import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import LabelEncoder

from sklearn.linear\_model import LogisticRegression

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

from sklearn.ensemble import RandomForestClassifier

# Load the dataset

df = sns.load\_dataset('titanic')

# Display the first few rows of the dataset

print(df.head())

# Display dataset info

print(df.info())

# Display summary statistics

print(df.describe())

# Check for missing values

print(df.isnull().sum())

# Fill missing values for 'age' with the median age

df['age'].fillna(df['age'].median(), inplace=True)

# Fill missing values for 'embarked' with the most common value

df['embarked'].fillna(df['embarked'].mode()[0], inplace=True)

# Drop rows with missing 'embark\_town' and 'deck' as these are not crucial for our analysis

df.drop(columns=['embark\_town', 'deck'], inplace=True)

# Drop the 'alive' column as it is redundant with 'survived'

df.drop(columns=['alive'], inplace=True)

# Encode categorical variables

label\_encoder = LabelEncoder()

df['sex'] = label\_encoder.fit\_transform(df['sex'])

df['embarked'] = label\_encoder.fit\_transform(df['embarked'])

df['class'] = label\_encoder.fit\_transform(df['class'])

# Drop columns that won't be used in the model

df.drop(columns=['who', 'adult\_male', 'embark\_town', 'alone'], inplace=True)

# Define features (X) and target (y)

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

y = df['survived']

# Split the data

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

# Initialize the model

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

# Fit the model

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

# Make predictions

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

# Calculate accuracy

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

print(f"Accuracy: {accuracy:.2f}")

# Display confusion matrix

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

print(f"Confusion Matrix:\n{conf\_matrix}")

# Display classification report

class\_report = classification\_report(y\_test, y\_pred)

print(f"Classification Report:\n{class\_report}")

# Get feature importances

feature\_importances = model.feature\_importances\_

# Create a DataFrame for visualization

features\_df = pd.DataFrame({

'Feature': X.columns,

'Importance': feature\_importances

})

# Sort the DataFrame by importance

features\_df = features\_df.sort\_values(by='Importance', ascending=False)

# Display the feature importances

print(features\_df)

# Plot the feature importances

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

sns.barplot(x='Importance', y='Feature', data=features\_df)

plt.title('Feature Importances')

plt.show()