import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

from sklearn.preprocessing import LabelEncoder

# Q1 Define the data source URL

data\_source = "https://raw.githubusercontent.com/HackBio-Internship/public\_datasets/main/R/nhanes.csv"

# Load the dataset into a pandas DataFrame

df = pd.read\_csv(data\_source, sep = ',')

# Remove rows with missing values (NaN)

df\_filled = df.fillna(0)

print(df\_filled.head())

# Q2. Create a figure with 2x2 subplots

fig, axes = plt.subplots(2, 2, figsize=(15, 12))

fig.suptitle('Distribution of Health Metrics in NHANES Dataset', fontsize=16, y=1.02)

# Flatten the axes array to access individual subplots easily

axes = axes.flatten() # <-- Add this line

# Create histograms with KDE

# BMI Distribution

sns.histplot(data=df, x='BMI', kde=True, ax=axes[0]) # <-- Now use axes[0]

axes[0].set\_title('BMI Distribution')

axes[0].set\_xlabel('BMI')

axes[0].set\_ylabel('Count')

# Weight Distribution (kg)

sns.histplot(data=df, x='Weight', kde=True, ax=axes[1]) # <-- Now use axes[1]

axes[1].set\_title('Weight Distribution (kg)')

axes[1].set\_xlabel('Weight (kg)')

axes[1].set\_ylabel('Count')

# Weight Distribution (lbs)

weight\_lbs = df['Weight'] \* 2.2

sns.histplot(x=weight\_lbs, kde=True, ax=axes[2]) # <-- Now use axes[2]

axes[2].set\_title('Weight Distribution (lbs)')

axes[2].set\_xlabel('Weight (lbs)')

axes[2].set\_ylabel('Count')

# Age Distribution

sns.histplot(data=df, x='Age', kde=True, ax=axes[3]) # <-- Now use axes[3]

axes[3].set\_title('Age Distribution')

axes[3].set\_xlabel('Age')

axes[3].set\_ylabel('Count')

# Adjust layout to prevent overlap

plt.tight\_layout()

# Show the plot

plt.show()

# Q3. Assuming the actual column name is "Pulse", adjust the code accordingly

print(df["Pulse"].mean()) # Access using the actual column name

# Q4. Replace 'dbp' with the actual column name for diastolic blood pressure \*\*\*

# For example, if the column is named 'DiastolicBP',

min\_dbp = df['BPDia'].min()

max\_dbp = df['BPDia'].max()

# If the column name is "BPXDIAS" as it appears in the dataset columns, use that.

print(f"Diastolic blood pressure range:")

print(f"Minimum: {min\_dbp}")

print(f"Maximum: {max\_dbp}")

# Q5. Calculate variance and standard deviation for income

income\_variance = df['Income'].var()

income\_std = df['Income'].std()

print(f"Income statistics:")

print(f"Variance: {income\_variance:.2f}")

print(f"Standard Deviation: {income\_std:.2f}")

# Q6a. Create a combination plot using relplot

# Use 'hue' for one categorical variable, and 'style' or 'size' for others if needed

g = sns.relplot(df, x="Weight", y="Height", hue="Gender")

# Add a title and adjust layout

plt.suptitle("Height vs Weight by Gender")

g.fig.tight\_layout() # Use tight\_layout() on the FacetGrid object

plt.show()

# Q6b. Create a combination plot using relplot

# Use 'hue' for one categorical variable, and 'style' or 'size' for others if needed

g = sns.relplot(df, x="Weight", y="Height", hue="Diabetes")

# Add a title and adjust layout

plt.suptitle("Height vs Weight by Diabetes")

g.fig.tight\_layout() # Use tight\_layout() on the FacetGrid object

plt.show()

# Q6c. Create a combination plot using relplot

# Use 'hue' for one categorical variable, and 'style' or 'size' for others if needed

g = sns.relplot(df, x="Weight", y="Height", hue="SmokingStatus")

# Add a title and adjust layout

plt.suptitle("Height vs Weight by Smoking Status")

g.fig.tight\_layout() # Use tight\_layout() on the FacetGrid object

plt.show()

# Q7a. Convert 'Gender' column to numerical representation (e.g., 0 for 'female', 1 for 'male')

df['Gender\_Numeric'] = df['Gender'].map({'female': 0, 'male': 1}) # Create a new column for numerical Gender

# Assign the numerical 'Gender' column and 'Age' column to variables

Gender\_Numeric = df['Gender\_Numeric']

Age = df['Age']

# Perform an independent t-test using the numerical Gender representation

t\_stat, p\_value = stats.ttest\_ind(Gender\_Numeric, Age)

print(f"T-statistic: {t\_stat}")

print(f"P-value: {p\_value}")

# Interpretation

if p\_value < 0.05:

print("The difference between Age and Gender is statistically significant.")

else:

print("The difference between age and gender is not statistically significant.")

# Q7b. Drop all NAS

df =df.dropna()

# Convert 'Diabetes' to numerical representation if it's categorical

# For example, if it has values like 'Yes', 'No', etc., you can use label encoding:

encoder = LabelEncoder()

df['Diabetes\_Numerical'] = encoder.fit\_transform(df['Diabetes'])

BMI = (df['BMI'])

#print(BMI)

Diabetes = (df['Diabetes'])

#print('Diabetes')

# Perform an independent t-test

t\_stat, p\_value = stats.ttest\_ind(BMI,df['Diabetes\_Numerical']) # Use the numerical column

print(f"T-statistic: {t\_stat}")

print(f"P-value: {p\_value}")

# Interpretation

if p\_value < 0.05:

print("The difference between BMI and Diabetes is statistically significant.")

else:

print("The difference between BMI and Diabetes is not statistically significant.")

# Q7c. Drop all NAS

df =df.dropna()

AlcoholYear = (df['AlcoholYear'])

#print(AlcoholYear)

RelationshipStatus = (df['RelationshipStatus'])

#print('RelationshipStatus')

# Convert 'RelationshipStatus' to numerical representation if it's categorical

# For example, if it has values like 'Single', 'Married', etc., you can use label encoding:

encoder = LabelEncoder()

df['RelationshipStatus\_Numerical'] = encoder.fit\_transform(df['RelationshipStatus'])

# Now use the numerical representation for the t-test

t\_stat, p\_value = stats.ttest\_ind(AlcoholYear, df['RelationshipStatus\_Numerical']) # Use the numerical column

print(f"T-statistic: {t\_stat}")

print(f"P-value: {p\_value}")

# Interpretation

if p\_value < 0.05:

print("The difference between AlcoholYear and RelationshipStatus is statistically significant.")

else:

print("The difference between AlcoholYear and RelationshipStatus is not statistically significant.")