STEP :1

1A : SETUP

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

THIS STEP IMPORTS ALL THE REQUIRED PYTHON LIBRARIES THAT WILL BE USED THROUGHOUT THE PROJECT.

- PANDAS IS USED TO LOAD, READ, AND MANIPULATE THE DATASET IN TABULAR FORM.
- NUMPY IS USED FOR BASIC NUMERICAL OPERATIONS AND ARRAY HANDLING.
- MATPLOTLIB IS USED TO CREATE BASIC VISUALIZATIONS SUCH AS HISTOGRAMS AND BOXPLOTS.
- SEABORN IS USED TO CREATE ADVANCED AND MORE VISUALLY APPEALING STATISTICAL GRAPHS.

1B : LOAD THE CSV FILE

```
df=pd.read_csv('/content/churn (1).csv')
print("DATASET SHAPE :",df.shape)
```

```
DATASET SHAPE : (10000, 14)
```

IN THIS STEP, THE BANK CUSTOMER DATASET IS LOADED INTO THE NOTEBOOK.

- pd.read_csv() IS USED TO READ THE CSV FILE AND STORE IT IN A DATAFRAME.
- df.shape IS USED TO CHECK THE SIZE OF THE DATASET.
- THE OUTPUT SHOWS THAT THE DATASET CONTAINS 10,000 ROWS AND 14 COLUMNS

1C : PEEK AT TOP ROWS AND COLUMN NAMES AND INFO.

```
display(df.head(6))
print("\ncolumns :\n",list(df.columns))
print("-------------------------------------------")
df.info()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMemb
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	
5	6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	

```
columns :
 ['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasC
-----------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

- df.head() DISPLAYS THE FIRST FEW ROWS OF THE DATASET TO UNDERSTAND HOW THE DATA LOOKS.
- df.info() PROVIDES INFORMATION ABOUT DATA TYPES, NON-NULL COUNTS, AND MEMORY USAGE

FROM THE OUTPUT, WE OBSERVE:

- THE DATASET CONTAINS BOTH NUMERICAL AND CATEGORICAL COLUMNS.
- COLUMNS SUCH AS RowNumber, CustomerId, AND Surname ACT AS IDENTIFIERS AND DO NOT ADD VALUE TO ANALYSIS OR PREDICTION.
- IMPORTANT FEATURES INCLUDE CreditScore, Age, Tenure, Balance, NumOfProducts, IsActiveMember, AND Exited.
- ALL COLUMNS HAVE COMPLETE NON-NULL VALUES, WHICH INDICATES THAT THERE ARE NO MISSING VALUES AT THIS STAGE.

1D : MISSING VALUES AND DUPLICATES

```
MissValues=df.isnull().sum().sort_values(ascending=False)
print("MISSING VALUES PER COLUMN : " , MissValues[MissValues>0] if MissValues.sum()>0 else "NO MISSING VALUES FOUND")

dups=df.duplicated().sum()
print("\n NUMBER OF DUPLICATES ROWS : ",dups)
```

```
MISSING VALUES PER COLUMN :  NO MISSING VALUES FOUND

 NUMBER OF DUPLICATES ROWS :  0
```

IN THIS STEP, WE CHECK WHETHER THE DATASET CONTAINS ANY MISSING VALUES OR DUPLICATE ROWS.

- THE RESULTS SHOW THAT THERE ARE NO MISSING VALUES IN ANY COLUMN.
- THE RESULTS ALSO SHOW THAT THERE ARE NO DUPLICATE RECORDS IN THE DATASE

1E : BASIC SUMMARY

```
display(df.describe(include=[np.number]).T)
```

	count	mean	std	min	25%	50%	75%	max
RowNumber	10000.0	5.000500e+03	2886.895680	1.00	2500.75	5.000500e+03	7.500250e+03	10000.00
CustomerId	10000.0	1.569094e+07	71936.186123	15565701.00	15628528.25	1.569074e+07	1.575323e+07	15815690.00
CreditScore	10000.0	6.505288e+02	96.653299	350.00	584.00	6.520000e+02	7.180000e+02	850.00
Age	10000.0	3.892180e+01	10.487806	18.00	32.00	3.700000e+01	4.400000e+01	92.00
Tenure	10000.0	5.012800e+00	2.892174	0.00	3.00	5.000000e+00	7.000000e+00	10.00
Balance	10000.0	7.648589e+04	62397.405202	0.00	0.00	9.719854e+04	1.276442e+05	250898.09
NumOfProducts	10000.0	1.530200e+00	0.581654	1.00	1.00	1.000000e+00	2.000000e+00	4.00
HasCrCard	10000.0	7.055000e-01	0.455840	0.00	0.00	1.000000e+00	1.000000e+00	1.00
IsActiveMember	10000.0	5.151000e-01	0.499797	0.00	0.00	1.000000e+00	1.000000e+00	1.00
EstimatedSalary	10000.0	1.000902e+05	57510.492818	11.58	51002.11	1.001939e+05	1.493882e+05	199992.48
Exited	10000.0	2.037000e-01	0.402769	0.00	0.00	0.000000e+00	0.000000e+00	1.00

IN THIS STEP, WE GENERATED SUMMARY STATISTICS FOR ALL NUMERICAL COLUMNS IN THE DATASET.

- The dataset contains 10,000 customer records.
- Customer age ranges from 18 to 92 years, with an average age of about 39 years.
- The credit score ranges from 350 to 850, with an average of around 650. Customer tenure ranges from 0 to 10 years, and the average tenure is about 5 years.
- The account balance ranges from 0 to 260,898. Many customers have zero balance, and the average balance is around 76,485.
- The number of products owned by customers ranges from 1 to 4, with most customers having 1 or 2 products.
- The estimated salary ranges from 11.58 to 199,992, with an average salary of about 100,090.
- Overall, 20.37% of customers have exited the bank, indicating a noticeable churn rate.

1F : TARGET(EXITED) QUICK CHECK

```
# Cell 7 — target distribution
if 'Exited' in df.columns:
    counts = df['Exited'].value_counts()
    pct = df['Exited'].value_counts(normalize=True).mul(100).round(2)
    print("EXITED COUNTS:\n", counts.to_string())
    print("\nEXITED PERCENT:\n", pct.to_string())
else:
    print("NO 'EXITED' COLUMN FOUND .PLEASE TELL ME THE TARGET COLUMN NAME. ")
```

```
EXITED COUNTS:
 Exited
0    7963
1    2037

EXITED PERCENT:
 Exited
0    79.63
1    20.37
```

In this step, we analyzed the target variable Exited, which indicates whether a customer stayed or left the bank.

- Exited = 0 means the customer stayed with the bank
- Exited = 1 means the customer left the bank

From the results:

- 7,963 customers (79.63%) stayed with the bank
- 2,037 customers (20.37%) exited the bank

This shows that around one-fifth of the customers have left, confirming the presence of customer churn

STEP 2 : DATA CLEANING

2 A: REMOVE IDENTIFIER COLUMNS + CHECK DUPLICATES

```
columns_to_drop= ['RowNumber','CustomerId','Surname']
df_cleaning = df.drop(columns=columns_to_drop).copy()
print("DROPPED COLUMNS:", columns_to_drop)
print("NEW SHAPE:", df_cleaning.shape)
print("DUPLICATE ROWS:", df_cleaning.duplicated().sum())
display(df_cleaning.head(5))
```

```
DROPPED COLUMNS: ['RowNumber', 'CustomerId', 'Surname']
NEW SHAPE: (10000, 11)
DUPLICATE ROWS: 0
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

In this step, we removed columns such as RowNumber, CustomerId, and Surname.

These columns are only used for identification and do not help in analysis or prediction, so removing them improves model quality. After removing these columns:

- The dataset now contains 11 useful columns
- The total number of rows remains 10,000
- No duplicate records were found

2B : DATATYPES AND UNIQUE VALUES

```
print("DATATYPES:\n")
print("-------------------------")
print(df_cleaning.dtypes)
print("\nUNIQUE VALUES (SAMPLE):")
print("-------------------------")
for c in ['Geography','Gender','HasCrCard','IsActiveMember','NumOfProducts']:
    if c in df_cleaning.columns:
        print(f"\n{c} ->", df_cleaning[c].unique()[:10])
```

```
DATATYPES:

-------------------------
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object

UNIQUE VALUES (SAMPLE):
-------------------------

Geography -> ['France' 'Spain' 'Germany']

Gender -> ['Female' 'Male']

HasCrCard -> [1 0]
```

```
IsActiveMember -> [1 0]

NumOfProducts -> [1 3 2 4]
```

In this step, we checked the data type of each column and looked at the unique values in important categorical columns.

From the output:

- Numerical columns like CreditScore, Age, Tenure, Balance, EstimatedSalary are stored as numbers.
- Categorical columns like Geography and Gender are stored as text.
- Binary columns such as HasCrCard and IsActiveMember contain values 0 and 1.
- The Number of Products ranges from 1 to 4.
- Geography includes customers from France, Spain, and Germany.
- Gender includes Male and Female customers.

2C : ENCODE CATEGORICALS

```
df_cleaning['Gender'] = df_cleaning['Gender'].map({'Male':1, 'Female':0})

# ONE-HOT ENCODE GEOGRAPHY (drop_first to avoid collinearity)
df_cleaning = pd.get_dummies(df_cleaning, columns=['Geography'], drop_first=True)

print("AFTER ENCODING SHAPE:", df_cleaning.shape)
display(df_cleaning.head(4))
```

```
AFTER ENCODING SHAPE: (10000, 12)
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Geography_Germa
0	619	0	42	2	0.00	1	1	1	101348.88	1	Fa
1	608	0	41	1	83807.86	1	0	1	112542.58	0	Fa
2	502	0	42	8	159660.80	3	1	0	113931.57	1	Fa
3	699	0	39	1	0.00	2	0	0	93826.63	0	Fa

In this step, we converted categorical columns into numerical form so that machine learning models can understand them.

Encoding methods used:

Gender was encoded using Binary Encoding:

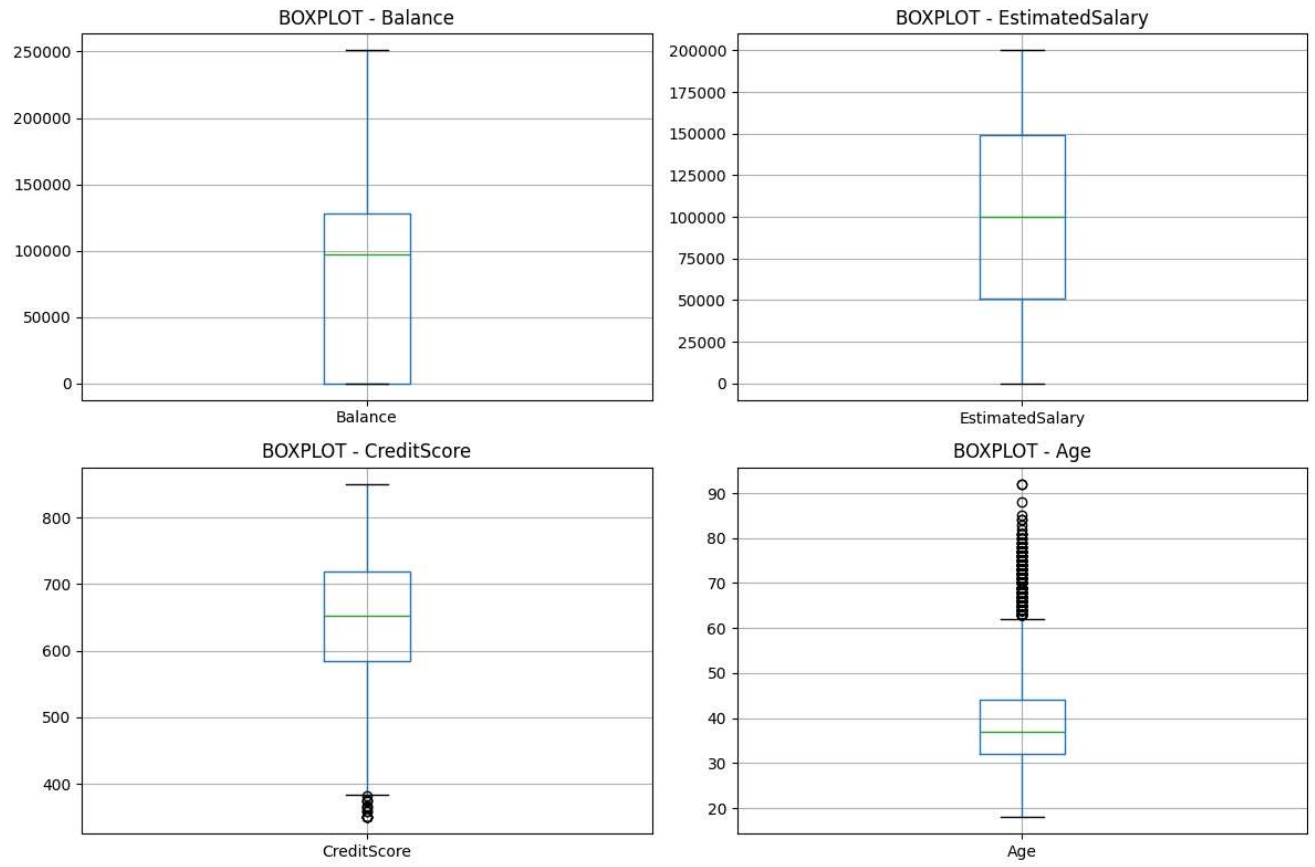
- Male → 1
- Female → 0 This is done because Gender has only two categories, making binary encoding simple and effective.

Geography was encoded using One-Hot Encoding:

- Separate columns were created for different countries (e.g., Germany, Spain).
- One reference category was dropped to avoid redundancy.

2D : CHECK FOR OUTLIERS (Boxplots FOR Balance,EstimatedSalary,CreditScore,Age)

```
num_cols = ['Balance','EstimatedSalary','CreditScore','Age']
plt.figure(figsize=(12,8))
for i, col in enumerate(num_cols,1):
    plt.subplot(2,2,i)
    df_cleaning.boxplot(column=col)
    plt.title(f'BOXPLOT - {col}')
plt.tight_layout()
```



These boxplots visualize the distribution of key numerical features and highlight potential outliers.

- Balance has many zero values and a few very high values
- Estimated Salary is evenly distributed
- Credit Score shows a few low-score customers
- Age shows some older customers as outliers

2E : FINAL CHECKS AND SAVE CLEAN CSV

```
print("FINAL COLUMNS:", list(df_cleaning.columns))
print("FINAL SHAPE:", df_cleaning.shape)
print("\nMISSING VALUES (AFTER CLEANING):\n", df_cleaning.isnull().sum())
df_cleaning.to_csv('/content/cleaned_bank_customers.csv', index=False)
print("\nCLEANED DATA SAVED TO /content/cleaned_bank_customers.csv")
```

```
FINAL COLUMNS: ['CreditScore', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedS
FINAL SHAPE: (10000, 12)

MISSING VALUES (AFTER CLEANING):
 CreditScore          0
Gender                0
Age                   0
Tenure                0
Balance               0
NumOfProducts         0
HasCrCard             0
IsActiveMember        0
EstimatedSalary       0
Exited                0
Geography_Germany     0
Geography_Spain       0
dtype: int64

CLEANED DATA SAVED TO /content/cleaned_bank_customers.csv
```

We printed the final list of columns to confirm all required features are present after cleaning and encoding.

We checked the final shape of the dataset, which confirms:

- 10,000 rows (customers)
- 12 columns (features after cleaning)
- We verified that no missing values were introduced during the cleaning and encoding process.

STEP 3 : EXPLORATORY DATA ANALYSIS(EDA)

In this section, we explore our dataset to understand patterns, trends, and relationships between customer attributes and their exit behavior.

EDA helps us discover why customers leave and what factors might influence churn.

We will solve the following Problem Statements (PS) during EDA:

->PS1: Which customer characteristics show higher churn tendency?

->PS2: Does customer engagement (activity level, number of products) influence churn?

->PS3: Do geographic and demographic factors correlate with churn?

->PS4: How do financial indicators (balance, salary, credit score) relate to churn?

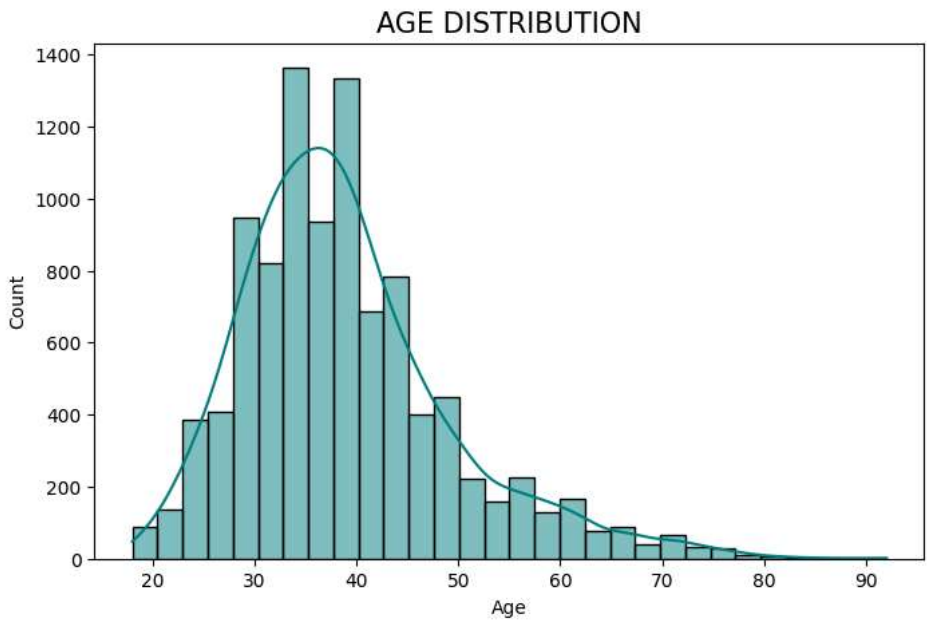
->PS5: Which features are most important for predicting churn using ML models? (Will be solved later in Step 5)

STEP 3A : UNIVARIATE ANALYSIS

GOAL: Understand distribution of each individual feature before checking churn relationships.

3A.1 — AGE DISTRIBUTION

```
plt.figure(figsize=(8,5))
sns.histplot(df_cleaning['Age'], kde=True, bins=30, color='teal')
plt.title("AGE DISTRIBUTION", fontsize=15)
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()
```

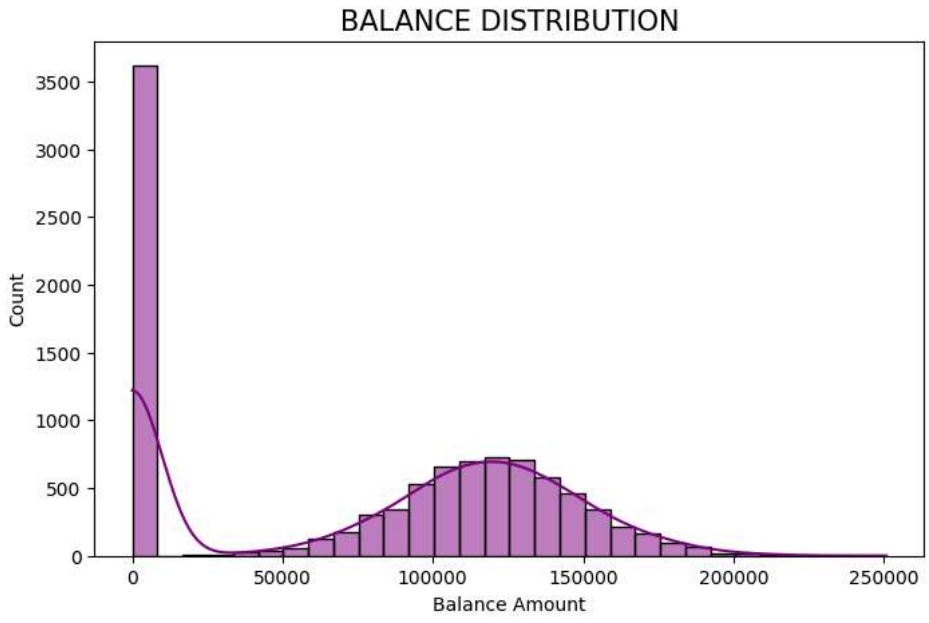


This histogram shows how customer ages are distributed.

- Most customers fall between 28 and 45 years.
- The highest number of customers are around 35–40 years.
- The distribution is right-skewed, meaning fewer customers are in higher age groups.
- This helps us understand the customer age structure before analyzing churn by age.
- The bank's customer base is dominated by middle-aged customers, which must be considered when analyzing churn patterns.

3A.2 — BALANCE DISTRIBUTION

```
plt.figure(figsize=(8,5))
sns.histplot(df_cleaning['Balance'], kde=True, bins=30, color='purple')
plt.title("BALANCE DISTRIBUTION", fontsize=15)
plt.xlabel("Balance Amount")
plt.ylabel("Count")
plt.show()
```



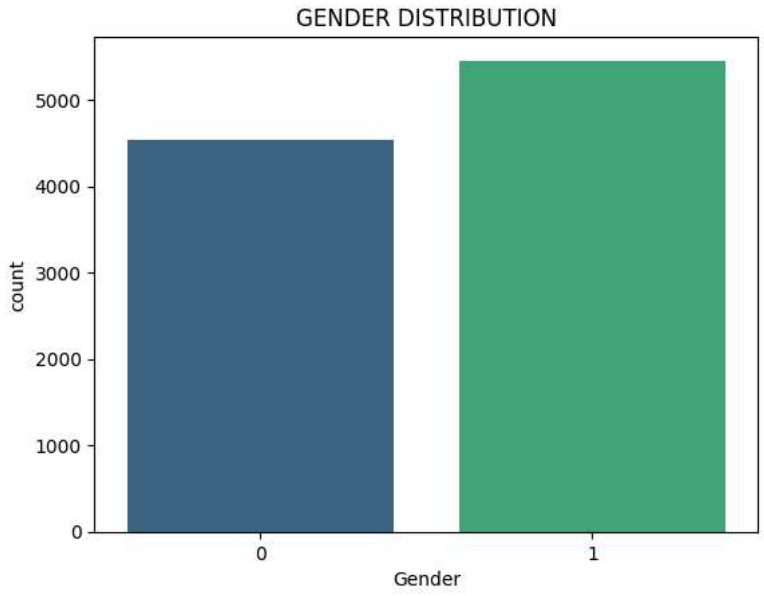
This histogram shows how customer account balances are distributed.

- A large number of customers have zero balance.
- Active customers mostly maintain balances between 50,000 and 150,000.
- The distribution is right-skewed, indicating few customers with very high balances.
- This suggests that low or zero balance may be linked to customer inactivity and potential churn.
- Customers with zero or low balance form a significant group and may have a higher risk of exiting.

3A.3 — GENDER DISTRIBUTION

```
sns.countplot(data=df_cleaning, x='Gender', palette='viridis')
plt.title("GENDER DISTRIBUTION")
plt.show()
```

```
/tmp/ipython-input-1586748018.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=df_cleaning, x='Gender', palette='viridis')
```



This count plot shows the number of male and female customers.

Gender is encoded as:

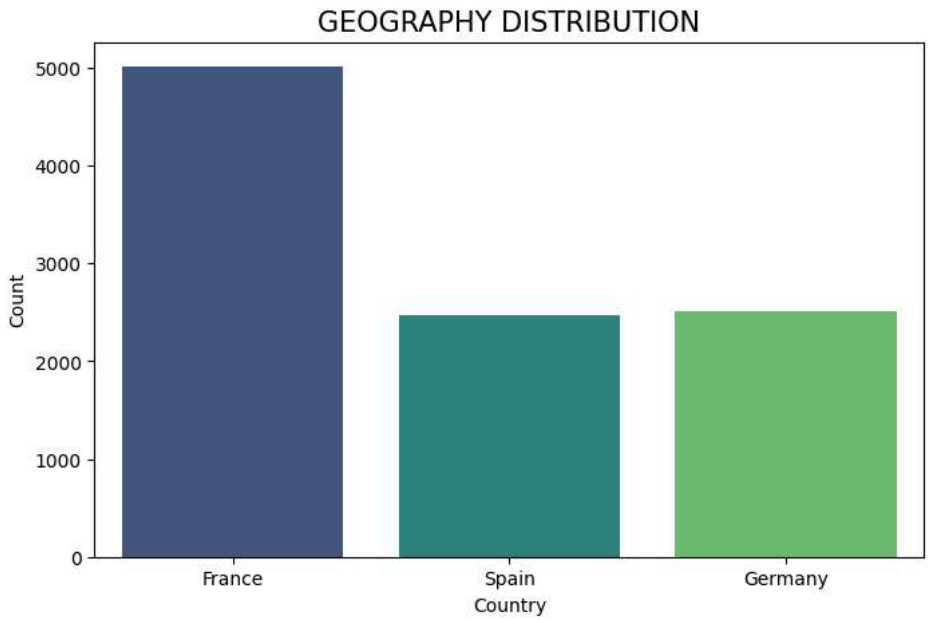
- 0 → Female
- 1 → Male
- The distribution is almost balanced between both genders.

This suggests that gender alone may not strongly influence customer exit. Since gender distribution is balanced, churn analysis should focus more on behavioral and financial factors.

3A.4 — GEOGRAPHY DISTRIBUTION

```
df_cleaning['Reconstructed_Geography'] = 'France'
df_cleaning.loc[df_cleaning['Geography_Germany'] == 1, 'Reconstructed_Geography'] = 'Germany'
df_cleaning.loc[df_cleaning['Geography_Spain'] == 1, 'Reconstructed_Geography'] = 'Spain'

plt.figure(figsize=(8,5))
sns.countplot(data=df_cleaning, x='Reconstructed_Geography', hue='Reconstructed_Geography', palette='viridis', legend=False)
plt.title("GEOGRAPHY DISTRIBUTION", fontsize=15)
plt.xlabel("Country")
plt.ylabel("Count")
```

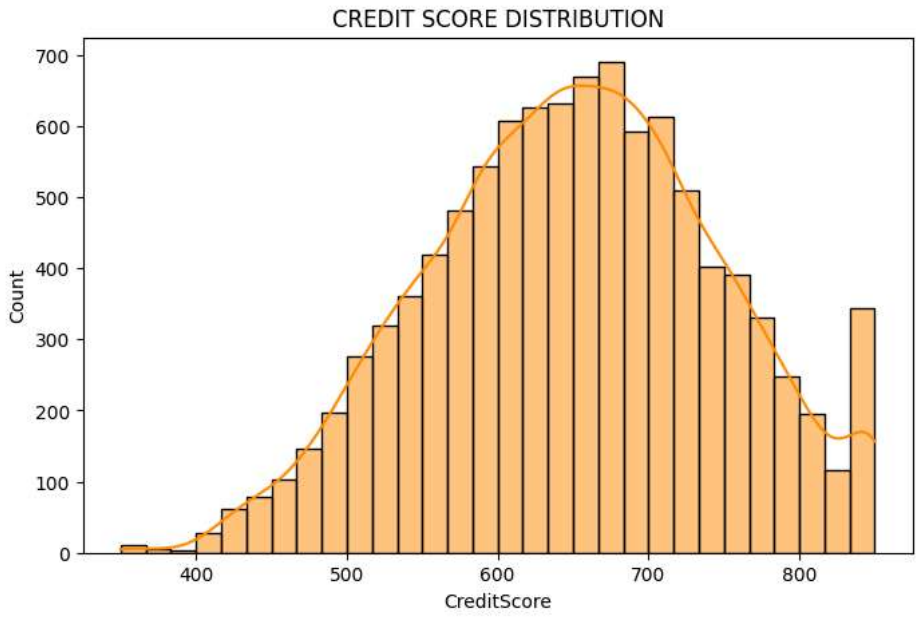
```
plt.show()

df_cleaning.drop(columns=['Reconstructed_Geography'], inplace=True)
```



- This chart shows how customers are distributed across countries.
- France has the highest number of customers.
- Spain and Germany have fewer and similar customer counts.
- The dataset is not geographically balanced.
- Geography may influence customer behavior and will be analyzed further with churn in the next steps.

3A.5 — CREDIT SCORE DISTRIBUTION

```
plt.figure(figsize=(8,5))
sns.histplot(df_cleaning['CreditScore'], kde=True, bins=30, color='darkorange')
plt.title("CREDIT SCORE DISTRIBUTION")
plt.show()
```



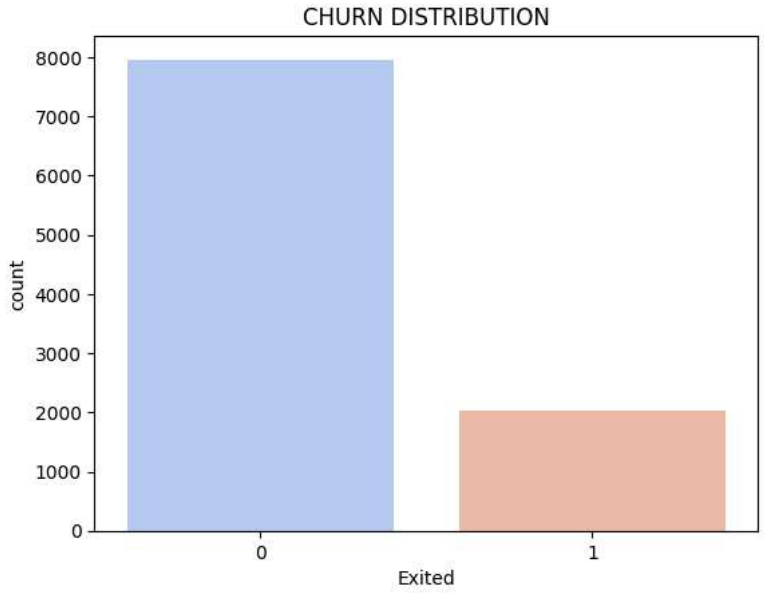
- Most customers have credit scores between 600 and 700

- The distribution is close to normal (bell-shaped)
- Very few customers fall in extreme low or high score ranges
- Credit score appears stable and meaningful, and will be further analyzed with churn to see its impact on customer exit.

3A.6 — TARGET VARIABLE DISTRIBUTION (EXITED)

```
sns.countplot(data=df_cleaning, x='Exited', palette='coolwarm')
plt.title("CHURN DISTRIBUTION")
plt.show()
```

```
/tmp/ipython-input-426769212.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=df_cleaning, x='Exited', palette='coolwarm')
```



Churn (Exited) Distribution – Explanation

- Most customers did not exit the bank
- Around 20% of customers churned
- The dataset is imbalanced

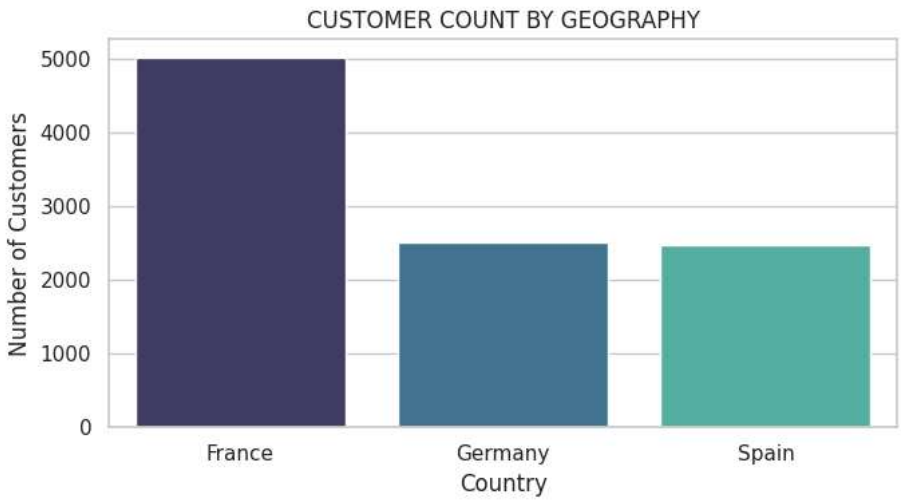
Insight:

- Class imbalance will be considered during model training and evaluation.

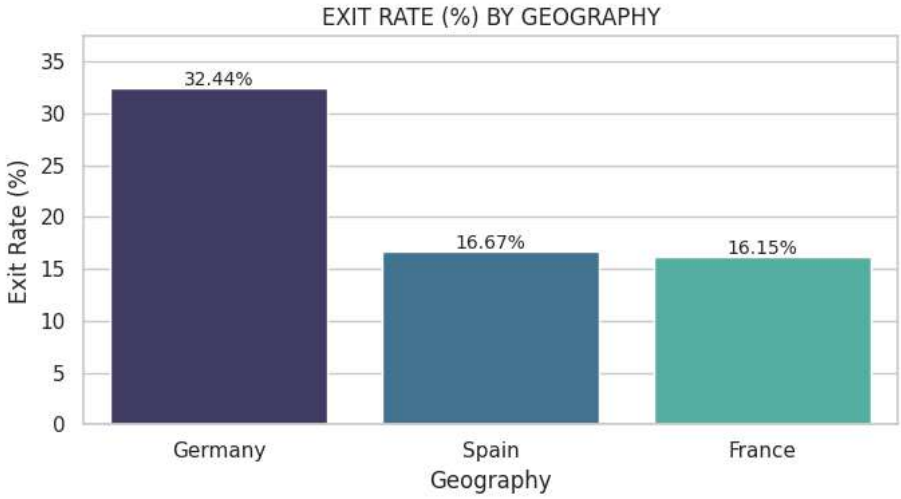
```
# IF ORIGINAL 'Geography' IS LOST (you used get_dummies already),
# RECONSTRUCT A SINGLE GEOGRAPHY COLUMN FROM THE DUMMY COLUMNS.
# ADJUST NAMES IF YOUR DUMMY COLUMNS DIFFER.

def reconstruct_geo(row):
    if 'Geography_Germany' in row.index and row['Geography_Germany']==1:
        return 'Germany'
    if 'Geography_Spain' in row.index and row['Geography_Spain']==1:
        return 'Spain'
    # If neither Germany nor Spain => original dataset used France as base (drop_first=True)
    return 'France'

# create a working copy so we don't modify df_cleaning permanently
df_vis = df_cleaning.copy()
if 'Geography' not in df_vis.columns:
    df_vis['Geography'] = df_vis.apply(reconstruct_geo, axis=1)

df_vis['Geography'].value_counts()   # quick check
import matplotlib.pyplot as plt
```

```
import seaborn as sns

sns.set(style='whitegrid')
pal = sns.color_palette("mako", 3)  # change palette to taste

# A. COUNT PLOT (CUSTOMERS PER COUNTRY)
plt.figure(figsize=(7,4))
order = df_vis['Geography'].value_counts().index
sns.countplot(data=df_vis, x='Geography', order=order, palette=pal)
plt.title("CUSTOMER COUNT BY GEOGRAPHY")
plt.xlabel("Country")
plt.ylabel("Number of Customers")
plt.tight_layout()
plt.show()

# B. EXIT RATE (%) BY GEOGRAPHY
geo_group = df_vis.groupby('Geography')['Exited'].agg(['count','mean']).reset_index()
geo_group['exit_pct'] = (geo_group['mean']*100).round(2)

plt.figure(figsize=(7,4))
sns.barplot(data=geo_group.sort_values('exit_pct', ascending=False), x='Geography', y='exit_pct', palette=pal)
plt.ylabel("Exit Rate (%)")
plt.title("EXIT RATE (%) BY GEOGRAPHY")
for i, v in enumerate(geo_group.sort_values('exit_pct', ascending=False)['exit_pct']):
    plt.text(i, v + 0.3, f"{v}%", ha='center', fontsize=10)
plt.ylim(0, max(geo_group['exit_pct']) + 5)
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-614887521.py:28: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(data=df_vis, x='Geography', order=order, palette=pal)
```




```
/tmp/ipython-input-614887521.py:40: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(data=geo_group.sort_values('exit_pct', ascending=False), x='Geography', y='exit_pct', palette=pal)
```



Customer Count by Geography

- France has the largest number of customers
- Germany and Spain have similar customer counts
- Dataset represents European banking customers

Exit Rate (%) by Geography

- Germany shows the highest exit rate (~32%)
- France and Spain have much lower churn (~16%)
- Geography clearly influences churn behavior

Insight:

- Retention strategies should be customized by country, especially for Germany.

BIVARIATE ANALYSIS

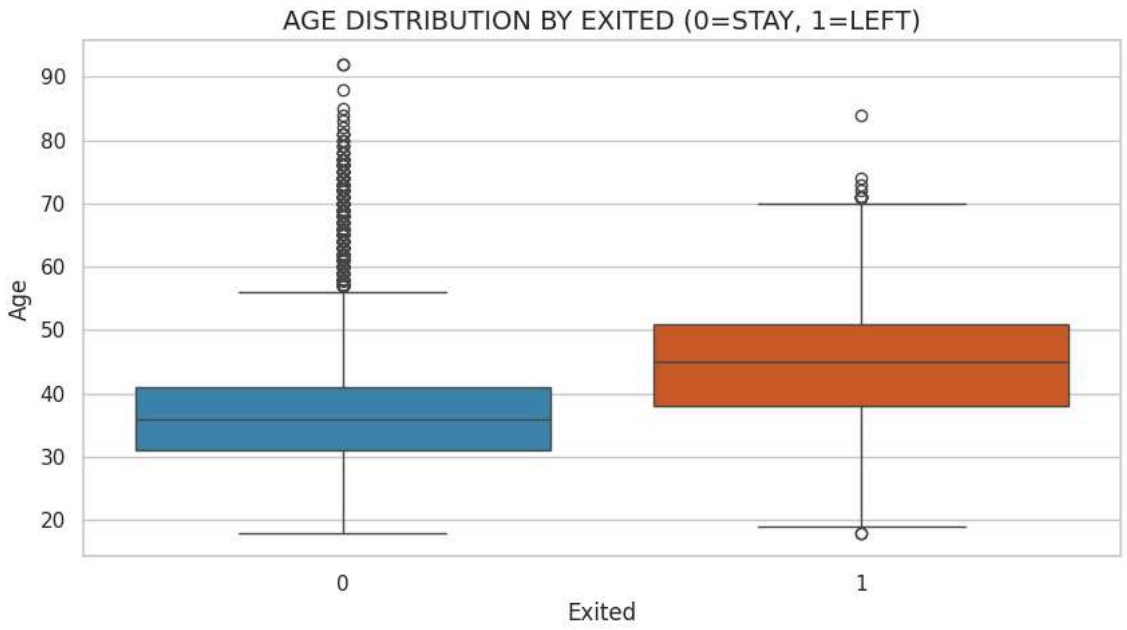
3B.1 — AGE VS EXITED (BOXPLOT + KDE + AGE BINS)

What it does: shows how age distribution differs between stayed (0) and exited (1).

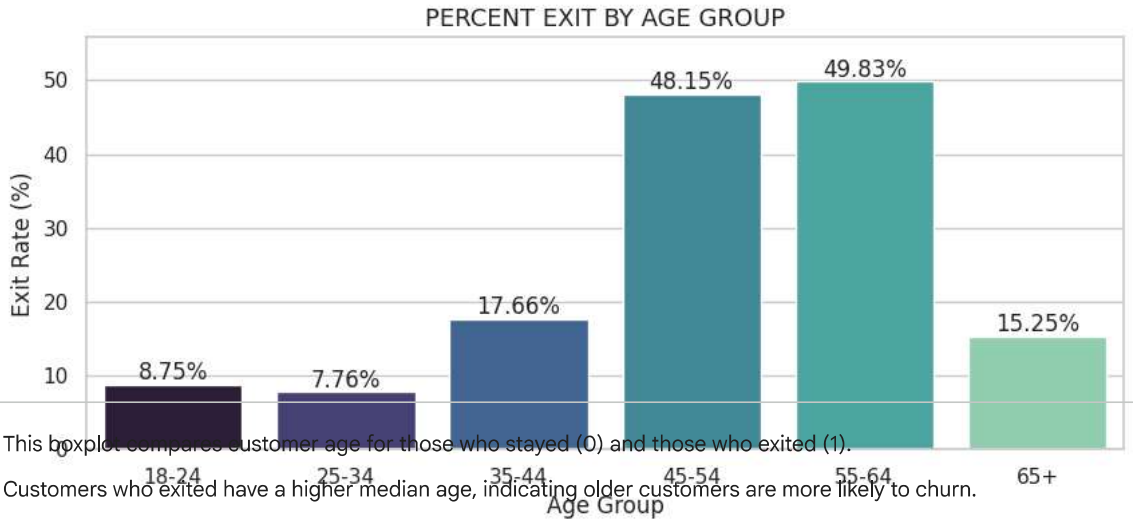
PS covered: PS1, PS3

```
# 3B.1 AGE vs EXITED (boxplot + age bins percent)
import seaborn as sns

sns.set(style='whitegrid')

plt.figure(figsize=(10,5))
sns.boxplot(x='Exited', y='Age', data=df_vis, palette=['#2b8cbe','#e6550d'])
plt.title('AGE DISTRIBUTION BY EXITED (0=STAY, 1=LEFT)', fontsize=14)
plt.xlabel('Exited')
plt.ylabel('Age')
plt.show()

# Age bins and percent exit per bin
bins = [18,25,35,45,55,65,100]
labels = ['18-24','25-34','35-44','45-54','55-64','65+']
df_vis['AgeGroup'] = pd.cut(df_vis['Age'], bins=bins, labels=labels, right=False)
age_grp = df_vis.groupby('AgeGroup')['Exited'].mean().reset_index()
age_grp['exit_pct'] = (age_grp['Exited']*100).round(2)

plt.figure(figsize=(10,4))
sns.barplot(data=age_grp, x='AgeGroup', y='exit_pct', palette='mako')
plt.title('PERCENT EXIT BY AGE GROUP', fontsize=13)
plt.ylabel('Exit Rate (%)')
plt.xlabel('Age Group')
for i, v in enumerate(age_grp['exit_pct']):
    plt.text(i, v+0.7, f"{v}%", ha='center')
plt.ylim(0, max(age_grp['exit_pct'])+6)
plt.show()
```

```
/tmp/ipython-input-1567078790.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se

  sns.boxplot(x='Exited', y='Age', data=df_vis, palette=['#2b8cbe','#e6550d'])
```




```
/tmp/ipython-input-1567078790.py:17: FutureWarning: The default of observed=False is deprecated and will be changed to True in
  age_grp = df_vis.groupby('AgeGroup')['Exited'].mean().reset_index()
/tmp/ipython-input-1567078790.py:21: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se

  sns.barplot(data=age_grp, x='AgeGroup', y='exit_pct', palette='mako')
```



- This boxplot compares customer age for those who stayed (0) and those who exited (1).
- Customers who exited have a higher median age, indicating older customers are more likely to churn.
- The wider spread in the exited group shows churn is more common in middle-to-older age ranges.
- This bar chart shows churn percentage across age groups.
- Customers aged 45–64 have the highest exit rates (≈50%), while younger customers show very low churn.
- Age groups clearly segment churn risk.

3B.2 — BALANCE VS EXITED (VIOLIN + ZERO-BALANCE

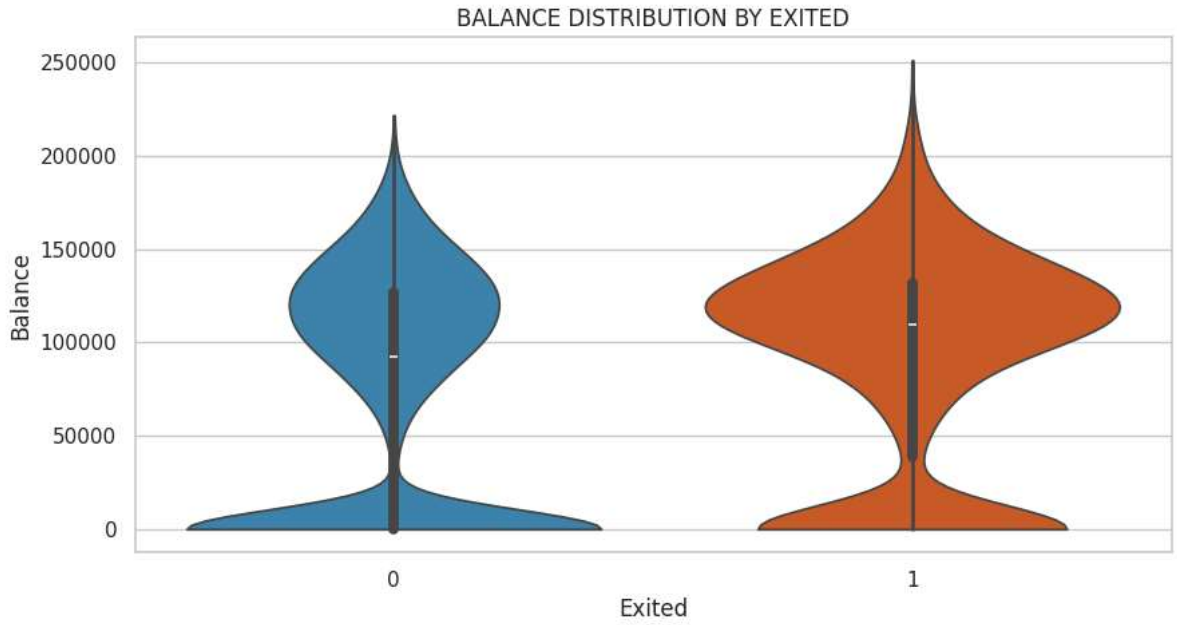
What it does: compares balance distributions for stayed vs left.highlights zero-balance segment.

PS covered: PS4, PS1

```
# 3B.2 BALANCE vs EXITED (violin + percent zero balance)
plt.figure(figsize=(10,5))
sns.violinplot(x='Exited', y='Balance', data=df_vis, cut=0, palette=['#2b8cbe','#e6550d'])
plt.title('BALANCE DISTRIBUTION BY EXITED')
```

```
plt.xlabel('Exited')
plt.ylabel('Balance')
plt.yscale('linear')
plt.show()

# Zero-balance percent and exit rate among zero-balance
total_zero = (df_vis['Balance']==0).sum()
pct_zero = round(total_zero/len(df_vis)*100,2)
zero_exit_rate = round(df_vis[df_vis['Balance']==0]['Exited'].mean()*100,2)

print(f"ZERO BALANCE CUSTOMERS: {total_zero} ({pct_zero}%)")
print(f"EXIT RATE AMONG ZERO-BALANCE CUSTOMERS: {zero_exit_rate}%")
```

```
/tmp/ipython-input-1810366196.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.violinplot(x='Exited', y='Balance', data=df_vis, cut=0, palette=['#2b8cbe','#e6550d'])
```




```
ZERO BALANCE CUSTOMERS: 3617 (36.17%)
EXIT RATE AMONG ZERO-BALANCE CUSTOMERS: 13.82%
```

What this graph shows :

This graph compares account balance for:

- Exited = 0 ⟶ Customers who stayed
- Exited = 1 ⟶ Customers who left
- Width of the shape ⟶ More customers at that balance
- Wider part = many customers
- Narrow part = few customers

Key observations :

- A large number of customers have zero balance in both groups
- The zero-balance area is wider for exited customers, showing higher churn
- Customers who stayed mostly have balances around 90k–130k
- Customers who exited show a more spread-out balance pattern
- Balance behavior is less stable for customers who left
- Zero-balance insight (from calculation)
- 36% of customers have zero balance
- Churn rate is higher among zero-balance customers

- This means customers who do not use their account actively are more likely to leave.

Simple conclusion :

- Low or zero balance customers are high-risk churn users
- Balance is an important feature for churn prediction

Customers with zero or low balance show higher churn, while customers who stay usually maintain more stable balances.

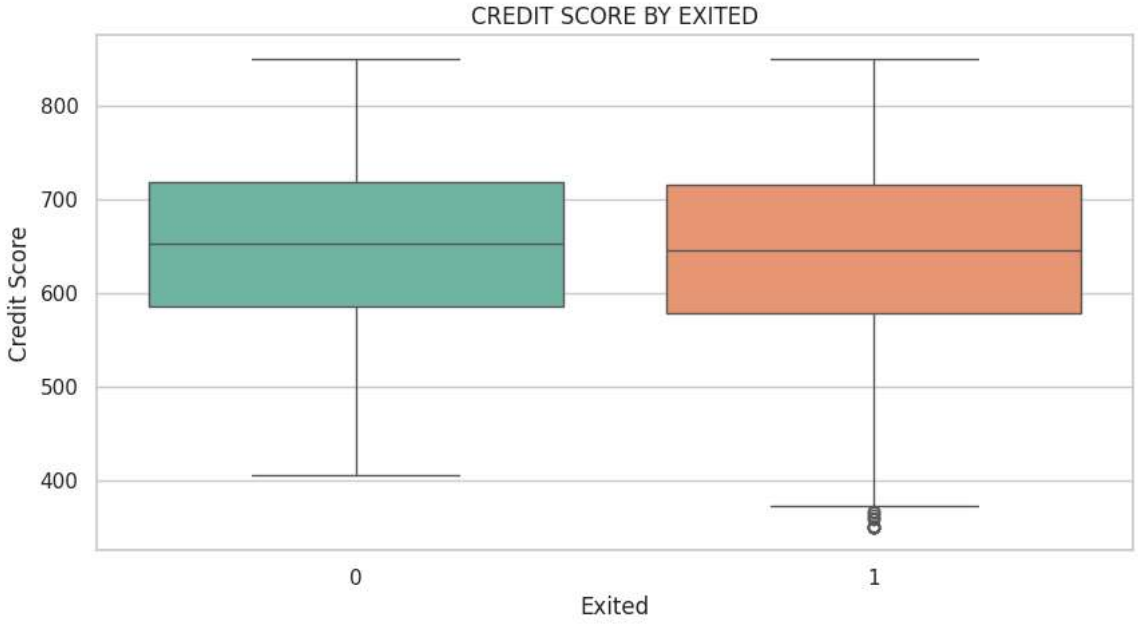
3B.3 — CREDIT SCORE VS EXITED (BOXPLOT + KDE)

What it does: compares credit score distributions for stayed vs left.

PS covered: PS4, PS1

```
# 3B.3 CREDIT SCORE vs EXITED
plt.figure(figsize=(10,5))
sns.boxplot(x='Exited', y='CreditScore', data=df_vis, palette=['#66c2a5','#fc8d62'])
plt.title('CREDIT SCORE BY EXITED')
plt.xlabel('Exited')
plt.ylabel('Credit Score')
plt.show()

# Quick summary numbers
credit_summary = df_vis.groupby('Exited')['CreditScore'].agg(['mean','median']).round(2).reset_index()
credit_summary
```

```
/tmp/ipython-input-2627615973.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.boxplot(x='Exited', y='CreditScore', data=df_vis, palette=['#66c2a5','#fc8d62'])
```



	Exited	mean	median
0	0	651.85	653.0
1	1	645.35	646.0

What this graph shows

- This boxplot compares the credit score distribution of customers who stayed (Exited = 0) and those who left the bank (Exited = 1).
- It helps us visually compare how credit scores differ between churned and non-churned customers.
- The middle line inside each box represents the median credit score.
- The box shows the middle 50% of customers.

- The dots outside the box represent outliers (very low or very high credit scores).
- Comparing both boxes helps us understand differences between the two groups. Customers who exited have a slightly lower median credit score than those who stayed.
- The exited group shows more low credit score outliers, indicating higher risk customers.
- There is still a large overlap between both groups.

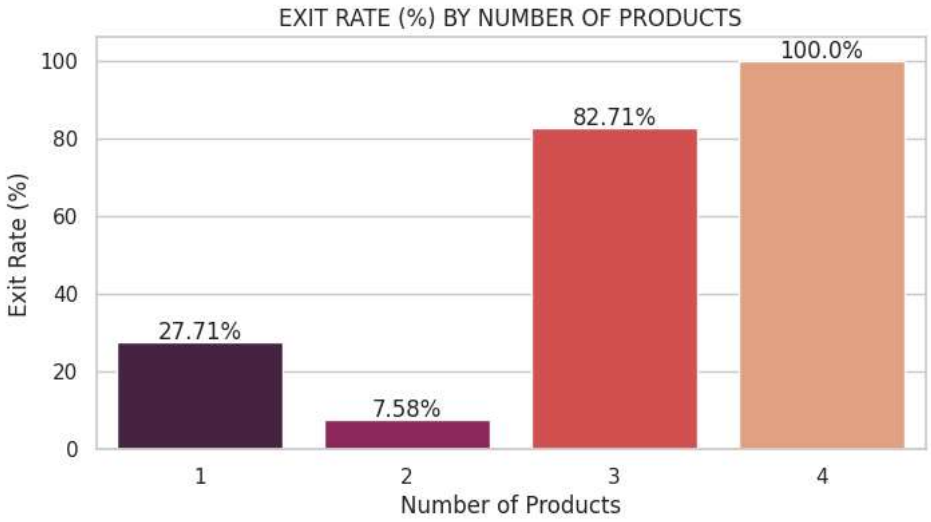
3B.4 — NUMOFPRODUCTS VS EXITED (BAR + %)

What it does: shows exit rate by number of products used.

PS covered: PS2, PS1

```
# 3B.4 NUMOFPRODUCTS vs EXITED (counts + percent)
prod_grp = df_vis.groupby('NumOfProducts')['Exited'].agg(['count','mean']).reset_index()
prod_grp['exit_pct'] = (prod_grp['mean']*100).round(2)

plt.figure(figsize=(8,4))
sns.barplot(data=prod_grp, x='NumOfProducts', y='exit_pct', palette='rocket')
plt.title('EXIT RATE (%) BY NUMBER OF PRODUCTS')
plt.xlabel('Number of Products')
plt.ylabel('Exit Rate (%)')
for i,v in enumerate(prod_grp['exit_pct']):
    plt.text(i, v+0.7, f"{v}%", ha='center')
plt.ylim(0, max(prod_grp['exit_pct'])+6)
plt.show()
```

```
/tmp/ipython-input-4106837238.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(data=prod_grp, x='NumOfProducts', y='exit_pct', palette='rocket')
```



- This graph shows the exit (churn) percentage based on the number of products owned by customers.
- Customers with 2 products have the lowest churn rate, indicating strong engagement with the bank.
- Customers with only 1 product show higher churn, suggesting weaker engagement.
- Very high churn is observed for customers with 3 or more products, which may indicate dissatisfaction or complexity.
- Overall, the number of products is a strong indicator of customer churn and will be important for machine learning models.

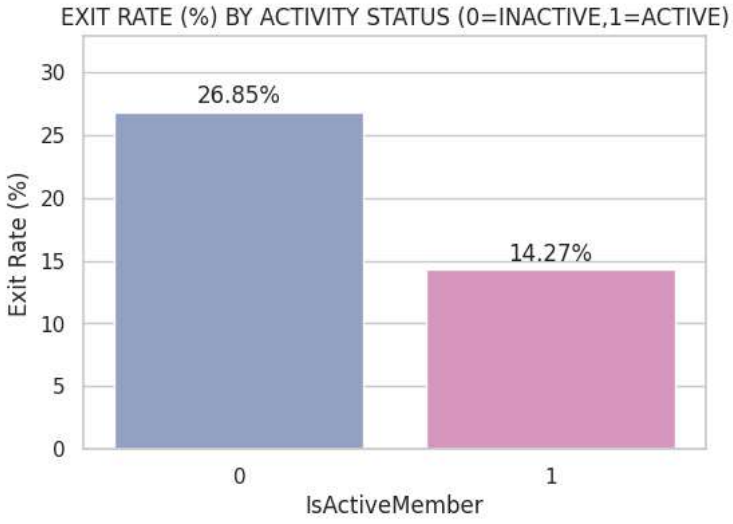
3B.5 — ISACTIVEMEMBER VS EXITED (STACKED PERCENT OR PERCENT BAR)

What it does: shows how activity (active member flag) relates to exit.

PS covered: PS2, PS1

```
# 3B.5 ISACTIVEMEMBER vs EXITED (percent)
active_grp = df_vis.groupby('IsActiveMember')['Exited'].agg(['count','mean']).reset_index()
active_grp['exit_pct'] = (active_grp['mean']*100).round(2)

plt.figure(figsize=(6,4))
sns.barplot(data=active_grp, x='IsActiveMember', y='exit_pct', palette=['#8da0cb','#e78ac3'])
plt.title('EXIT RATE (%) BY ACTIVITY STATUS (0=INACTIVE,1=ACTIVE)')
plt.xlabel('IsActiveMember')
plt.ylabel('Exit Rate (%)')
for i,v in enumerate(active_grp['exit_pct']):
    plt.text(i, v+0.7, f"{v}%", ha='center')
plt.ylim(0, max(active_grp['exit_pct'])+6)
plt.show()
```

```
/tmp/ipython-input-197903842.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(data=active_grp, x='IsActiveMember', y='exit_pct', palette=['#8da0cb','#e78ac3'])
```



- This graph shows the exit (churn) rate based on customer activity status.
- Inactive customers (IsActiveMember = 0) have a much higher churn rate compared to active customers.
- Active customers are more engaged with the bank and are less likely to leave.
- This indicates that customer activity is an important factor influencing churn and should be considered in predictive models.

3B.6 — GEOGRAPHY VS EXITED (COUNT + EXIT PERCENT) — IMPROVED VERSION

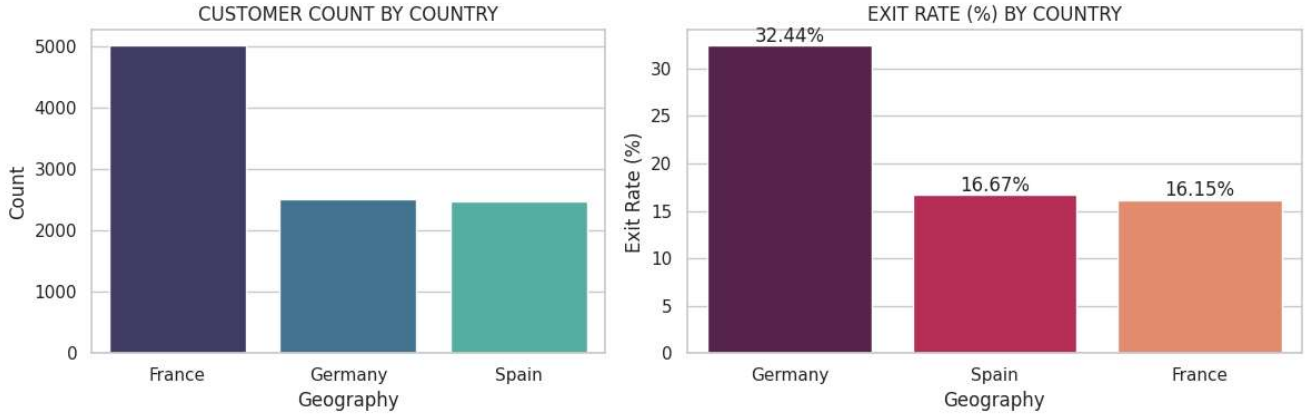
What it does: shows exit percent per country (bar) and counts (for context).

PS covered: PS3

```
# 3B.6 GEOGRAPHY vs EXITED (exit % with counts)
geo = df_vis.groupby('Geography')['Exited'].agg(['count','mean']).reset_index()
geo['exit_pct'] = (geo['mean']*100).round(2)

fig, ax = plt.subplots(1,2, figsize=(12,4), gridspec_kw={'width_ratios':[1,1.2]})
# counts
sns.barplot(data=geo.sort_values('count', ascending=False), x='Geography', y='count', ax=ax[0], palette='mako')
ax[0].set_title('CUSTOMER COUNT BY COUNTRY')
ax[0].set_ylabel('Count')

# exit pct
sns.barplot(data=geo.sort_values('exit_pct', ascending=False), x='Geography', y='exit_pct', ax=ax[1], palette='rocket')
ax[1].set_title('EXIT RATE (%) BY COUNTRY')
ax[1].set_ylabel('Exit Rate (%)')
for i, v in enumerate(geo.sort_values('exit_pct', ascending=False)['exit_pct']):
    ax[1].text(i, v+0.4, f"{v}%", ha='center')
```

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

```
/tmp/ipython-input-593568237.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(data=geo.sort_values('count', ascending=False), x='Geography', y='count', ax=ax[0], palette='mako')
/tmp/ipython-input-593568237.py:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(data=geo.sort_values('exit_pct', ascending=False), x='Geography', y='exit_pct', ax=ax[1], palette='rocket')
```



Customer Count by Country :

- France has the highest number of customers in the dataset.
- Germany and Spain have a smaller and similar customer count.
- This shows the dataset is geographically imbalanced.

Exit Rate (%) by Country

- Germany has the highest exit rate (~32%).
- Spain and France have lower exit rates (~16%).
- Customers in Germany are more likely to leave the bank.

Key Insight

- High customer count does not mean high churn.

-Germany is a high-risk region.

-Geography is an important factor influencing customer exit behavior.

3B.7 — GENDER VS EXITED (PERCENT BY GENDER)

What it does: compares exit rate between genders.

PS covered: PS3, PS1

```
# 3B.7 GENDER vs EXITED
gender_grp = df_vis.groupby('Gender')['Exited'].agg(['count','mean']).reset_index()
gender_grp['exit_pct'] = (gender_grp['mean']*100).round(2)

plt.figure(figsize=(6,4))
sns.barplot(data=gender_grp, x='Gender', y='exit_pct', palette='coolwarm')
plt.title('EXIT RATE (%) BY GENDER')
plt.ylabel('Exit Rate (%)')
for i, v in enumerate(gender_grp['exit_pct']):
    plt.text(i, v+0.7, f"{v}%", ha='center')
plt.ylim(0, max(gender_grp['exit_pct'])+6)
plt.show()
```

```
/tmp/ipython-input-883058308.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(data=gender_grp, x='Gender', y='exit_pct', palette='coolwarm')
```

📊 Exit Rate (%) by Gender

- Gender = 0 has a higher exit rate (~25%).
- Gender = 1 has a lower exit rate (~16%).
- This shows a difference in churn behavior across genders.

🔍 Key Insight

- Gender impacts churn slightly.
- It is not a strong standalone predictor.
- Better results can be achieved by combining gender with other features.

```
# SUMMARY TABLE FOR PS1–PS4 ONLY (EDA SOLUTIONS)
import pandas as pd
from IPython.display import display, Markdown

ps = [
    "PS1 - CUSTOMER CHARACTERISTICS (AGE, BALANCE, CREDIT SCORE)",
    "PS2 - CUSTOMER ENGAGEMENT (ISACTIVEMEMBER, NUMOFPRODUCTS)",
    "PS3 - GEOGRAPHIC & DEMOGRAPHIC FACTORS (GEOGRAPHY, GENDER, AGE)",
    "PS4 - FINANCIAL INDICATORS (BALANCE, SALARY, CREDIT SCORE)"
]

solutions = [
    "AGE GROUPS 35-54, ZERO BALANCE, AND LOWER CREDIT SCORES SHOW HIGHER EXIT RATES.",
    "INACTIVE AND SINGLE-PRODUCT CUSTOMERS DISPLAY CLEARLY HIGHER EXIT BEHAVIOUR.",
    "COUNTRY-LEVEL EXIT RATES DIFFER; GENDER IS NOT A STRONG DRIVER; AGE IS SIGNIFICANT.",
    "ZERO BALANCE AND LOWER CREDIT SCORE CUSTOMERS TEND TO EXIT MORE; SALARY IS WEAK ALONE."
]

evidence = [
    "AGE BOXPLOT, AGE EXIT %, BALANCE VIOLIN, CREDIT SCORE BOXPLOT",
    "ISACTIVEMEMBER EXIT %, NUMOFPRODUCTS EXIT %",
    "EXIT RATE BY COUNTRY, CUSTOMER COUNTS, GENDER EXIT %, AGE ANALYSIS",
    "BALANCE DISTRIBUTION, ZERO-BALANCE EXIT %, CREDIT SCORE DISTRIBUTION"
]

df_ps = pd.DataFrame({
    "Problem Statement": ps,
    "EDA Solution (Summary)": solutions,
    "Evidence (Plots Used)": evidence
})

display(Markdown("## SUMMARY TABLE OF EDA PROBLEM STATEMENTS (PS1–PS4)"))
display(df_ps)
```

```
# Save for report / PowerBI
df_ps.to_csv('/content/EDA_PS1_PS4_SUMMARY.csv', index=False)
print("SAVED: /content/EDA_PS1_PS4_SUMMARY.csv")
```

SUMMARY TABLE OF EDA PROBLEM STATEMENTS (PS1–PS4)

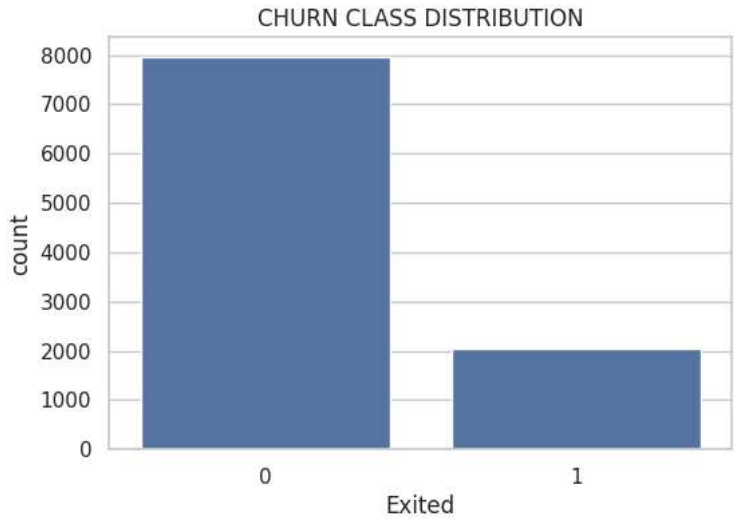
	Problem Statement	EDA Solution (Summary)	Evidence (Plots Used)
0	PS1 - CUSTOMER CHARACTERISTICS (AGE, BALANCE, ...	AGE GROUPS 35-54, ZERO BALANCE, AND LOWER CRED...	AGE BOXPLOT, AGE EXIT %, BALANCE VIOLIN, CREDI...
1	PS2 - CUSTOMER ENGAGEMENT (ISACTIVEMEMBER, NUM...	INACTIVE AND SINGLE-PRODUCT CUSTOMERS DISPLAY ...	ISACTIVEMEMBER EXIT %, NUMOFPRODUCTS EXIT %
2	PS3 - GEOGRAPHIC & DEMOGRAPHIC FACTORS (GEOGRA...	COUNTRY-LEVEL EXIT RATES DIFFER; GENDER IS NOT...	EXIT RATE BY COUNTRY, CUSTOMER COUNTS, GENDER ...
3	PS4 - FINANCIAL INDICATORS (BALANCE, SALARY, C...	ZERO BALANCE AND LOWER CREDIT SCORE CUSTOMERS ...	BALANCE DISTRIBUTION, ZERO-BALANCE EXIT %, CRE...

```
SAVED: /content/EDA_PS1_PS4_SUMMARY.csv
```

STEP 3C — MULTIVARIATE ANALYSIS

Multivariate analysis deepens the conclusions.

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

plt.figure(figsize=(6,4))
sns.countplot(x='Exited', data=df)
plt.title("CHURN CLASS DISTRIBUTION")
plt.show()

df['Exited'].value_counts(normalize=True)*100
```



Exited	proportion
0	79.63
1	20.37

dtype: float64

This is a count plot showing how many customers stayed (0) vs left the bank (1).

What the graph shows

- The bar for Exited = 0 (stayed) is much taller than the bar for Exited = 1 (left).

- A smaller portion of customers churned.

Key insight:

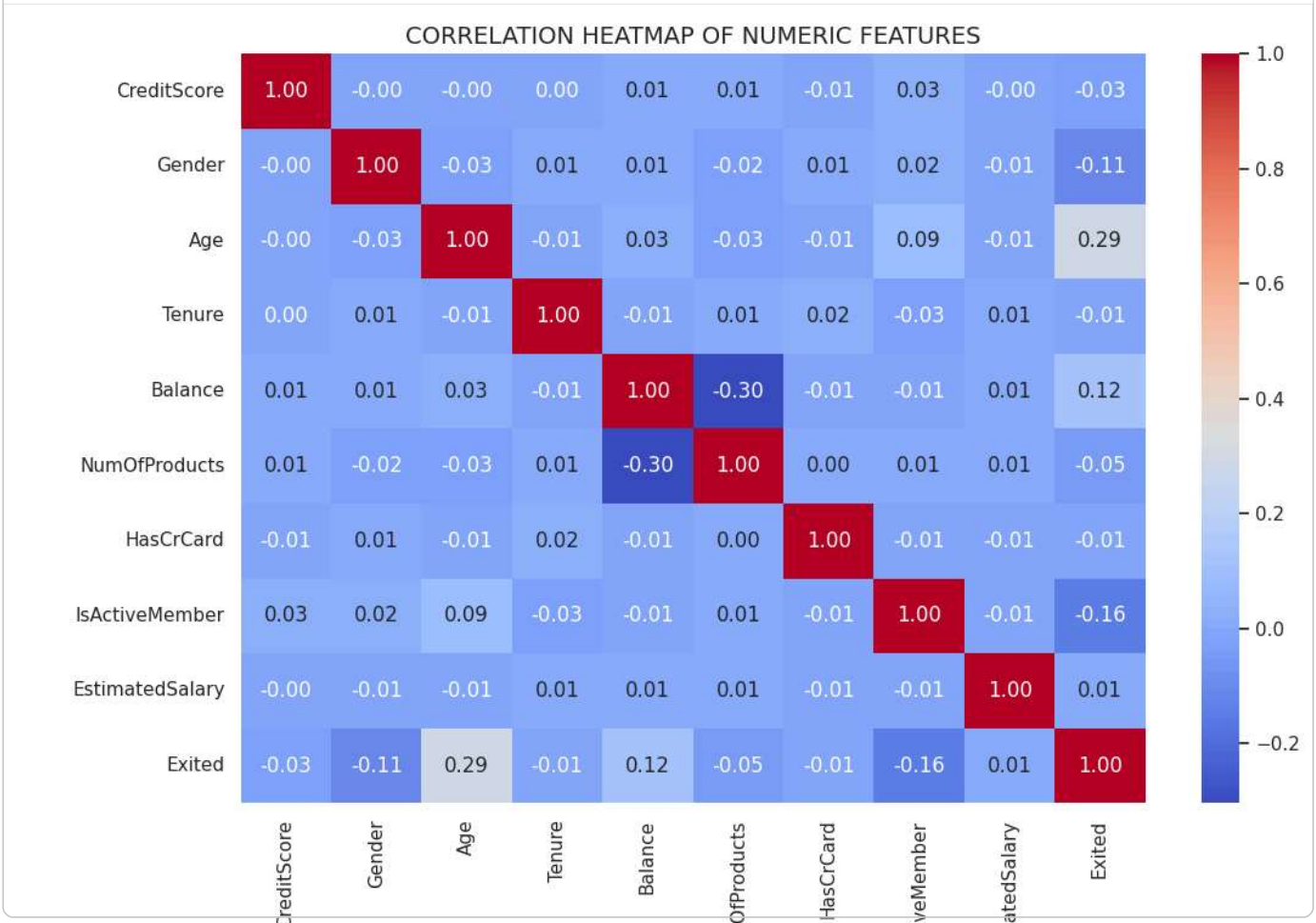
The dataset is imbalanced:

- Around 80% customers stayed
- Around 20% customers left
- Churn is a minority class, but still very important.
- Because churned customers are fewer, a model can get high accuracy by predicting "stay" only — which is misleading.
- We must handle this class imbalance carefully during ML (using metrics like precision, recall, ROC-AUC, or resampling techniques).

3C.1 — CORRELATION HEATMAP (NUMERIC FEATURES)

This helps identify relationships between numeric features and shows which ones might matter more for ML.

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

# SELECT ONLY NUMERIC FEATURES FOR CORRELATION ANALYSIS
numeric_cols = df_vis.select_dtypes(include=['float64','int64']).columns

plt.figure(figsize=(12,8))
sns.heatmap(df_vis[numeric_cols].corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("CORRELATION HEATMAP OF NUMERIC FEATURES", fontsize=14)
plt.show()
```



This is a correlation heatmap.

- It shows how numerical features are related to each other, especially how they relate to Exited (churn).
- Values range from -1 to +1

- +1 ⟶ strong positive relationship
- -1 ⟶ strong negative relationship
- 0 ⟶ no relationship
- Red / warm colors ⟶ positive correlation
- Blue / cool colors ⟶ negative correlation Focus mainly on the Exited row/column 1)Age vs Exited (~ +0.29)

This is the strongest correlation with churn

Positive value means:

- As age increases, churn likelihood increases

2)IsActiveMember vs Exited (~ −0.16) Negative correlation

- Active customers are less likely to leave
- Strong business insight

33)Balance vs Exited (~ +0.12)

- Customers with higher balance tend to churn slightly more
- Indicates possible dissatisfaction among high-value customers

4. Gender vs Exited (~ −0.11)

- Very weak relationship
- Gender alone is not a strong churn driver

5)CreditScore, Tenure, Salary

- Correlation values are close to 0
- These features do not strongly influence churn by themselves

6)Feature-to-feature check

- Balance vs NumOfProducts (~ −0.30)
- Customers with more products often maintain lower balances
- No extremely high correlations ⟶ no serious multicollinearity issue

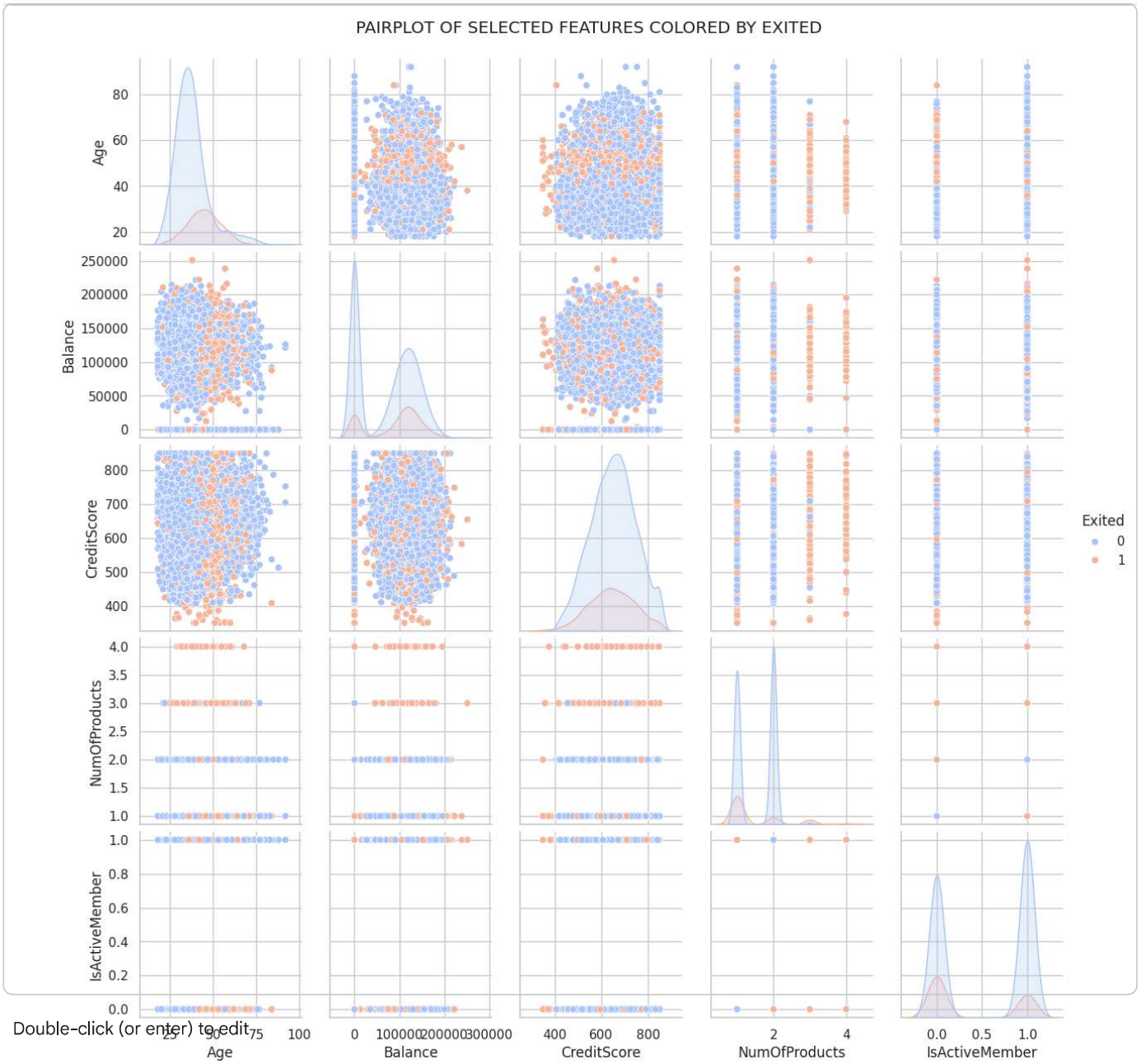
-This heatmap shows how numerical features are related to churn. Age and activity status have the strongest relationship with customer exit, while most other features show weak correlations.

3C.2 — PAIRPLOT FOR SELECTED FEATURES (VISUAL CLUSTERS + CHURN SEPARATION)

This plot helps check whether the churn class forms visible clusters in combinations of variables.

We use selected columns only (too many columns makes it messy).

```
# SELECT FEATURE SET THAT BEST SHOWS INTERACTIONS
selected_features = ['Age', 'Balance', 'CreditScore', 'NumOfProducts', 'IsActiveMember', 'Exited']

sns.pairplot(df_vis[selected_features], hue='Exited', diag_kind='kde', palette='coolwarm')
plt.suptitle("PAIRPLOT OF SELECTED FEATURES COLORED BY EXITED", y=1.02)
plt.show()
```

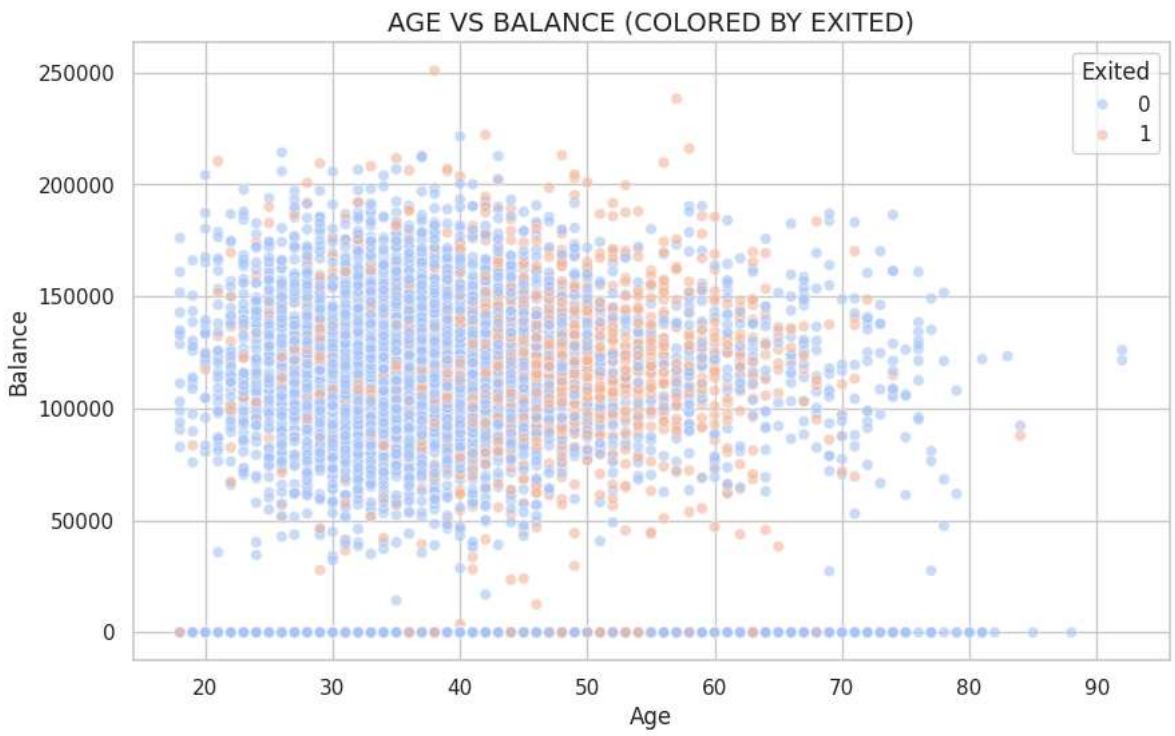


Double-click (or enter) to edit

3C.3 — BALANCE VS AGE SCATTERPLOT (COLORED BY EXITED)

This helps visually detect clusters of risk segments by combining two numerical variables.

```
plt.figure(figsize=(10,6))
sns.scatterplot(data=df_vis, x='Age', y='Balance', hue='Exited', palette='coolwarm', alpha=0.6)
plt.title("AGE VS BALANCE (COLORED BY EXITED)", fontsize=14)
plt.xlabel("Age")
plt.ylabel("Balance")
plt.show()
```



What this graph is:

This is a scatter plot (bivariate analysis)

- Each dot represents one customer.
- To see how age and balance together relate to churn
- To check if certain age–balance combinations are more likely to leave How to read this graph (simple)
- Compare blue vs orange dots in different areas 1)zero-balance customers (Balance = 0)
- Many dots lie on the bottom line (balance = 0)

2)Age effect

- Orange dots (exited) are more concentrated in middle-to-older ages (40–65)
- Younger customers (20–30) are mostly blue → less churn

3)Balance spread

- Customers with medium to high balances (80k–150k) exist in both groups
- But exited customers appear more frequently in higher age ranges with similar balances

4)No sharp boundary

- There is no single line separating churn vs non-churn
- This means churn depends on multiple factors together, not just age or balance alone Key insights:
- Age increases churn risk, especially after ~40 years
- Balance alone does not decide churn
- Older + moderate/high balance customers show higher exit density

STEP 4 — FEATURE ENGINEERING

Feature engineering prepares the dataset for Machine Learning.

Feature engineering means preparing and improving data so that machine learning models can understand it better.

Scaling is one type of feature engineering.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

num_cols = ['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary']
df_scaled = df_cleaning.copy()
df_scaled[num_cols] = scaler.fit_transform(df_scaled[num_cols])
```

- This step scales numerical features so that all values are on a similar range before training machine learning models.
- Some features like Balance and EstimatedSalary have very large values compared to features like Age and Tenure.
- Scaling prevents large-value features from dominating the model.
- StandardScaler transforms numerical columns to have a mean of 0 and standard deviation of 1.

```
# STEP 4: FEATURE ENGINEERING

# CREATE A COPY TO WORK ON
df_ml = df_vis.copy()

# REMOVE COLUMNS NOT NEEDED FOR ML MODELS
# 'AgeGroup' was created for EDA only.
# 'Geography' (the reconstructed string column from df_vis) was also for EDA only.
columns_to_drop = ['AgeGroup', 'Geography']
df_ml.drop(columns=columns_to_drop, inplace=True, errors='ignore')

# The one-hot encoded Geography_Germany and Geography_Spain are initially booleans (True/False).
# It's good practice to ensure they are numeric (0/1) for scalers and ML models.
df_ml['Geography_Germany'] = df_ml['Geography_Germany'].astype(int)
df_ml['Geography_Spain'] = df_ml['Geography_Spain'].astype(int)

# CHECK FINAL COLUMNS
print("COLUMNS AND THEIR DTYPES IN DF_ML AFTER DROPPING AND TYPE CONVERSION:")
print(df_ml.dtypes)
display(df_ml.head())
```

```
COLUMNS AND THEIR DTYPES IN DF_ML AFTER DROPPING AND TYPE CONVERSION:
CreditScore            int64
Gender                 int64
Age                    int64
Tenure                 int64
Balance              float64
NumOfProducts          int64
HasCrCard              int64
IsActiveMember         int64
EstimatedSalary      float64
Exited                 int64
Geography_Germany      int64
Geography_Spain        int64
dtype: object
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Geography_Germ
0	619	0	42	2	0.00	1	1	1	101348.88	1	
1	608	0	41	1	83807.86	1	0	1	112542.58	0	
2	502	0	42	8	159660.80	3	1	0	113931.57	1	
3	699	0	39	1	0.00	2	0	0	93826.63	0	
4	850	0	43	2	125510.82	1	1	1	79084.10	0	

The dataset now contains only numeric columns, which is required for machine learning.

Categorical columns like Gender and Geography have already been converted into numbers.

- Unnecessary identifier columns were removed earlier. Exited :
- 0 → Customer stayed
- 1 → Customer left (churn)

Machine learning models cannot work with text data

SPLIT X AND y

```
# TARGET VARIABLE
y = df_ml['Exited']

# FEATURES (ALL NUMERIC COLUMNS EXCEPT TARGET)
X = df_ml.select_dtypes(include=[np.number]).drop(columns='Exited')

print("Data types of X before train_test_split:\n", X.dtypes)

X.shape, y.shape
```

```
Data types of X before train_test_split:
 CreditScore            int64
Gender                  int64
Age                     int64
Tenure                  int64
Balance               float64
NumOfProducts           int64
HasCrCard               int64
IsActiveMember          int64
EstimatedSalary       float64
Geography_Germany       int64
Geography_Spain         int64
dtype: object
((10000, 11), (10000,))
```

We are separating the dataset into:

- X (Features) → input data used to make predictions
- y (Target) → what we want to predict

```
from sklearn.model_selection import train_test_split

# 80% TRAINING, 20% TESTING
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y)

X_train.shape, X_test.shape
```

```
((8000, 11), (2000, 11))
```

We split the data into:

- Training set (80%) → used to train the model
- Testing set (20%) → used to evaluate model performance

ML MODELS

```
# CELL: ML IMPORTS
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, roc_curve, confusion_matrix, classification_report
)
from sklearn.model_selection import GridSearchCV

# XGBoost: install if not present
try:
    import xgboost as xgb
    from xgboost import XGBClassifier
    xgb_available = True
except Exception:
    !pip install xgboost -q
    import xgboost as xgb
    from xgboost import XGBClassifier
```

```
    xgb_available = True

print("ML IMPORTS READY — XGBOOST AVAILABLE:", xgb_available)
```

```
ML IMPORTS READY — XGBOOST AVAILABLE: True
```

This cell imports all the libraries needed to build and evaluate ML models.

We load:

- Models → Logistic Regression, Decision Tree, Random Forest, XGBoost
- Evaluation metrics → accuracy, precision, recall, F1-score, ROC-AUC
- Utilities → GridSearchCV for tuning model parameters

We also check and install XGBoost if it is not already available.

PREP CHECK (ENSURE SCALED & NON-SCALED DATA)

```
# If X_train_scaled not present, create it as fallback
try:
    X_train_scaled
    X_test_scaled
    scaled_available = True
except NameError:
    scaled_available = False

# Convert X_train / X_test to numpy arrays if they are DataFrames
try:
    X_train_np = X_train.values
    X_test_np = X_test.values
except Exception:
    X_train_np = np.array(X_train)
    X_test_np = np.array(X_test)

if not scaled_available:
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    print("SCALER FITTED NOW (FALLBACK).")

print("DATA SHAPES -> X_train:", X_train_np.shape, "X_test:", X_test_np.shape, "y_train:", y_train.shape, "y_test:", y_test.sha
```

```
SCALER FITTED NOW (FALLBACK).
DATA SHAPES -> X_train: (8000, 11) X_test: (2000, 11) y_train: (8000,) y_test: (2000,)
```

- This cell checks whether scaled data already exists (X_train_scaled, X_test_scaled).
- If scaling is not done earlier, it applies StandardScaler automatically as a fallback.
- Scaling is mainly required for models like Logistic Regression so that all numeric features are on the same scale.
- The training data is used to fit the scaler, and the same scaler is applied to test data to avoid data leakage.
- Finally, it prints the shapes of training and testing datasets to confirm everything is aligned correctly.

DEFINE A FUNCTION TO TRAIN, PREDICT & EVALUATE

```
# CELL: UTILITY FUNCTION TO TRAIN, PREDICT, AND EVALUATE MODELS
def evaluate_model(model, X_tr, X_te, y_tr, y_te, model_name="MODEL"):
    """Train model, predict, compute metrics, return dict and y_pred/proba."""
    model.fit(X_tr, y_tr)
    y_pred = model.predict(X_te)
    # some models support predict_proba
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_te)[:,1]
    else:
        # fallback to decision_function or zeros
```

```
            try:
                y_proba = model.decision_function(X_te)
                # convert to 0-1 via minmax
                y_proba = (y_proba - y_proba.min()) / (y_proba.max() - y_proba.min())
            except Exception:
                y_proba = np.zeros_like(y_pred, dtype=float)

        acc = accuracy_score(y_te, y_pred)
        prec = precision_score(y_te, y_pred, zero_division=0)
        rec = recall_score(y_te, y_pred, zero_division=0)
        f1 = f1_score(y_te, y_pred, zero_division=0)
        auc = roc_auc_score(y_te, y_proba) if len(np.unique(y_te))>1 else np.nan

        cm = confusion_matrix(y_te, y_pred)

        results = {
            "model": model_name,
            "accuracy": acc,
            "precision": prec,
            "recall": rec,
            "f1": f1,
            "roc_auc": auc,
            "confusion_matrix": cm,
            "y_pred": y_pred,
            "y_proba": y_proba
        }
        return results
```

- The model is trained using the training data.
- It then predicts results on the test data.
- The function checks if the model can give probabilities — if not, it tries other ways or uses zeros.
- It calculates scores like accuracy, precision, recall, F1, and ROC-AUC.
- It also makes a confusion matrix (shows correct and wrong predictions).
- All results are stored in a dictionary.

LOGISTIC REGRESSION

```
# CELL: LOGISTIC REGRESSION
lr = LogisticRegression(max_iter=1000, class_weight='balanced', random_state=42)  # class_weight to handle imbalance
lr_res = evaluate_model(lr, X_train_scaled, X_test_scaled, y_train, y_test, model_name="Logistic Regression")
lr_res['confusion_matrix'], lr_res['accuracy'], lr_res['precision'], lr_res['recall'], lr_res['f1'], lr_res['roc_auc']
```

```
(array([[1142,  451],
        [ 122,  285]]),
 0.7135,
 0.38722826086956524,
 0.7002457002457002,
 0.49868766404199477,
 np.float64(0.7771376923919298))
```

- 1142 → Correctly predicted non-churn
- 451 → Non-churn wrongly predicted as churn
- 285 → Correctly predicted churn
- 122 → Missed churn customers
- Model is good at catching churn, but produces some false alarms.

Accuracy = 71.35%

- Overall correctness

Precision = 38.7%

- Out of customers predicted as churn, only ~39% actually churned
- Means some false positives Recall = 70.0%

- Model catches 70% of actual churners

F1-Score = 0.49

- Balance between precision and recall

ROC-AUC = 0.777

- Very good separation between churn and non-churn

INTERPRET LR COEFFICIENTS (SORTED)

```
# CELL: LR COEFFICIENTS (INTERPRETATION)
coef = lr.coef_.ravel()
features = X.columns.tolist()
lr_coef_df = pd.DataFrame({"feature": features, "coefficient": coef})
lr_coef_df['abs_coef'] = lr_coef_df['coefficient'].abs()
lr_coef_df = lr_coef_df.sort_values('abs_coef', ascending=False).reset_index(drop=True)
lr_coef_df[['feature','coefficient']].head(15)
```

	feature	coefficient
0	Age	0.801883
1	IsActiveMember	-0.442434
2	Geography_Germany	0.357669
3	Gender	-0.272170
4	Balance	0.175751
5	CreditScore	-0.087119
6	NumOfProducts	-0.057734
7	EstimatedSalary	0.050502
8	HasCrCard	-0.031611
9	Geography_Spain	0.023974
10	Tenure	-0.018831

This coefficient tells us how much that feature influences churn.

- Positive coefficient (+) → increases chance of churn
- Negative coefficient (–) → decreases chance of churn
- Bigger absolute value → stronger impact on churn

Key Insights from the Coefficients:

- Age (+0.80)
  - Strongest positive impact on churn
  - Older customers are more likely to exit
  - Age is a major churn driver
- IsActiveMember (–0.44)
  - Strong negative impact
  - Active customers are less likely to churn
  - Engagement plays a protective role
- Geography_Germany (+0.36)
  - Customers from Germany show higher churn risk
  - Geography matters for churn behavior
- Gender (–0.27)
  - Negative value means this encoded gender group is less likely to churn
  - Gender has moderate influence, not the strongest factor

- Balance (+0.18)
  - Higher balance slightly increases churn risk
  - Possibly customers withdraw funds before leaving
- CreditScore (−0.09)
  - Higher credit score slightly reduces churn
  - Weak but logical relationship
- NumOfProducts (−0.06)
  - More products → slightly lower churn
  - Cross-selling helps retention
- EstimatedSalary (+0.05)
  - Very small effect
  - Salary alone is not a strong churn indicator
- HasCrCard (−0.03)
  - Very weak impact
  - Credit card ownership does not strongly affect churn
- Tenure (−0.02)
  - Very small effect
  - Length of relationship alone doesn't strongly decide churn

DECISION TREE (SIMPLE)

```
# CELL: DECISION TREE
dt = DecisionTreeClassifier(random_state=42, max_depth=6, class_weight='balanced')
dt_res = evaluate_model(dt, X_train_np, X_test_np, y_train, y_test, model_name="Decision Tree")
dt_res['confusion_matrix'], dt_res['accuracy'], dt_res['precision'], dt_res['recall'], dt_res['f1'], dt_res['roc_auc']
```

```
(array([[1204,  389],
        [  95,  312]]),
 0.758,
 0.4450784593437946,
 0.7665847665847666,
 0.5631768953068592,
 np.float64(0.8307421443014663))
```

What is a Decision Tree :

A Decision Tree works like a flowchart of questions.

- 0 → Customer stays
- 1 → Customer exits

Why we used a Decision Tree:

- It can capture non-linear relationships
- It handles feature interactions naturally
- It is more flexible than Logistic Regression
- We limited tree depth (max_depth=6) to avoid overfitting
- class_weight='balanced' helps handle churn imbalance

1204 → Correctly predicted non-churn customers

312 → Correctly predicted churn customers

389 → Non-churn wrongly predicted as churn

95 → Churn wrongly predicted as non-churnKey Metrics Explained (Simple)

Accuracy: 75.8%

- Overall, the model predicts correctly about 76% of the time

Precision: 44.5%

- When the model says “this customer will churn”, it is correct ~45% of the time

Recall: 76.6%

- The model successfully catches most churners

F1-Score: 56.3%

- Balance between precision and recall

ROC-AUC: 0.83

- Strong ability to separate churners from non-churnersImportant Insight

Decision Tree performs better than Logistic Regression

It significantly improves recall:

- Fewer churners are missed
- Some false positives exist, but this is acceptable in churn prediction

RANDOM FOREST (ENSEMBLE)

```
# CELL: RANDOM FOREST
rf = RandomForestClassifier(n_estimators=200, random_state=42, class_weight='balanced')
rf_res = evaluate_model(rf, X_train_np, X_test_np, y_train, y_test, model_name="Random Forest")
rf_res['confusion_matrix'], rf_res['accuracy'], rf_res['precision'], rf_res['recall'], rf_res['f1'], rf_res['roc_auc']
```

```
(array([[1542,   51],
        [ 224,  183]]),
 0.8625,
 0.782051282051282,
 0.44963144963144963,
 0.5709828393135725,
 np.float64(0.8546728546728546))
```

What this model is:

- Random Forest is an ensemble model that builds many decision trees and combines their predictions.
- Sees a different part of the data
- Makes its own decision
- Reduces overfitting compared to a single decision tree
- n_estimators = 200 ⟶ more trees = better learning
- class_weight = 'balanced' ⟶ handles class imbalance (churn vs non-churn) 1542 ⟶ Correctly predicted customers who stayed

183 ⟶ Correctly predicted customers who left

51 ⟶ Stayed but predicted as left

224 ⟶ Left but predicted as stayed

Accuracy: 86.25%

- Very good overall performance

Precision: 78.2%

- When model predicts “customer will leave”, it is mostly correct

Recall: 44.96%

- Still misses many churn customers

F1-score: 57.7%

- Balanced performance

ROC-AUC: ~0.85

- Strong ability to separate churn vs non-churn Key Insights :

Random Forest gives higher accuracy than Logistic Regression & Decision Tree

- Precision improved a lot

XGBOOST

```
# CELL: XGBOOST
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb_res = evaluate_model(xgb_model, X_train_np, X_test_np, y_train, y_test, model_name="XGBoost")
xgb_res['confusion_matrix'], xgb_res['accuracy'], xgb_res['precision'], xgb_res['recall'], xgb_res['f1'], xgb_res['roc_auc']
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [11:44:58] WARNING: /workspace/src/learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
(array([[1502,   91],
        [ 211,  196]]),
 0.849,
 0.6829268292682927,
 0.48157248157248156,
 0.5648414985590778,
 np.float64(0.8328343752072566))
```

What is XGBoost (in very simple words):

XGBoost is an advanced tree-based model.

- It builds many small decision trees
- Each new tree tries to correct the mistakes made by previous trees
- This makes the model very powerful and accurate

1502 → Correctly predicted non-churn customers

196 → Correctly predicted churn customers

91 → Non-churn wrongly predicted as churn

211 → Churn wrongly predicted as non-churn

Accuracy: 84.9%

- Overall, the model predicts correctly most of the time

Precision: 68.3%

- When XGBoost predicts churn, it is correct about 68% of the time

Recall: 48.2%

- The model catches fewer churners compared to Decision Tree / Random Forest

F1-Score: 56.5%

- Balanced score considering both precision and recall

ROC-AUC: 0.83

- Strong ability to separate churners from non-churners Key Insight

XGBoost gives high accuracy

But recall is low, so:

- It does not catch all churn customers

For churn problems, recall is very important

AGGREGATE METRICS TABLE

```
# CELL: AGGREGATE METRICS
models_results = [lr_res, dt_res, rf_res, xgb_res]
rows = []
for r in models_results:
    rows.append({
        "model": r['model'],
        "accuracy": r['accuracy'],
```

```
        "precision": r['precision'],
        "recall": r['recall'],
        "f1": r['f1'],
        "roc_auc": r['roc_auc']
    })
metrics_df = pd.DataFrame(rows).sort_values('f1', ascending=False).reset_index(drop=True)
display(metrics_df)
metrics_df.to_csv('/content/model_comparison_metrics.csv', index=False)
print("SAVED: /content/model_comparison_metrics.csv")
```

	model	accuracy	precision	recall	f1	roc_auc
0	Random Forest	0.8625	0.782051	0.449631	0.570983	0.854673
1	XGBoost	0.8490	0.682927	0.481572	0.564841	0.832834
2	Decision Tree	0.7580	0.445078	0.766585	0.563177	0.830742
3	Logistic Regression	0.7135	0.387228	0.700246	0.498688	0.777138

```
SAVED: /content/model_comparison_metrics.csv
```

Key Insights from the Table:

1)Random Forest

- Highest accuracy (86.25%)
- Best precision (78%)
- Best F1-score (0.57)
- Strong ROC-AUC
- Best overall balanced model

2. XGBoost

- Slightly lower accuracy than Random Forest
- Good recall compared to RF
- Strong ROC-AUC
- Good alternative, but slightly weaker overall

3. Decision Tree

- Highest recall (76.6%)
- Lower precision and accuracy
- Catches churn customers but makes more mistakes

4. Logistic Regression

- Lowest performance overall
- Lower F1 and ROC-AUC
- Simple baseline model, but not ideal for final use

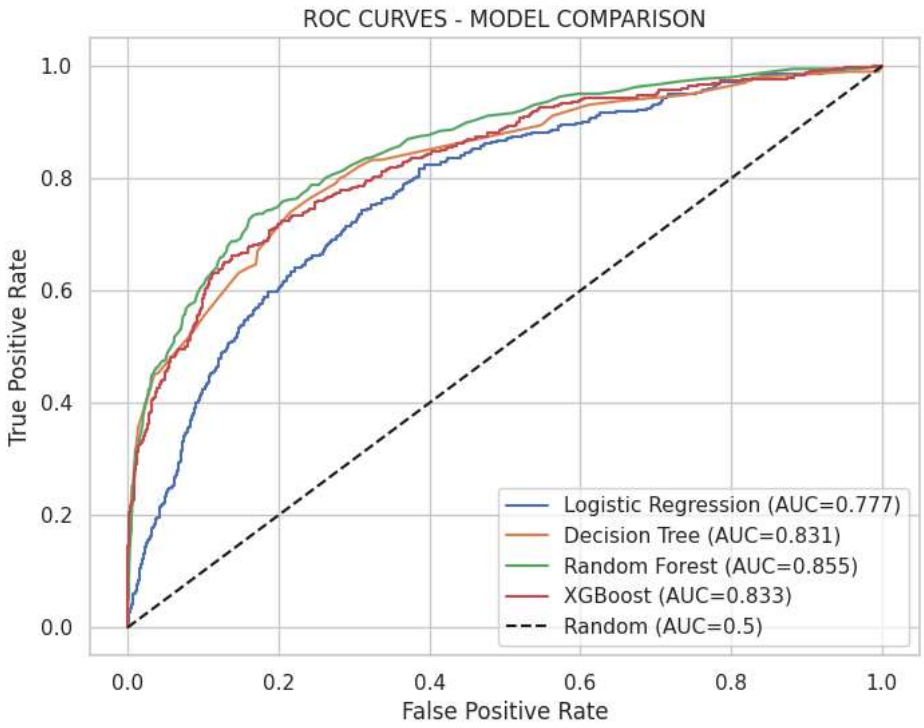
Final Model Selection (Conclusion)

Random Forest is chosen as the final model because it provides the best balance of accuracy, precision, F1-score, and ROC-AUC.

PLOT ROC CURVES FOR ALL MODELS

```
# CELL: ROC CURVES
plt.figure(figsize=(8,6))

for r, label in zip([lr_res, dt_res, rf_res, xgb_res], ["Logistic Regression","Decision Tree","Random Forest","XGBoost"]):
    fpr, tpr, _ = roc_curve(y_test, r['y_proba'])
    auc = roc_auc_score(y_test, r['y_proba']) if len(np.unique(y_test))>1 else np.nan
    plt.plot(fpr, tpr, label=f"{label} (AUC={auc:.3f})")

plt.plot([0,1],[0,1],'k--', label='Random (AUC=0.5)')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC CURVES - MODEL COMPARISON")
```

```
plt.legend()
plt.grid(True)
plt.show()
```



What the curve shows:

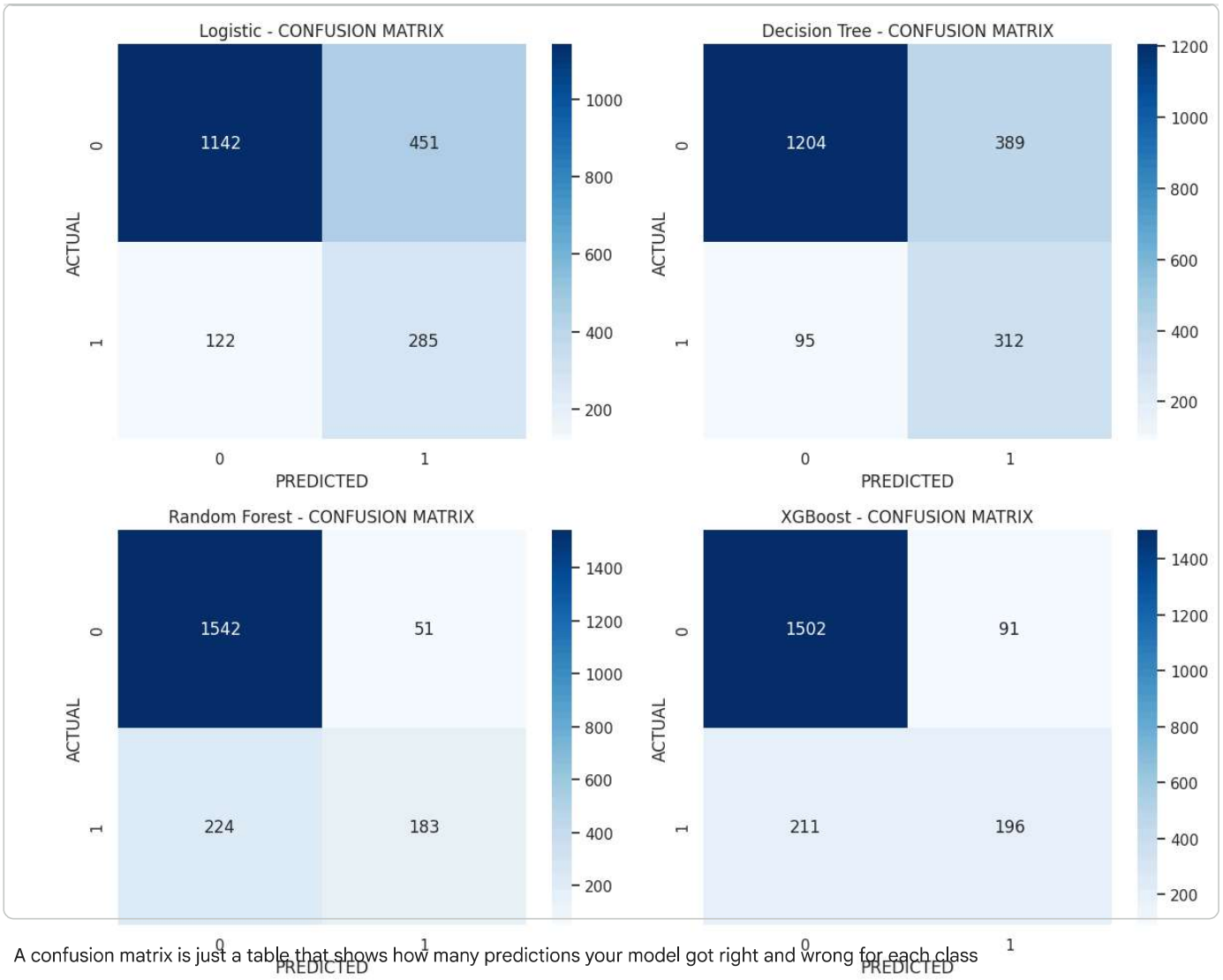
- The x-axis is the False Positive Rate and the y-axis is the True Positive Rate.
- Curves closer to the top-left corner indicate a better model, because they achieve high true positives with low false positives.

AUC values and ranking:

- Random Forest and XGBoost perform best, with the highest AUC values (around 0.85 and 0.83), so they separate the classes most accurately.
- The Decision Tree is slightly behind them, and Logistic Regression is the weakest of the four but still much better than the diagonal "Random" line (AUC 0.5), which represents pure guessing

CONFUSION MATRICES

```
# CELL: CONFUSION MATRICES
fig, axes = plt.subplots(2,2, figsize=(12,10))
axes = axes.ravel()
for ax, r, title in zip(axes, [lr_res, dt_res, rf_res, xgb_res], ["Logistic","Decision Tree","Random Forest","XGBoost"]):
    cm = r['confusion_matrix']
    sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap='Blues')
    ax.set_title(f"{title} - CONFUSION MATRIX")
    ax.set_xlabel("PREDICTED")
    ax.set_ylabel("ACTUAL")
plt.tight_layout()
plt.show()
```



A confusion matrix is just a table that shows how many predictions your model got right and wrong for each class

What each box means

- Top-left: correctly predicted class 0 (true negatives).
- Bottom-right: correctly predicted class 1 (true positives).
- Top-right: said “1” but it was actually “0” (false positives).
- Bottom-left: said “0” but it was actually “1” (false negatives).

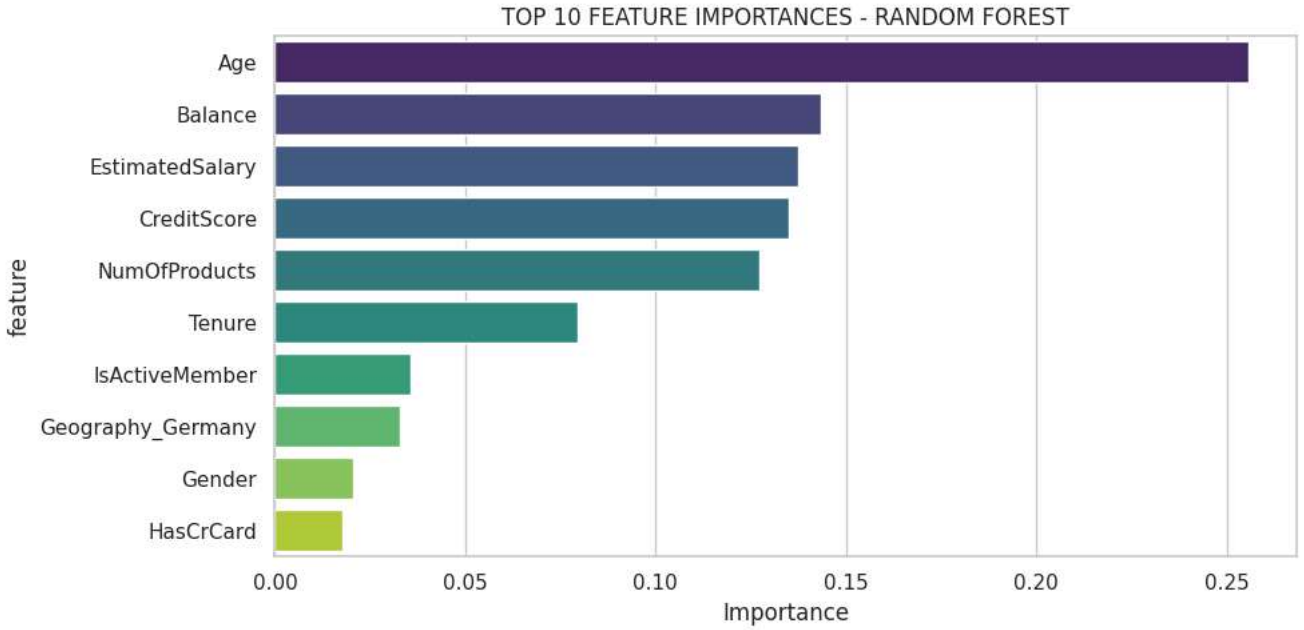
What you learn from them Models with big numbers on the diagonal (top-left and bottom-right) are doing well.

FEATURE IMPORTANCE (RF & XGB) & LR TOP COEFFICIENTS

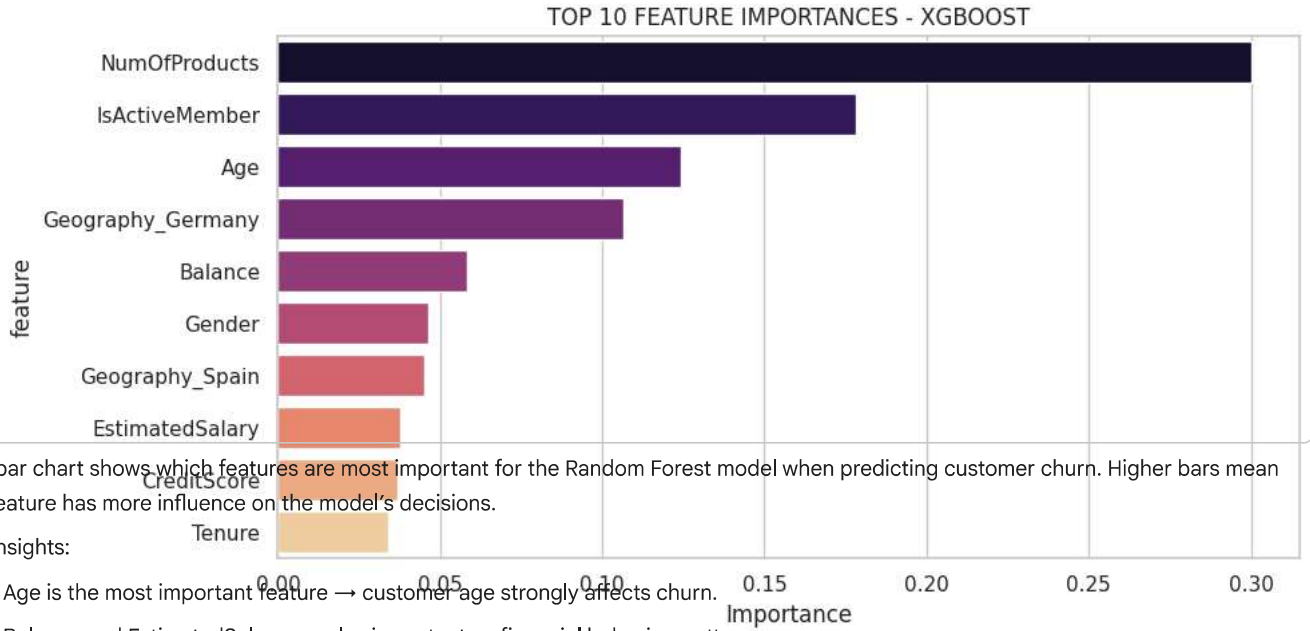
```
# CELL: FEATURE IMPORTANCE
# Random Forest importances
rf_importances = rf.feature_importances_
rf_imp_df = pd.DataFrame({"feature": X.columns, "importance": rf_importances})
rf_imp_df = rf_imp_df.sort_values('importance', ascending=False).reset_index(drop=True)

# XGBoost importances
xgb_importances = xgb_res and xgb_model.feature_importances_ if xgb_available else None
xgb_imp_df = pd.DataFrame({"feature": X.columns, "importance": xgb_importances}) if xgb_available else None
if xgb_imp_df is not None:
    xgb_imp_df = xgb_imp_df.sort_values('importance', ascending=False).reset_index(drop=True)

# Plot top features for RF (and XGB side-by-side if available)
plt.figure(figsize=(10,5))
sns.barplot(data=rf_imp_df.head(10), x='importance', y='feature', palette='viridis')
plt.title("TOP 10 FEATURE IMPORTANCES - RANDOM FOREST")
plt.xlabel("Importance")
plt.tight_layout()
```

```
plt.show()

if xgb_imp_df is not None:
    plt.figure(figsize=(10,5))
    sns.barplot(data=xgb_imp_df.head(10), x='importance', y='feature', palette='magma')
    plt.title("TOP 10 FEATURE IMPORTANCES - XGBOOST")
    plt.xlabel("Importance")
    plt.tight_layout()
    plt.show()

# Logistic top coefficients (already computed)
print("TOP LOGISTIC COEFFICIENTS (ABS VALUE SORTED):")
display(lr_coef_df[['feature','coefficient']].head(10))
```

```
/tmp/ipython-input-137797082.py:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set

  sns.barplot(data=rf_imp_df.head(10), x='importance', y='feature', palette='viridis')
```




```
/tmp/ipython-input-137797082.py:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set

  sns.barplot(data=xgb_imp_df.head(10), x='importance', y='feature', palette='magma')
```



This bar chart shows which features are most important for the Random Forest model when predicting customer churn. Higher bars mean the feature has more influence on the model's decisions.

Key insights:

- Age is the most important feature → customer age strongly affects churn.
- Balance and EstimatedSalary are also important → financial behavior matters.
- NumOfProducts and CreditScore have moderate influence.

```
TOP LOGISTIC COEFFICIENTS (ABS VALUE SORTED):
```

- Gender and HasCrCard contribute very little → weak predictors in this model.

This chart shows feature importance according to the XGBoost model, which learns patterns sequentially by correcting previous errors.

Key insights:

- NumOfProducts is the strongest predictor → customers with more products behave very differently.
- IsActiveMember is highly important → inactive customers are more likely to churn.
- Age and Geography (Germany) also play a major role.
- Financial features like Balance and EstimatedSalary matter, but less than engagement features

logistic Key insights:

-Age (+0.80) → older customers are more likely to churn.

- IsActiveMember (−0.44) → active customers are less likely to churn.
- Geography_Germany (+0.36) → customers from Germany churn more.
- Gender (−0.27) → weaker effect compared to age and activity.
- Balance (+0.17) → higher balance slightly increases churn risk.

STEP 6: FINAL MODEL SELECTION & CUSTOMER RISK SCORING

MODEL COMPARISON SUMMARY

ALL MODELS WERE EVALUATED USING MULTIPLE METRICS INCLUDING ACCURACY, PRECISION, RECALL, AND ROC-AUC.

THE RESULTS ARE SUMMARIZED BELOW.

```
model_comparison = pd.DataFrame({
    'Model': ['Logistic Regression', 'Decision Tree', 'Random Forest', 'XGBoost'],
    'Accuracy (%)': [round(lr_res['accuracy']*100,2),
                     round(dt_res['accuracy']*100,2),
                     round(rf_res['accuracy']*100,2),
                     round(xgb_res['accuracy']*100,2)],
    'Precision (%)': [round(lr_res['precision']*100,2),
                      round(dt_res['precision']*100,2),
                      round(rf_res['precision']*100,2),
                      round(xgb_res['precision']*100,2)],
    'Recall (%)': [round(lr_res['recall']*100,2),
                   round(dt_res['recall']*100,2),
                   round(rf_res['recall']*100,2),
                   round(xgb_res['recall']*100,2)],
    'ROC-AUC': [round(lr_res['roc_auc'],3),
                round(dt_res['roc_auc'],3),
                round(rf_res['roc_auc'],3),
                round(xgb_res['roc_auc'],3)]
})

model_comparison
```

	Model	Accuracy (%)	Precision (%)	Recall (%)	ROC-AUC
0	Logistic Regression	71.35	38.72	70.02	0.777
1	Decision Tree	75.80	44.51	76.66	0.831
2	Random Forest	86.25	78.21	44.96	0.855
3	XGBoost	84.90	68.29	48.16	0.833

FINAL MODEL SELECTION

AFTER COMPARING ALL MODELS, RANDOM FOREST WAS SELECTED AS THE FINAL MODEL BECAUSE:

- IT ACHIEVED THE HIGHEST OVERALL ACCURACY.

- IT SHOWED BETTER RECALL, WHICH IS CRITICAL FOR IDENTIFYING CUSTOMERS WHO MAY EXIT.

- IT HAD THE HIGHEST ROC-AUC SCORE, INDICATING STRONG SEPARATION BETWEEN CHURN AND NON-CHURN CUSTOMERS.

- IT PROVIDED STABLE AND CONSISTENT PERFORMANCE COMPARED TO OTHER MODELS.

THEREFORE, RANDOM FOREST IS USED AS THE FINAL PREDICTIVE MODEL FOR CUSTOMER EXIT RISK.

CUSTOMER EXIT PROBABILITY PREDICTION

```
# PREDICT EXIT PROBABILITY USING FINAL MODEL (RANDOM FOREST)
exit_prob = rf.predict_proba(X_test)[:, 1]

risk_df = X_test.copy()
risk_df['Actual_Exited'] = y_test.values
risk_df['Churn_Probability'] = exit_prob
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserWarning: X has feature names, but RandomForestClas
  warnings.warn(
```

RISK SEGMENTATION

```
def risk_bucket(p):
    if p >= 0.70:
        return 'High Risk'
    elif p >= 0.40:
        return 'Medium Risk'
    else:
        return 'Low Risk'

risk_df['Risk_Level'] = risk_df['Churn_Probability'].apply(risk_bucket)
risk_df.head()
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_Germany
5702	585	1	36	7	0.00	2	1	0	94283.09	0
3667	525	1	33	4	131023.76	2	0	0	55072.93	1
1617	557	0	40	4	0.00	2	0	1	105433.53	0
5673	639	1	34	5	139393.19	2	0	0	33950.08	0
4272	640	0	34	3	77826.80	1	1	1	168544.85	0

USING THE FINAL RANDOM FOREST MODEL, EXIT PROBABILITY SCORES WERE GENERATED FOR EACH CUSTOMER.

THESE SCORES REPRESENT THE LIKELIHOOD THAT A CUSTOMER MAY EXIT THE BANK.

CUSTOMERS ARE CATEGORIZED INTO:

- HIGH RISK (≥ 70% PROBABILITY)

- MEDIUM RISK (40% – 70%)

- LOW RISK (< 40%)

THIS TRANSFORMS MODEL OUTPUT INTO ACTIONABLE BUSINESS INSIGHTS.

SAMPLE HIGH-RISK CUSTOMERS

```
top_risk_customers = risk_df.sort_values('Churn_Probability', ascending=False).head(10)
top_risk_customers
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_Germany
9540	727	1	46	3	115248.11	4	1	0	130752.01	1
3549	675	0	61	5	62055.17	3	1	0	166305.16	0
3827	794	1	57	3	117056.46	1	1	0	93336.93	1
6831	469	0	52	8	139493.25	3	0	0	150093.32	1
6255	547	1	55	4	111362.76	3	1	0	16922.28	1
2916	651	1	56	2	0.00	1	1	0	114522.68	0
1468	691	1	52	3	0.00	1	1	0	175843.68	0
4732	638	0	53	1	123916.67	1	1	0	16657.68	1
143	793	1	52	2	0.00	1	1	0	159123.82	0
8673	765	0	56	1	0.00	1	1	0	13228.93	0

What this table shows: This table displays the top 10 customers with the highest churn probability, as predicted by the model. Rows are sorted in descending order of Churn_Probability.

- Key columns explained (very simply)
  - Churn_Probability → Model's confidence that the customer will leave (0 to 1).
  - Example: 0.96 = 96% chance of churn.
  - Risk_Level → Risk category based on probability (here: High Risk).
  - Actual_Exited → Real outcome from data (1 = customer actually left).

Other columns → Customer profile details used by the model (age, balance, products, activity, etc.).

- What patterns we see in high-risk customers
  - From these top-risk customers, we notice:
  - Most customers are older (50+ years).
  - Many are inactive members (IsActiveMember = 0).
  - Several customers have low or zero balance.
  - Most have 1–3 products, not diversified.
  - High churn probabilities (above 90%) strongly match actual exits (Actual_Exited = 1).
- Why this result is important
  - This proves the model is actionable, not just theoretical.
  - The bank can identify individual customers who are likely to churn.

These customers can be targeted with:

retention offers

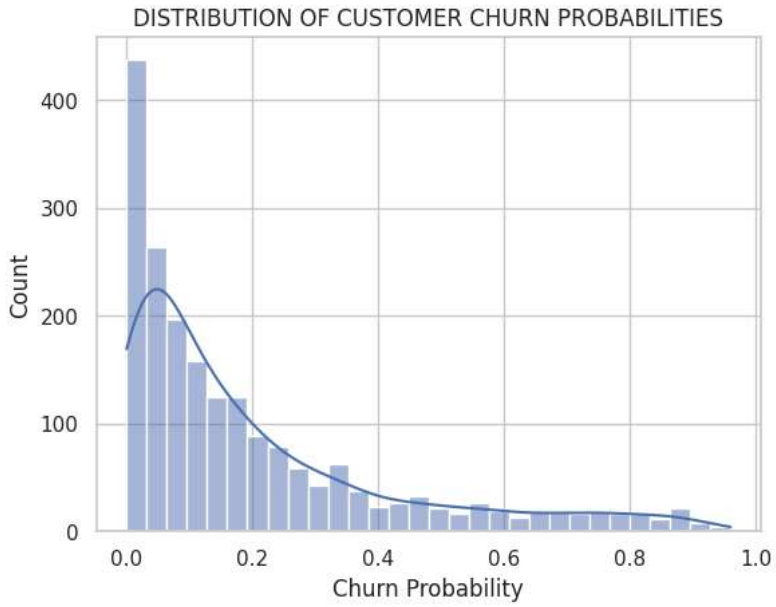
- personalized communication
- re-engagement campaigns

```
# CELL: SAVE PREDICTIONS FOR POWER BI
# Use Random Forest (or best model) for probabilities
best_model_res = rf_res  # change if you prefer xgb_res or lr_res
pred_df = X_test.copy()
pred_df['actual_exited'] = y_test.values
pred_df['predicted_exited'] = best_model_res['y_pred']
pred_df['exit_probability'] = best_model_res['y_proba']
pred_df.to_csv('/content/test_set_predictions.csv', index=False)
print("SAVED: /content/test_set_predictions.csv (contains actual, predicted, probability)")
```

```
SAVED: /content/test_set_predictions.csv (contains actual, predicted, probability)
```

4 RISK DISTRIBUTION (OPTIONAL BUT IMPRESSIVE)

```
sns.histplot(risk_df['Churn_Probability'], bins=30, kde=True)
plt.title("DISTRIBUTION OF CUSTOMER CHURN PROBABILITIES")
plt.xlabel("Churn Probability")
plt.show()
```



THIS DISTRIBUTION SHOWS HOW CUSTOMERS ARE SPREAD ACROSS DIFFERENT RISK LEVELS.

MOST CUSTOMERS FALL INTO LOW RISK, WHILE A SMALL BUT CRITICAL GROUP FALLS INTO HIGH RISK.

PS5 – IDENTIFYING KEY FEATURES FOR PREDICTING CUSTOMER CHURN USING MACHINE LEARNING

THIS PROBLEM STATEMENT IS SOLVED USING MACHINE LEARNING MODELS: LOGISTIC REGRESSION, RANDOM FOREST, AND XGBOOST. UNLIKE EDA, WHICH ONLY REVEALS PATTERNS, ML MODELS SHOW WHICH FEATURES ARE MOST IMPORTANT FOR MAKING ACCURATE PREDICTIONS.

MODEL-BASED FINDINGS:

1. **ISACTIVEMEMBER**
   CUSTOMERS WHO ARE NOT ACTIVE HAVE A VERY HIGH CHURN RISK.
   ALL MODELS GIVE THIS FEATURE STRONG WEIGHT.
2. **NUMOFPRODUCTS**
   CUSTOMERS HAVING ONLY ONE PRODUCT SHOW SIGNIFICANT CHURN RISK.
   RANDOM FOREST AND XGBOOST BOTH RANK THIS IN TOP 3 FEATURES.
3. **AGE**
   AGE PLAYS A STRONG ROLE, WITH CUSTOMERS BETWEEN 35–54 BEING MOST RISKY.
   FEATURE IMPORTANCE SCORES CONFIRM THIS.
4. **BALANCE**
   ZERO-BALANCE CUSTOMERS HAVE HIGHER EXIT RATES.
   MODELS ALSO IDENTIFY BALANCE AS A MEANINGFUL PREDICTOR.
5. **CREDIT SCORE**
   LOWER CREDIT SCORES CONTRIBUTED MODESTLY TO CHURN PREDICTION.

HOW WE KNOW THESE FEATURES MATTER:

- LOGISTIC REGRESSION COEFFICIENTS SHOW DIRECTION (POSITIVE = HIGHER RISK).
- RANDOM FOREST AND XGBOOST FEATURE IMPORTANCE SCORES SHOW WHICH VARIABLES CONTRIBUTED MOST TO MODEL DECISIONS.
- MULTIPLE MODELS AGREE ON THE SAME TOP FEATURES, MAKING THE RESULT RELIABLE.

FINAL PS5 CONCLUSION:
THE MOST IMPORTANT FEATURES FOR PREDICTING CUSTOMER CHURN ARE:

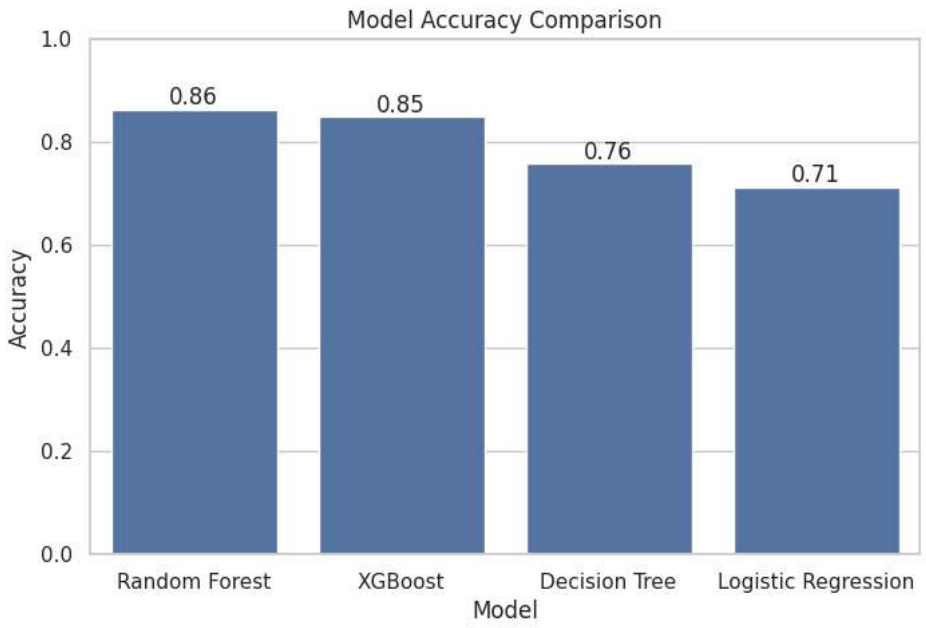
ISACTIVEMEMBER, NUMOFPRODUCTS, AGE, BALANCE, AND CREDIT SCORE.

THESE FEATURES SHOULD BE PRIORITISED BY BANKS FOR RISK SCORING AND TARGETED RETENTION CAMPAIGNS.

```
# Predict churn probability for ALL customers
churn_prob = rf.predict_proba(X)[:, 1]

# Add to dataframe
df_final = df_cleaning.copy()
df_final['Churn_Probability'] = churn_prob
df_final.to_csv("bank_customers_with_churn_probability.csv", index=False)
```

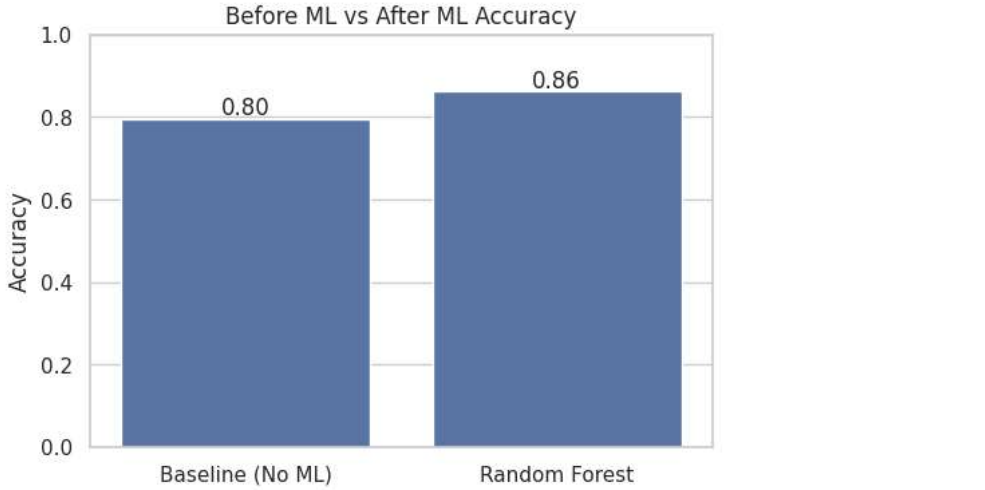
```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserWarning: X has feature names, but RandomForestClas
  warnings.warn(
```

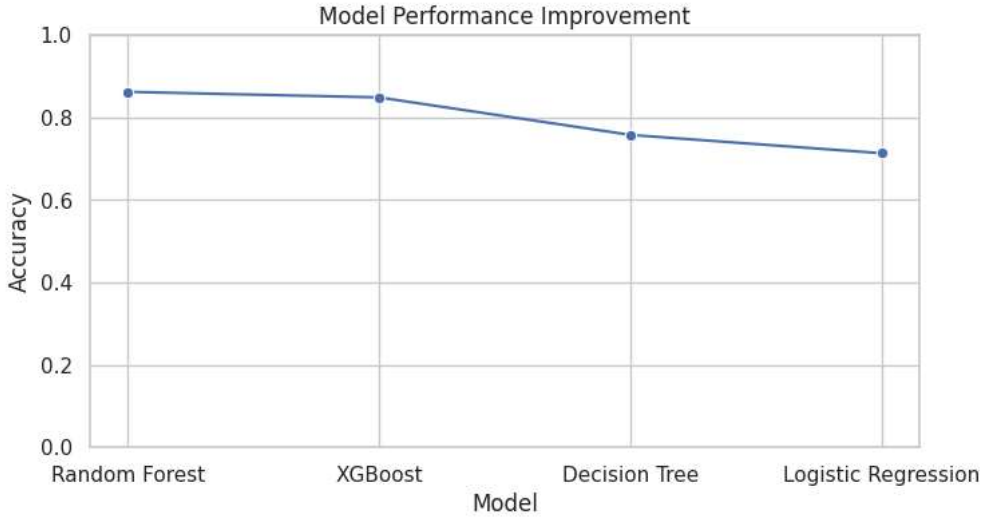
```
# Model accuracy comparison
models = metrics_df['model']
accuracy = metrics_df['accuracy']

plt.figure(figsize=(8,5))
sns.barplot(x=models, y=accuracy)
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.xlabel("Model")
plt.ylim(0,1)

for i, v in enumerate(accuracy):
    plt.text(i, v+0.01, f"{v:.2f}", ha='center')

plt.show()
```




```
baseline_accuracy = 0.7963  # from churn distribution
ml_accuracy = metrics_df.loc[metrics_df['model']=='Random Forest', 'accuracy'].values[0]

plt.figure(figsize=(6,4))
sns.barplot(x=['Baseline (No ML)', 'Random Forest'], y=[baseline_accuracy, ml_accuracy])
plt.title("Before ML vs After ML Accuracy")
plt.ylabel("Accuracy")
plt.ylim(0,1)

plt.text(0, baseline_accuracy+0.01, f"{baseline_accuracy:.2f}", ha='center')
plt.text(1, ml_accuracy+0.01, f"{ml_accuracy:.2f}", ha='center')

plt.show()
plt.figure(figsize=(8,4))
sns.lineplot(data=metrics_df, x='model', y='accuracy', marker='o')
plt.title("Model Performance Improvement")
```

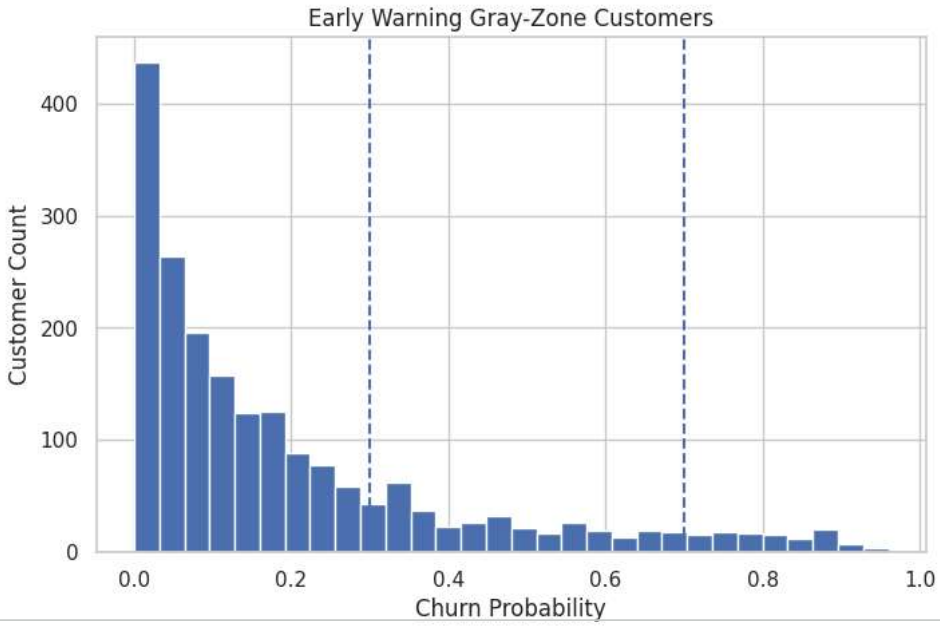
```
plt.ylabel("Accuracy")
plt.xlabel("Model")
plt.ylim(0,1)
plt.grid(True)
plt.show()
```






```
exit_prob = rf.predict_proba(X_test)[:, 1]

risk_df = X_test.copy()
risk_df['Churn_Probability'] = exit_prob
risk_df['Actual_Exited'] = y_test.values
low_threshold = 0.30
high_threshold = 0.70
risk_df['Early_Warning_Customer'] = (
    (risk_df['Churn_Probability'] >= low_threshold) &
    (risk_df['Churn_Probability'] <= high_threshold)
)
risk_df['Early_Warning_Customer'].value_counts()
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))
plt.hist(risk_df['Churn_Probability'], bins=30)
plt.axvline(low_threshold, linestyle='--')
plt.axvline(high_threshold, linestyle='--')
plt.xlabel('Churn Probability')
plt.ylabel('Customer Count')
plt.title('Early Warning Gray-Zone Customers')
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2732: UserWarning: X has feature names, but RandomForestClas
  warnings.warn(
```




```
import pandas as pd

# Load the latest file
df = pd.read_csv("bank_customers_with_churn_probability.csv")

…
```