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, LabelEncoder, OneHotEncoder

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

# Load the dataset

data = pd.read\_excel('Patient\_Data.xlsx')

# Data Preprocessing

# Expand the dataset to include additional factors such as demographics, clinical biomarkers, medication history, etc.

# Feature Engineering

# Include additional features based on demographic information, clinical biomarkers, medication history, etc.

# Model Training

# Define features (X) and target variable (y)

X = data.drop(columns=['Name']) # Update columns based on expanded dataset

y = data['Name']

# Split the data into training and testing sets

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

# Define numeric and categorical features

numeric\_features = X.select\_dtypes(include=np.number).columns

categorical\_features = X.select\_dtypes(include=object).columns

# Preprocessing pipeline for numeric features

numeric\_transformer = Pipeline(steps=[

('scaler', StandardScaler())

])

# Preprocessing pipeline for categorical features

categorical\_transformer = Pipeline(steps=[

('onehot', OneHotEncoder(handle\_unknown='ignore'))

])

# Column transformer to apply different preprocessing steps to numeric and categorical features

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_features),

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

])

# Model pipeline

pipeline = Pipeline([

('preprocessor', preprocessor),

('classifier', RandomForestClassifier())

])

# Train the model

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

# Model Evaluation

# Predict on the testing set

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

# Evaluate model performance

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

# Classification report

print(classification\_report(y\_test, y\_pred, zero\_division=1))

# Confusion matrix

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

sns.heatmap(conf\_matrix, annot=True, cmap="YlGnBu", fmt="d")

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

# Plot feature importance

importances = pipeline.named\_steps['classifier'].feature\_importances\_

# Get feature names after one-hot encoding

cat\_encoder = pipeline.named\_steps['preprocessor'].named\_transformers\_['cat'].named\_steps['onehot']

cat\_onehot\_features = list(cat\_encoder.get\_feature\_names\_out(categorical\_features))

all\_features = list(numeric\_features) + cat\_onehot\_features

indices = np.argsort(importances)

plt.figure(figsize=(10, 6))

plt.title('Feature Importances')

plt.barh(range(len(indices)), importances[indices], color='b', align='center')

plt.yticks(range(len(indices)), [all\_features[i] for i in indices])

plt.xlabel('Relative Importance')

plt.show()

# Plot correlation matrix

corr\_matrix = data.corr()

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

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

plt.title('Correlation Matrix')

plt.show()

# Check if selected features exist in the dataset

selected\_features = ['Age', 'BMI', 'Blood\_Pressure', 'Cholesterol']

valid\_selected\_features = [feature for feature in selected\_features if feature in data.columns]

# Pairplot for selected features

if valid\_selected\_features:

sns.pairplot(data[valid\_selected\_features])

plt.title('Pairplot of Selected Features')

plt.show()

else:

print("Selected features not found in the dataset.")