	Rolling_7D_Clothing	Rolling_7D_Electronics
0	1	0.0	0.0	0.0
1	2	0.0	0.0	0.0
2	3	0.0	0.0	0.0
3	4	0.0	0.0	0.0
4	5	0.0	0.0	0.0
...
995	996	0.0	0.0	0.0
996	997	0.0	0.0	0.0
997	998	0.0	0.0	0.0
998	999	0.0	0.0	0.0
999	1000	0.0	0.0	0.0

1000 rows × 7 columns

In [62]: `sns.violinplot(x="ChurnStatus" , y= "Rolling_7D_Books"
, data=test4, cut = 0)`

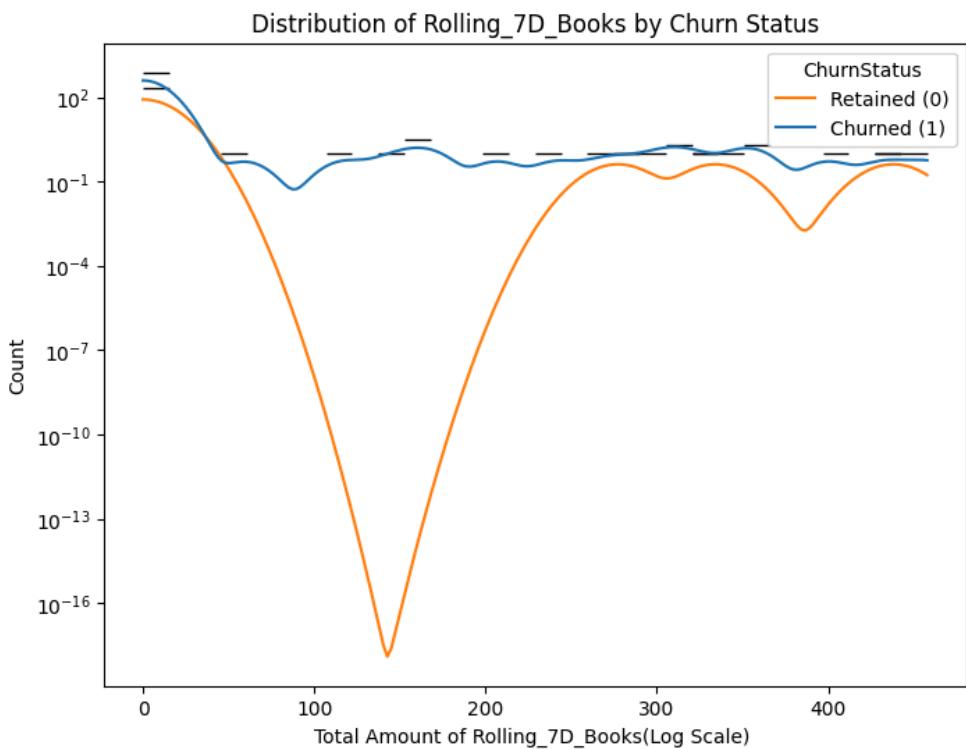
Out[62]: <Axes: xlabel='ChurnStatus', ylabel='Rolling_7D_Books'>



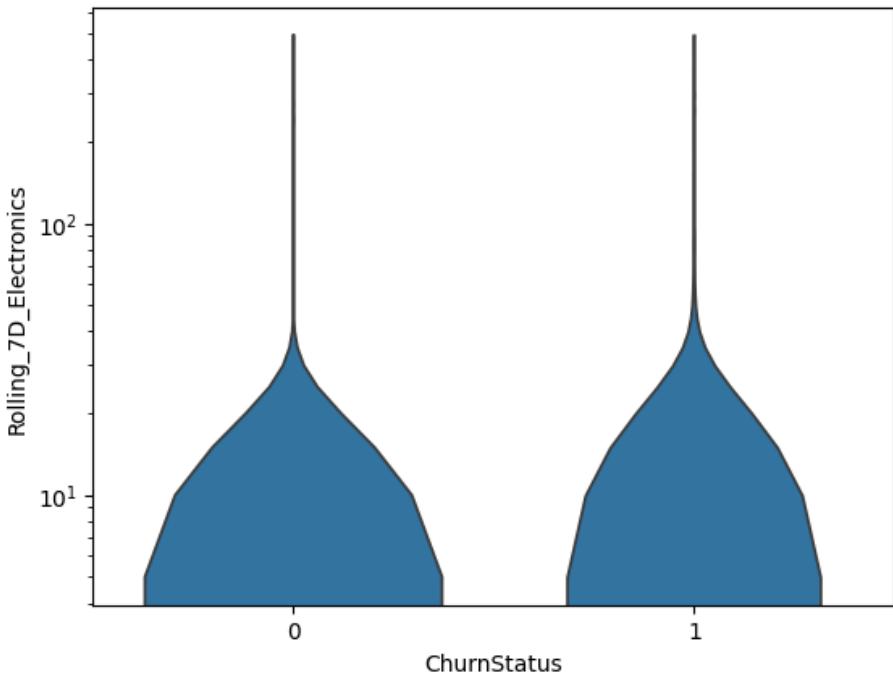
```
In [63]: plt.figure(figsize=(8,6))

sns.histplot(data=test4, x="Rolling_7D_Books", hue="ChurnStatus",
             bins=30, kde=True, log_scale=(False, True), alpha=0.5)

plt.title("Distribution of Rolling_7D_Books by Churn Status")
plt.xlabel("Total Amount of Rolling_7D_Books(Log Scale) ")
plt.ylabel("Count")
plt.legend(title="ChurnStatus", labels=["Retained (0)", "Churned (1)"]) # >
plt.show()
```



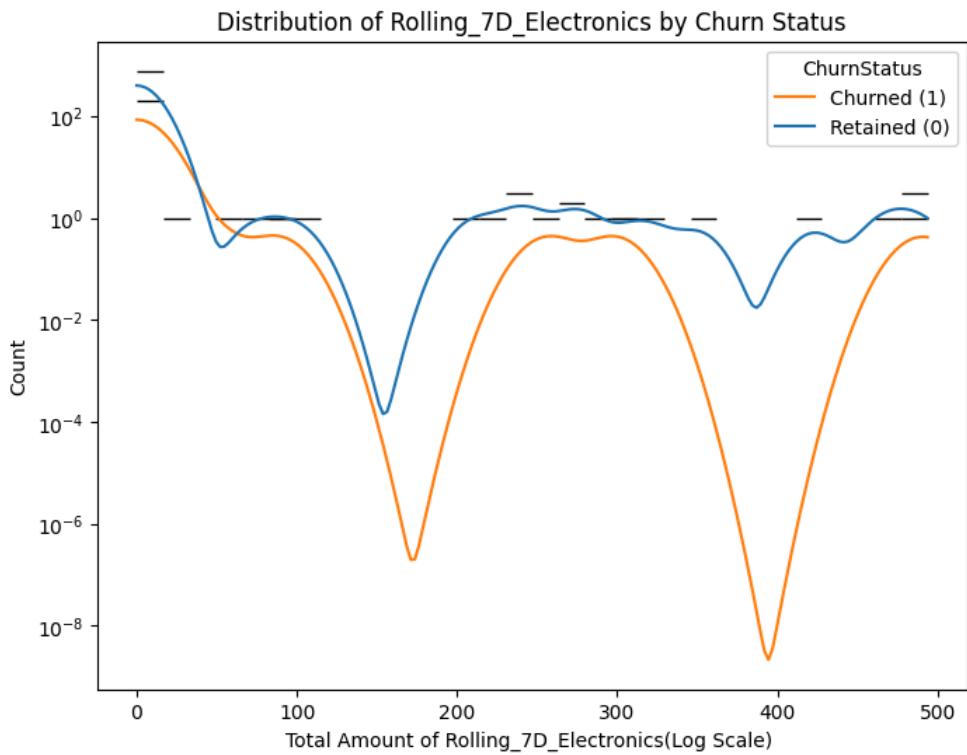
```
In [64]: sns.violinplot(x="ChurnStatus" , y= "Rolling_7D_Electronics"
                     , data=test4, cut = 0)
plt.yscale("log")
```



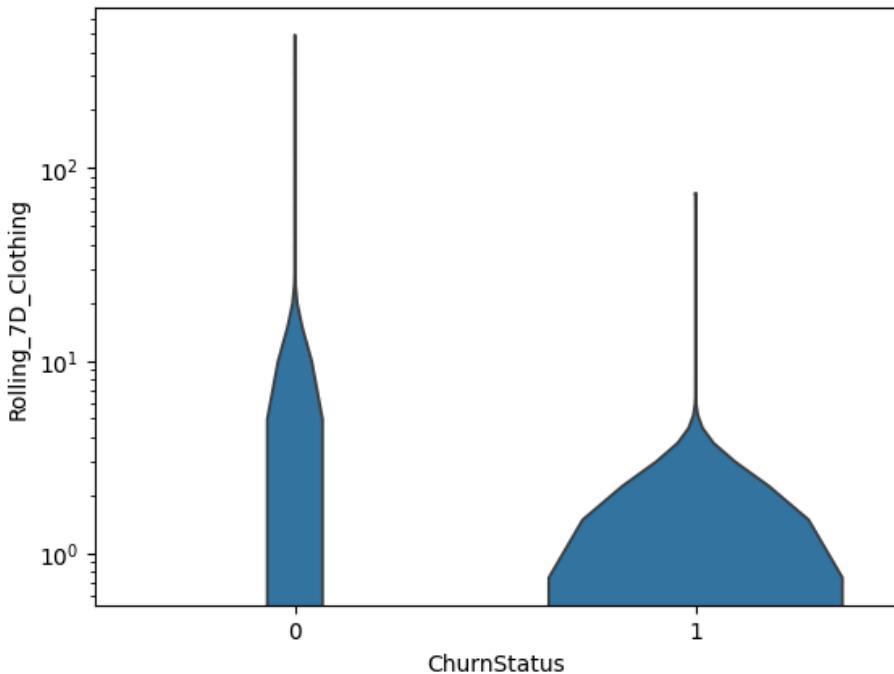
```
In [107]: plt.figure(figsize=(8,6))

sns.histplot(data=test4, x="Rolling_7D_Electronics", hue="ChurnStatus",
             bins=30, kde=True, log_scale=(False, True), alpha=0.5)

plt.title("Distribution of Rolling_7D_Electronics by Churn Status")
plt.xlabel("Total Amount of Rolling_7D_Electronics(Log Scale) ")
plt.ylabel("Count")
plt.legend(title="ChurnStatus", labels=["Churned (1)", "Retained (0)"])
plt.savefig("Plots/KDE_RollingElectronics.png", dpi=300, bbox_inches='tight')
plt.show()
```



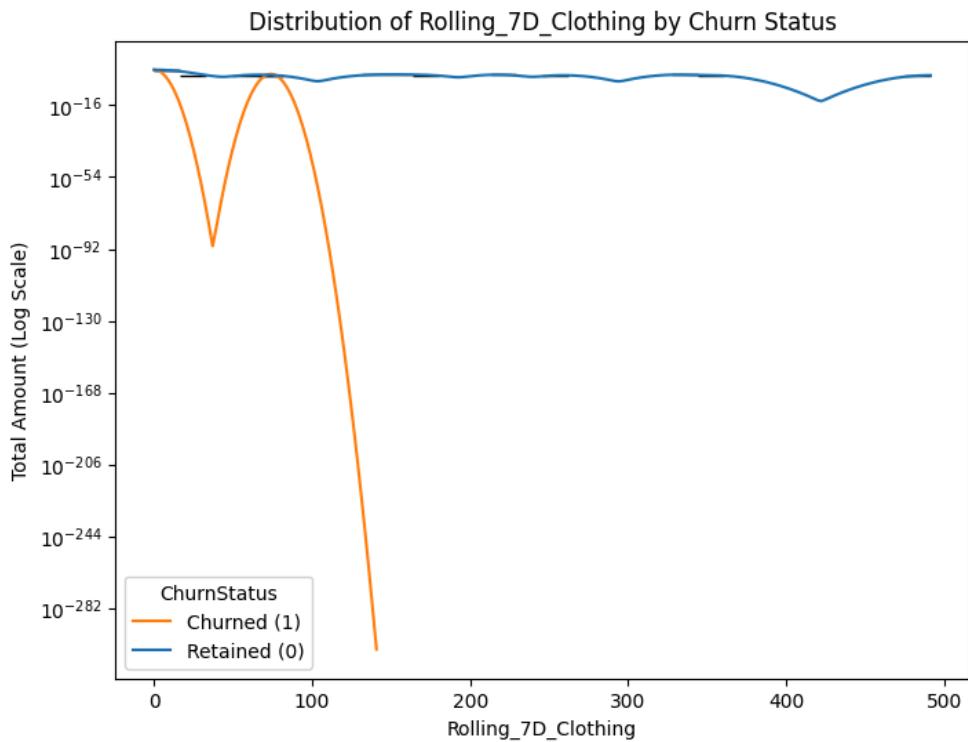
```
In [66]: sns.violinplot(x="ChurnStatus" , y= "Rolling_7D_Clothing"
                     , data=test4, cut = 0)
plt.yscale("log")
```



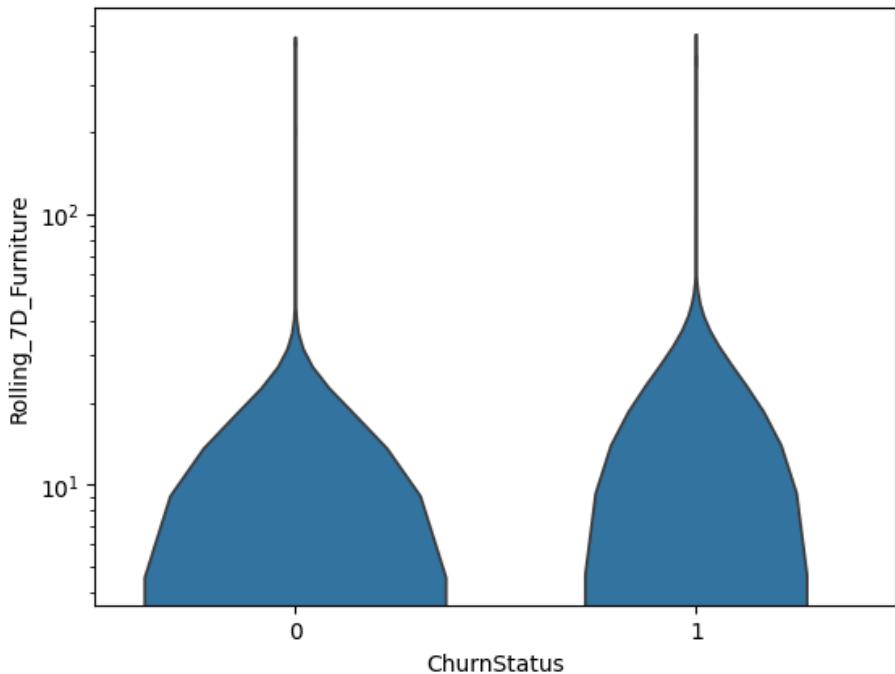
```
In [108]: plt.figure(figsize=(8,6))

sns.histplot(data=test4, x="Rolling_7D_Clothing", hue="ChurnStatus",
             bins=30, kde=True, log_scale=(False, True), alpha=0.5)

plt.title("Distribution of Rolling_7D_Clothing by Churn Status")
plt.xlabel("Rolling_7D_Clothing")
plt.ylabel("Total Amount (Log Scale)")
plt.legend(title="ChurnStatus", labels=["Churned (1)", "Retained (0)"])
plt.savefig("Plots/KDE_RollingClothing.png", dpi=300, bbox_inches='tight')
plt.show()
```



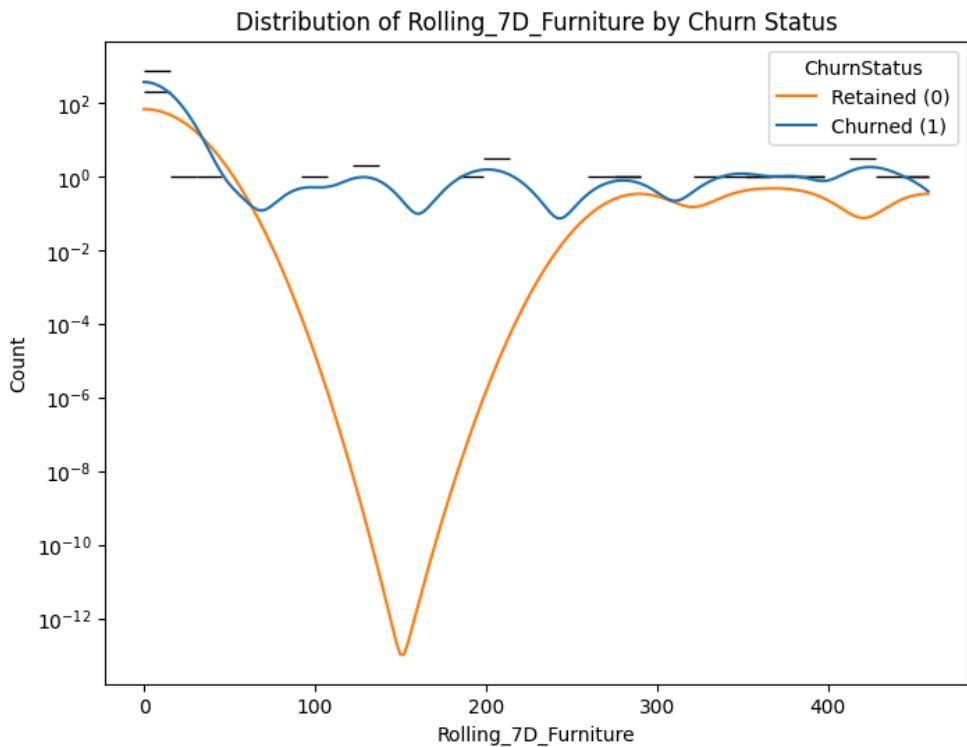
```
In [109]: sns.violinplot(x="ChurnStatus" , y= "Rolling_7D_Furniture"
                     , data=test4, cut = 0)
plt.yscale("log")
```



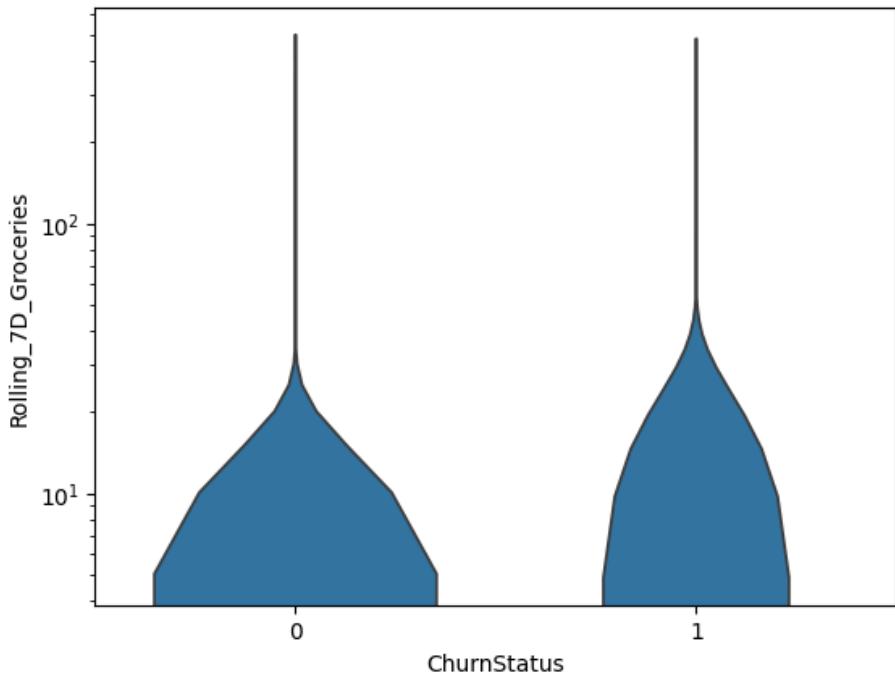
```
In [180]: plt.figure(figsize=(8,6))

sns.histplot(data=test4, x="Rolling_7D_Furniture", hue="ChurnStatus",
             bins=30, kde=True, log_scale=(False, True), alpha=0.5)

plt.title("Distribution of Rolling_7D_Furniture by Churn Status")
plt.xlabel("Rolling_7D_Furniture")
plt.ylabel("Count")
plt.legend(title="ChurnStatus", labels=["Retained (0)", "Churned (1)"])
plt.savefig("Plots/Rolling_7D_Furniture.png", dpi=300, bbox_inches='tight')
plt.show()
```



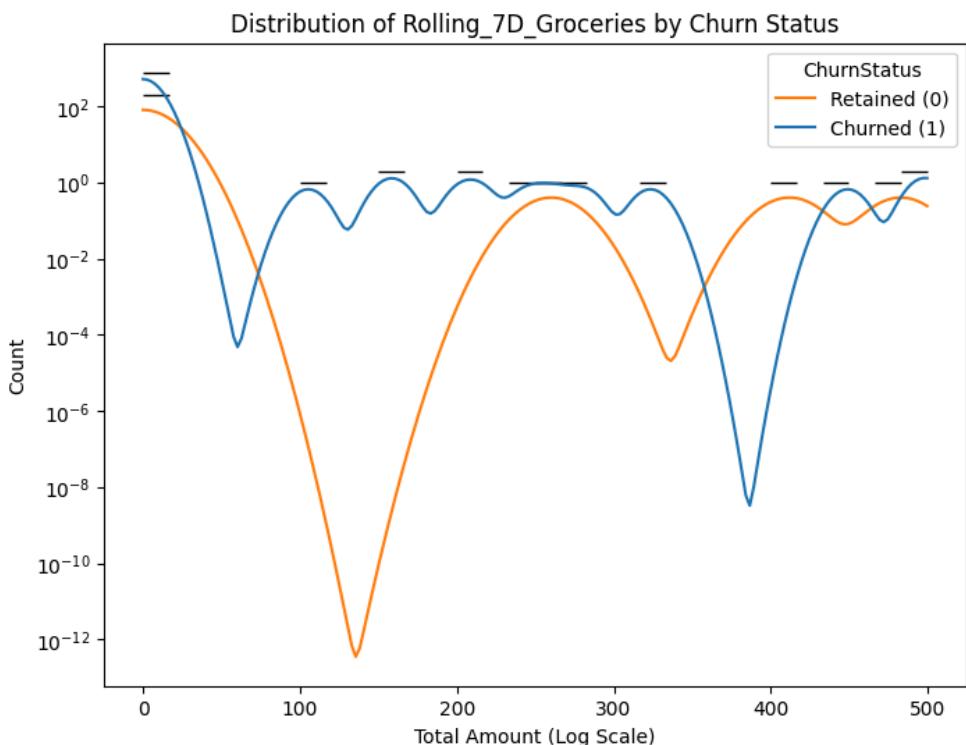
```
In [111]: sns.violinplot(x="ChurnStatus" , y= "Rolling_7D_Groceries"
                     , data=test4, cut = 0)
plt.yscale("log")
```



```
In [179]: plt.figure(figsize=(8,6))

sns.histplot(data=test4, x="Rolling_7D_Groceries", hue="ChurnStatus",
             bins=30, kde=True, log_scale=(False, True), alpha=0.5)

plt.title("Distribution of Rolling_7D_Groceries by Churn Status")
plt.xlabel("Total Amount (Log Scale)")
plt.ylabel("Count")
plt.legend(title="ChurnStatus", labels=["Retained (0)", "Churned (1)"])
plt.savefig("Plots/Rolling_7D_Groceries.png", dpi=300, bbox_inches='tight')
plt.show()
```



```
In [113... test4["Rolling_7D_Furniture_bined"] = pd.cut(test4["Rolling_7D_Furniture"], bins = [-np.inf, 0, 100, 200, 300, 400, 500, np.inf], labels = ["0-100", "100-200", "200-300", "300-400", "400-500", "500+"])

In [114... test4["Rolling_7D_Clothing_bined"] = pd.cut(test4["Rolling_7D_Clothing"], bins = [-np.inf, 0, 100, 200, 300, 400, 500, np.inf], labels = ["0-100", "100-200", "200-300", "300-400", "400-500", "500+"])

In [115... test4["Rolling_7D_Electronics_bined"] = pd.cut(test4["Rolling_7D_Electronics"], bins = [-np.inf, 0, 100, 200, 300, 400, 500, np.inf], labels = ["0-100", "100-200", "200-300", "300-400", "400-500", "500+"])

In [116... test4["Rolling_7D_Books_bined"] = pd.cut(test4["Rolling_7D_Books"], bins = [-np.inf, 0, 100, 200, 300, 400, 500, np.inf], labels = ["0-100", "100-200", "200-300", "300-400", "400-500", "500+"])

In [117... test4["Rolling_7D_Groceries_bined"] = pd.cut(test4["Rolling_7D_Groceries"], bins = [-np.inf, 0, 100, 200, 300, 400, 500, np.inf], labels = ["0-100", "100-200", "200-300", "300-400", "400-500", "500+"])

In [118... test3 = test3.groupby("CustomerID")["Rolling_7D_Sum"].max().reset_index()
test_count = test3.merge(data["Churn_Status"], how = "left", on = "CustomerID")

In [119... test_count.fillna(0)
```

Out[119...]

	CustomerID	Rolling_7D_Sum	ChurnStatus
0	1	0.00	0
1	2	397.37	1
2	3	0.00	0
3	4	241.45	0
4	5	0.00	0
...
995	996	0.00	0
996	997	0.00	0
997	998	0.00	0
998	999	80.28	0
999	1000	0.00	0

1000 rows × 3 columns

```
In [120...]: count_label = pd.cut(test_count.fillna(0)[ 'Rolling_7D_Sum' ], bins = [-1,200,
```

```
In [121...]: rolling_dummies = pd.get_dummies(count_label)
test_count = pd.concat([test_count, rolling_dummies], axis=1).fillna(0)
test_count = test_count.T.drop_duplicates().T
```

```
In [122...]: test_count = test_count.merge(test4, how="left", on = ["CustomerID", "ChurnSt
```

```
In [123...]: test_count
```

Out[123...]

	CustomerID	Rolling_7D_Sum	ChurnStatus	Rolling_7D_0-200	Rolling_7D_200-400
0	1	0.0	0	True	False
1	2	397.37	1	False	True
2	3	0.0	0	True	False
3	4	241.45	0	False	True
4	5	0.0	0	True	False
...
995	996	0.0	0	True	False
996	997	0.0	0	True	False
997	998	0.0	0	True	False
998	999	80.28	0	True	False
999	1000	0.0	0	True	False

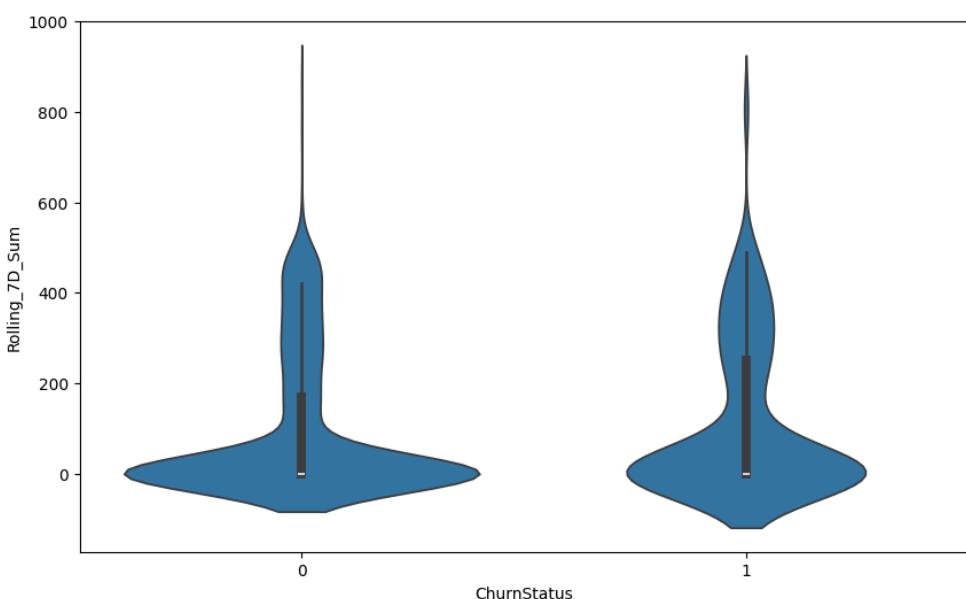
1000 rows × 17 columns

In [124...]

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.violinplot(x="ChurnStatus" , y= "Rolling_7D_Sum"
                , data=test_count)
```

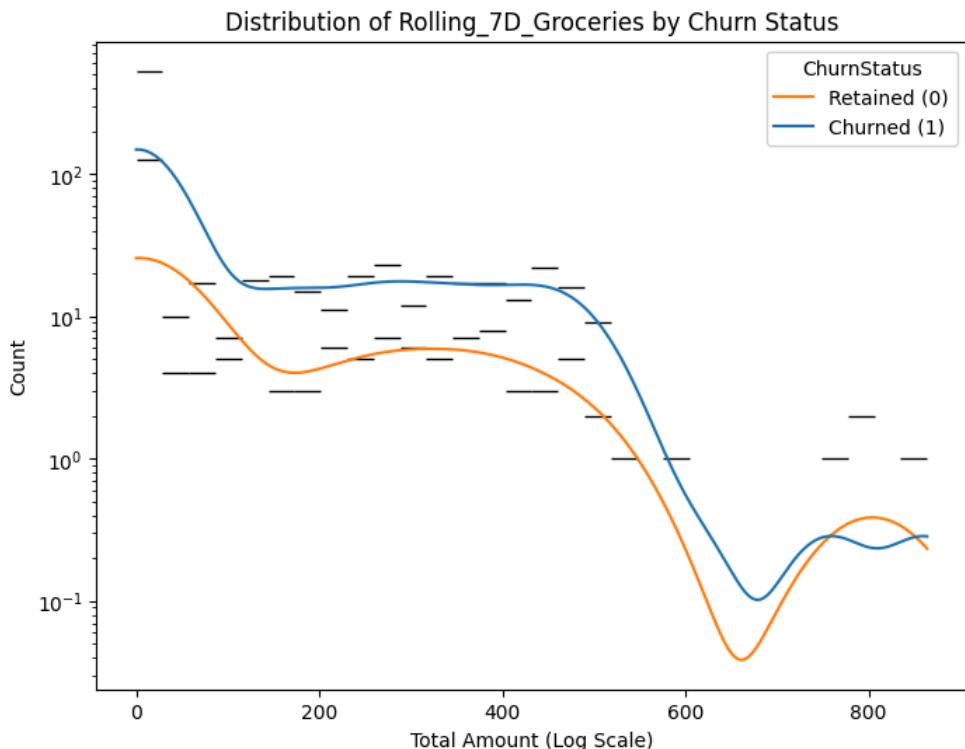
Out[124...]



```
In [125... plt.figure(figsize=(8,6))

sns.histplot(data=test_count, x="Rolling_7D_Sum", hue="ChurnStatus",
             bins=30, kde=True, log_scale=(False, True), alpha=0.5)

plt.title("Distribution of Rolling_7D_Groceries by Churn Status")
plt.xlabel("Total Amount (Log Scale)")
plt.ylabel("Count")
plt.legend(title="ChurnStatus", labels=["Retained (0)", "Churned (1)"]) # >
plt.show()
```



```
In [126... duplicate_ids = data["Churn_Status"]["CustomerID"].duplicated().sum()
print(f"repeated columns: {duplicate_ids}")
```

repeated columns: 0

```
In [127... df_grouped = df_diff.groupby(["ProductCategory","CustomerID"])[["First_Transaction",
                                                               "total_Transaction", "mean_Transaction",
                                                               "count_Transaction"]].sum().reset_index()

df_grouped
```

Out[127...]

	ProductCategory	CustomerID	First_Transaction	Last_Transaction	total
0	Books	3	241.06	241.06	
1	Books	6	365.57	365.57	
2	Books	7	86.73	86.73	
3	Books	8	240.92	263.24	
4	Books	11	163.78	426.06	
...	
3092	Groceries	991	201.31	257.17	
3093	Groceries	994	71.78	336.21	
3094	Groceries	995	494.90	494.90	
3095	Groceries	999	260.06	387.96	
3096	Groceries	1000	232.06	375.34	

3097 rows × 7 columns

In [128...]

```
df_pivot = df_diff.pivot_table(index="CustomerID", columns="ProductCategory"
df_pivot.columns = ['_'.join(col).strip() for col in df_pivot.columns.values]
df_pivot.reset_index(inplace=True)
```

In [129...]

```
df_pivot.columns
```

Out[129...]

```
Index(['CustomerID', 'Days_Difference_Books', 'Days_Difference_Clothing',
       'Days_Difference_Electronics', 'Days_Difference_Furniture',
       'Days_Difference_Groceries', 'First_Transaction_Books',
       'First_Transaction_Clothing', 'First_Transaction_Electronics',
       'First_Transaction_Furniture', 'First_Transaction_Groceries',
       'Last_Transaction_Books', 'Last_Transaction_Clothing',
       'Last_Transaction_Electronics', 'Last_Transaction_Furniture',
       'Last_Transaction_Groceries', 'count_Transaction_Books',
       'count_Transaction_Clothing', 'count_Transaction_Electronics',
       'count_Transaction_Furniture', 'count_Transaction_Groceries',
       'mean_Transaction_Books', 'mean_Transaction_Clothing',
       'mean_Transaction_Electronics', 'mean_Transaction_Furniture',
       'mean_Transaction_Groceries', 'total_Transaction_Books',
       'total_Transaction_Clothing', 'total_Transaction_Electronics',
       'total_Transaction_Furniture', 'total_Transaction_Groceries'],
      dtype='object')
```

In [130...]

```
category2 = ["Rolling_7D_Furniture_bined", "Rolling_7D_Electronics_bined", "test_count_endcoded = pd.get_dummies(test_count, columns=category2, drop_fir
test_count_endcoded.head()
```

Out[130...]

	CustomerID	Rolling_7D_Sum	ChurnStatus	Rolling_7D_0-200	Rolling_7D_200-400
0	1	0.0	0	True	False
1	2	397.37	1	False	True
2	3	0.0	0	True	False
3	4	241.45	0	False	True
4	5	0.0	0	True	False

5 rows × 27 columns

In [131...]

```
test_count_endcoded.columns
```

Out[131...]

```
Index(['CustomerID', 'Rolling_7D_Sum', 'ChurnStatus', 'Rolling_7D_0-200',
       'Rolling_7D_200-400', 'Rolling_7D_400-600', 'Rolling_7D_600+',
       'Rolling_7D_Books', 'Rolling_7D_Clothing', 'Rolling_7D_Electronics',
       'Rolling_7D_Furniture', 'Rolling_7D_Groceries',
       'Rolling_7D_Furniture_bined_Rolling_7D_Furniture_210',
       'Rolling_7D_Furniture_bined_Rolling_7D_Furniture_210+',
       'Rolling_7D_Electronics_bined_Rolling_7D_Electronics_310',
       'Rolling_7D_Electronics_bined_Rolling_7D_Electronics_360',
       'Rolling_7D_Electronics_bined_Rolling_7D_Electronics_440',
       'Rolling_7D_Electronics_bined_Rolling_7D_Electronics_440+',
       'Rolling_7D_Books_bined_Rolling_7D_Books_220+',
       'Rolling_7D_Books_bined_Rolling_7D_Books_80',
       'Rolling_7D_Groceries_bined_Rolling_7D_Groceries_280',
       'Rolling_7D_Groceries_bined_Rolling_7D_Groceries_320',
       'Rolling_7D_Groceries_bined_Rolling_7D_Groceries_380',
       'Rolling_7D_Groceries_bined_Rolling_7D_Groceries_410+',
       'Rolling_7D_Groceries_bined_Rolling_7D_Groceries_80',
       'Rolling_7D_Clothing_bined_Rolling_7D_Clothing_150+',
       'Rolling_7D_Clothing_bined_Rolling_7D_Clothing_40'],
      dtype='object')
```

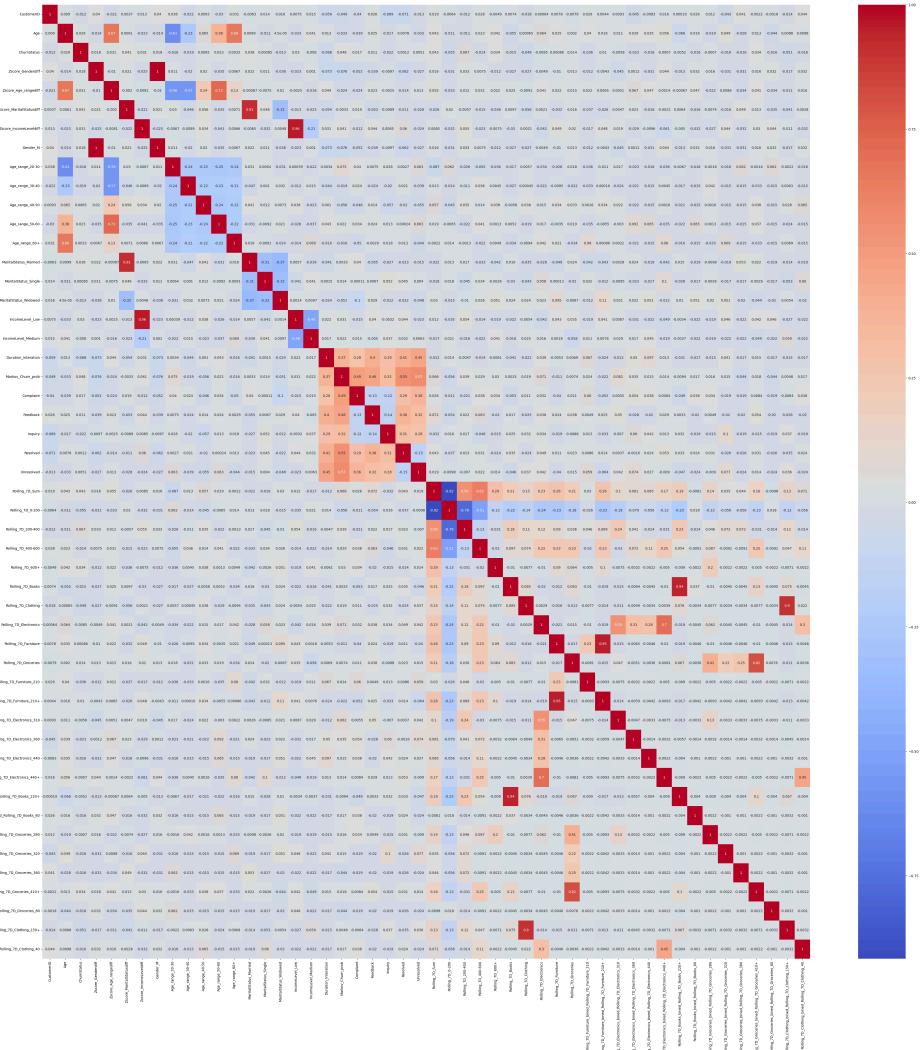
In [134...]

```
enhanced_Interaction_Customer_Transactions = enhanced_Interaction_Customer.m
```

In [135...]

```
corr_df = enhanced_Interaction_Customer_Transactions.corr()
plt.figure(figsize=(50, 50))
sns.heatmap(corr_df, annot=True, cmap='coolwarm')
```

Out[135...]



Online Activity Sheet

```
In [137]: Online_Activity = data['Online_Activity']
```

```
In [138...]: Online_Activity = Online_Activity.merge(data['Churn_Status'], how = "left",
```

In [139... Online_Activity

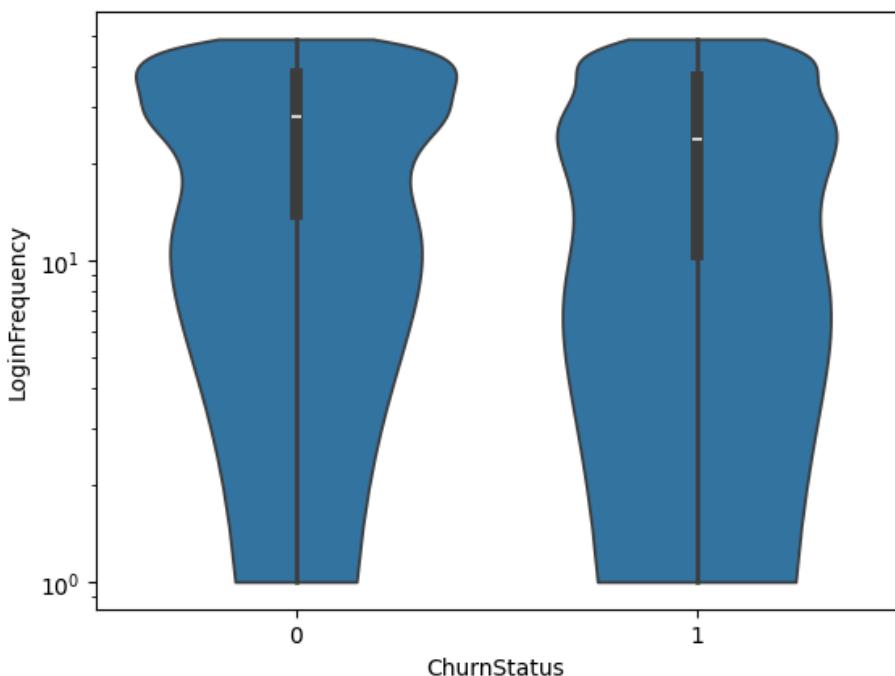
Out[139...]

	CustomerID	LastLoginDate	LoginFrequency	ServiceUsage	ChurnStatus
0	1	2023-10-21	34	Mobile App	0
1	2	2023-12-05	5	Website	1
2	3	2023-11-15	3	Website	0
3	4	2023-08-25	2	Website	0
4	5	2023-10-27	41	Website	0
...
995	996	2023-01-29	38	Mobile App	0
996	997	2023-04-01	5	Mobile App	0
997	998	2023-07-10	47	Website	0
998	999	2023-01-08	23	Website	0
999	1000	2023-08-13	22	Mobile App	0

1000 rows × 5 columns

In [140...]

```
sns.violinplot(x="ChurnStatus" , y= "LoginFrequency"  
                 , data=Online_Activity, cut = 0)  
plt.yscale("log")
```



In [141...]

```
plt.figure(figsize=(8,6))
```

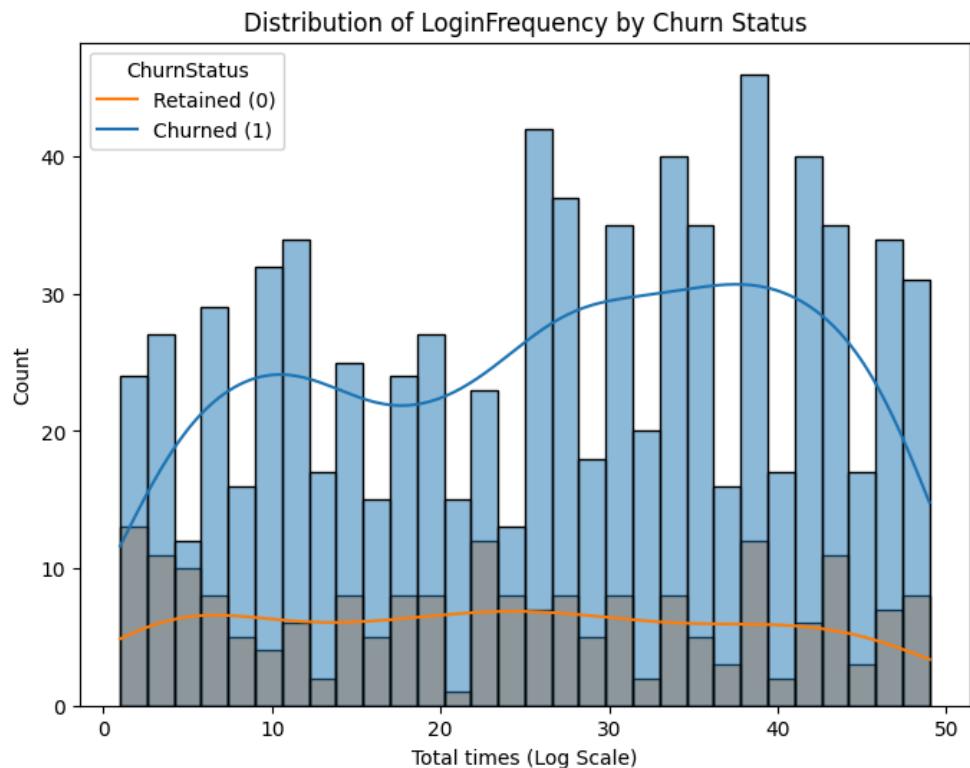
Loading [MathJax]/extensions/Safe.js

```

sns.histplot(data=Online_Activity, x="LoginFrequency", hue="ChurnStatus",
             bins=30, kde=True, log_scale=(False, False), alpha=0.5)

plt.title("Distribution of LoginFrequency by Churn Status")
plt.xlabel("Total times (Log Scale)")
plt.ylabel("Count")
plt.legend(title="ChurnStatus", labels=["Retained (0)", "Churned (1)"])
plt.show()

```

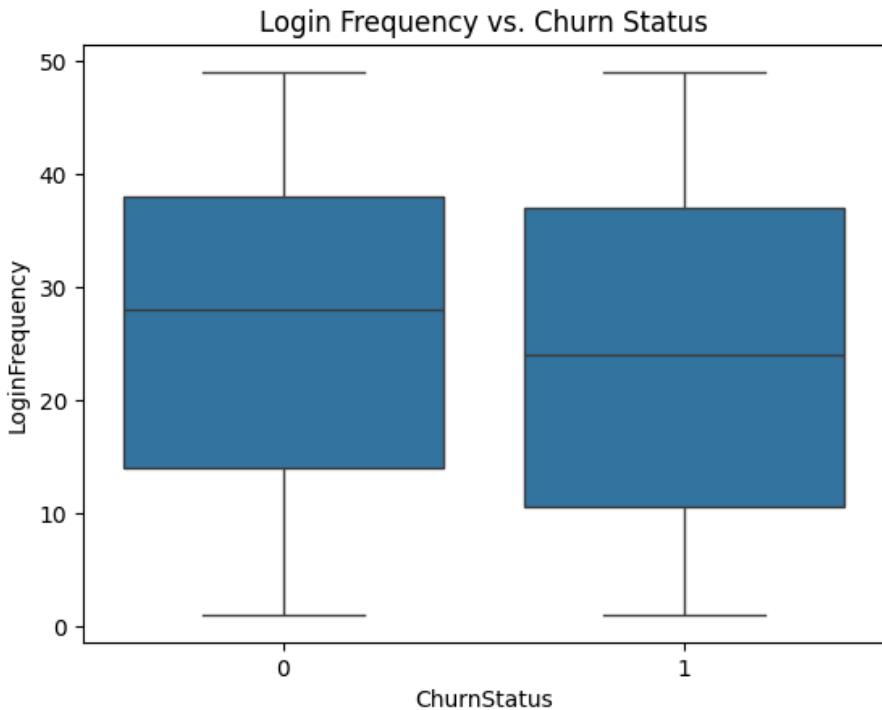


```

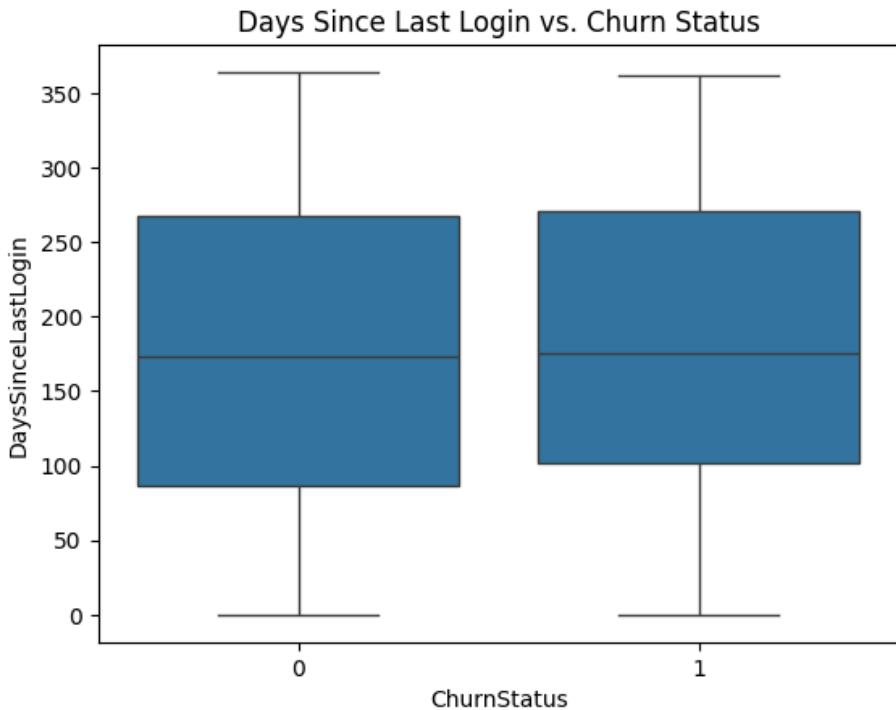
In [142]: import seaborn as sns
          import matplotlib.pyplot as plt

sns.boxplot(x="ChurnStatus", y="LoginFrequency", data=Online_Activity)
plt.title("Login Frequency vs. Churn Status")
plt.show()

```



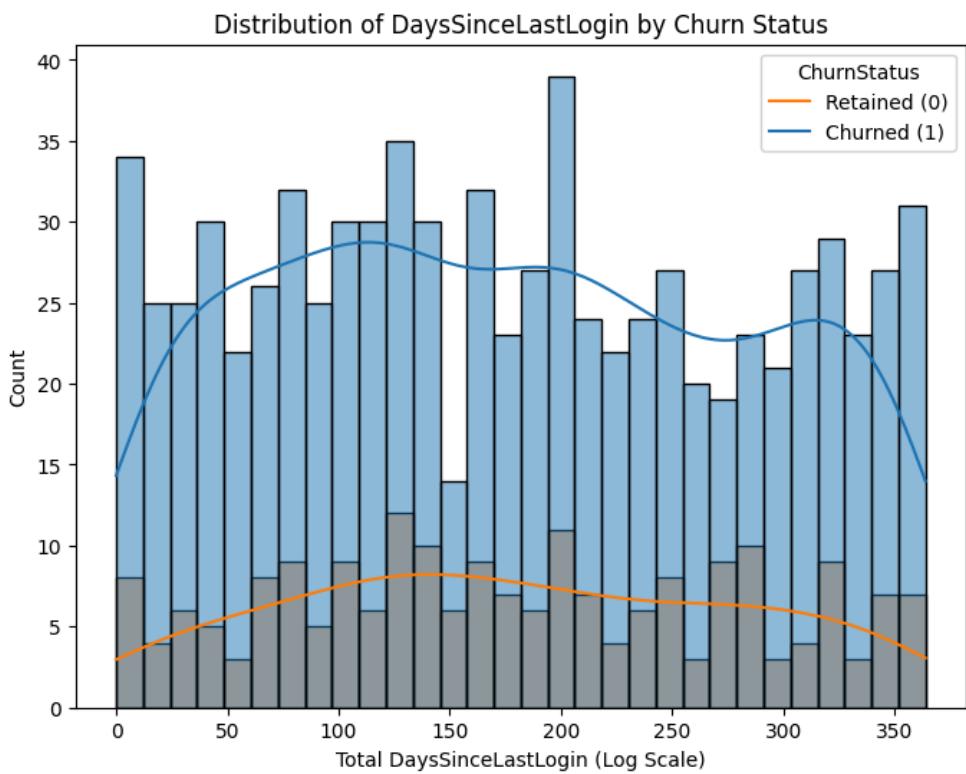
```
In [143]: from datetime import datetime  
  
Online_Activity["DaysSinceLastLogin"] = (Online_Activity["LastLoginDate"].map(lambda x: x.date() - datetime.now().date())).dt.days  
  
sns.boxplot(x="ChurnStatus", y="DaysSinceLastLogin", data=Online_Activity)  
plt.title("Days Since Last Login vs. Churn Status")  
plt.show()
```



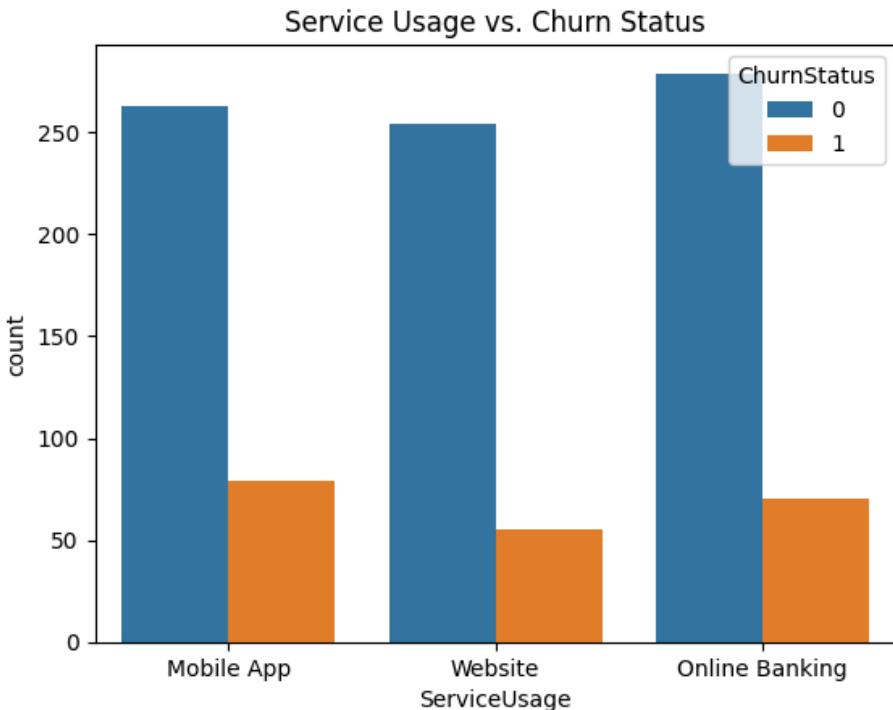
```
In [144]: plt.figure(figsize=(8,6))

sns.histplot(data=Online_Activity, x="DaysSinceLastLogin", hue="ChurnStatus",
             bins=30, kde=True, log_scale=(False, False), alpha=0.5)

plt.title("Distribution of DaysSinceLastLogin by Churn Status")
plt.xlabel("Total DaysSinceLastLogin (Log Scale)")
plt.ylabel("Count")
plt.legend(title="ChurnStatus", labels=["Retained (0)", "Churned (1)"])
plt.show()
```



```
In [145...]: sns.countplot(x="ServiceUsage", hue="ChurnStatus", data=Online_Activity)
plt.title("Service Usage vs. Churn Status")
plt.show()
```



```
In [146]: import scipy.stats as stats

t_stat, p_value = stats.ttest_ind(
    Online_Activity.loc[Online_Activity["ChurnStatus"] == 1, "LoginFrequency"],
    Online_Activity.loc[Online_Activity["ChurnStatus"] == 0, "LoginFrequency"]
)
print(f"T-test for LoginFrequency: p-value = {p_value:.4f}")

T-test for LoginFrequency: p-value = 0.0098
```

```
In [104]: import scipy.stats as stats

t_stat, p_value = stats.ttest_ind(
    Online_Activity.loc[Online_Activity["ChurnStatus"] == 1, "DaysSinceLastLogin_Normalized"],
    Online_Activity.loc[Online_Activity["ChurnStatus"] == 0, "DaysSinceLastLogin_Normalized"]
)
print(f"T-test for DaysSinceLastLogin_Normalized: p-value = {p_value:.4f}")

T-test for DaysSinceLastLogin_Normalized: p-value = 0.7749
```

```
In [105]: import pandas as pd
import scipy.stats as stats

contingency_table = pd.crosstab(Online_Activity["ServiceUsage"], Online_Activity["ChurnStatus"])

# Kai^2 test
chi2, p, dof, expected = stats.chi2_contingency(contingency_table)
```

```

print(f"Kai squared statistics: {chi2:.4f}")
print(f"P value: {p:.4f}")

alpha = 0.05
if p < alpha:
    print("ServiceUsage and ChurnStatus have obvious correlation")
else:
    print("ServiceUsage and ChurnStatus have no obvious correlation")

```

Kai squared statistics: 2.8469
P value: 0.2409
ServiceUsage and ChurnStatus have no obvious correlation

```

In [160]: import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

churned = Online_Activity.loc[Online_Activity["ChurnStatus"] == 1, "LoginFrequency"]
retained = Online_Activity.loc[Online_Activity["ChurnStatus"] == 0, "LoginFrequency"]

churned1 = Online_Activity.loc[Online_Activity["ChurnStatus"] == 1, "DaysSinceLastLogin"]
retained1 = Online_Activity.loc[Online_Activity["ChurnStatus"] == 0, "DaysSinceLastLogin"]

n1 = len(churned)
n2 = len(retained)
df = n1 + n2 - 2

x = np.linspace(-4, 4, 1000)
y = stats.t.pdf(x, df)

t_stat, p_value = stats.ttest_ind(churned, retained)
t_stat1, p_value1 = stats.ttest_ind(churned1, retained1)

plt.figure(figsize=(12, 8))
plt.plot(x, y, label=f"T-Distribution (df={df})", color='blue')

alpha = 0.05
t_critical = stats.t.ppf(1 - alpha / 2, df)
plt.fill_between(x, y, where=(x < -t_critical) | (x > t_critical), color='red')

plt.axvline(t_stat, color='green', linestyle='dashed', linewidth=2, label=f"t-stat ({t_stat})")
plt.axvline(t_stat1, color='orange', linestyle='dashed', linewidth=2, label=f"t-stat1 ({t_stat1})")

plt.title("T-test Distribution Plot for LoginFrequency and DaysSinceLastLogin")
plt.xlabel("T-score")
plt.ylabel("Probability Density")

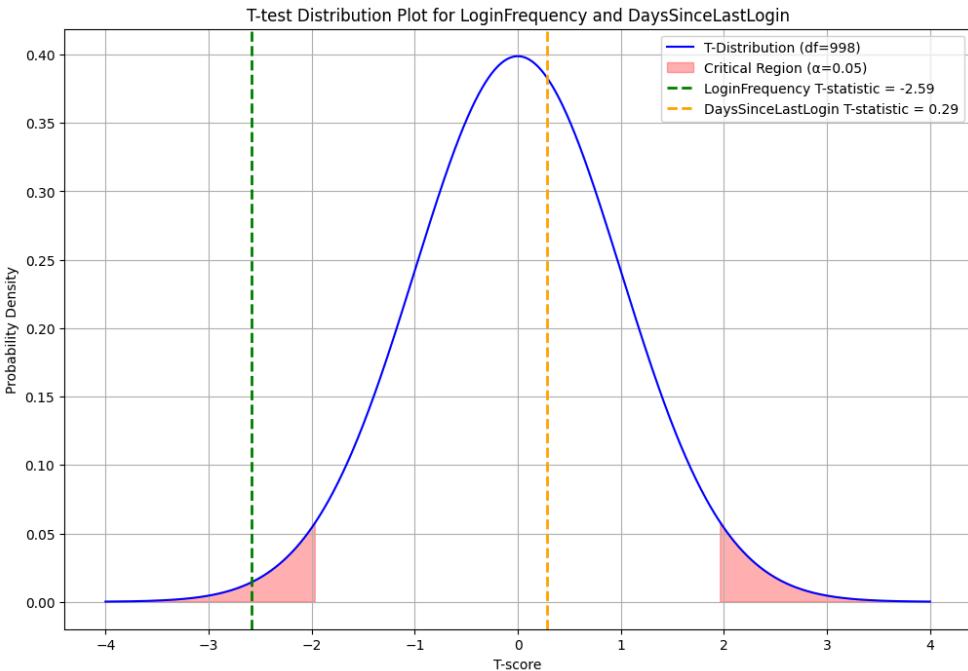
```

```

plt.legend()
plt.grid()
plt.savefig("Plots/T_test_Distribution.png", dpi=300, bbox_inches='tight')
plt.show()

print(f"T-statistic: {t_stat:.4f}, p-value: {p_value:.4f}")

```



T-statistic: -2.5870, p-value: 0.0098

In [159]:

```

import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

df = 1

x = np.linspace(0, 10, 1000)
y = stats.chi2.pdf(x, df)

contingency_table = pd.crosstab(Online_Activity["ServiceUsage"], Online_Acti

chi2_stat, p, dof, expected = stats.chi2_contingency(contingency_table)

plt.figure(figsize=(12, 8))
nlt nlt(x, y, label=f"Chi-Square Distribution (df={df})", color='blue')

```

Loading [MathJax]/extensions/Safe.js

```

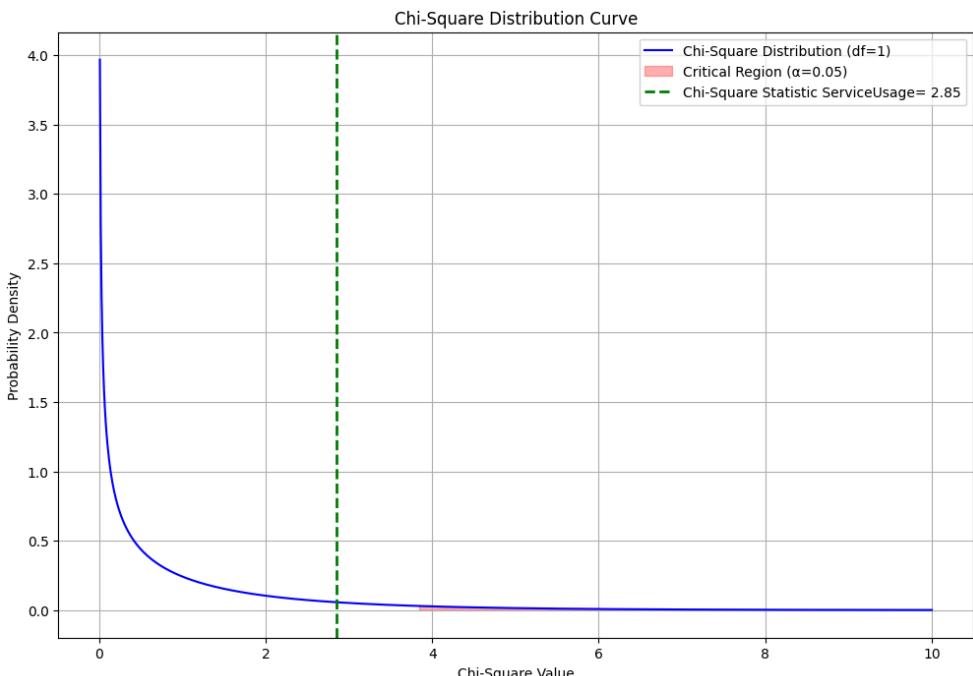
alpha = 0.05
chi2_critical = stats.chi2.ppf(1 - alpha, df)
plt.fill_between(x, y, where=(x > chi2_critical), color='red', alpha=0.3, label="Critical Region (α=0.05)")

plt.axvline(chi2_stat, color='green', linestyle='dashed', linewidth=2, label="Chi-Square Statistic ServiceUsage= 2.85")

plt.title("Chi-Square Distribution Curve")
plt.xlabel("Chi-Square Value")
plt.ylabel("Probability Density")
plt.legend()
plt.grid()
plt.savefig("Plots/Chi-Square_ServiceUsage.png", dpi=300, bbox_inches='tight')

plt.show()

```



In [170]:

```

from scipy.stats import spearmanr
corr, p_value = spearmanr(enhanced_Interaction_Customer_Transaction_Login["A"]
print(f"Spearman Correlation: {corr:.4f}, p-value: {p_value:.4f}")

```

Spearman Correlation: 0.6096, p-value: 0.0000

In [162]:

```

Online_Activity["LoginFrequency_Ratio"] = Online_Activity["LoginFrequency"]
Online_Activity = pd.get_dummies(Online_Activity, columns=["ServiceUsage"], c

```

```
In [164... Online_Activity = Online_Activity.drop(columns=["LastLoginDate"])

In [165... Online_Activity["LoginCategory"] = pd.cut(Online_Activity["LoginFrequency"],

In [166... Online_Activity = pd.get_dummies(Online_Activity, columns=["LoginCategory"],

In [167... enhanced_Interaction_Customer_Transaction_Login = enhanced_Interaction_Cust

In [172... for col in enhanced_Interaction_Customer_Transaction_Login.columns:
    try:
        enhanced_Interaction_Customer_Transaction_Login[col] = pd.to_numeric(
    except:
        print(f"failed at {col}")

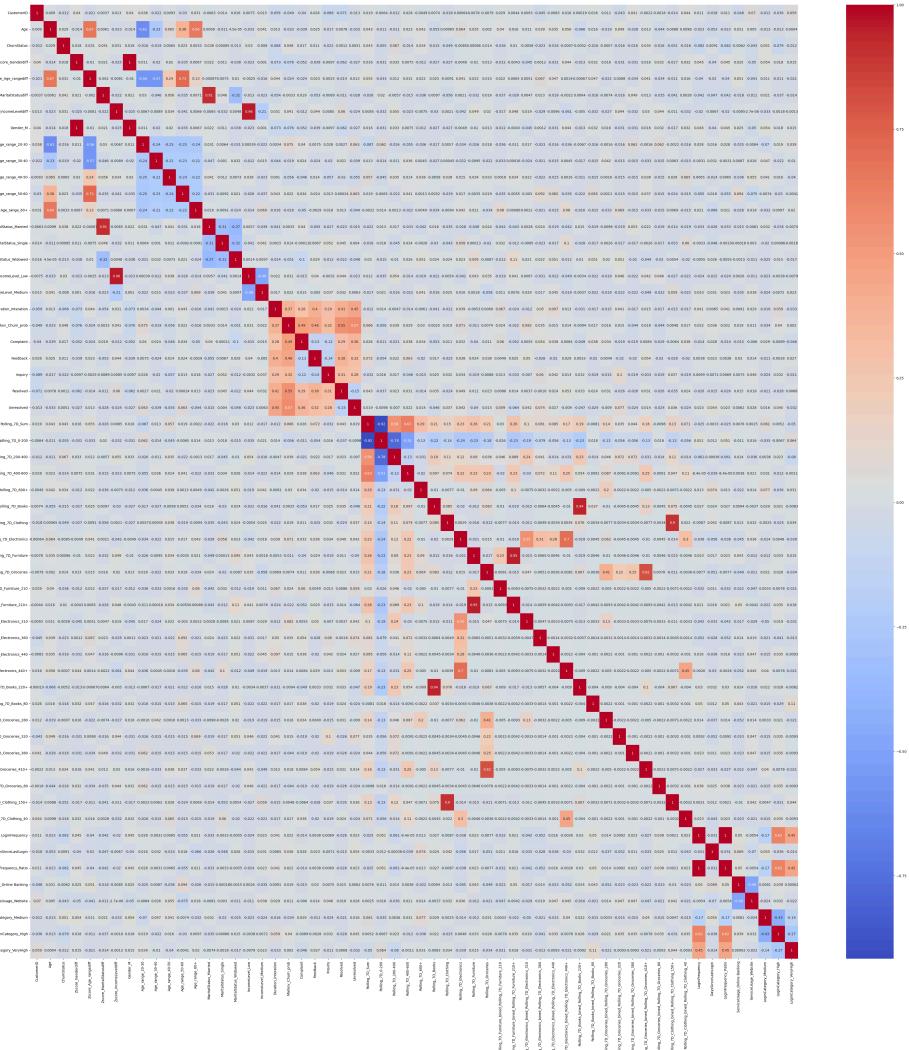
/var/folders/pt/0nf2m3pj1f3b373wsyfc6glh0000gn/T/ipykernel_52421/2591289443.
py:3: FutureWarning:

  errors='ignore' is deprecated and will raise in a future version. Use to_num
  eric without passing `errors` and catch exceptions explicitly instead

In [173... enhanced_Interaction_Customer_Transaction_Login["Duration_Interation"] = enh

In [178... corr = enhanced_Interaction_Customer_Transaction_Login.corr()

plt.figure(figsize=(50, 50))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.savefig("Plots/Correlation.png", dpi=300, bbox_inches='tight')
plt.show()
```



```
In [177]: enhanced_Interaction_Customer_Transaction_Login.to_csv("enhanced_customer_da
```

This notebook was converted with [convert.ploomber.io](#)

Loading [MathJax]/extensions/Safe.js

```
In [1]: from imblearn.under_sampling import RandomUnderSampler
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import networkx as nx
from collections import defaultdict
import plotly.graph_objects as go
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from xgboost import XGBClassifier
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

```
In [2]: enhanced_Interaction_Customer_Transaction_Login = pd.read_csv("enhanced_cust
```

```
In [3]: enhanced_Interaction_Customer_Transaction_Login
```

Out[3]:

	CustomerID	Age	ChurnStatus	Zscore_Genderdiff	Zscore_Age_rangediff
0	1	62	0	21.898796	3.071478
1	2	65	1	21.898796	3.071478
2	3	18	0	21.898796	2.238200
3	4	21	0	21.898796	-17.033005
4	5	21	0	21.898796	-17.033005
...
995	996	54	0	-23.096539	20.951841
996	997	19	0	21.898796	2.238200
997	998	47	0	21.898796	6.257337
998	999	23	0	21.898796	-17.033005
999	1000	34	0	21.898796	-19.062154

1000 rows × 58 columns

```
In [4]: enhanced_Interaction_Customer_Transaction_Login.columns
```

```
Out[4]: Index(['CustomerID', 'Age', 'ChurnStatus', 'Zscore_Genderdiff',
   'Zscore_Age_rangediff', 'Zscore_MaritalStatusdiff',
   'Zscore_IncomeLeveldiff', 'Gender_M', 'Age_range_20-30',
   'Age_range_30-40', 'Age_range_40-50', 'Age_range_50-60',
   'Age_range_60+', 'MaritalStatus_Married', 'MaritalStatus_Single',
   'MaritalStatus_Widowed', 'IncomeLevel_Low', 'IncomeLevel_Medium',
   'Duration_Interation', 'Markov_Churn_prob', 'Complaint', 'Feedback',
   'Inquiry', 'Resolved', 'Unresolved', 'Rolling_7D_Sum',
   'Rolling_7D_0-200', 'Rolling_7D_200-400', 'Rolling_7D_400-600',
   'Rolling_7D_600+', 'Rolling_7D_Books', 'Rolling_7D_Clothing',
   'Rolling_7D_Electronics', 'Rolling_7D_Furniture',
   'Rolling_7D_Groceries',
   'Rolling_7D_Furniture_bined_Rolling_7D_Furniture_210',
   'Rolling_7D_Furniture_bined_Rolling_7D_Furniture_210+',
   'Rolling_7D_Electronics_bined_Rolling_7D_Electronics_310',
   'Rolling_7D_Electronics_bined_Rolling_7D_Electronics_360',
   'Rolling_7D_Electronics_bined_Rolling_7D_Electronics_440',
   'Rolling_7D_Electronics_bined_Rolling_7D_Electronics_440+',
   'Rolling_7D_Books_bined_Rolling_7D_Books_220+',
   'Rolling_7D_Books_bined_Rolling_7D_Books_80',
   'Rolling_7D_Groceries_bined_Rolling_7D_Groceries_280',
   'Rolling_7D_Groceries_bined_Rolling_7D_Groceries_320',
   'Rolling_7D_Groceries_bined_Rolling_7D_Groceries_380',
   'Rolling_7D_Groceries_bined_Rolling_7D_Groceries_410+',
   'Rolling_7D_Groceries_bined_Rolling_7D_Groceries_80',
   'Rolling_7D_Clothing_bined_Rolling_7D_Clothing_150+',
   'Rolling_7D_Clothing_bined_Rolling_7D_Clothing_40', 'LoginFrequenc
y',
   'DaysSinceLastLogin', 'LoginFrequency_Ratio',
   'ServiceUsage_Online Banking', 'ServiceUsage_Website',
   'LoginCategory_Medium', 'LoginCategory_High', 'LoginCategory_VeryHig
h'],
   dtype='object')
```

```
In [50]: baseline_data = (enhanced_Interaction_Customer_Transaction_Login.copy())[['Cu
```

```
In [51]: baseline_data.columns
```

```
Out[51]: Index(['CustomerID', 'Age', 'ChurnStatus', 'Duration_Interation', 'Gender_M',
   'MaritalStatus_Married', 'MaritalStatus_Single',
   'MaritalStatus_Widowed', 'IncomeLevel_Low', 'IncomeLevel_Medium',
   'Age_range_20-30', 'Age_range_30-40', 'Age_range_40-50',
   'Age_range_50-60', 'Age_range_60+', 'LoginFrequency',
   'DaysSinceLastLogin', 'ServiceUsage_Online Banking',
   'ServiceUsage_Website', 'LoginCategory_Medium', 'LoginCategory_Hig
h',
   'LoginCategory_VeryHigh'],
   dtype='object')
```

```
In [53]: X = baseline_data.drop(columns=['ChurnStatus'])
y = baseline_data['ChurnStatus'].astype(int)
```

```

for col in X.columns:
    try:
        X[col] = pd.to_numeric(X[col], errors='ignore')
    except:
        print(f"can not transform: {col}")

X_train, X_test_baseline, y_train, y_test_baseline = train_test_split(X, y,
xgb_model_baseline = XGBClassifier(
    n_estimators=100,
    max_depth=10,
    learning_rate=0.1,
    random_state=42
)

xgb_model_baseline.fit(X_train, y_train)
y_pred = xgb_model_baseline.predict(X_test_baseline)

report = classification_report(y_test_baseline, y_pred)
print("report:", report)

report:          precision    recall   f1-score   support
               0       0.80      0.93      0.86      159
               1       0.27      0.10      0.14       41
               accuracy           0.76      200
            macro avg       0.53      0.51      0.50      200
     weighted avg       0.69      0.76      0.71      200

```

```

/var/folders/pt/0nf2m3pj1f3b373wsyfc6glh0000gn/T/ipykernel_91558/2402704606.py:7: FutureWarning: errors='ignore' is deprecated and will raise in a future version. Use to_numeric without passing `errors` and catch exceptions explicitly instead
X[col] = pd.to_numeric(X[col], errors='ignore')

```

In [8]:

```

X = enhanced_Interaction_Customer_Transaction_Login.drop(columns=["ChurnStatus"])
y = enhanced_Interaction_Customer_Transaction_Login[["ChurnStatus"]].astype(int)

for col in X.columns:
    try:
        X[col] = pd.to_numeric(X[col], errors='ignore')
    except:
        print(f"can not transform {col}")

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

```

```
/var/folders/pt/0nf2m3pj1f3b373wsyfc6glh0000gn/T/ipykernel_91558/4119102381.py:7: FutureWarning: errors='ignore' is deprecated and will raise in a future version. Use to_numeric without passing `errors` and catch exceptions explicitly instead
    X[col] = pd.to_numeric(X[col] , errors='ignore') # 'ignore' 让非数值列保持不变
```

```
In [9]: def focal_loss(alpha, gamma):
    def focal_loss_func(y_pred, y_true):
        p_t = np.exp(-y_pred)
        loss = alpha * (1 - p_t) ** gamma * y_pred
        return "focal_loss", np.mean(loss)
    return focal_loss_func
```

```
In [10]: from sklearn.feature_selection import mutual_info_classif

mi_scores = mutual_info_classif(enhanced_Interaction_Customer_Transaction_Lo
, random_state=123)
mi_scores_df = pd.DataFrame({'Feature': enhanced_Interaction_Customer_Transa
mi_scores_df = mi_scores_df.sort_values(by="MI Score", ascending=False)

print(mi_scores_df.head(40))
```

	Feature	MI Score
53	ServiceUsage_Website	0.023989
30	Rolling_7D_Clothing	0.020915
54	LoginCategory_Medium	0.020584
4	Zscore_MaritalStatusdiff	0.019372
12	MaritalStatus_Married	0.018458
29	Rolling_7D_Books	0.018309
49	LoginFrequency	0.017536
8	Age_range_30-40	0.016729
7	Age_range_20-30	0.016718
26	Rolling_7D_200-400	0.013307
46	Rolling_7D_Groceries_bined_Rolling_7D_Grocerie...	0.013058
36	Rolling_7D_Electronics_bined_Rolling_7D_Electr...	0.012295
37	Rolling_7D_Electronics_bined_Rolling_7D_Electr...	0.011920
51	LoginFrequency_Ratio	0.009713
10	Age_range_50-60	0.007855
39	Rolling_7D_Electronics_bined_Rolling_7D_Electr...	0.007560
52	ServiceUsage_Online Banking	0.007440
27	Rolling_7D_400-600	0.006443
23	Unresolved	0.006023
14	MaritalStatus_Widowed	0.004336
9	Age_range_40-50	0.004278
11	Age_range_60+	0.004106
0	CustomerID	0.003323
2	Zscore_Genderdiff	0.002922
22	Resolved	0.002770
17	Duration_Interation	0.001987
43	Rolling_7D_Groceries_bined_Rolling_7D_Grocerie...	0.000479
47	Rolling_7D_Clothing_bined_Rolling_7D_Clothing_...	0.000000
44	Rolling_7D_Groceries_bined_Rolling_7D_Grocerie...	0.000000
48	Rolling_7D_Clothing_bined_Rolling_7D_Clothing_40	0.000000
55	LoginCategory_High	0.000000
42	Rolling_7D_Groceries_bined_Rolling_7D_Grocerie...	0.000000
41	Rolling_7D_Books_bined_Rolling_7D_Books_80	0.000000
40	Rolling_7D_Books_bined_Rolling_7D_Books_220+	0.000000
50	DaysSinceLastLogin	0.000000
38	Rolling_7D_Electronics_bined_Rolling_7D_Electr...	0.000000
45	Rolling_7D_Groceries_bined_Rolling_7D_Grocerie...	0.000000
28	Rolling_7D_600+	0.000000
35	Rolling_7D_Furniture_bined_Rolling_7D_Furnitur...	0.000000
19	Complaint	0.000000

```
In [19]: correlation_matrix[abs(correlation_matrix["ChurnStatus"]) > 0.001].index
```

```
Out[19]: Index(['ServiceUsage_Website', 'Rolling_7D_Clothing', 'LoginCategory_Medium',
       'Zscore_MaritalStatusdiff', 'MaritalStatus_Married', 'Rolling_7D_Books',
       'LoginFrequency', 'Age_range_30-40', 'Age_range_20-30',
       'Rolling_7D_200-400',
       'Rolling_7D_Groceries_bined_Rolling_7D_Groceries_80',
       'Rolling_7D_Electronics_bined_Rolling_7D_Electronics_310',
       'Rolling_7D_Electronics_bined_Rolling_7D_Electronics_360',
       'LoginFrequency_Ratio', 'Age_range_50-60',
       'ServiceUsage_Online Banking', 'Rolling_7D_400-600', 'Unresolved',
       'MaritalStatus_Widowed', 'Age_range_40-50', 'Age_range_60+',
       'CustomerID', 'Zscore_Genderdiff', 'ChurnStatus'],
      dtype='object')
```

```
In [48]: from ctgan import CTGAN
import pandas as pd
from tqdm import tqdm
import time

os.environ["OMP_NUM_THREADS"] = "1"
os.environ["MKL_NUM_THREADS"] = "1"

data = enhanced_Interaction_Customer_Transaction_Login.copy()

data_to_augment = data[data['ChurnStatus'] == 1]
num_samples = len(data_to_augment) * 5

epochs = 500

ctgan = CTGAN(epochs=epochs, batch_size=500, verbose=False)

print("Training CTGAN...")
for epoch in tqdm(range(epochs), desc="CTGAN Training Progress"):

    ctgan.fit(data_to_augment)

print("CTGAN Training Complete!")

print("Generating synthetic data...")
new_data = ctgan.sample(num_samples)

for col in data_to_augment.columns:
    if col in new_data.columns:
        new_data[col] = new_data[col].astype(data_to_augment[col].dtype)

print("Synthetic Data Generated!")
```

```
augmented_data = pd.concat([data, new_data], ignore_index=True)
```

Training CTGAN...

```
CTGAN Training Progress: 100%|██████████| 50/50 [02:14<00:00,  2.69 s/it]
```

CTGAN Training Complete!

- ⌚ Generating synthetic data...
- ✓ Synthetic Data Generated!
- ⌚ Training XGBoost Model...
- ✓ XGBoost Training Complete!

📊 Classification Report:

	precision	recall	f1-score	support
0	0.76	0.92	0.83	159
1	0.94	0.81	0.87	245
accuracy			0.85	404
macro avg	0.85	0.86	0.85	404
weighted avg	0.87	0.85	0.85	404

```
In [70]: X = augmented_data.drop(columns=["ChurnStatus"])
y = augmented_data["ChurnStatus"]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

from xgboost import XGBClassifier
print("⌚ Training XGBoost Model...")
model_enhanced = XGBClassifier(n_estimators=30, max_depth=10, learning_rate=
model_enhanced.fit(X_train, y_train)
print("✓ XGBoost Training Complete!")
```

```
y_pred = model_enhanced.predict(X_test)
```

```
from sklearn.metrics import classification_report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

⌚ Training XGBoost Model...

✓ XGBoost Training Complete!

📊 Classification Report:

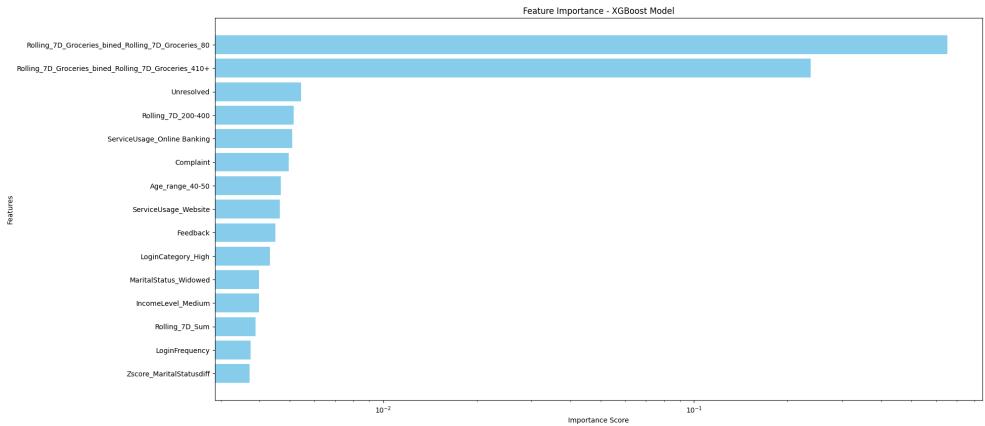
	precision	recall	f1-score	support
0	0.76	0.95	0.85	159
1	0.96	0.81	0.88	245
accuracy			0.86	404
macro avg	0.86	0.88	0.86	404
weighted avg	0.88	0.86	0.87	404

```
In [78]: feature_importances = model_enhanced.feature_importances_
feature_names = X_train.columns

importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(20, 10))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.xlabel("Importance Score")
plt.ylabel("Features")

plt.title("Feature Importance - XGBoost Model")
plt.xscale("log")
plt.gca().invert_yaxis()
plt.savefig("Plots/importance_rank.png", dpi=300, bbox_inches='tight')
plt.show()
```



```
In [76]: importance_df
```

Out[76]:

		Feature	Importance
46	Rolling_7D_Groceries_bined_Rolling_7D_Grocerie...	0.654101	
45	Rolling_7D_Groceries_bined_Rolling_7D_Grocerie...	0.237621	
23	Unresolved	0.005438	
26	Rolling_7D_200-400	0.005156	
52	ServiceUsage_Online Banking	0.005088	
19	Complaint	0.004958	
9	Age_range_40-50	0.004691	
53	ServiceUsage_Website	0.004653	
20	Feedback	0.004492	
55	LoginCategory_High	0.004325	
14	MaritalStatus_Widowed	0.003990	
16	IncomeLevel_Medium	0.003987	
24	Rolling_7D_Sum	0.003883	
49	LoginFrequency	0.003739	
4	Zscore_MaritalStatusdiff	0.003722	
1	Age	0.003688	
0	CustomerID	0.003681	
18	Markov_Churn_prob	0.003673	
54	LoginCategory_Medium	0.003600	
2	Zscore_Genderdiff	0.003585	
17	Duration_Interation	0.003534	
50	DaysSinceLastLogin	0.003356	
22	Resolved	0.003296	
5	Zscore_IncomeLeveldiff	0.003220	
13	MaritalStatus_Single	0.003203	
32	Rolling_7D_Furniture	0.002988	
21	Inquiry	0.002813	
33	Rolling_7D_Groceries	0.002531	
30	Rolling_7D_Clothing	0.002237	
31	Rolling_7D_Electronics	0.002114	
7	Age_range_20-30	0.001662	
3	Zscore_Age_rangediff	0.000976	
51	LoginFrequency_Ratio	0.000000	

	Feature	Importance
48	Rolling_7D_Clothing_bined_Rolling_7D_Clothing_40	0.000000
47	Rolling_7D_Clothing_bined_Rolling_7D_Clothing_...	0.000000
40	Rolling_7D_Books_bined_Rolling_7D_Books_220+	0.000000
44	Rolling_7D_Groceries_bined_Rolling_7D_Grocerie...	0.000000
43	Rolling_7D_Groceries_bined_Rolling_7D_Grocerie...	0.000000
42	Rolling_7D_Groceries_bined_Rolling_7D_Grocerie...	0.000000
41	Rolling_7D_Books_bined_Rolling_7D_Books_80	0.000000
28	Rolling_7D_600+	0.000000
39	Rolling_7D_Electronics_bined_Rolling_7D_Electr...	0.000000
38	Rolling_7D_Electronics_bined_Rolling_7D_Electr...	0.000000
37	Rolling_7D_Electronics_bined_Rolling_7D_Electr...	0.000000
36	Rolling_7D_Electronics_bined_Rolling_7D_Electr...	0.000000
35	Rolling_7D_Furniture_bined_Rolling_7D_Furnitur...	0.000000
34	Rolling_7D_Furniture_bined_Rolling_7D_Furnitur...	0.000000
29	Rolling_7D_Books	0.000000
27	Rolling_7D_400-600	0.000000
25	Rolling_7D_0-200	0.000000
15	IncomeLevel_Low	0.000000
12	MaritalStatus_Married	0.000000
11	Age_range_60+	0.000000
10	Age_range_50-60	0.000000
8	Age_range_30-40	0.000000
6	Gender_M	0.000000
56	LoginCategory_VeryHigh	0.000000

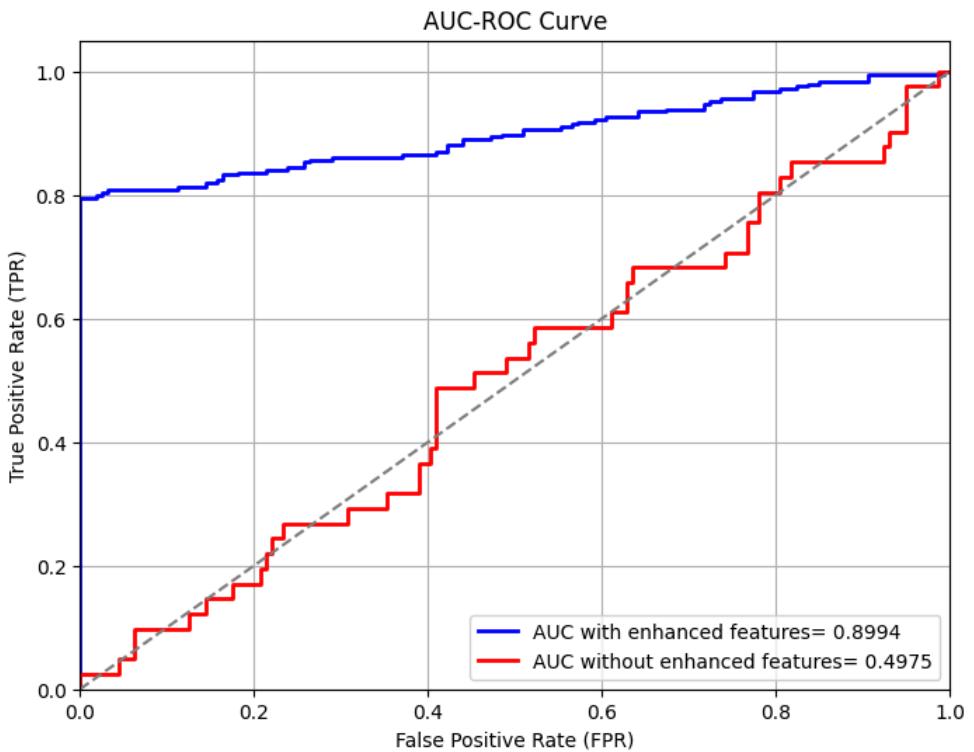
```
In [58]: import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

y_pred_proba = model_enhanced.predict_proba(X_test)[:, 1]
y_pred_proba_baseline = xgb_model_baseline.predict_proba(X_test_baseline)[:, 1]

fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
fpr_baseline, tpr_baseline, _1 = roc_curve(y_test_baseline, y_pred_proba_baseline)

roc_auc = auc(fpr, tpr)
roc_auc_baseline = auc(fpr_baseline, tpr_baseline)
```

```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color="blue", lw=2, label=f"AUC with enhanced features= {auc_enhanced:.4f}")
plt.plot(fpr_baseline, tpr_baseline, color="red", lw=2, label=f"AUC without enhanced features= {auc_no_enhanced:.4f}")
plt.plot([0, 1], [0, 1], color="grey", linestyle="--") # 参考线
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.title("AUC-ROC Curve")
plt.legend(loc="lower right")
plt.grid()
plt.savefig("Plots/AUCs.png", dpi=300, bbox_inches='tight')
plt.show()
```



5.2. Simulation Certificate

Thanks to Vesna Perisic for the patient guidance during the term. Through this coursework, I successfully applied my knowledge to complete the Lloyds Bank job simulation combined with the coursework at the same time, which I believe will be very beneficial for my future career!



Zhe GUAN

Data Science Job Simulation

Certificate of Completion

February 10th, 2025

Over the period of November 2024 to February 2025, Zhe GUAN has completed practical tasks in:

Data gathering and exploratory analysis
Building a machine learning model

A handwritten signature in black ink.

Tom Brunskill
CEO, Co-Founder of
Forage

Enrolment Verification Code 6kRLi9SmuTcwZKd9F | User Verification Code ubLX6a75Xx7YKf8Pv | Issued by Forage