```
import pandas as pd

# Load the dataset
file_path = "heart_attack_youngsters_india.csv"
df = pd.read_csv(file_path)

# Display basic information and first few rows
df.info(), df.head()
```

```
 22  Exercise Induced Angina                10000 non-null  object
 23  Blood Oxygen Levels (SpO2%)            10000 non-null  float64
 24  Triglyceride Levels (mg/dL)            10000 non-null  int64
 25  Heart Attack Likelihood                10000 non-null  object
dtypes: float64(2), int64(7), object(17)
memory usage: 2.0+ MB
(None,
    Age  Gender Region Urban/Rural     SES Smoking Status Alcohol Consumption
\
 0   30    Male   East       Urban  Middle          Never           Regularly
 1   24  Female   East       Urban     Low   Occasionally        Occasionally
 2   24  Female  North       Urban     Low   Occasionally        Occasionally
 3   27    Male   East       Urban  Middle   Occasionally               Never
 4   21  Female   West       Rural     Low   Occasionally        Occasionally

         Diet Type Physical Activity Level  Screen Time (hrs/day)  ...  \
 0  Non-Vegetarian               Sedentary                      3  ...
 1  Non-Vegetarian               Sedentary                     15  ...
 2           Vegan                    High                     15  ...
 3      Vegetarian               Sedentary                      6  ...
 4      Vegetarian                Moderate                      4  ...

    Stress Level Blood Pressure (systolic/diastolic mmHg)  \
 0          High                                177.0/63.1
 1          High                               137.5/110.7
 2           Low                                138.3/76.6
 3        Medium                                177.1/90.0
 4           Low                               130.7/108.8

   Resting Heart Rate (bpm) ECG Results  Chest Pain Type  \
 0                       82      Normal      Non-anginal
 1                       76      Normal      Non-anginal
 2                       86      Normal          Typical
 3                      106      Normal      Non-anginal
 4                       73      Normal         Atypical

    Maximum Heart Rate Achieved Exercise Induced Angina  \
 0                          183                       No
 1                          118                       No
 2                          164                       No
 3                          188                       No
 4                          216                       No

   Blood Oxygen Levels (SpO2%)  Triglyceride Levels (mg/dL)  \
 0                        94.1                           58
 1                        97.1                          341
 2                        92.7                          373
 3                        98.4                          102
 4                        94.9                          235

   Heart Attack Likelihood
 0                      No
 1                      No
 2                     Yes
 3                     Yes
 4                      No

 [5 rows x 26 columns])
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report



# Encode categorical variables
label_encoders = {}
for col in df.select_dtypes(include=['object']).columns:
    if col != 'Heart Attack Likelihood':  # Keep target variable separate
        label_encoders[col] = LabelEncoder()
        df[col] = label_encoders[col].fit_transform(df[col])
```

or UnboundLocalError ValueError •••

Please explain the error:

```
ValueError: Found input variables with inconsisten
```

Hypothesis

The error "ValueError: Found input variables with inconsistent numbers of samples: [7163, 0]" indicates that the `X` and `y` variables used in the `train_test_split` function have different numbers of samples. `X` has 7163 samples, while `y` has 0 samples. This likely happened because you're using the original `df_genz` to define `X`, but a potentially empty or modified version of `df_genz` to define `y`. Remember that in cell 10, you converted 'Heart Attack Likelihood' to numeric and then in cell 13, you dropped NaNs from the 'Heart Attack Likelihood'. Because `X` is defined before and `y` is defined after these steps, they end up having a different number of rows.

Suggested Changes

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_s
from sklearn.linear_model import LogisticRegress
from sklearn.metrics import accuracy_score, clas
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline



# --- 1. Data Preparation ---
# Filter Gen Z (adjust age threshold if needed)
df_genz = df[df["Age"] <= 30]
# --- Ensure 'Heart Attack Likelihood' is numeri
df_genz['Heart Attack Likelihood'] = df_genz['He
# --- Drop rows with NaN in 'Heart Attack Likeli
df_genz = df_genz.dropna(subset=['Heart Attack L

# Define features (X) and target (y) *after* cle
X = df_genz[[
    "Sleep Duration (hrs/day)",
    "Screen Time (hrs/day)",
    "Diet Type",
    "Stress Level",
    "Physical Activity Level",
    "BMI (kg/m²)",
    "Cholesterol Levels (mg/dL)",
    "Triglyceride Levels (mg/dL)"
]]

y = df_genz["Heart Attack Likelihood"]  # Binary

# Train-test split
X_train, X_test, y_train, y_test = train_test_sp

# --- 2. Preprocessing Pipeline ---
# Categorical features (to encode)
cat_cols = ["Diet Type", "Stress Level", "Physic
# Numeric features (to scale)
num_cols = ["Sleep Duration (hrs/day)", "Screen
            "Cholesterol Levels (mg/dL)", "Trigl
```

```
# Convert target variable to binary (0 = No, 1 = Yes)
df['Heart Attack Likelihood'] = df['Heart Attack Likelihood'].map({'No': 0, 'Yes': 1

# Define features and target
X = df.drop(columns=['Heart Attack Likelihood'])
y = df['Heart Attack Likelihood']

# Split into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

# Standardize numerical features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train multiple models
models = {
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "SVM": SVC(kernel='rbf', probability=True),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_
}

# Evaluate models
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    results[name] = accuracy
    print(f"{name} Accuracy: {accuracy:.4f}")
    print(classification_report(y_test, y_pred))

# Identify the best model
best_model_name = max(results, key=results.get)
print(f"Best Model: {best_model_name} with Accuracy: {results[best_model_name]:.4f}"
```

```
Random Forest Accuracy: 0.7960
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1592
           1       0.00      0.00      0.00       408

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.44      2000
weighted avg       0.63      0.80      0.71      2000

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
SVM Accuracy: 0.7960
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1592
           1       0.00      0.00      0.00       408

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.44      2000
weighted avg       0.63      0.80      0.71      2000

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [18:25
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
XGBoost Accuracy: 0.7845
              precision    recall  f1-score   support

           0       0.80      0.97      0.88      1592
           1       0.34      0.06      0.10       408

    accuracy                           0.78      2000
   macro avg       0.57      0.51      0.49      2000
weighted avg       0.71      0.78      0.72      2000

Best Model: Random Forest with Accuracy: 0.7960
```

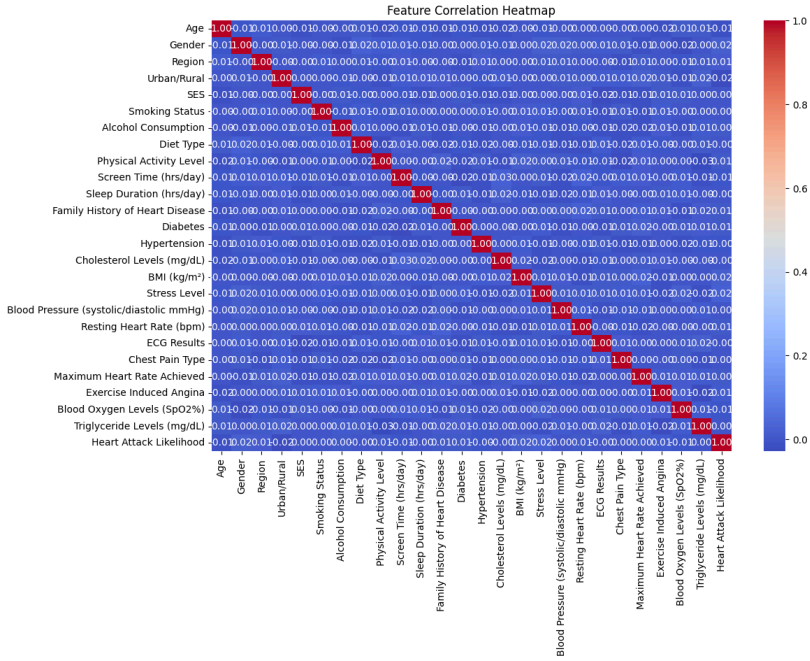
```
# Preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), num_cols),
        ("cat", OneHotEncoder(), cat_cols)
    ])

# --- 3. Model Training ---
# Pipeline with logistic regression
model = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("classifier", LogisticRegression(max_iter=1(
])

model.fit(X_train, y_train)

# --- 4. Model Evaluation ---
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]

print("=== Model Performance ===")
print(f"Accuracy: {accuracy_score(y_test, y_pred
print(f"ROC-AUC: {roc_auc_score(y_test, y_pred_p
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# --- 5. Feature Importance ---
# Extract coefficients (for interpretation)
log_reg = model.named_steps["classifier"]
feature_names = (preprocessor.named_transformers_
feature_names = num_cols + feature_names

coefficients = pd.DataFrame({
    "Feature": feature_names,
    "Coefficient": log_reg.coef_[0]
}).sort_values(by="Coefficient", ascending=False

print("\n=== Top Risk Factors ===")
print(coefficients.head(10))
```

Use code with caution

▶ Sources

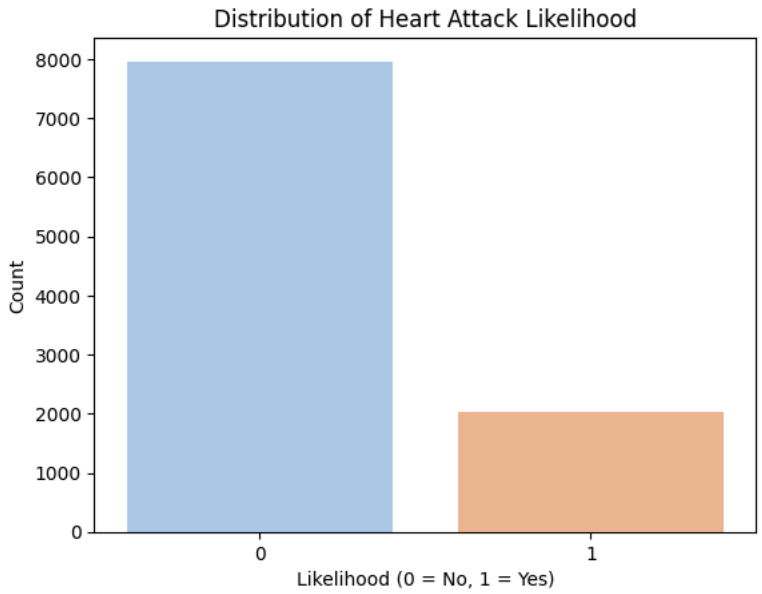
Rate this answer

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()
```




```
sns.countplot(x=df["Heart Attack Likelihood"], palette="pastel")
plt.title("Distribution of Heart Attack Likelihood")
plt.xlabel("Likelihood (0 = No, 1 = Yes)")
plt.ylabel("Count")
plt.show()
```

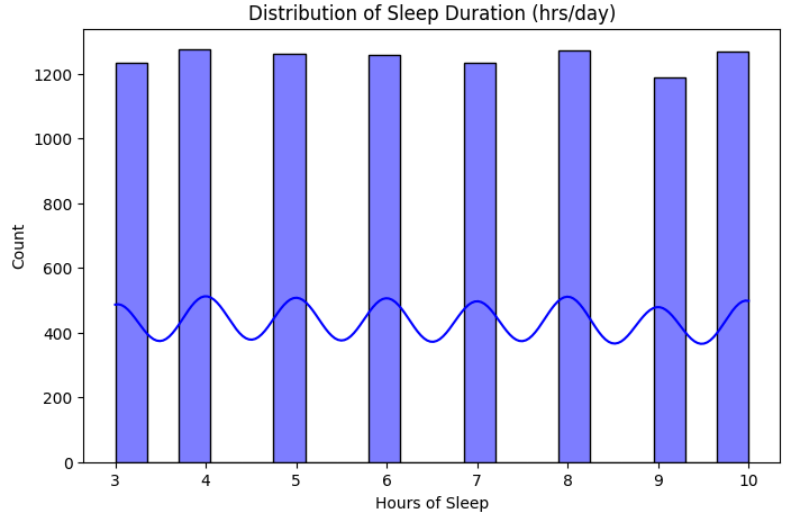
```
<ipython-input-7-faa6bcfc8dfb>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v

  sns.countplot(x=df["Heart Attack Likelihood"], palette="pastel")
```



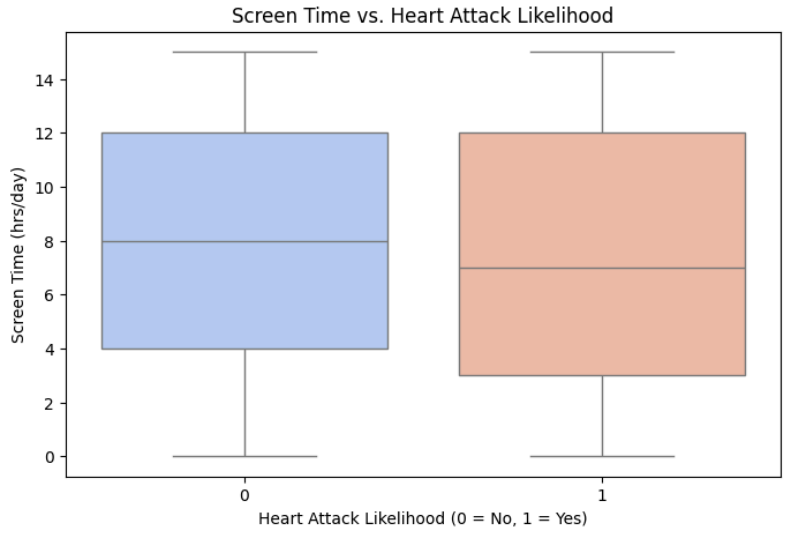

```
df.columns
```

```
Index(['Age', 'Gender', 'Region', 'Urban/Rural', 'SES', 'Smoking Status',
       'Alcohol Consumption', 'Diet Type', 'Physical Activity Level',
       'Screen Time (hrs/day)', 'Sleep Duration (hrs/day)',
       'Family History of Heart Disease', 'Diabetes', 'Hypertension',
       'Cholesterol Levels (mg/dL)', 'BMI (kg/m²)', 'Stress Level',
       'Blood Pressure (systolic/diastolic mmHg)', 'Resting Heart Rate (bpm)',
       'ECG Results', 'Chest Pain Type', 'Maximum Heart Rate Achieved',
       'Exercise Induced Angina', 'Blood Oxygen Levels (SpO2%)',
       'Triglyceride Levels (mg/dL)', 'Heart Attack Likelihood'],
      dtype='object')
```

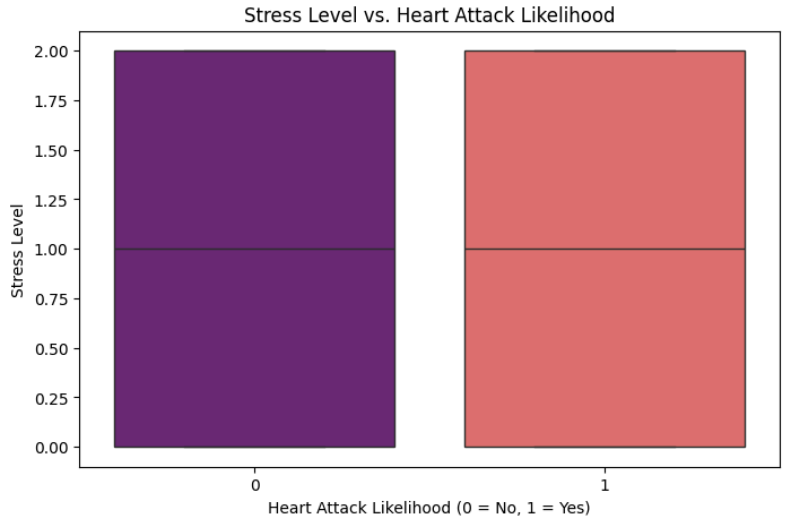
```
Start coding or generate with AI.
```

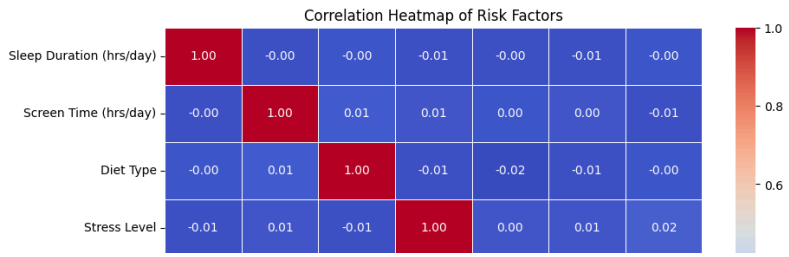


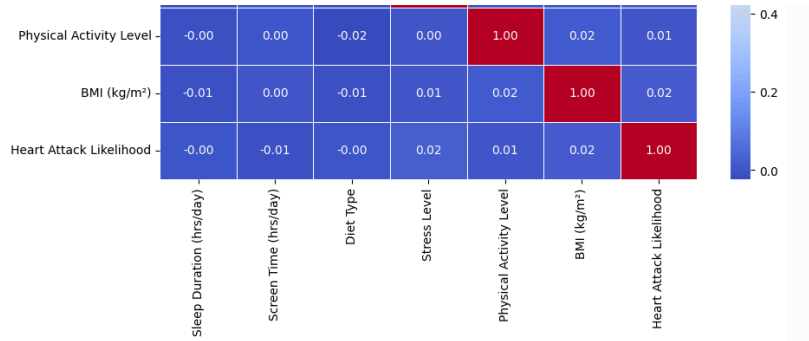

```
<ipython-input-11-3311393d136e>:27: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed i

  sns.boxplot(x="Heart Attack Likelihood", y="Screen Time (hrs/day)", data=df
```




```
<ipython-input-11-3311393d136e>:35: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed i

  sns.boxplot(x="Heart Attack Likelihood", y="Stress Level", data=df_filtered
```







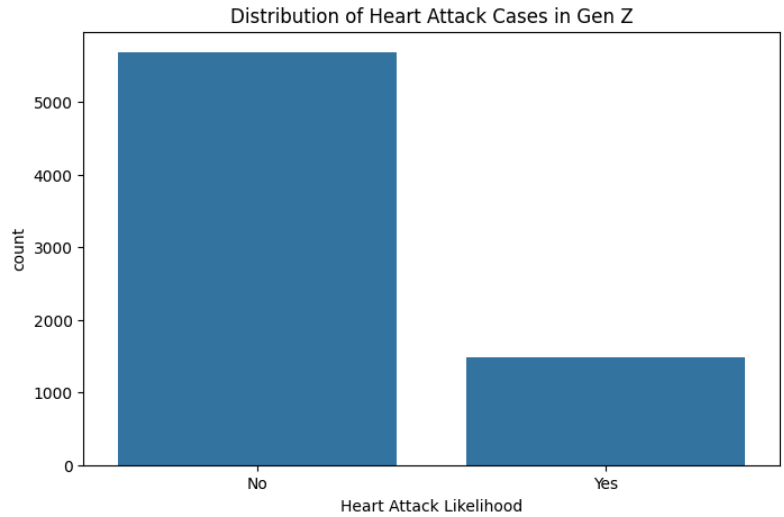

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency, ttest_ind
import statsmodels.api as sm

# Load data
df = pd.read_csv("heart_attack_youngsters_india.csv")

# Filter Gen Z (assuming age <=24)
df_genz = df[df["Age"] <= 30]

# --- Convert 'Heart Attack Likelihood' to numerical before groupby ---
# This is crucial to avoid the TypeError
df_genz['Heart Attack Likelihood'] = df_genz['Heart Attack Likelihood'].map({'No': 0,

# --- 1. Descriptive Analysis ---
print("=== Descriptive Statistics ===")
# Prevalence of risk factors
print("Sleep <6hrs:", round((df_genz["Sleep Duration (hrs/day)"] < 6).mean() * 100, 1
print("Screen Time >4hrs:", round((df_genz["Screen Time (hrs/day)"] > 4).mean() * 100
print("Unhealthy Diet:", round((df_genz["Diet Type"] == "Unhealthy").mean() * 100, 1)
print("Low Activity:", round((df_genz["Physical Activity Level"] == "Low").mean() * 1
print("High Stress:", round((df_genz["Stress Level"] == "High").mean() * 100, 1), "%"

# Heart attack rates by risk factors
print("\n=== Heart Attack Likelihood by Risk Factor ===")
print("Screen Time >4hrs:")
print(df_genz.groupby("Screen Time (hrs/day)")["Heart Attack Likelihood"].mean())
print("\nSleep <6hrs:")
print(df_genz.groupby("Sleep Duration (hrs/day)")["Heart Attack Likelihood"].mean())

# --- 2. Correlation Analysis ---
# ... (rest of the code remains the same)
```

```
=== Descriptive Statistics ===
Sleep <6hrs: 37.6 %
Screen Time >4hrs: 68.5 %
Unhealthy Diet: 0.0 %
Low Activity: 0.0 %
High Stress: 29.1 %

=== Heart Attack Likelihood by Risk Factor ===
Screen Time >4hrs:
```

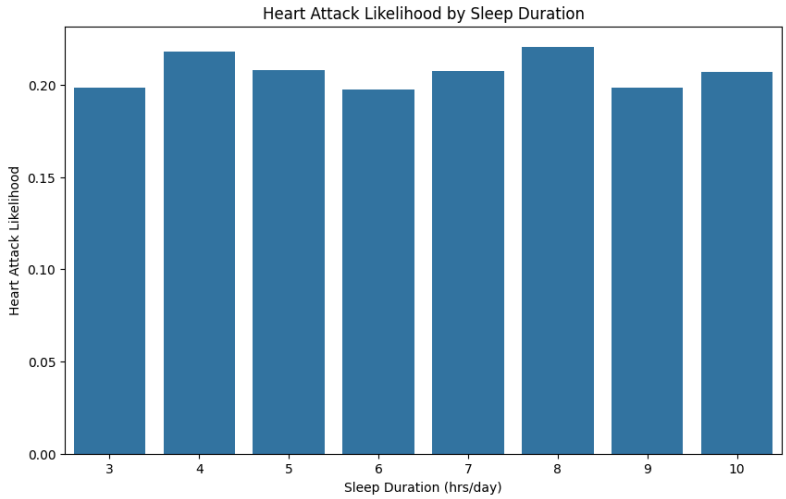
```
Screen Time (hrs/day)
0     0.207188
1     0.187225
2     0.205817
3     0.230415
4     0.205817
5     0.207792
6     0.217703
7     0.193258
8     0.210300
9     0.206089
10    0.192941
11    0.222222
12    0.217489
13    0.213220
14    0.193548
15    0.202299
Name: Heart Attack Likelihood, dtype: float64

Sleep <6hrs:
Sleep Duration (hrs/day)
3     0.198422
4     0.217920
5     0.207778
6     0.197309
7     0.207399
8     0.220573
9     0.198307
10    0.206972
Name: Heart Attack Likelihood, dtype: float64
<ipython-input-7-8d8e78534442>:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stal
  df_genz['Heart Attack Likelihood'] = df_genz['Heart Attack Likelihood'].map({
```

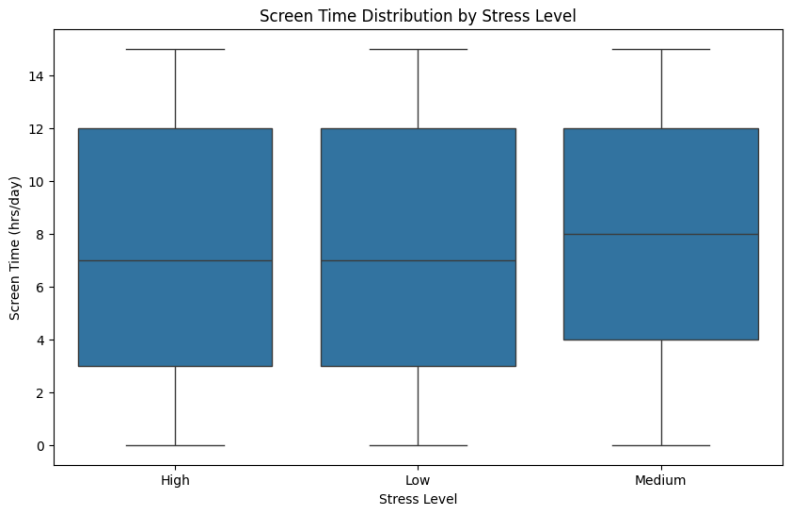
```
# Filter Gen Z (adjust age threshold if needed)
df_genz = df[df["Age"] <= 30]

# --- 1. Distribution of Heart Attack Likelihood ---
plt.figure(figsize=(8, 5))
sns.countplot(x="Heart Attack Likelihood", data=df_genz)
plt.title("Distribution of Heart Attack Cases in Gen Z")
plt.show()

# --- 2. Sleep Duration vs. Heart Attack Likelihood ---
plt.figure(figsize=(10, 6))
# --- Ensure 'Heart Attack Likelihood' is numerical for barplot ---
df_genz['Heart Attack Likelihood'] = df_genz['Heart Attack Likelihood'].map({'No': 0,
sns.barplot(
    x="Sleep Duration (hrs/day)",
    y="Heart Attack Likelihood",
    data=df_genz,
    ci=None
)
plt.title("Heart Attack Likelihood by Sleep Duration")
plt.show()

# --- 3. Screen Time vs. Stress Level ---
plt.figure(figsize=(10, 6))
sns.boxplot(
    x="Stress Level",
    y="Screen Time (hrs/day)",
    data=df_genz
)
plt.title("Screen Time Distribution by Stress Level")
plt.show()

# --- 4. Diet Type vs. Cholesterol Levels ---
plt.figure(figsize=(10, 6))
sns.boxplot(
    x="Diet Type",
    y="Cholesterol Levels (mg/dL)",
    hue="Heart Attack Likelihood",
    data=df_genz
)
plt.title("Cholesterol Levels by Diet Type (Heart Attack Cases Highlighted)")
plt.show()

# --- 6. Correlation Heatmap ---
corr_vars = df_genz[[
    "Screen Time (hrs/day)",
    "Sleep Duration (hrs/day)",
```

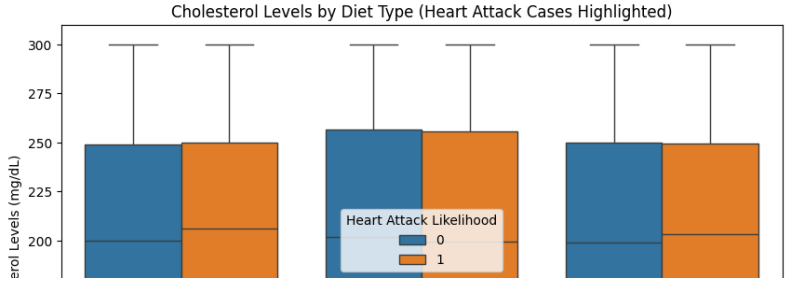
```
    "BMI (kg/m²)",
    "Cholesterol Levels (mg/dL)",
    "Triglyceride Levels (mg/dL)",
    "Heart Attack Likelihood"  # This column likely has 'Yes'/'No'
]].copy()  # Create a copy to avoid modifying the original df_genz

# Convert 'Heart Attack Likelihood' to numerical for correlation
corr_vars["Heart Attack Likelihood"] = corr_vars["Heart Attack Likelihood"].map({"No"

plt.figure(figsize=(12, 8))
sns.heatmap(corr_vars.corr(), annot=True, cmap="coolwarm", vmin=-1, vmax=1)
plt.title("Correlation Between Risk Factors and Heart Disease")
plt.show()

# --- 7. Heart Attack Cases by Risk Factor Combinations ---
# Create binary risk factors
df_genz["High Screen Time"] = (df_genz["Screen Time (hrs/day)"] > 4).astype(int)
df_genz["Poor Sleep"] = (df_genz["Sleep Duration (hrs/day)"] < 6).astype(int)
df_genz["Unhealthy Diet"] = (df_genz["Diet Type"] == "Unhealthy").astype(int)

# Aggregate data
# --- Ensure 'Heart Attack Likelihood' is numerical before groupby ---
df_genz['Heart Attack Likelihood'] = df_genz['Heart Attack Likelihood'].map({'No': 0,
risk_factors = df_genz.groupby(
    ["High Screen Time", "Poor Sleep", "Unhealthy Diet"]
)["Heart Attack Likelihood"].mean().reset_index()

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(
    x="High Screen Time",
    y="Heart Attack Likelihood",
    hue="Unhealthy Diet",
    data=risk_factors
)
plt.title("Heart Attack Likelihood by Combined Risk Factors")
plt.show()
```

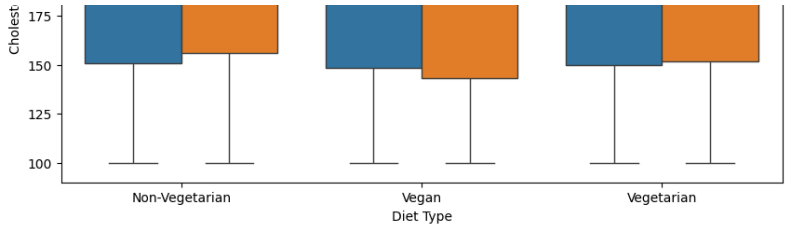



```
<ipython-input-10-6a47dd7505be>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  df_genz['Heart Attack Likelihood'] = df_genz['Heart Attack Likelihood'].map
<ipython-input-10-6a47dd7505be>:14: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(
```

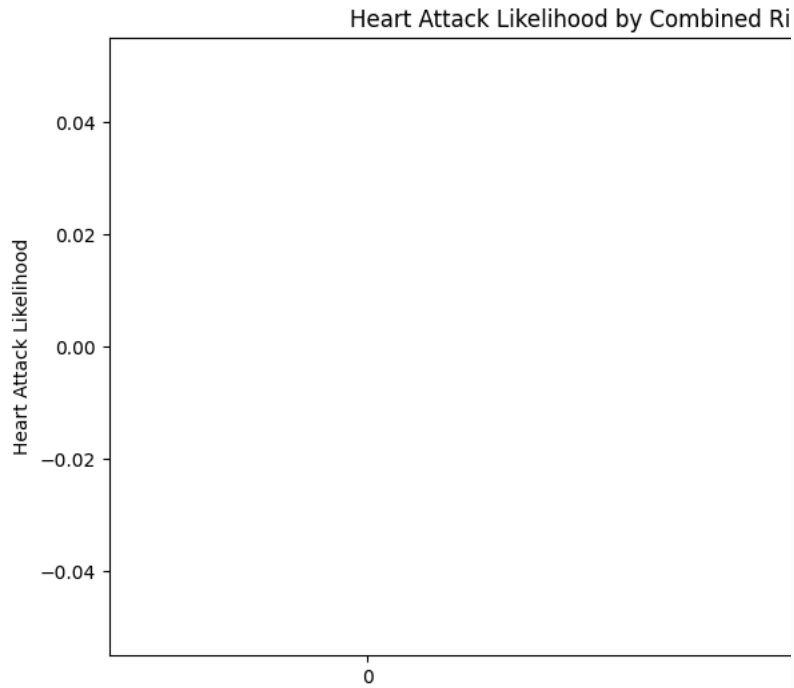


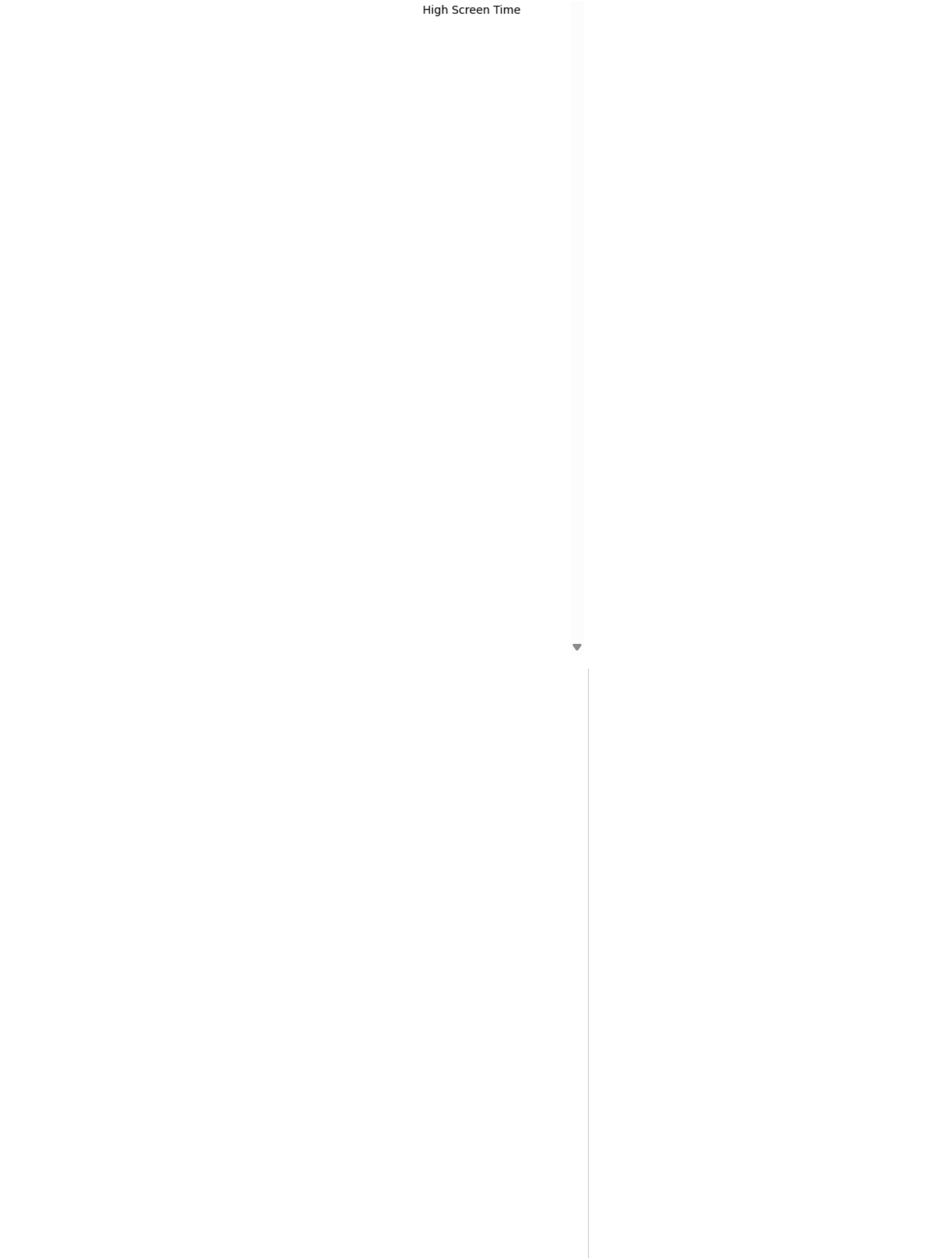







```
<ipython-input-10-6a47dd7505be>:64: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  df_genz["High Screen Time"] = (df_genz["Screen Time (hrs/day)"] > 4).astype
<ipython-input-10-6a47dd7505be>:65: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  df_genz["Poor Sleep"] = (df_genz["Sleep Duration (hrs/day)"] < 6).astype(in
<ipython-input-10-6a47dd7505be>:66: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  df_genz["Unhealthy Diet"] = (df_genz["Diet Type"] == "Unhealthy").astype(in
<ipython-input-10-6a47dd7505be>:70: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  df_genz['Heart Attack Likelihood'] = df_genz['Heart Attack Likelihood'].map
```




High Screen Time