```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import xgboost as xgb


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

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

```
# prompt: import all the python libraries for doing churn analysis

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from google.colab import drive
drive.mount('/content/drive')
```

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

New section

```
# Importing the dataset from my computer hard drive
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Bank Customer Churn Prediction.csv to Bank Customer Churn Prediction.csv
```

Data Cleaning and Transformation

```
# prompt: Handle Missing Values in the uploaded dataset (Bank Customer Churn Prediction.csv)

import io
df = pd.read_csv(io.BytesIO(uploaded['Bank Customer Churn Prediction.csv']))


# Check for missing values
print(df.isnull().sum())

# Handle missing values (e.g., by replacing with mean, median, or mode)
# For numerical features, replace missing values with the mean
for column in df.columns:
  if pd.api.types.is_numeric_dtype(df[column]):
    df[column].fillna(df[column].mean(), inplace=True)

# For categorical features, replace missing values with the mode
for column in df.columns:
  if not pd.api.types.is_numeric_dtype(df[column]):
    df[column].fillna(df[column].mode()[0], inplace=True)


# Verify if missing values are handled
print(df.isnull().sum())
```

```
customer_id         0
credit_score        0
country             0
gender              0
age                 0
tenure              0
```

```
balance            0
products_number    0
credit_card        0
active_member      0
estimated_salary   0
churn              0
dtype: int64
customer_id        0
credit_score       0
country            0
gender             0
age                0
tenure             0
balance            0
products_number    0
credit_card        0
active_member      0
estimated_salary   0
churn              0
dtype: int64
<ipython-input-9-d9f90218f292>:14: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df[column].fillna(df[column].mean(), inplace=True)
<ipython-input-9-d9f90218f292>:19: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained ass
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col


  df[column].fillna(df[column].mode()[0], inplace=True)
```

```
# prompt: call the dataset and show all columns

# Display all columns of the dataset
df
```

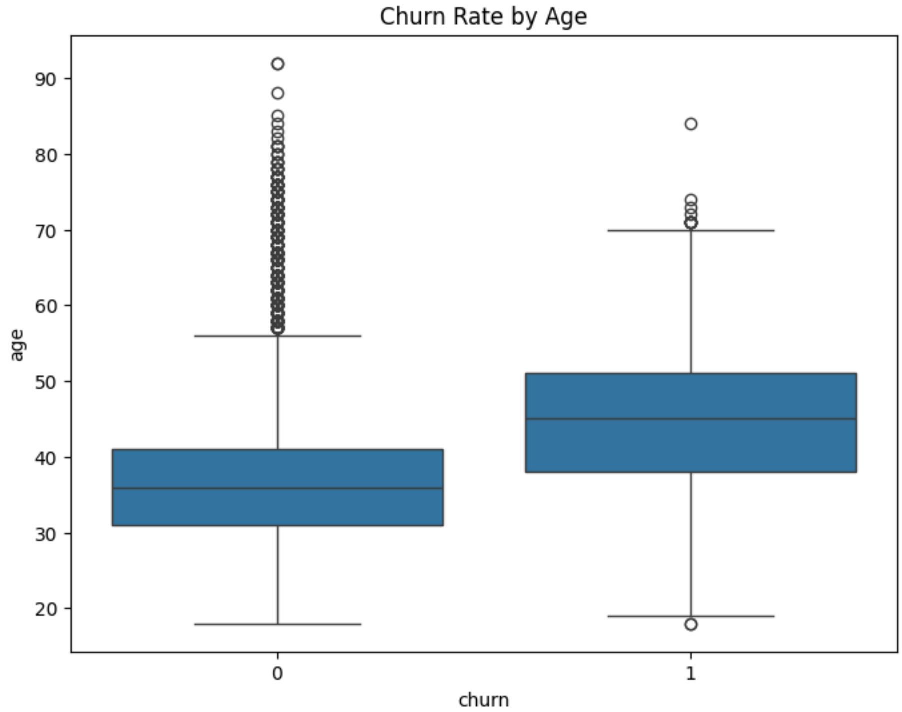
	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_sala
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084
...	...	...	...	...	...	...	...	...	...	...	
9995	15606229	771	France	Male	39	5	0.00	2	1	0	96270
9996	15569892	516	France	Male	35	10	57369.61	1	1	1	101699
9997	15584532	709	France	Female	36	7	0.00	1	0	1	42085
9998	15682355	772	Germany	Male	42	3	75075.31	2	1	0	92888
9999	15628319	792	France	Female	28	4	130142.79	1	1	0	38190

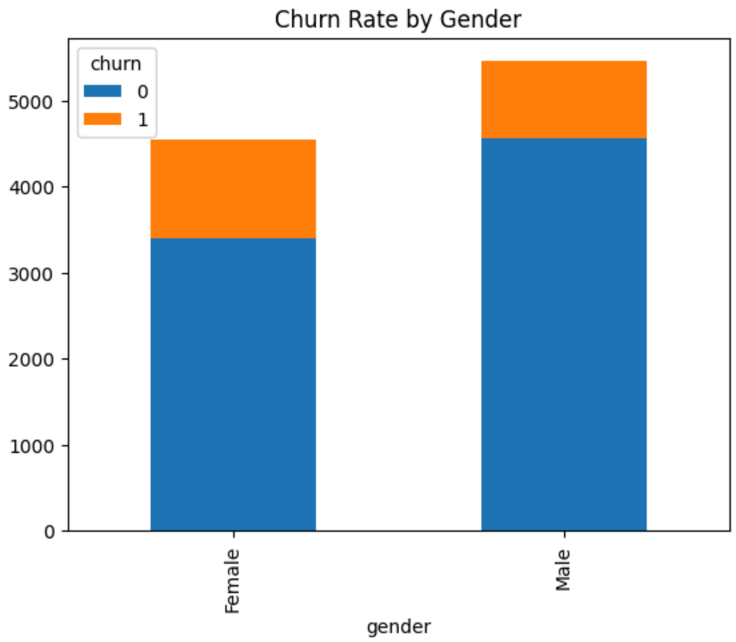
10000 rows × 12 columns

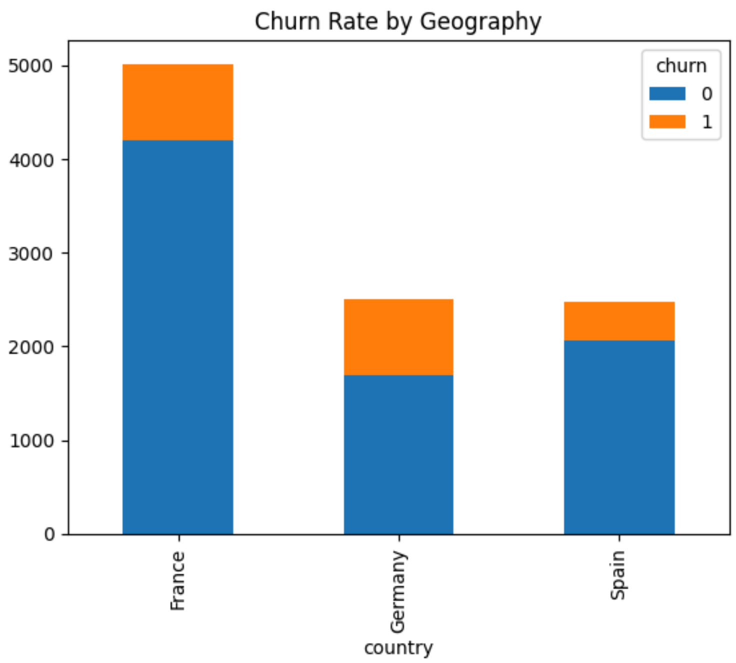
Exploratory Data Analysis (EDA)

```
# prompt: Exploratory Data Analysis (A. Customer Demographics
# Age: Analyze if younger or older customers are more likely to churn.
# Gender: Check for any gender-based trends in churn rates.
# Geography: Compare churn rates across different regions to see if any particular region has higher churn.
# B. Account Tenure and Balance
# Tenure: Plot churn rates by the duration of time customers have been with the bank. Customers with very short or very long tenures migh
# Balance: Investigate if low or zero account balances correlate with higher churn rates.
# C. Financial Behavior
# Credit Score: Analyze if there is a relationship between credit score ranges and churn. Low credit scores might indicate financial stre
# Number of Products: Banks often find that customers with multiple products (e.g., loan, savings, credit card) are less likely to churn.
# Estimated Salary: Observe if income level has any relationship with churn behavior.)use df column names as they are

import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'churn' is the correct column name for customer churn
# Replace 'churn' with the actual column name if it's different
churn_column = 'churn'
```

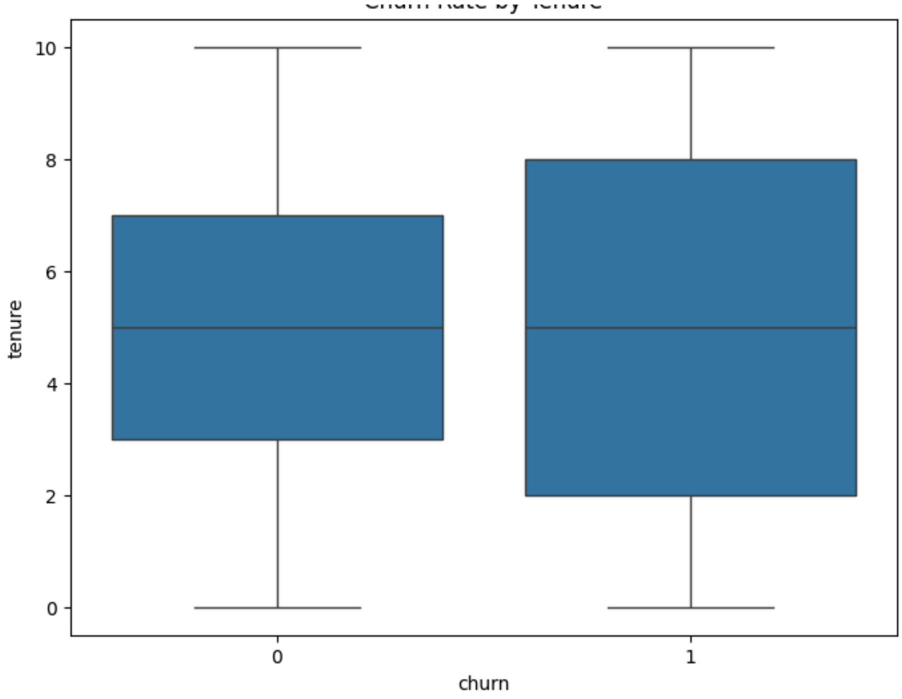
```
# If you don't know the exact column name for customer churn:
# 1. Print the column names of your DataFrame: print(df.columns)
# 2. Identify the column representing churn (e.g., 'Exited', 'Churned')
# 3. Update the `churn_column` variable accordingly


# A. Customer Demographics

# Age
plt.figure(figsize=(8, 6))
sns.boxplot(x=churn_column, y='age', data=df)
plt.title('Churn Rate by Age')
plt.show()

# Gender
gender_churn = df.groupby(['gender', churn_column])[churn_column].count().unstack()
gender_churn.plot(kind='bar', stacked=True)
plt.title('Churn Rate by Gender')
plt.show()

# Geography
geography_churn = df.groupby(['country', churn_column])[churn_column].count().unstack()
geography_churn.plot(kind='bar', stacked=True)
plt.title('Churn Rate by Geography')
plt.show()


# B. Account Tenure and Balance

# Tenure
plt.figure(figsize=(8, 6))
sns.boxplot(x=churn_column, y='tenure', data=df)
plt.title('Churn Rate by Tenure')
plt.show()

# Balance
plt.figure(figsize=(8, 6))
sns.boxplot(x=churn_column, y='balance', data=df)
plt.title('Churn Rate by Balance')
plt.show()


# C. Financial Behavior

# Credit Score
plt.figure(figsize=(8, 6))
sns.boxplot(x=churn_column, y='credit_score', data=df)
plt.title('Churn Rate by Credit Score')
plt.show()

# Number of Products
product_churn = df.groupby(['products_number', churn_column])[churn_column].count().unstack()
product_churn.plot(kind='bar', stacked=True)
plt.title('Churn Rate by Number of Products')
plt.show()

# Estimated Salary
plt.figure(figsize=(8, 6))
sns.boxplot(x=churn_column, y='estimated_salary', data=df)
plt.title('Churn Rate by Estimated Salary')
plt.show()
```

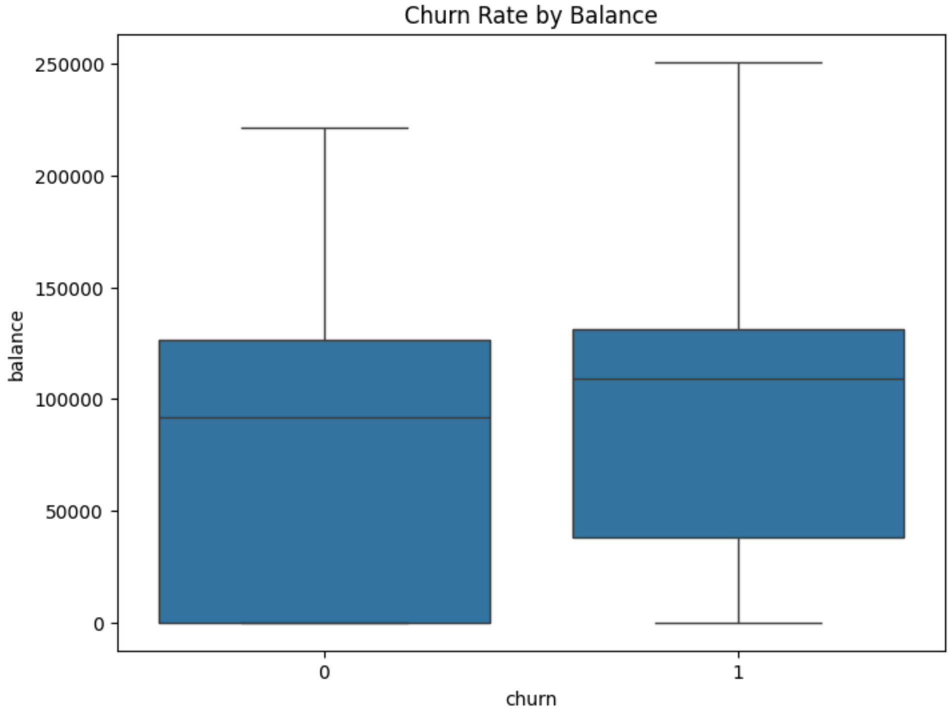


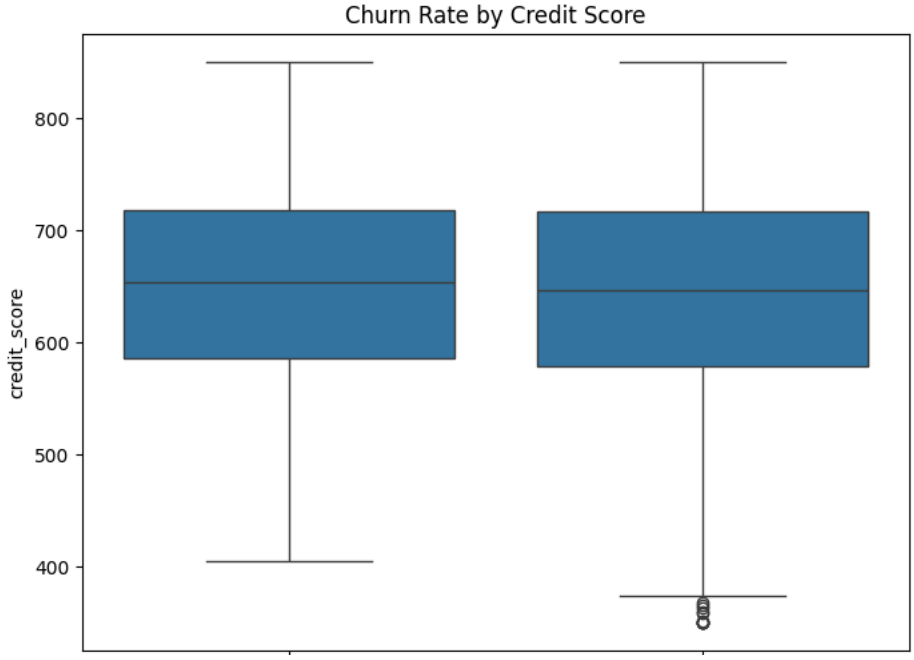




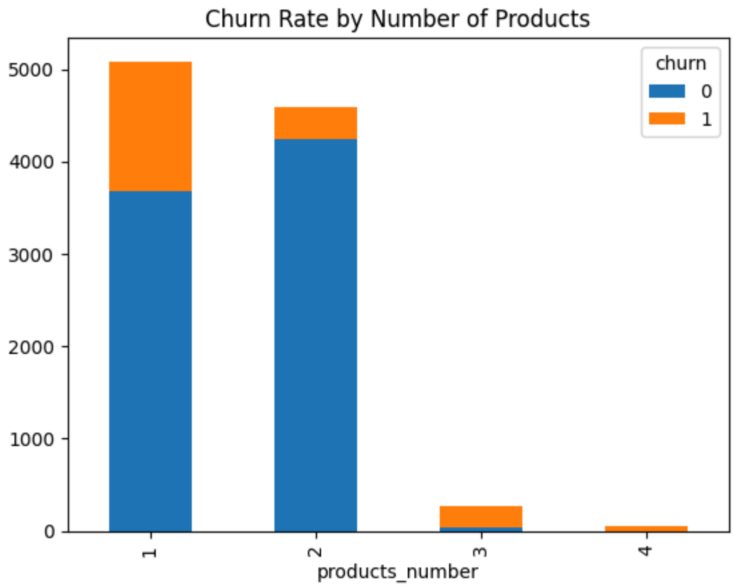
Churn Rate by Tenure

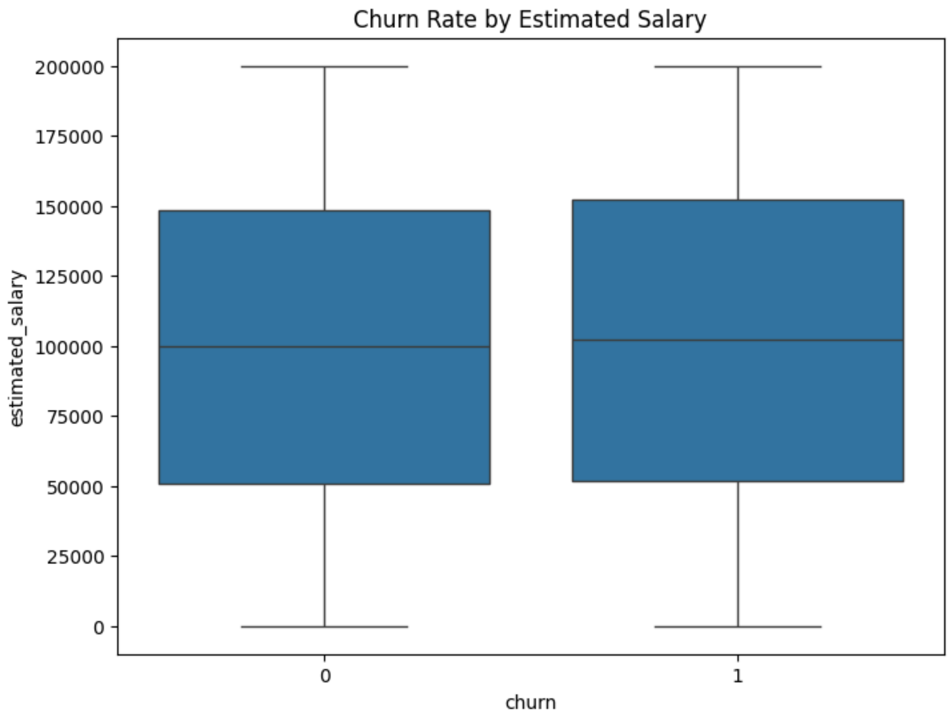

10
8
6
4
2
0
tenure
0
1
churn


Churn Rate by Balance
250000
200000
150000
100000
50000
0
balance
0
1
churn


Churn Rate by Credit Score
800
700
600
500
400
credit_score


0
1
churn


Churn Rate by Number of Products
churn
0
1
5000
4000
3000
2000
1000
0
1
2
3
4
products_number


Churn Rate by Estimated Salary
200000
175000
150000
125000
100000
75000
50000
25000
0
estimated_salary
0
1
churn

Feature Engineering

```
# prompt: Feature Engineering
# Feature engineering helps you derive new insights:
# Interaction Terms: Create interaction features such as "Credit Score * Balance" or "Tenure * Number of Products" to reveal compounded
# Risk Score: Consider developing a custom "risk score" by weighing factors like low balance, low tenure, and single-product holding.

# Feature Engineering

# 1. Interaction Terms
df['Tenure_Balance'] = df['tenure'] * df['balance']
df['CreditScore_Balance'] = df['credit_score'] * df['balance']
df['Age_EstimatedSalary'] = df['age'] * df['estimated_salary']

# 2. Risk Score (Example)
df['RiskScore'] = (
    (df['balance'] < 5000) * 1 +
    (df['tenure'] < 2) * 1 +
    (df['products_number'] == 1) * 1
)


# You can further engineer features based on domain knowledge and insights from EDA.
# For example, creating features based on customer segments identified during EDA.

# Display the updated dataframe with new features
df
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_sal
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084
...	...	...	...	...	...	...	...	...	...	...	
9995	15606229	771	France	Male	39	5	0.00	2	1	0	96270
9996	15569892	516	France	Male	35	10	57369.61	1	1	1	101699
9997	15584532	709	France	Female	36	7	0.00	1	0	1	42085
9998	15682355	772	Germany	Male	42	3	75075.31	2	1	0	92888
9999	15628319	792	France	Female	28	4	130142.79	1	1	0	38190

10000 rows × 17 columns

Model Building

```
# prompt: Model Building and Evaluation
# For customer churn prediction, consider these modeling steps:
# A. Split the Data
# Divide the dataset into training and testing sets (e.g., 80/20 split) to evaluate model performance on unseen data.
# B. Choose Models
# Start with basic models such as Logistic Regression and Decision Trees, then consider advanced ones like Random Forests or XGBoost.
# Logistic Regression can help identify key drivers of churn due to its interpretability, while ensemble methods like Random Forests are
# C. Evaluate the Model
# Use metrics such as accuracy, precision, recall, and F1-score, but focus on ROC-AUC as it evaluates the model’s ability to distinguish
# Confusion Matrix: Helps to visualize false positives (predicting a customer will stay, but they churn) and false negatives (predicting

# Define features (X) and target variable (y)
X = df.drop(churn_column, axis=1)
y = df[churn_column]

# Convert categorical features to numerical using one-hot encoding
X = pd.get_dummies(X, drop_first=True)

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

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Handle class imbalance using SMOTE (if needed)
# smote = SMOTE(random_state=42)
# X_train, y_train = smote.fit_resample(X_train, y_train)

# Choose models
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "XGBoost": xgb.XGBClassifier()
}

# Train and evaluate each model
results = {}
for model_name, model in models.items():
  model.fit(X_train, y_train)
  y_pred = model.predict(X_test)
  y_prob = model.predict_proba(X_test)[:, 1]

  accuracy = accuracy_score(y_test, y_pred)
  precision = precision_score(y_test, y_pred)
  recall = recall_score(y_test, y_pred)
  f1 = f1_score(y_test, y_pred)
  roc_auc = roc_auc_score(y_test, y_prob)
  confusion = confusion_matrix(y_test, y_pred)

  results[model_name] = {
      "accuracy": accuracy,
      "precision": precision,
      "recall": recall,
      "f1": f1,
      "roc_auc": roc_auc,
      "confusion_matrix": confusion
  }

# Print evaluation results for each model
for model_name, metrics in results.items():
  print(f"Model: {model_name}")
  print(f"Accuracy: {metrics['accuracy']:.4f}")
  print(f"Precision: {metrics['precision']:.4f}")
  print(f"Recall: {metrics['recall']:.4f}")
  print(f"F1-score: {metrics['f1']:.4f}")
  print(f"ROC-AUC: {metrics['roc_auc']:.4f}")
  print(f"Confusion Matrix:\n{metrics['confusion_matrix']}")
  print("-" * 30)

# You can choose the best performing model based on these metrics
# and further tune it for optimal performance.
```

```
Model: Logistic Regression
Accuracy: 0.8275
Precision: 0.6364
Recall: 0.2850
F1-score: 0.3937
ROC-AUC: 0.7966
Confusion Matrix:
[[1543   64]
 [ 281  112]]
------------------------------
Model: Decision Tree
Accuracy: 0.7815
Precision: 0.4511
Recall: 0.5165
F1-score: 0.4816
ROC-AUC: 0.6814
Confusion Matrix:
[[1360  247]
 [ 190  203]]
------------------------------
Model: Random Forest
Accuracy: 0.8675
Precision: 0.7623
Recall: 0.4733
F1-score: 0.5840
ROC-AUC: 0.8590
Confusion Matrix:
[[1549   58]
 [ 207  186]]
------------------------------
Model: XGBoost
Accuracy: 0.8610
Precision: 0.6990
Recall: 0.5140
F1-score: 0.5924
ROC-AUC: 0.8567
```