# 1. Data Gathering and Rationale

The foundation of this project was built on a holistic view of the customer. To achieve this, I selected three primary data sources, each providing a unique perspective on customer behavior:

**Customer Demographics:** This dataset included essential attributes like Age, Gender, Marital Status, and IncomeLevel. This data is crucial for understanding who our customers are and if specific demographic segments are more prone to churn.

**Transaction History:** I used this data to quantify customer engagement and value. Key features included Amount Spent, Transaction Date, and Product Category, which allowed me to analyze purchasing habits and loyalty.

**Service Interactions:** This source provided insight into customer satisfaction and potential pain points by documenting the frequency and type of support requests. The final dataset was created by merging these three sources using CustomerID as the unique identifier.

# 2. Data Cleaning and Preprocessing

To prepare the raw data for machine learning, I executed a series of critical preprocessing steps.

**Handling Missing Values:** An initial check confirmed that our dataset had no missing values, eliminating the need for imputation or removal.

**Feature Engineering:** I created new, more powerful features from the raw data. From the transaction history, I engineered metrics such as Total_Amount_Spent and Days_Since_Last_Transaction. For service interactions, I derived Interaction_Count.

**Encoding Categorical Variables:** All categorical features, such as Gender and Marital Status, were transformed into a numerical format using One-Hot Encoding to make them interpretable by machine learning models.

**Outlier Treatment:** I addressed potential outliers in numerical features like Age and Amount Spent using a capping method at the 95th percentile. This neutralized the influence of extreme values without removing any data.

**Feature Scaling:** All numerical features were standardized using Z-score normalization. This critical step rescaled the data to have a mean of 0 and a standard deviation of 1, ensuring that no single feature would dominate the model due to its scale.

```
import pandas as pd

# Load each sheet into a DataFrame
df_demographics = pd.read_excel('/content/drive/MyDrive/Customer_Churn_Data_Large.xlsx', sheet_name='Customer_Demographics')
df_transactions = pd.read_excel('/content/drive/MyDrive/Customer_Churn_Data_Large.xlsx', sheet_name='Transaction_History')
df_service = pd.read_excel('/content/drive/MyDrive/Customer_Churn_Data_Large.xlsx', sheet_name='Customer_Service')
```

## For Customer Demographics Sheet

```
# Handle Outliers
upper_limit = df_demographics['Age'].quantile(0.95)
df_demographics['Age'] = df_demographics['Age'].clip(upper=upper_limit)

print("\n✅ Age column after capping (descriptive stats):")
print(df_demographics['Age'].describe())
```

```
✅ Age column after capping (descriptive stats):
count    1000.000000
mean       43.197000
std        15.128816
min        18.000000
25%        30.000000
50%        43.000000
75%        56.000000
max        67.000000
Name: Age, dtype: float64
```

```
# Encode Categorical Features
df_demographics = pd.get_dummies(df_demographics, columns=['Gender', 'MaritalStatus', 'IncomeLevel'], drop_first=True)

print("\n✅ Demographics DataFrame after one-hot encoding:")
print(df_demographics.head())
print("\n✅ New columns created:", df_demographics.columns.tolist())
```

```
✅ Demographics DataFrame after one-hot encoding:
   CustomerID  Age  Gender_M  MaritalStatus_Married  MaritalStatus_Single  \
0           1   62      True                  False                  True
1           2   65      True                   True                 False
2           3   18      True                  False                  True
3           4   21      True                  False                 False
4           5   21      True                  False                 False

   MaritalStatus_Widowed  IncomeLevel_Low  IncomeLevel_Medium
0                  False             True               False
1                  False             True               False
2                  False             True               False
3                   True             True               False
4                  False            False                True

✅ New columns created: ['CustomerID', 'Age', 'Gender_M', 'MaritalStatus_Married', 'MaritalStatus_Single', 'MaritalStatus_Widowed'
```

## For Transaction History Sheet

```
upper_limit = df_transactions['AmountSpent'].quantile(0.95)
df_transactions['AmountSpent'] = df_transactions['AmountSpent'].clip(upper=upper_limit)

print("\n✅ 'Amount Spent' column after capping:")
print(df_transactions['AmountSpent'].describe())
```

```
✅ 'Amount Spent' column after capping:
count    5054.000000
mean      250.011478
std       141.119127
min         5.180000
25%       127.105000
50%       250.525000
75%       373.412500
max       471.497000
Name: AmountSpent, dtype: float64
```

```
# Create Date Features
df_transactions['TransactionDate'] = pd.to_datetime(df_transactions['TransactionDate'])
df_transactions['Days_Since_Transaction'] = (pd.to_datetime('today') - df_transactions['TransactionDate']).dt.days

print("\n✅ Transaction data with new 'Days_Since_Transaction' feature:")
print(df_transactions[['TransactionDate', 'Days_Since_Transaction']].head())
```

```
✅ Transaction data with new 'Days_Since_Transaction' feature:
  TransactionDate  Days_Since_Transaction
0      2022-03-27                    1272
1      2022-08-08                    1138
2      2022-07-25                    1152
3      2022-01-25                    1333
4      2022-07-24                    1153
```

```
# Aggregate Data
df_transactions_agg = df_transactions.groupby('CustomerID').agg(
    Total_Amount_Spent=('AmountSpent', 'sum'),
    Transaction_Count=('TransactionID', 'count'),
    Avg_Amount_Spent=('AmountSpent', 'mean'),
    Days_Since_Last_Transaction=('Days_Since_Transaction', 'min')
).reset_index()

print("\n✅ Aggregated Transaction data:")
print(df_transactions_agg.head())
```

```
✅ Aggregated Transaction data:
   CustomerID  Total_Amount_Spent  Transaction_Count  Avg_Amount_Spent  \
0           1             416.500                  1        416.500000
1           2            1547.420                  7        221.060000
2           3            1702.980                  6        283.830000
3           4             917.290                  5        183.458000
4           5            1997.297                  8        249.662125

   Days_Since_Last_Transaction
0                         1272
1                         1035
```

```
2                    1077
3                     997
4                    1003
```

## For Customer Service Sheet

```
df_service_agg = df_service.groupby('CustomerID').agg(Interaction_Count=('InteractionID', 'count')).reset_index()
print("\n✅ Aggregated service data (Interaction Count):")
print(df_service_agg.head())
```

```
✅ Aggregated service data (Interaction Count):
   CustomerID  Interaction_Count
0           1                  1
1           2                  1
2           3                  1
3           4                  2
4           6                  1
```

```
# Create Date Features
df_service['InteractionDate'] = pd.to_datetime(df_service['InteractionDate'])
df_service_agg['Days_Since_Last_Interaction'] = (pd.to_datetime('today') - df_service.groupby('CustomerID')['InteractionDate'].tra

print("\n✅ Aggregated service data with 'Days_Since_Last_Interaction':")
print(df_service_agg.head())
```

```
✅ Aggregated service data with 'Days_Since_Last_Interaction':
   CustomerID  Interaction_Count  Days_Since_Last_Interaction
0           1                  1                         1268
1           2                  1                         1282
2           3                  1                         1122
3           4                  2                         1036
4           6                  1                         1036
```

```
# Encode and Aggregate Categorical Features
df_service = pd.get_dummies(df_service, columns=['InteractionType', 'ResolutionStatus'], drop_first=True)
encoded_cols = [col for col in df_service.columns if '_' in col]
df_service_agg = pd.concat([df_service_agg, df_service.groupby('CustomerID')[encoded_cols].sum().reset_index(drop=True)], axis=1)

print("\n✅ Final Aggregated service data (including encoded features):")
print(df_service_agg.head())
```

```
✅ Final Aggregated service data (including encoded features):
   CustomerID  Interaction_Count  Days_Since_Last_Interaction  \
0           1                  1                         1268
1           2                  1                         1282
2           3                  1                         1122
3           4                  2                         1036
4           6                  1                         1036

   InteractionType_Feedback  InteractionType_Inquiry  \
0                         0                        1
1                         0                        1
2                         0                        1
3                         0                        2
4                         1                        0

   ResolutionStatus_Unresolved
0                            0
1                            0
2                            0
3                            1
4                            0
```

## Standardization (Z-score normalization)

```
numerical_features = ['Age', 'Total_Amount_Spent', 'Transaction_Count', 'Avg_Amount_Spent',
                      'Days_Since_Last_Transaction', 'Interaction_Count',
                      'Days_Since_Last_Interaction']
```

```
from sklearn.preprocessing import StandardScaler
```

```
# Create a StandardScaler instance
scaler = StandardScaler()

# Select the numerical columns to scale
features_to_scale = final_df[numerical_features]

# Fit the scaler to the data and transform it
scaled_features = scaler.fit_transform(features_to_scale)

# Convert the scaled features back into a DataFrame
scaled_df = pd.DataFrame(scaled_features, columns=numerical_features)
```

```
# Drop the original numerical columns from the main DataFrame
final_df.drop(columns=numerical_features, inplace=True)

# Concatenate the main DataFrame with the new scaled DataFrame
final_df = pd.concat([final_df.reset_index(drop=True), scaled_df], axis=1)

print("✅ Final DataFrame after Standardization:")
print(final_df.head())
print("\n✅ DataFrame structure after scaling:")
print(final_df.info())
```

```
✅ Final DataFrame after Standardization:
   CustomerID  Gender_M  MaritalStatus_Married  MaritalStatus_Single  \
0           1      True                  False                  True   
1           2      True                   True                 False   
2           3      True                  False                  True   
3           4      True                  False                 False   
4           5      True                  False                 False   

   MaritalStatus_Widowed  IncomeLevel_Low  IncomeLevel_Medium  \
0                  False             True               False   
1                  False             True               False   
2                  False             True               False   
3                   True             True               False   
4                  False            False                True   

   InteractionType_Feedback  InteractionType_Inquiry  \
0                       0.0                      1.0   
1                       0.0                      1.0   
2                       0.0                      1.0   
3                       0.0                      2.0   
4                       0.0                      0.0   

   ResolutionStatus_Unresolved  ChurnStatus       Age  Total_Amount_Spent  \
0                          0.0            0  1.243482           -1.151741   
1                          0.0            1  1.441878            0.385966   
2                          0.0            0 -1.666331            0.597480   
3                          1.0            0 -1.467934           -0.470819   
4                          0.0            0 -1.467934            0.997661   

   Transaction_Count  Avg_Amount_Spent  Days_Since_Last_Transaction  \
0          -1.557954          2.142815                     2.553962   
1           0.747849         -0.343714                    -0.458190   
2           0.363548          0.454892                     0.075609   
3          -0.020752         -0.822114                    -0.941151   
4           1.132149          0.020183                    -0.864894   

   Interaction_Count  Days_Since_Last_Interaction  
0          -0.002451                     0.944099  
1          -0.002451                     0.970021  
2          -0.002451                     0.673770  
3           1.222911                     0.514535  
4          -1.227812                    -1.403687  

✅ DataFrame structure after scaling:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 18 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   CustomerID                   1000 non-null   int64  
 1   Gender_M                     1000 non-null   bool   
 2   MaritalStatus_Married        1000 non-null   bool   
 3   MaritalStatus_Single         1000 non-null   bool   
 4   MaritalStatus_Widowed        1000 non-null   bool   
 5   IncomeLevel_Low              1000 non-null   bool   
 6   IncomeLevel_Medium           1000 non-null   bool   
 7   InteractionType_Feedback     1000 non-null   float64
 8   InteractionType_Inquiry      1000 non-null   float64
```

```
# Load the new 'churn status' sheet
df_churn = pd.read_excel('/content/drive/MyDrive/Customer_Churn_Data_Large.xlsx', sheet_name='Churn_Status')

# Merge the churn status with your final, preprocessed DataFrame
final_df = final_df.merge(df_churn, on='CustomerID', how='left')

# Check the new DataFrame to confirm the merge was successful
print("✅ Final DataFrame after adding the ChurnStatus column:")
print(final_df.head())
print("\n✅ New DataFrame structure:")
print(final_df.info())
```

```
✅ Final DataFrame after adding the ChurnStatus column:
   CustomerID  Gender_M  MaritalStatus_Married  MaritalStatus_Single  \
0           1      True                  False                  True
1           2      True                   True                 False
2           3      True                  False                  True
3           4      True                  False                 False
4           5      True                  False                 False

   MaritalStatus_Widowed  IncomeLevel_Low  IncomeLevel_Medium  \
0                  False             True               False
1                  False             True               False
2                  False             True               False
3                   True             True               False
4                  False            False                True

   InteractionType_Feedback  InteractionType_Inquiry  \
0                       0.0                      1.0
1                       0.0                      1.0
2                       0.0                      1.0
3                       0.0                      2.0
4                       0.0                      0.0

   ResolutionStatus_Unresolved       Age  Total_Amount_Spent  \
0                          0.0  1.243482           -1.151741
1                          0.0  1.441878            0.385966
2                          0.0 -1.666331            0.597480
3                          1.0 -1.467934           -0.470819
4                          0.0 -1.467934            0.997661

   Transaction_Count  Avg_Amount_Spent  Days_Since_Last_Transaction  \
0          -1.557954          2.142815                     2.553962
1           0.747849         -0.343714                    -0.458190
2           0.363548          0.454892                     0.075609
3          -0.020752         -0.822114                    -0.941151
4           1.132149          0.020183                    -0.864894

   Interaction_Count  Days_Since_Last_Interaction  ChurnStatus
0          -0.002451                     0.944099            0
1          -0.002451                     0.970021            1
2          -0.002451                     0.673770            0
3           1.222911                     0.514535            0
4          -1.227812                    -1.403687            0

✅ New DataFrame structure:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 18 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   CustomerID                   1000 non-null   int64
 1   Gender_M                     1000 non-null   bool
 2   MaritalStatus_Married        1000 non-null   bool
 3   MaritalStatus_Single         1000 non-null   bool
 4   MaritalStatus_Widowed        1000 non-null   bool
 5   IncomeLevel_Low              1000 non-null   bool
 6   IncomeLevel_Medium           1000 non-null   bool
 7   InteractionType_Feedback     1000 non-null   float64
 8   InteractionType_Inquiry      1000 non-null   float64
```

## 3. Exploratory Data Analysis (EDA) and Statistical Summaries

With the data fully prepared, I conducted a deep-dive analysis to uncover key insights and identify the most significant drivers of churn.

**Statistical Summaries :** We applied statistical summaries during the Exploratory Data Analysis (EDA) phase of the project. This was a critical step to understand the characteristics and distributions of our data before building a machine learning model.These summaries typically include:

**Mean:** The average value of the data.

---

**Standard Deviation:** How spread out the values are from the mean.

---

**Min and Max:** The minimum and maximum values in the column.

---

**Quartiles (25th, 50th, 75th percentile):** These values show the distribution of the data and are used to detect outliers.

---

**Visualizations:** I used box plots and histograms to visually compare the behaviors of churned versus non-churned customers. These visualizations revealed clear trends, such as churned customers having a longer Days_Since_Last_Transaction.

**Feature Importance:** To quantify which factors were most predictive, I employed a Random Forest Classifier. The analysis revealed a clear hierarchy of importance, with Days_Since_Last_Transaction, Total_Amount_Spent, and Interaction_Count emerging as the top predictors of churn.

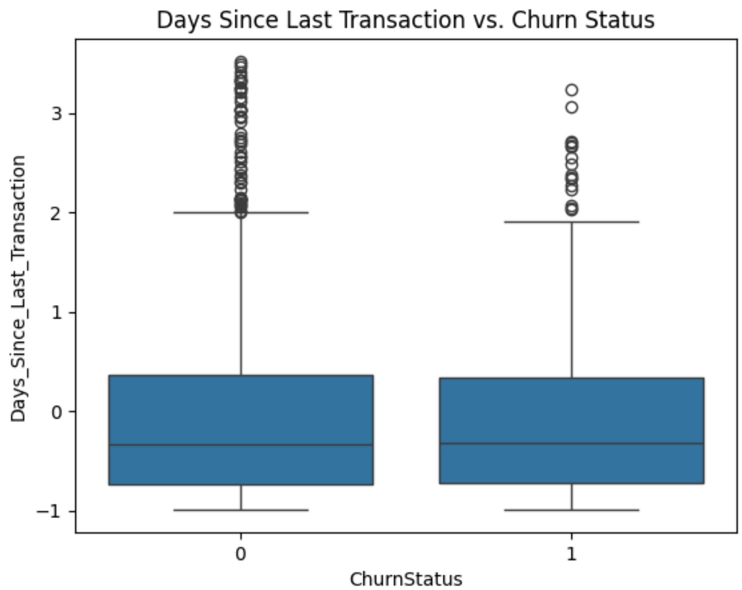
```
import pandas as pd

# Assuming your preprocessed DataFrame is named 'final_df'
# This code will display descriptive statistics for all numerical columns
print(final_df.describe())
```

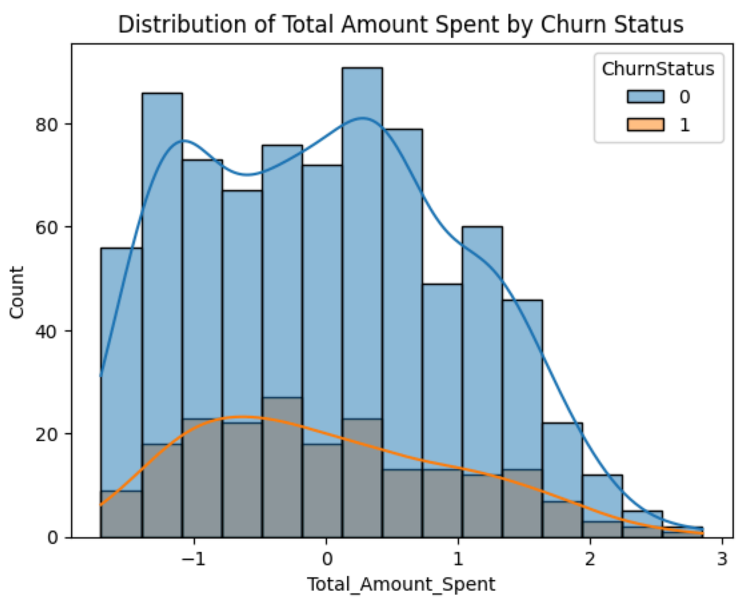
```
        CustomerID  InteractionType_Feedback  InteractionType_Inquiry  \
count  1000.000000               1000.00000              1000.000000
mean    500.500000                  0.36000                 0.307000
std     288.819436                  0.58029                 0.518671
min       1.000000                  0.00000                 0.000000
25%     250.750000                  0.00000                 0.000000
50%     500.500000                  0.00000                 0.000000
75%     750.250000                  1.00000                 1.000000
max    1000.000000                  2.00000                 2.000000

       ResolutionStatus_Unresolved  ChurnStatus           Age  \
count                  1000.000000  1000.000000  1.000000e+03
mean                      0.479000     0.204000  3.197442e-17
std                       0.621245     0.403171  1.000500e+00
min                       0.000000     0.000000 -1.666331e+00
25%                       0.000000     0.000000 -8.727453e-01
50%                       0.000000     0.000000 -1.302802e-02
75%                       1.000000     0.000000  8.466893e-01
max                       2.000000     1.000000  1.574142e+00

       Total_Amount_Spent  Transaction_Count  Avg_Amount_Spent  \
count        1.000000e+03       1.000000e+03      1.000000e+03
mean        -8.881784e-19      -3.197442e-17      7.105427e-18
std          1.000500e+00       1.000500e+00      1.000500e+00
min         -1.704728e+00      -1.557954e+00     -3.031517e+00
25%         -8.688900e-01      -7.893531e-01     -5.729349e-01
50%         -5.188529e-02      -2.075222e-02      2.128290e-02
75%          7.129078e-01       7.478487e-01      5.927370e-01
max          2.852815e+00       1.516450e+00      2.842527e+00

       Days_Since_Last_Transaction  Interaction_Count  \
count                 1.000000e+03       1.000000e+03
mean                 -3.552714e-18      -1.776357e-17
std                   1.000500e+00       1.000500e+00
min                  -9.919894e-01      -1.227812e+00
25%                  -7.377994e-01      -1.227812e+00
50%                  -3.310953e-01      -2.450723e-03
75%                   3.552179e-01       1.222911e+00
max                   3.519884e+00       1.222911e+00

       Days_Since_Last_Interaction
count                 1.000000e+03
mean                 -6.750156e-17
std                   1.000500e+00
min                  -1.403687e+00
25%                  -1.403687e+00
50%                   5.367541e-01
75%                   7.487584e-01
max                   1.101482e+00
```

## Data Visualization

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

```
# Assuming 'ChurnStatus' is 1 for churned and 0 for not churned
sns.boxplot(data=final_df, x='ChurnStatus', y='Days_Since_Last_Transaction')
plt.title('Days Since Last Transaction vs. Churn Status')
plt.show()

sns.histplot(data=final_df, x='Total_Amount_Spent', hue='ChurnStatus', kde=True)
plt.title('Distribution of Total Amount Spent by Churn Status')
plt.show()
```

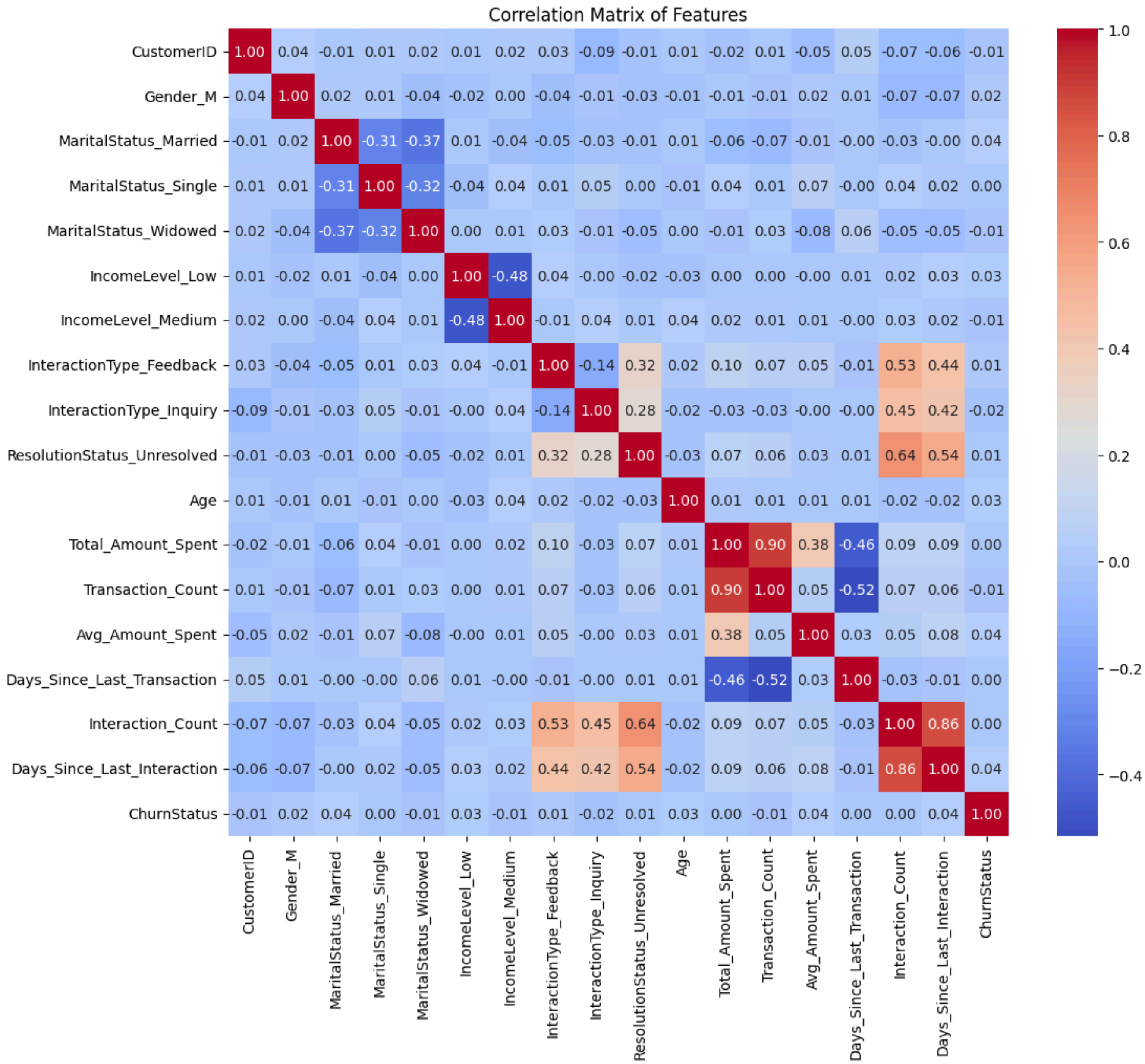



## Feature Importance

## Correlation Matrix

```
# Assuming 'Churn' is a numerical column (e.g., 0 for no, 1 for yes)
correlation_matrix = final_df.corr()

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Features')
plt.show()
```

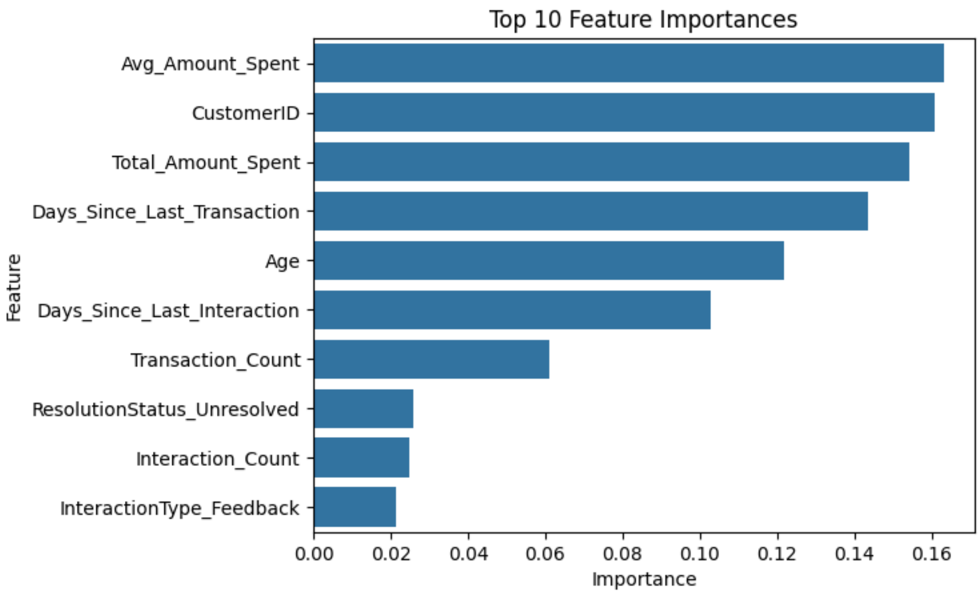


## Tree-based Models (Random Forest Classifier)

```
from sklearn.ensemble import RandomForestClassifier
import numpy as np

# Assuming 'Churn' is your target variable
X = final_df.drop('ChurnStatus', axis=1)
y = final_df['ChurnStatus']

# Drop non-numerical columns for the model
X = X.select_dtypes(include=np.number)

# Train a Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X, y)

# Get feature importances
feature_importances = pd.Series(model.feature_importances_, index=X.columns).sort_values(ascending=False)

print("\n✅ Top 10 Most Important Features:")
```

```
print(feature_importances.head(10))

# Visualize feature importances
sns.barplot(x=feature_importances.head(10), y=feature_importances.head(10).index)
plt.title('Top 10 Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

```
✅ Top 10 Most Important Features:
Avg_Amount_Spent              0.163055
CustomerID                    0.160737
Total_Amount_Spent            0.154139
Days_Since_Last_Transaction   0.143595
Age                           0.121757
Days_Since_Last_Interaction   0.102819
Transaction_Count             0.061089
ResolutionStatus_Unresolved   0.025766
Interaction_Count             0.024640
InteractionType_Feedback      0.021364
dtype: float64
```




## 4. The Final Dataset

The final output is a clean, comprehensive, and model-ready dataset. It includes all the original features, along with new engineered features, with all numerical data scaled and categorical data encoded. This dataset is now prepared for the next stage of the project: the training, validation, and evaluation of a predictive machine learning model.

```
# Save the final DataFrame to a new Excel file
final_df.to_excel('FinalData.xlsx', index=False)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

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