```
# Upload Dataset
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 Churn_Modelling.csv to Churn_Modelling (2).csv
```

```
#Load Dataset
import pandas as pd
df = pd.read_csv('Churn_Modelling.csv')
```

```
# Data Exploration
print(df.info())
print(df.describe())
print(df['Exited'].value_counts())
```

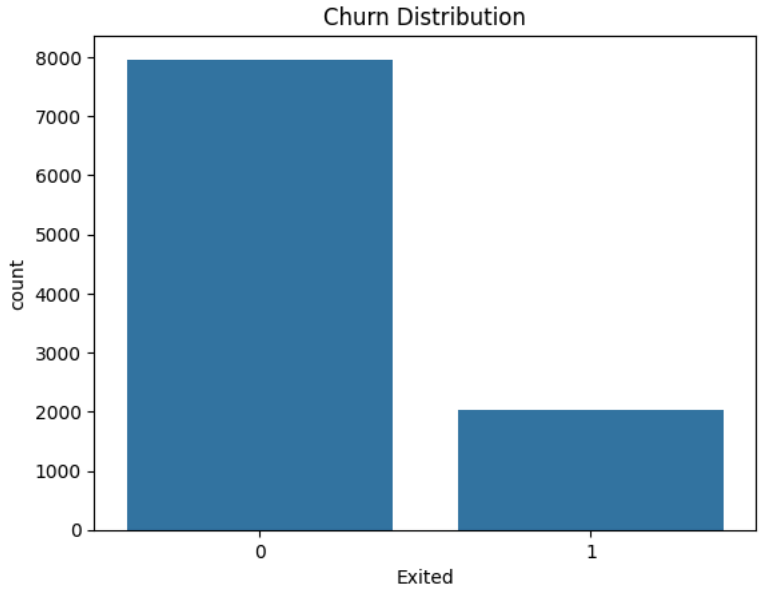
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          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
None
         RowNumber    CustomerId   CreditScore           Age        Tenure  \
count  10000.00000  1.000000e+04  10000.000000  10000.000000  10000.000000
mean    5000.50000  1.569094e+07    650.528800     38.921800      5.012800
std     2886.89568  7.193619e+04     96.653299     10.487806      2.892174
min        1.00000  1.556570e+07    350.000000     18.000000      0.000000
25%     2500.75000  1.562853e+07    584.000000     32.000000      3.000000
50%     5000.50000  1.569074e+07    652.000000     37.000000      5.000000
75%     7500.25000  1.575323e+07    718.000000     44.000000      7.000000
max    10000.00000  1.581569e+07    850.000000     92.000000     10.000000

             Balance  NumOfProducts    HasCrCard  IsActiveMember  \
count   10000.000000   10000.000000  10000.00000    10000.000000
mean    76485.889288       1.530200      0.70550        0.515100
std     62397.405202       0.581654      0.45584        0.499797
min         0.000000       1.000000      0.00000        0.000000
25%         0.000000       1.000000      0.00000        0.000000
50%     97198.540000       1.000000      1.00000        1.000000
75%    127644.240000       2.000000      1.00000        1.000000
max    250898.090000       4.000000      1.00000        1.000000

       EstimatedSalary        Exited
count     10000.000000  10000.000000
mean     100090.239881      0.203700
std       57510.492818      0.402769
min          11.580000      0.000000
25%       51002.110000      0.000000
50%      100193.915000      0.000000
75%      149388.247500      0.000000
max      199992.480000      1.000000
Exited
0    7963
1    2037
Name: count, dtype: int64
```

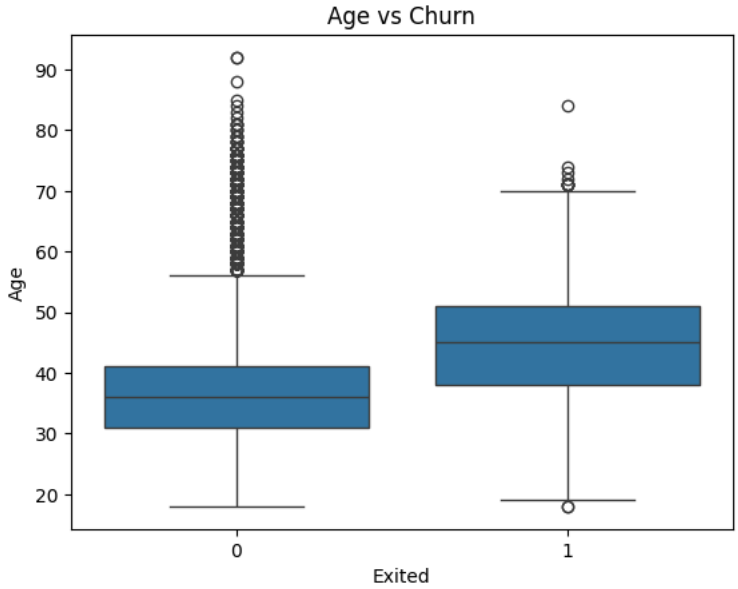
```
# Check for Missing Values and Duplicates
print("Missing Values:\n", df.isnull().sum())
print("Duplicate Rows:", df.duplicated().sum())
```

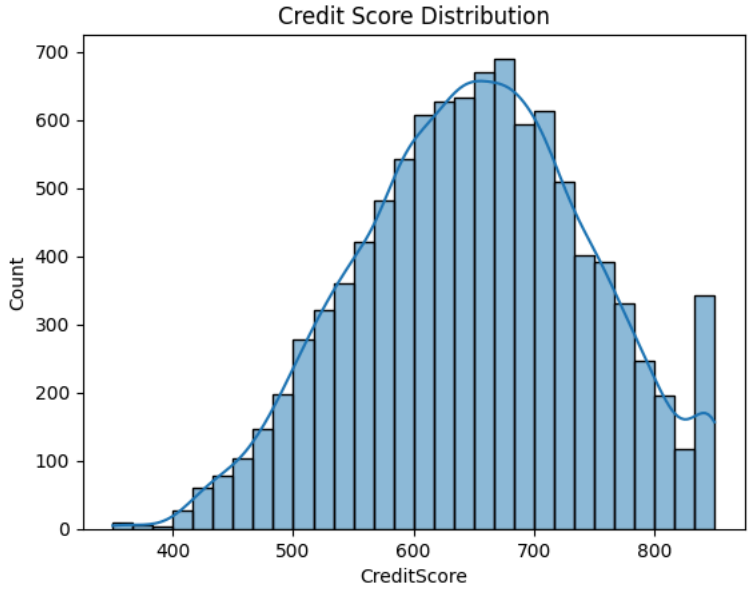
```
Missing Values:
 RowNumber          0
CustomerId         0
```

```
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
Duplicate Rows: 0
```

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

sns.countplot(x='Exited', data=df)
plt.title("Churn Distribution")
plt.show()

sns.boxplot(x='Exited', y='Age', data=df)
plt.title("Age vs Churn")
plt.show()

sns.histplot(df['CreditScore'], bins=30, kde=True)
plt.title("Credit Score Distribution")
plt.show()
```








```
#  Preprocessing
X = df.drop(columns=['Exited', 'RowNumber', 'CustomerId', 'Surname'])  # Keep Geography!
y = df['Exited']

# Encode Gender
```

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

# One-Hot Encode Geography
X = pd.get_dummies(X, columns=['Geography'], drop_first=True)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Train-Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```
#  Model Building
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
#  Evaluation
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
y_pred = model.predict(X_test)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Confusion Matrix:
 [[1552   55]
 [ 213  180]]
Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.97      0.92      1607
           1       0.77      0.46      0.57       393

    accuracy                           0.87      2000
   macro avg       0.82      0.71      0.75      2000
weighted avg       0.86      0.87      0.85      2000

Accuracy: 0.866
```

```
# Single Prediction
new_data = {
    'CreditScore': [650],
    'Geography': ['France'],
    'Gender': ['Female'],
    'Age': [40],
    'Tenure': [3],
    'Balance': [60000],
    'NumOfProducts': [2],
    'HasCrCard': [1],
    'IsActiveMember': [1],
    'EstimatedSalary': [50000]
}
new_df = pd.DataFrame(new_data)

# Match preprocessing
new_df['Gender'] = new_df['Gender'].map({'Male': 1, 'Female': 0})
new_df = pd.get_dummies(new_df, columns=['Geography'])

# Add missing columns
for col in X.columns:
    if col not in new_df.columns:
        new_df[col] = 0
new_df = new_df[X.columns]

# Scale and predict
new_scaled = scaler.transform(new_df)
prediction = model.predict(new_scaled)
print("Churn Prediction:", "Yes" if prediction[0] == 1 else "No")
```

```
Churn Prediction: No
```

```
#  Gradio App
!pip install gradio
import gradio as gr

def predict_churn(CreditScore, Geography, Gender, Age, Tenure, Balance,
                  NumOfProducts, HasCrCard, IsActiveMember, EstimatedSalary):
    input_data = {
        'CreditScore': [CreditScore],
        'Geography': [Geography],
        'Gender': [Gender],
        'Age': [Age],
        'Tenure': [Tenure],
        'Balance': [Balance],
        'NumOfProducts': [NumOfProducts],
        'HasCrCard': [HasCrCard],
        'IsActiveMember': [IsActiveMember],
        'EstimatedSalary': [EstimatedSalary]
    }
    input_df = pd.DataFrame(input_data)
    input_df['Gender'] = input_df['Gender'].map({'Male': 1, 'Female': 0})
    input_df = pd.get_dummies(input_df, columns=['Geography'])

    for col in X.columns:
        if col not in input_df.columns:
            input_df[col] = 0
    input_df = input_df[X.columns]

    input_scaled = scaler.transform(input_df)
    result = model.predict(input_scaled)
    return "Customer is likely to churn" if result[0] == 1 else "Customer is likely to stay"

# Gradio Interface
interface = gr.Interface(
    fn=predict_churn,
    inputs=[
        gr.Number(label="Credit Score"),
        gr.Radio(["France", "Germany", "Spain"], label="Geography"),
        gr.Radio(["Male", "Female"], label="Gender"),
        gr.Number(label="Age"),
        gr.Number(label="Tenure"),
        gr.Number(label="Balance"),
        gr.Number(label="Number of Products"),
        gr.Radio([1, 0], label="Has Credit Card (1 = Yes, 0 = No)"),
        gr.Radio([1, 0], label="Is Active Member (1 = Yes, 0 = No)"),
        gr.Number(label="Estimated Salary")
    ],
    outputs="text",
    title="Customer Churn Predictor"
)

interface.launch(share=True)
```