**Source code** :

[**Predicting Customer Churn using machine learning to uncover hidden pattern ]**

# Import Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

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

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.ensemble import RandomForestClassifier

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

import gradio as gr

# Load Dataset

try:

df = pd.read\_csv("churn\_data.csv") # Replace with your actual dataset path

except FileNotFoundError:

print("Dataset not found. Please check the path.")

exit()

# Basic Info

print("Dataset Shape:", df.shape)

print("\nData Types:\n", df.dtypes)

print("\nMissing Values:\n", df.isnull().sum())

print("\nDuplicate Rows:", df.duplicated().sum())

print("\nFirst 5 Rows:\n", df.head())

# =============================

# 🌟 Exploratory Data Analysis

# =============================

# 1. Summary Statistics

print("\nSummary Statistics:\n", df.describe(include='all'))

# 2. Target Variable Distribution

sns.countplot(data=df, x='Churn', palette='Set2')

plt.title("Churn Distribution")

plt.show()

churn\_rate = df['Churn'].value\_counts(normalize=True) \* 100

print(f"\nChurn Rate:\n{churn\_rate}")

# 3. Correlation Heatmap

plt.figure(figsize=(12, 8))

sns.heatmap(df.corr(numeric\_only=True), annot=True, fmt='.2f', cmap='coolwarm')

plt.title("Correlation Heatmap")

plt.show()

# 4. Categorical Features vs. Churn

categorical\_cols = df.select\_dtypes(include='object').columns

for col in categorical\_cols:

if col != 'Churn':

plt.figure(figsize=(6, 4))

sns.countplot(data=df, x=col, hue='Churn', palette='Set3')

plt.title(f'{col} vs Churn')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

# 5. Numerical Features Distribution

numeric\_cols = df.select\_dtypes(include=['int64', 'float64']).columns

df[numeric\_cols].hist(bins=20, figsize=(15, 10), layout=(len(numeric\_cols) // 3 + 1, 3))

plt.tight\_layout()

plt.show()

# 6. Boxplots for Outliers

for col in numeric\_cols:

plt.figure(figsize=(6, 4))

sns.boxplot(data=df, x='Churn', y=col)

plt.title(f'{col} Boxplot by Churn')

plt.tight\_layout()

plt.show()

# 7. Optional: Interactive Plotly Scatter Matrix

fig = px.scatter\_matrix(df, dimensions=numeric\_cols, color="Churn", title="Scatter Matrix by Churn")

fig.show()

# =============================

# Data Preprocessing

# =============================

df = df.dropna() # Drop missing values

label\_encoders = {}

# Encode categorical features

for col in df.select\_dtypes(include=['object']).columns:

if col != 'Churn': # Churn will be handled separately

le = LabelEncoder()

df[col] = le.fit\_transform(df[col])

label\_encoders[col] = le

# Encode target

df['Churn'] = df['Churn'].map({'Yes': 1, 'No': 0})

# Feature and Target Split

X = df.drop('Churn', axis=1)

y = df['Churn']

# Feature Scaling

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# 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 Training

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

model.fit(X\_train, y\_train)

# Model Evaluation

y\_pred = model.predict(X\_test)

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

# Confusion Matrix Heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(confusion\_matrix(y\_test, y\_pred), annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

# Churn Distribution Histogram

fig = px.histogram(df, x='Churn', color='Churn', title='Churn Distribution')

fig.show()

# =============================

# Gradio Interface

# =============================

def predict\_churn(\*\*inputs):

input\_df = pd.DataFrame([inputs])

for col, le in label\_encoders.items():

input\_df[col] = le.transform(input\_df[col])

input\_scaled = scaler.transform(input\_df)

prediction = model.predict(input\_scaled)[0]

probability = model.predict\_proba(input\_scaled)[0][1]

return {"Churn": "Yes" if prediction == 1 else "No", "Probability": f"{probability:.2f}"}

# Create input components dynamically

input\_components = []

for col in X.columns:

if col in label\_encoders:

options = list(label\_encoders[col].classes\_)

input\_components.append(gr.Dropdown(label=col, choices=options))

else:

input\_components.append(gr.Number(label=col))

gr.Interface(

fn=predict\_churn,

inputs=input\_components,

outputs=[gr.Label(label="Churn Prediction"), gr.Label(label="Churn Probability")],

title="Customer Churn Predictor",

description="Predict whether a customer will churn based on input features."

).launch()