```python
import pandas as pd
import numpy as np

# Load
df = pd.read_csv("customer_level.csv")

# Strip whitespace from string columns
for col in df.select_dtypes(include="object").columns:
    df[col] = df[col].astype(str).str.strip().replace({"nan": np.nan})

# Normalize InteractionType (split multi-values)
df["InteractionType"] = df["InteractionType"].str.split(",")
df = df.explode("InteractionType")
df["InteractionType"] = df["InteractionType"].str.strip()
```

```python
df.columns = df.columns.str.strip()
df = df.drop([col for col in df.columns if "LoginBin_equal" in col], axis=1)
```

```python
for col in ["Gender","MaritalStatus","IncomeLevel",
            "ServiceUsage","InteractionType","ResolutionStatus","LoginBin_quartile"]:
    df[col] = df[col].astype(str).str.strip().str.lower()

# Replace string "nan" with actual NaN
df.replace("nan", np.nan, inplace=True)

# Check cleaned categories
for col in ["Gender","MaritalStatus","IncomeLevel","ServiceUsage"]:
    print(col, df[col].unique())
```

```
Gender ['m' 'f']
MaritalStatus ['single' 'married' 'widowed' 'divorced']
IncomeLevel ['low' 'high' 'medium']
ServiceUsage ['mobile app' 'online banking' 'website']
```

```python
df.head()
```

	C...		Marita...	Inc...		Am...	LoginFre...	HasInter...	ProductC...
0	1	m	single	low	62	416.5	34	1	Electronics
1	10	m	married	high	68	1397.36	203	0	Furniture
2	100	f	married	low	41	2217.12	360	1	Clothing
3	100	f	married	low	41	2217.12	360	1	Clothing
4	1000	m	widowed	low	34	1670.79	132	0	Furniture

Rows: 5 Expand

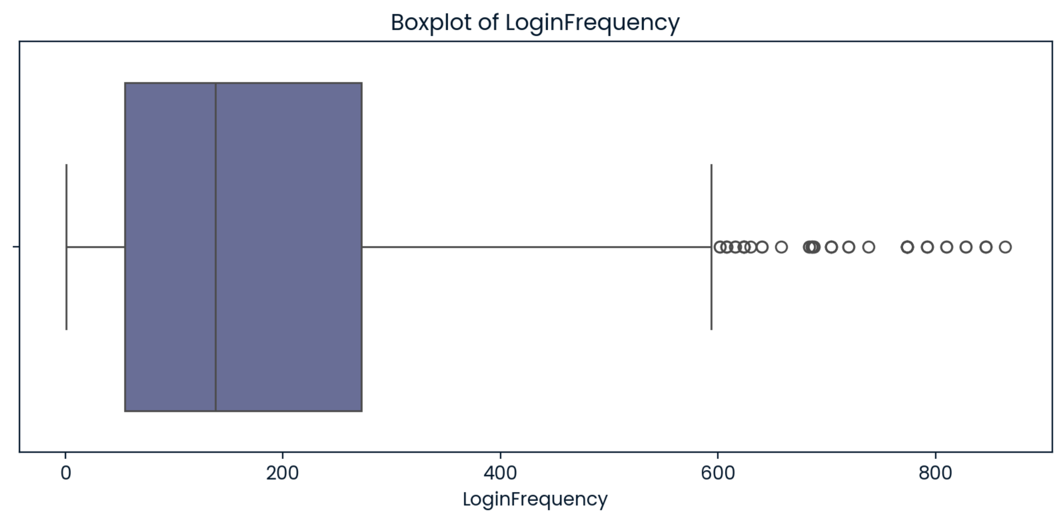
```python
# Display unique values for all categorical (object) columns in df
for col in df.select_dtypes(include="object").columns:
    print(f"{col}: {df[col].unique()}\n")
```

```
Gender: ['m' 'f']

MaritalStatus: ['single' 'married' 'widowed' 'divorced']

IncomeLevel: ['low' 'high' 'medium']

ProductCategory: ['Electronics' 'Furniture' 'Clothing' 'Groceries' 'Books']

ServiceUsage: ['mobile app' 'online banking' 'website']

InteractionType: ['inquiry' 'no interaction' 'complaint' 'feedback']

ResolutionStatus: ['resolved' 'no interaction' 'unresolved']

ChurnStatusLabel: ['Active' 'Churned']

LoginBin_quartile: ['q1 (lowest)' 'q3' 'q4 (highest)' 'q2']
```

```python
df.select_dtypes(exclude="object").columns
```

```
Index(['CustomerID', 'Age', 'AmountSpent', 'LoginFrequency', 'HasInteraction',
       'ChurnStatus'],
      dtype='object')
```

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

# Visualize LoginFrequency to spot outliers
plt.figure(figsize=(10, 4))
sns.boxplot(x=df["LoginFrequency"])
plt.title("Boxplot of LoginFrequency")
plt.show()

# Calculate outlier thresholds using IQR
Q1 = df["LoginFrequency"].quantile(0.25)
Q3 = df["LoginFrequency"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Find outliers
outliers = df[(df["LoginFrequency"] < lower_bound) | (df["LoginFrequency"] > upper_bound)]
```



```python
outliers = df[(df["LoginFrequency"] < lower_bound) | (df["LoginFrequency"] > upper_bound)]
outliers[["CustomerID", "LoginFrequency", "LoginBin_quartile"]].describe(include="all")
```

	Custom...	LoginFre...	LoginBin_qua...
count	48	48	48
unique			1
top			q4 (highest)
freq			48
mean	567.1458333333	712.8333333333	null
std	312.9555735981	81.0545740328	null
min	15	602	null
25%	399	628.5	null
50%	584	704	null
75%	887	774	null
max	986	864	null

Rows: 11 Expand

Count: 48 customers flagged as outliers LoginFrequency range:

- Min: 602
- Max: 864
- Mean: 713
- Median: 704 These are not data errors — they're real customers with extremely high login activity. They represent a distinct segment: highly engaged users who may be loyal or at risk if their behavior changes. Their presence distorts the scale of LoginFrequency, but they're valuable for churn modeling.

```python
print(df.columns.tolist())
```

```
['CustomerID', 'Gender', 'MaritalStatus', 'IncomeLevel', 'Age', 'AmountSpent', 'LoginFrequency', 'HasInteraction',
'ProductCategory', 'ServiceUsage', 'InteractionType', 'ResolutionStatus', 'ChurnStatus', 'ChurnStatusLabel',
'LoginBin_quartile']
```

```python
df["flag_login_outlier"] = (df["LoginFrequency"] > 600).astype(int)
```

```
Write Python code or tell our AI what to do
```

```python
print(df.columns.tolist())
```

```
['CustomerID', 'Gender', 'MaritalStatus', 'IncomeLevel', 'Age', 'AmountSpent', 'LoginFrequency', 'HasInteraction',
'ProductCategory', 'ServiceUsage', 'InteractionType', 'ResolutionStatus', 'ChurnStatus', 'ChurnStatusLabel',
'LoginBin_quartile', 'flag_login_outlier']
```

Predictive Modeling for Customer Churn

In this notebook, we will develop and implement a predictive model using Random Forest and other machine learning algorithms to predict customer churn. Our goal is to achieve an ROC-AUC score of at least 0.82. We will use the `customer_level.csv` dataset and perform exploratory data analysis (EDA) as needed.

```python
import pandas as pd

# Load the dataset
df = pd.read_csv('customer_level.csv')
df.head()
```

	C...		Marita...	Inc...		Am...	LoginFre...	HasInter...	ProductC...
0	1	M	Single	Low	62	416.5	34	1	Electronics
1	10	M	Married	High	68	1397.36	203	0	Furniture
2	100	F	Married	Low	41	2217.12	360	1	Clothing
3	100	F	Married	Low	41	2217.12	360	1	Clothing
4	1000	M	Widowed	Low	34	1670.79	132	0	Furniture

Rows: 5 Expand

Exploratory Data Analysis (EDA)

Let's explore the dataset to understand its structure, check for missing values, and get basic statistics.

```
# Check dataset info and missing values
df.info()
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1215 entries, 0 to 1214
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   CustomerID         1215 non-null   int64
 1   Gender             1215 non-null   object
 2   MaritalStatus      1215 non-null   object
 3   IncomeLevel        1215 non-null   object
 4   Age                1215 non-null   int64
 5   AmountSpent        1215 non-null   float64
 6   LoginFrequency     1215 non-null   int64
 7   HasInteraction     1215 non-null   int64
 8   ProductCategory    1215 non-null   object
 9   ServiceUsage       1215 non-null   object
 10  InteractionType    1215 non-null   object
 11  ResolutionStatus   1215 non-null   object
 12  ChurnStatus        1215 non-null   int64
 13  ChurnStatusLabel   1215 non-null   object
 14  LoginBin_equal     282 non-null    object
 15  LoginBin_quartile  1215 non-null   object
dtypes: float64(1), int64(5), object(10)
memory usage: 152.0+ KB
```

index	
CustomerID	0
Gender	0
MaritalStatus	0
IncomeLevel	0
Age	0
AmountSpent	0
LoginFrequency	0
HasInteraction	0
ProductCategory	0
ServiceUsage	0
InteractionType	0
ResolutionStatus	0
ChurnStatus	0
ChurnStatusLabel	0
LoginBin_equal	933
LoginBin_quartile	0

Rows: 16 Expand

```
# Display basic statistics
df.describe(include='all')
```

	Custom...		Marita...	Inc...	Age	AmountSp...	LoginFre...	HasInter...
count	1215	1215	1215	1215	1215	1215	1215	
unique		2	4	3				
top		F	Widowed	High				
freq		630	322	423				
mean	496.8576131687	null	null	null	43.4082304527	1877.384781893	189.5094650206	0.726
std	287.6945747523	null	null	null	15.2065817953	1346.8658953484	176.2837817209	0.445
min	1	null	null	null	18	13.86	1	
25%	247.5	null	null	null	30	865.485	54.5	
50%	492	null	null	null	44	1562.83	138	
75%	748.5	null	null	null	56	2571.24	272	
max	1000	null	null	null	69	6440.6	864	

Rows: 11 Expand

Data Preprocessing

We will preprocess the data by handling missing values, encoding categorical variables, and splitting the data into features and target.

```
# Example preprocessing (update as needed based on EDA results)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Assume 'Churn' is the target variable (update if different)
target_col = 'ChurnStatus'

# Encode categorical variables
df_encoded = df.copy()
for col in df_encoded.select_dtypes(include=['object', 'category']).columns:
    if col != target_col:
        df_encoded[col] = LabelEncoder().fit_transform(df_encoded[col].astype(str))

# Handle missing values (simple fill for demonstration)
df_encoded = df_encoded.fillna(df_encoded.median(numeric_only=True))

# Split features and target
X = df_encoded.drop([target_col, "ChurnStatusLabel"], axis=1)

y = df_encoded[target_col]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

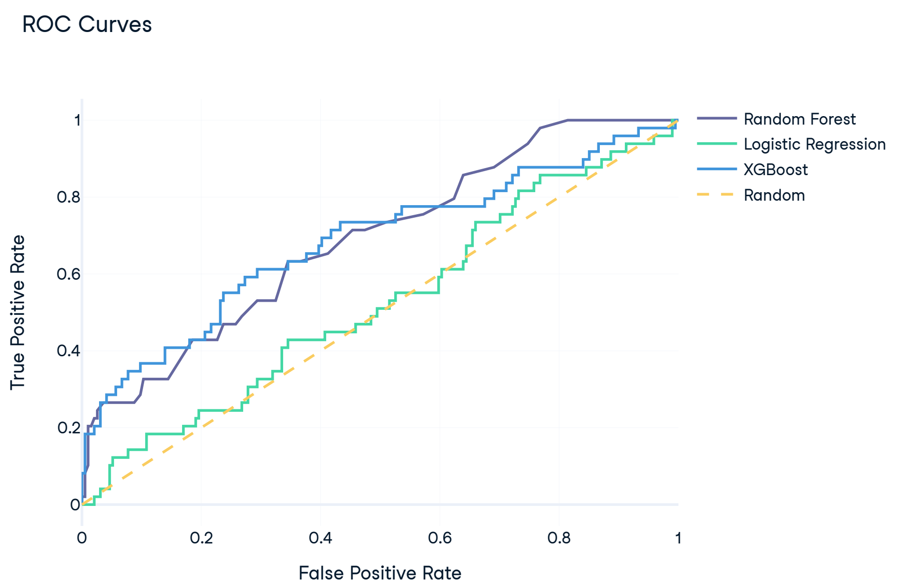
Model Training and Evaluation

We will train a Random Forest classifier and compare its performance with other algorithms such as Logistic Regression and XGBoost. The main evaluation metric will be ROC-AUC.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score, roc_curve

# Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
rf_probs = rf.predict_proba(X_test)[:,1]
rf_auc = roc_auc_score(y_test, rf_probs)

# Train Logistic Regression
lr = LogisticRegression(max_iter=1000, random_state=42)
lr.fit(X_train, y_train)
lr_probs = lr.predict_proba(X_test)[:,1]
lr_auc = roc_auc_score(y_test, lr_probs)

# Train XGBoost
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb.fit(X_train, y_train)
xgb_probs = xgb.predict_proba(X_test)[:,1]
xgb_auc = roc_auc_score(y_test, xgb_probs)

print(f"Random Forest ROC-AUC: {rf_auc:.3f}")
print(f"Logistic Regression ROC-AUC: {lr_auc:.3f}")
print(f"XGBoost ROC-AUC: {xgb_auc:.3f}")
```

```
Random Forest ROC-AUC: 0.690
Logistic Regression ROC-AUC: 0.524
XGBoost ROC-AUC: 0.684
```

```
import plotly.graph_objs as go

# Plot ROC curves
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_probs)
fpr_lr, tpr_lr, _ = roc_curve(y_test, lr_probs)
fpr_xgb, tpr_xgb, _ = roc_curve(y_test, xgb_probs)

fig = go.Figure()
fig.add_trace(go.Scatter(x=fpr_rf, y=tpr_rf, mode='lines', name='Random Forest'))
fig.add_trace(go.Scatter(x=fpr_lr, y=tpr_lr, mode='lines', name='Logistic Regression'))
fig.add_trace(go.Scatter(x=fpr_xgb, y=tpr_xgb, mode='lines', name='XGBoost'))
fig.add_trace(go.Scatter(x=[0,1], y=[0,1], mode='lines', name='Random', line=dict(dash='dash')))
fig.update_layout(title='ROC Curves', xaxis_title='False Positive Rate', yaxis_title='True Positive Rate', width=700,
height=500)
fig.show()
```



```python
print(df_encoded.corr()[target_col].sort_values(ascending=False))
```

```
ChurnStatus          1.000000
ChurnStatusLabel     1.000000
HasInteraction       0.026155
Age                  0.025463
IncomeLevel          0.019466
Gender               0.014467
ResolutionStatus     0.010732
ProductCategory      0.008132
MaritalStatus        0.001630
CustomerID          -0.011954
AmountSpent         -0.024454
InteractionType     -0.035769
LoginBin_equal      -0.065084
LoginBin_quartile   -0.065314
ServiceUsage        -0.068383
LoginFrequency      -0.072942
Name: ChurnStatus, dtype: float64
```