# -\*- coding: utf-8 -\*-

"""phase 3

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1CArcQr6U4Md9hn\_YLj5mtbyoP-deP51m

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import StandardScaler

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

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

!pip install gradio

import gradio as gr

df = pd.read\_csv('/content/people-100 (2).csv')

df.head()

print("Missing values in each column:", df.isnull().sum())

df.dropna(inplace=True)

df.info()

print(df.describe())

df\_reset = df.reset\_index()

sns.countplot(x='index', data=df\_reset)

plt.title('Distribution of Fraud and Non-Fraud Transactions')

plt.show()

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

numeric\_df = df.select\_dtypes(include=np.number)

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

plt.title('Correlation Matrix')

plt.show()

print(df.columns)

df = df.drop(columns=['Index'])

print(df.head())

df = pd.read\_csv('/content/people-100 (2).csv')

df = df.drop(columns=['Index'])

if 'Index' in df.columns:

df = df.drop(columns=['Index'])

else:

print("'Index' column not found in the dataframe.")

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

X = df.drop(columns=['User Id'])

y = df['User Id']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

scaler = StandardScaler()

print(X.dtypes)

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

print("Original columns:", X.columns)

if any(X.dtypes == 'object'):

print("Encoding categorical columns...")

X = pd.get\_dummies(X, drop\_first=True)

print("New columns after encoding:", X.columns)

else:

print("No categorical columns to encode.")

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

print(X.dtypes)

print(type(X))

original\_column\_names = X.columns

print(X.shape)

print(len(original\_column\_names))

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

print(type(X\_train))

print(X\_train.shape)

print(X\_train.dtypes)

import numpy as np

print(np.any(np.isnan(X\_train)))

print(np.any(np.isinf(X\_train)))

from sklearn.preprocessing import StandardScaler

import pandas as pd

import numpy as np

X\_train = pd.DataFrame({

'feature1': [1.0, 2.0, 3.0],

'feature2': [4.0, 5.0, 6.0]

})

scaler = StandardScaler()

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

print(X\_train\_scaled)

print(X\_train.shape)

print(X\_test.shape)

import numpy as np

print(np.any(np.isnan(X\_test)))

print(np.any(np.isinf(X\_test)))

from sklearn.preprocessing import StandardScaler

import pandas as pd

X\_train = pd.DataFrame({

'feature1': [1, 2, 3],

'feature2': [4, 5, 6]

})

X\_test = pd.DataFrame({

'feature1': [2, 3],

'feature2': [5, 6]

})

scaler = StandardScaler()

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

X\_test\_scaled = scaler.transform(X\_test)

print(X\_test\_scaled)

scaler = StandardScaler()

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

X\_test\_scaled = scaler.transform(X\_test)

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

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

print(X\_train\_scaled.shape)

print(y\_train.shape)

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

df = pd.read\_csv('/people-100 (2) (1).csv')

X = df.drop(columns=['Index'])

y = df['Index']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

X\_train = X\_train.reset\_index(drop=True)

y\_train = y\_train.reset\_index(drop=True)

!pip install scikit-learn

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

categorical\_features = X\_train.select\_dtypes(include=['object']).columns.tolist()

numerical\_pipeline = Pipeline([

('scaler', StandardScaler()),

])

categorical\_pipeline = Pipeline([

('onehot', OneHotEncoder(sparse\_output=False, handle\_unknown='ignore')),

])

preprocessor = ColumnTransformer(

transformers=[

('num', numerical\_pipeline, numerical\_features),

('cat', categorical\_pipeline, categorical\_features),

])

l

pipeline = Pipeline([

('preprocessor', preprocessor),

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

])

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

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

!pip install scikit-learn

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

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

df = pd.read\_csv('/people-100 (2) (1).csv')

X = df.drop(columns=['User Id'])

y = df['User Id']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

numerical\_features = X\_train.select\_dtypes(include=np.number).columns.tolist()

categorical\_features = X\_train.select\_dtypes(include=['object']).columns.tolist()

numerical\_pipeline = Pipeline([

('scaler', StandardScaler()),

])

categorical\_pipeline = Pipeline([

('onehot', OneHotEncoder(sparse\_output=False, handle\_unknown='ignore')),

])

preprocessor = ColumnTransformer(

transformers=[

('num', numerical\_pipeline, numerical\_features),

('cat', categorical\_pipeline, categorical\_features),

])

pipeline = Pipeline([

('preprocessor', preprocessor),

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

])

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

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

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

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

!pip install gradio

import gradio as gr

import pandas as pd

def predict\_fraud(v\_inputs):

# Define X within the function or pass it as an argument

df = pd.read\_csv('/people-100 (2) (1).csv') # Assuming this is the correct path

X = df.drop(columns=['User Id']) # Assuming 'User Id' is the target column

input\_df = pd.DataFrame([v\_inputs], columns=X.columns)

# Assuming 'model' is defined and trained in a previous cell

# If not, you need to load or train the model here as well.

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

# Check if predict\_proba is available for your model

try:

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

except AttributeError:

prob = 0.5 # Or some default value

return {"Prediction": "Fraud" if pred == 1 else "Not Fraud", "Probability": round(prob, 3)}

input\_components = [gr.Number(label=col) for col in pd.read\_csv('/content/people-100 (2) (1).csv').drop(columns=['User Id']).columns]

gr.Interface(

fn=predict\_fraud,

inputs=input\_components,

outputs=["text", "number"],

title="🔐 AI-Powered Credit Card Fraud Detection",

description="Enter transaction features to detect fraud in real-time."

).launch()