# main.py

from database.mysqldb import connect\_db, display\_all\_customers, insert\_data\_from\_csv

from scripts.data\_loader import load\_csv\_data

from scripts.data\_cleaner import clean\_data

from scripts.eda import plot\_churn\_distribution, plot\_age\_distribution, plot\_correlation\_matrix

from scripts.feature\_engineering import one\_hot\_encode, scale\_features

from scripts.model\_training import train\_model

import os

if \_\_name\_\_ == "\_\_main\_\_":

    file\_path = "data/raw/exl\_credit\_card\_churn\_data.csv"

    #SQL

    connect\_db()

    # insert\_data\_from\_csv(file\_path)

    display\_all\_customers()

    # 1. Load raw data

    df = load\_csv\_data(file\_path)

    if df.empty:

        print("Failed to load data. Exiting.")

        exit()

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_DataFrame shape:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n", df.shape)

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Data types:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n", df.dtypes)

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Head of data:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n", df.head())

    # 2. Clean data

    df\_clean = clean\_data(df)

    # Save cleaned data to processed folder

    df\_clean.to\_csv("data/processed/churn\_cleaned.csv", index=False)

    print("Cleaned data saved to data/processed/churn\_cleaned.csv")

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_CLEANED DATA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

    print(df\_clean.head())

    # 3. Run EDA and save plots

    os.makedirs("feature/eda", exist\_ok=True)

    plot\_churn\_distribution(df\_clean)

    plot\_age\_distribution(df\_clean)

    plot\_correlation\_matrix(df\_clean)

    # 4. Encode and scale

    df\_encoded = one\_hot\_encode(df\_clean)

    df\_encoded.to\_csv("data/processed/churn\_encoded.csv", index=False)

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_DATA AFTER ONE HOT ENCODING\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

    print(df\_encoded.head())

    numeric\_cols = ['Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']

    df\_scaled = scale\_features(df\_encoded, numeric\_cols)

    df\_scaled.to\_csv("data/processed/churn\_scaled.csv", index=False)

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_DATA AFTER SCALING\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

    print(df\_scaled.head())

    # Check class distribution

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_Class distribution after scaling:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

    print(df\_scaled['Churn'].value\_counts(normalize=True))

    models\_to\_train = ['knn', 'random\_forest', 'logistic\_regression']

    for model\_name in models\_to\_train:

        print(f"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Training {model\_name.upper()} model \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

        model = train\_model(df\_scaled, model\_name)

    print(" Pipeline complete.")

import mysql.connector

def connect\_db():

    return mysql.connector.connect(

        host="localhost",

        user="root",

        password="!@Qwerty25",

        database="churn\_db"

    )

import csv

def insert\_data\_from\_csv(file\_path):

    conn = connect\_db()

    cursor = conn.cursor()

    with open(file\_path, newline='') as csvfile:

        reader = csv.DictReader(csvfile)

        count = 0  # initialize a counter

        for row in reader:

            if count >= 20:

                break  # stop after 20 rows

            query = """

                INSERT INTO customers

                (CustomerID, Gender, Age, Tenure, Balance, NumOfProducts, HasCrCard, IsActiveMember, EstimatedSalary, Churn)

                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

            """

            values = (

                row['CustomerID'],

                row['Gender'] if row['Gender'] else None,

                int(row['Age']) if row['Age'] else None,

                int(row['Tenure']) if row['Tenure'] else None,

                float(row['Balance']) if row['Balance'] else 0.0,

                int(row['NumOfProducts']) if row['NumOfProducts'] else None,

                1 if row['HasCrCard'].strip().lower() == 'yes' else 0,

                1 if row['IsActiveMember'].strip().lower() == 'yes' else 0,

                float(row['EstimatedSalary']) if row['EstimatedSalary'] else 0.0,

                1 if row['Churn'].strip().lower() in ['yes', 'maybe'] else 0

            )

            cursor.execute(query, values)

            count += 1  # increment the counter

    conn.commit()

    cursor.close()

    conn.close()

def display\_all\_customers():

    conn = connect\_db()

    cursor = conn.cursor()

    # cursor.execute("SELECT \* FROM customers")

    # results = cursor.fetchall()

    # for row in results:

    #     print(row)

    query = "SELECT \* FROM customers LIMIT 20"

    cursor.execute(query)

    results = cursor.fetchall()

    # Print column headers

    column\_names = [i[0] for i in cursor.description]

    print("\t".join(column\_names))

    # Print each row

    print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_DB DATA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

    for row in results:

        print("\t".join(str(item) for item in row))

    cursor.close()

    conn.close()

import pandas as pd

# Load credit card churn data from a CSV file into a Pandas DataFrame.

def load\_csv\_data(file\_path: str) -> pd.DataFrame:

    try:

        df = pd.read\_csv(file\_path)

        print(f"Loaded data shape: {df.shape}")

        return df

    except Exception as e:

        print(f"Error loading CSV: {e}")

        return pd.DataFrame()

import pandas as pd

def clean\_data(df: pd.DataFrame) -> pd.DataFrame:

    # Show initial row count

    print(f"Original rows: {df.shape[0]}")

    # Drop duplicate rows

    df = df.drop\_duplicates()

    # Strip whitespace from string columns

    str\_cols = df.select\_dtypes(include=['object']).columns

    for col in str\_cols:

        df[col] = df[col].str.strip()

    # Keep only rows with valid binary values

    df = df[df['Churn'].isin(['0', '1'])]

    df = df[df['HasCrCard'].isin(['0', '1'])]

    df = df[df['IsActiveMember'].isin(['0', '1'])]

    # Normalize gender

    if 'Gender' in df.columns:

        df['Gender'] = df['Gender'].str.strip().str.lower()

    # Convert numerical columns

    numeric\_cols = ['Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']

    for col in numeric\_cols:

        if col in df.columns:

            df[col] = pd.to\_numeric(df[col], errors='coerce')

    # Drop missing values

    df = df.dropna()

    # Remove invalid values

    df = df[df['Age'] >= 0]

    df = df[df['Tenure'] >= 0]

    df = df[df['Balance'] >= 0]

    df = df[df['NumOfProducts'] > 0]

    df = df[df['EstimatedSalary'] >= 0]

    # Outlier removal using IQR method

    for col in numeric\_cols:

        if col in df.columns:

            Q1 = df[col].quantile(0.25)

            Q3 = df[col].quantile(0.75)

            IQR = Q3 - Q1

            lower\_bound = Q1 - 1.5 \* IQR

            upper\_bound = Q3 + 1.5 \* IQR

            original\_count = df.shape[0]

            df = df[(df[col] >= lower\_bound) & (df[col] <= upper\_bound)]

            removed = original\_count - df.shape[0]

            if removed > 0:

                print(f"Removed {removed} outliers from '{col}'")

    # Reset index

    df.reset\_index(drop=True, inplace=True)

    # Final row count

    print(f"Rows after cleaning: {df.shape[0]}")

    return df

# scripts/eda.py

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

# Set a "cute" color theme

sns.set\_style("whitegrid")

sns.set\_palette("pastel")

def plot\_churn\_distribution(df):

    plt.figure(figsize=(6, 4))

    sns.countplot(x='Churn', data=df)

    plt.title('Churn Count', fontsize=14)

    plt.xlabel('Churn')

    plt.ylabel('Count')

    plt.savefig('feature/eda/churn\_count.png', bbox\_inches='tight')

    plt.clf()

def plot\_age\_distribution(df):

    plt.figure(figsize=(6, 4))

    sns.histplot(df['Age'], kde=True, color=sns.color\_palette("muted")[3])

    plt.title('Age Distribution', fontsize=14)

    plt.xlabel('Age')

    plt.ylabel('Frequency')

    plt.savefig('feature/eda/age\_distribution.png', bbox\_inches='tight')

    plt.clf()

def plot\_correlation\_matrix(df):

    # Select only numeric columns

    df\_numeric = df.select\_dtypes(include=[np.number])

    # Handle NaNs and Infs

    df\_numeric.replace([np.inf, -np.inf], np.nan, inplace=True)

    df\_numeric.dropna(inplace=True)

    # Compute correlation

    correlation = df\_numeric.corr()

    # Use a visually appealing color map

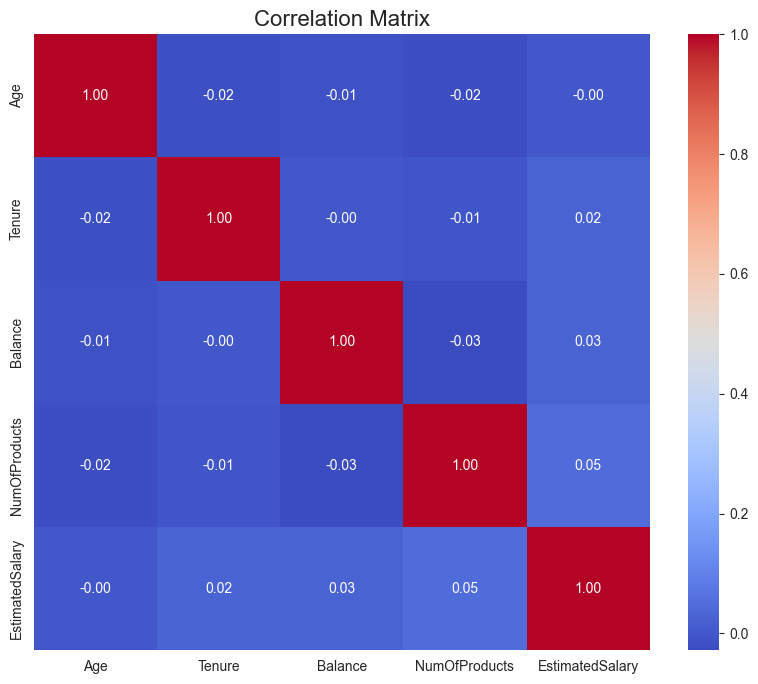
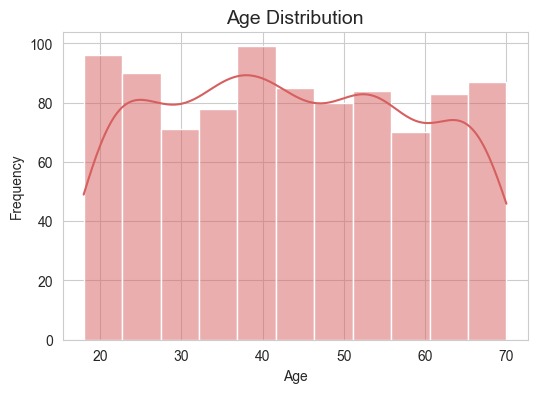
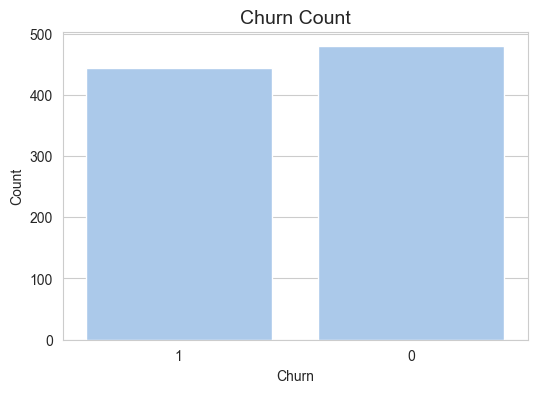
    plt.figure(figsize=(10, 8))

    sns.heatmap(correlation, annot=True, fmt='.2f', cmap='coolwarm', square=True, cbar=True)

    plt.title('Correlation Matrix', fontsize=16)

    plt.savefig('feature/eda/correlation\_matrix.png', bbox\_inches='tight')

    plt.clf()



# scripts/feature\_engineering.py

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

def one\_hot\_encode(df: pd.DataFrame) -> pd.DataFrame:

    return pd.get\_dummies(df, columns=['Gender'], drop\_first=True)

def scale\_features(df: pd.DataFrame, numerical\_cols: list) -> pd.DataFrame:

    scaler = MinMaxScaler()

    df[numerical\_cols] = scaler.fit\_transform(df[numerical\_cols])

    return df

# scripts/model\_training.py

import os

import pickle

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, confusion\_matrix

def get\_model(model\_name):

    if model\_name == 'random\_forest':

        return RandomForestClassifier(n\_estimators=100, random\_state=42)

    elif model\_name == 'logistic\_regression':

        return LogisticRegression(max\_iter=1000, class\_weight='balanced')

    elif model\_name == 'knn':

        return KNeighborsClassifier(n\_neighbors=5)

    else:

        raise ValueError(f"Model '{model\_name}' is not supported.")

def train\_model(df, model\_name='random\_forest', target\_col='Churn'):

    # 1. Split data

    X = df.drop(columns=[target\_col, 'CustomerID'])

    y = df[target\_col]

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    test\_data = df.iloc[y\_test.index]  # Keep original test rows for CustomerId

    # 2. Train model

    model = get\_model(model\_name)

    model.fit(X\_train, y\_train)

    # 3. Predict

    y\_pred = model.predict(X\_test)

    # 4. Save model

    os.makedirs('model', exist\_ok=True)

    with open(f'model/{model\_name}\_churn\_model.pkl', 'wb') as f:

        pickle.dump(model, f)

    # 5. Save metrics

    with open(f'result/test\_data/{model\_name}\_model\_metrics.txt', 'w') as f:

        f.write(classification\_report(y\_test, y\_pred))

    # 7. Visualizations

    os.makedirs("plots", exist\_ok=True)

    pastel\_palette = ['#FFB6C1', '#B0E0E6']  # light pink and light blue

    # 7a. Confusion matrix

    cm = confusion\_matrix(y\_test, y\_pred)

    plt.figure(figsize=(5, 4))

    sns.heatmap(cm, annot=True, fmt='d', cmap=sns.light\_palette("seagreen", as\_cmap=True),

                xticklabels=['No Churn', 'Churn'],

                yticklabels=['No Churn', 'Churn'])

    plt.xlabel('Predicted')

    plt.ylabel('Actual')

    plt.title(f'{model\_name} - Confusion Matrix')

    plt.tight\_layout()

    plt.savefig(f'plots/{model\_name}\_confusion\_matrix.png')

    plt.close()

    # 7b. Churn distribution bar plot

    churn\_counts = pd.Series(y\_pred).value\_counts().sort\_index()

    churn\_labels = ['No Churn', 'Churn']

    plt.figure()

    sns.barplot(x=churn\_labels, y=churn\_counts, palette=pastel\_palette)

    plt.title(f'{model\_name} - Predicted Churn Distribution')

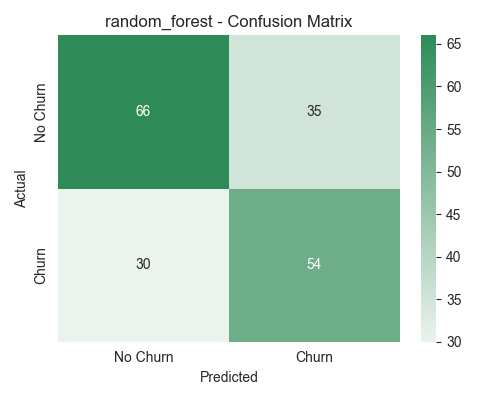
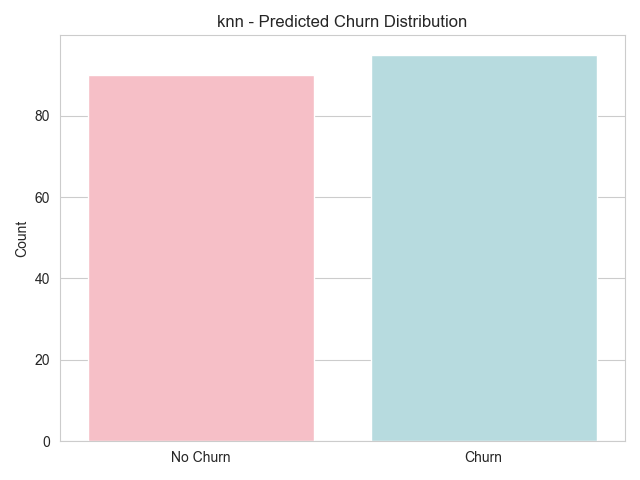
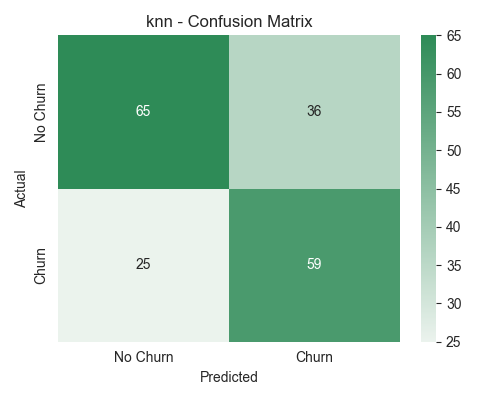
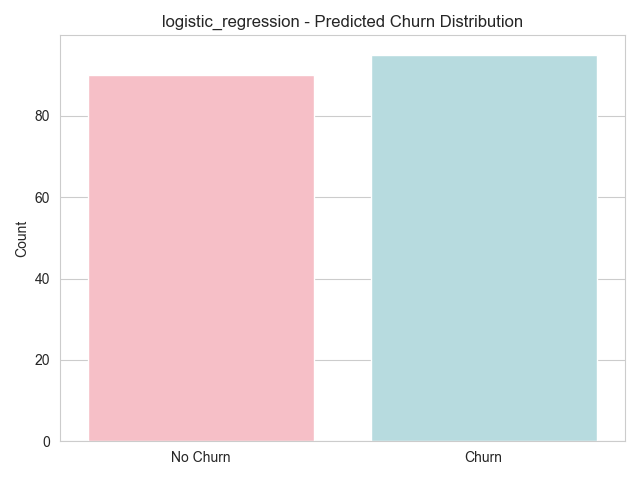
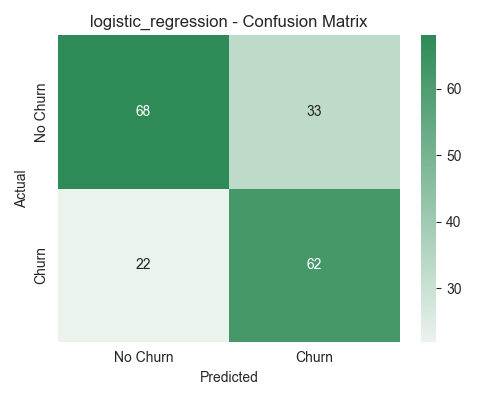
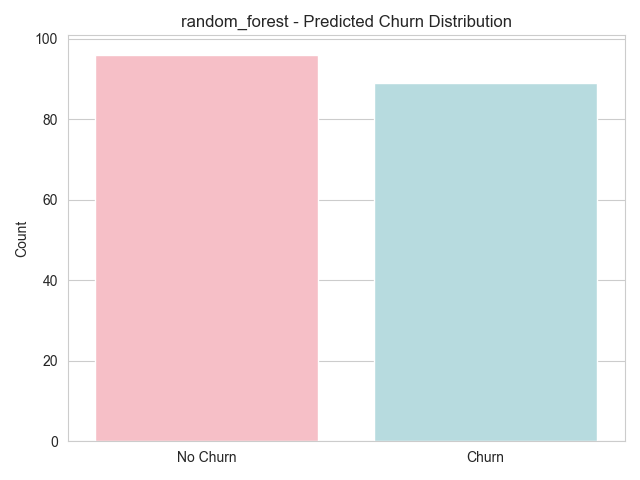
    plt.ylabel('Count')

    plt.tight\_layout()

    plt.savefig(f'plots/{model\_name}\_churn\_bar.png')

    plt.close()

    return model



import pandas as pd

from scripts.data\_loader import load\_csv\_data

from scripts.eda import plot\_churn\_distribution, plot\_age\_distribution, plot\_correlation\_matrix

from scripts.feature\_engineering import one\_hot\_encode, scale\_features

from scripts.model\_training import train\_model

test\_data = pd.DataFrame({

    'CustomerId': [1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009],

    'Gender': ['Male', 'Female', 'Female', 'male', 'FEMALE', 'female', 'male', 'female', 'male'],

    'Age': [35, 42, 28, -5, 51, 30, 45, 38, 60],  # -5 is invalid

    'Tenure': [3, 5, 2, 1, 7, 4, 6, 3, 8],

    'Balance': [50000.0, 0.0, -100.0, 30000.0, 75000.0, 25000.0, 60000.0, 40000.0, 85000.0],  # -100 is invalid

    'NumOfProducts': [2, 1, 1, 3, 2, 2, 1, 2, 2],

    'HasCrCard': [1, 0, 1, 1, 0, 1, 1, 0, 1],

    'IsActiveMember': [1, 0, 1, 1, 0, 1, 1, 0, 1],

    'EstimatedSalary': [60000, 52000, 45000, 72000, 68000, 56000, 47000, 63000, 75000],

    'Churn': ['1', '0', 'yes', 'No', '1', '0', '1', '0', '1']  # 'yes'/'No' are inconsistent

})

def clean\_data(df):

    df = df.copy()

    # Fix case for Gender and Churn

    df['Gender'] = df['Gender'].str.lower().str.strip()

    df['Churn'] = df['Churn'].astype(str).str.lower().str.strip()

    # Clean invalid ages and balances

    df = df[df['Age'] > 0]

    df = df[df['Balance'] >= 0]

    # Normalize churn labels

    churn\_map = {'yes': 1, '1': 1, 'no': 0, '0': 0}

    df['Churn'] = df['Churn'].map(churn\_map)

    # Keep only known genders

    df = df[df['Gender'].isin(['male', 'female'])]

    # Drop rows with missing target

    df = df.dropna(subset=['Churn'])

    # Convert to int

    df['Churn'] = df['Churn'].astype(int)

    return df

df\_test\_clean = clean\_data(test\_data)

print(df\_test\_clean.head())

df\_test\_encoded = one\_hot\_encode(df\_test\_clean)

print(df\_test\_encoded.head())

numeric\_cols = ['Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']

df\_test\_scaled = scale\_features(df\_test\_encoded, numeric\_cols)

print(df\_test\_scaled.head())

import joblib

# Drop unneeded columns

X\_test = df\_test\_scaled.drop(columns=['CustomerId', 'Churn'])

print ("Test features: ")

print(X\_test.head())

y\_test = df\_test\_scaled['Churn']

# Load your trained model

model = joblib.load('model/random\_forest\_churn\_model.pkl')

# Predict

predictions = model.predict(X\_test)

# Output

for i, pred in enumerate(predictions):

    print(f"Customer ID: {test\_data.iloc[i]['CustomerId']} --> Predicted Churn: {pred} | Actual: {y\_test.iloc[i]}")

import os

# Prepare output directory

os.makedirs("result", exist\_ok=True)

# Define output file name

model\_name = model.\_\_class\_\_.\_\_name\_\_  # e.g., KNeighborsClassifier

output\_file = f"result/test\_sample/{model\_name.lower()}\_test\_result.txt"

# Write predictions to the file

with open(output\_file, "w") as f:

    for i, pred in enumerate(predictions):

        line = f"Customer ID: {df\_test\_clean.iloc[i]['CustomerId']} --> Predicted Churn: {pred} | Actual: {y\_test.iloc[i]}"

        f.write(line + "\n")

output

Customer ID: 1001 --> Predicted Churn: 0 | Actual: 1

Customer ID: 1002 --> Predicted Churn: 1 | Actual: 0

Customer ID: 1005 --> Predicted Churn: 0 | Actual: 1

Customer ID: 1006 --> Predicted Churn: 0 | Actual: 0

Customer ID: 1007 --> Predicted Churn: 0 | Actual: 1

Customer ID: 1008 --> Predicted Churn: 1 | Actual: 0

Customer ID: 1009 --> Predicted Churn: 0 | Actual: 1