# SOURCE CODE:

# Install libraries

!pip install pandas scikit-learn

# Import Libraries

import pandas as pd

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

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\\_score

# Load Dataset

df = pd.read\\_csv("Telco-Customer-Churn.csv")

# Basic Preprocessing

df = df\[df\['TotalCharges'] != " "]

df\['TotalCharges'] = df\['TotalCharges'].astype(float)

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

df.drop(\['customerID'], axis=1, inplace=True)

# Encode categorical variables

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

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

# Train/Test Split

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

y = df\['Churn']

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

# Train Model

model = RandomForestClassifier()

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

# Predict and Evaluate

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

print("Accuracy:", accuracy\\_score(y\\_test, y\\_pred))

# Save Predictions to CSV

results\\_df = X\\_test.copy()

results\\_df\['Actual\\_Churn'] = y\\_test.values

results\\_df\['Predicted\\_Churn'] = y\\_pred

results\\_df.to\\_csv('churn\\_predictions.csv', index=False)

print("Predictions saved to 'churn\\_predictions.csv'")

import gradio as gr

import numpy as np

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

import seaborn as sns

import os # Import os module for file operations

import tempfile # Import tempfile for creating temporary files

# Seed for reproducibility

np.random.seed(42)

# Generate synthetic dataset for customer churn

def generate\\_data(n=1000):

data = pd.DataFrame()

# Numeric features

data['tenure'] = np.random.randint(0, 72, n) # months

data['MonthlyCharges'] = np.round(np.random.uniform(20, 120, n), 2)

data['TotalCharges'] = np.round(data['MonthlyCharges'] \* data['tenure'] + np.random.uniform(-10, 50, n), 2)

data['NumCustomerServiceCalls'] = np.random.randint(0, 10, n)

# Categorical features

data['Contract'] = np.random.choice(['Month-to-month', 'One year', 'Two year'], n, p=[0.55, 0.25, 0.2])

data['PaymentMethod'] = np.random.choice(['Electronic check', 'Mailed check', 'Bank transfer', 'Credit card'], n)

data['InternetService'] = np.random.choice(['DSL', 'Fiber optic', 'No'], n, p=[0.35, 0.45, 0.2])

# Target variable churn: let's simulate churn with noise depending on features

churn\_prob = (

0.3 \* (data['Contract'] == 'Month-to-month').astype(int) +

0.25 \* (data['InternetService'] == 'Fiber optic').astype(int) +

0.15 \* (data['NumCustomerServiceCalls'] > 3).astype(int) +

0.1 \* (data['MonthlyCharges'] > 80).astype(int) +

0.1 \* (data['tenure'] < 12).astype(int) +

np.random.normal(0, 0.05, n)

)

data['Churn'] = (churn\_prob > 0.4).astype(int)

return data

# Prepare data and model

df = generate\\_data(1000)

# Encode categorical variables

categorical\\_cols = \['Contract', 'PaymentMethod', 'InternetService']

label\\_encoders = {}

for col in categorical\\_cols:

le = LabelEncoder()

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

label\\_encoders\[col] = le

feature\\_cols = \['tenure', 'MonthlyCharges', 'TotalCharges', 'NumCustomerServiceCalls'] + categorical\\_cols

X = df\[feature\\_cols]

y = df\['Churn']

# Train model

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

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

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

# Feature importance for pattern uncovering

feature\importance = pd.Series(model.feature\\_importances\, index=feature\\_cols).sort\\_values(ascending=False)

# Prediction function for Gradio

def predict\\_churn(tenure, monthly\\_charges, total\\_charges, num\\_calls, contract, payment\\_method, internet\\_service):

\# Encode inputs the same way as training

input\\_data = {

'tenure': tenure,

'MonthlyCharges': monthly\\_charges,

'TotalCharges': total\\_charges,

'NumCustomerServiceCalls': num\\_calls,

'Contract': label\\_encoders\['Contract'].transform(\[contract])\[0],

'PaymentMethod': label\\_encoders\['PaymentMethod'].transform(\[payment\\_method])\[0],

'InternetService': label\\_encoders\['InternetService'].transform(\[internet\\_service])\[0]

}

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

pred\_proba = model.predict\_proba(input\_df)[0,1]

pred\_label = model.predict(input\_df)[0]

churn\_text = "Yes" if pred\_label == 1 else "No"

confidence = f"{pred\_proba\*100:.2f}%"

# Construct explanation text

explanation = "Feature importances in prediction:\n"

# simple contributions with feature importance \* standardized input (optional for sophistication)

importance\_df = pd.DataFrame({

'Feature': feature\_importance.index,

'Importance': feature\_importance.values,

'Value': [input\_df[feat].values[0] for feat in feature\_importance.index]

})

explanation += "\n".join([f"{row['Feature']}: Importance {row['Importance']:.3f}, Input value {row['Value']}" for \_, row in importance\_df.iterrows()])

return churn\_text, confidence, explanation

# Plot feature importance

def plot\\_feature\\_importance():

plt.figure(figsize=(8,5))

sns.barplot(x=feature\\_importance.values, y=feature\\_importance.index, palette="coolwarm")

plt.title("Feature Importance for Customer Churn Prediction")

plt.xlabel("Importance")

plt.tight\\_layout()

# Save the plot to a temporary file and return the path

with tempfile.NamedTemporaryFile(suffix=".png", delete=False) as tmpfile:

plt.savefig(tmpfile.name)

plot\_path = tmpfile.name

plt.close() # Close the figure after saving

return plot\_path # Return the path to the saved image file

# Define Gradio components

with gr.Blocks() as demo:

gr.Markdown("<h1 style='text-align:center;color:#2E86C1'>Customer Churn Prediction App</h1>")

gr.Markdown("""

This app predicts customer churn probability using a Random Forest Classifier trained on synthetic data.

Enter customer details below to see whether they might churn and uncover important factors driving the prediction.

""")

with gr.Row():

with gr.Column():

tenure = gr.Slider(0, 72, step=1, label="Tenure (months)", value=12)

monthly\\_charges = gr.Number(label="Monthly Charges (\$)", value=70)

total\\_charges = gr.Number(label="Total Charges (\$)", value=2000)

num\\_calls = gr.Slider(0, 10, step=1, label="Number of Customer Service Calls", value=1)

with gr.Column():

contract = gr.Dropdown(choices=label\encoders\['Contract'].classes\.tolist(), label="Contract Type", value='Month-to-month')

payment\method = gr.Dropdown(choices=label\\_encoders\['PaymentMethod'].classes\.tolist(), label="Payment Method", value='Electronic check')

internet\service = gr.Dropdown(choices=label\\_encoders\['InternetService'].classes\.tolist(), label="Internet Service", value='DSL')

churn\_output = gr.Textbox(label="Churn Prediction", interactive=False)

confidence\_output = gr.Textbox(label="Prediction Confidence", interactive=False)

explanation\_output = gr.Textbox(label="Feature Importance Explanation", lines=8, interactive=False)

# Now the value is a function that returns a file path

feature\_imp\_plot = gr.Image(value=plot\_feature\_importance, label="Overall Feature Importance", interactive=False)

btn = gr.Button("Predict Churn")

btn.click(fn=predict\_churn, inputs=[tenure, monthly\_charges, total\_charges, num\_calls, contract, payment\_method, internet\_service],

outputs=[churn\_output, confidence\_output, explanation\_output])

if \*name\* == "\*main\*":

demo.launch()