

```
In [1]: import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings("ignore")
```

```
In [2]: df = pd.read_csv("fitness_dataset.csv")
print(df.head())
print(df.info())
print(df.columns)
```

	age	height_cm	weight_kg	heart_rate	blood_pressure	sleep_hours	\
0	56	152	65	69.6	117.0	NaN	
1	69	186	95	60.8	114.8	7.5	
2	46	192	103	61.4	116.4	NaN	
3	32	189	83	60.2	130.1	7.0	
4	60	175	99	58.1	115.8	8.0	

	nutrition_quality	activity_index	smokes	gender	is_fit
0	2.37	3.97	no	F	1
1	8.77	3.19	0	F	1
2	8.20	2.03	0	F	0
3	6.18	3.68	0	M	1
4	9.95	4.83	yes	F	1

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2000 entries, 0 to 1999

Data columns (total 11 columns):

#	Column	Non-Null Count	Dtype
0	age	2000 non-null	int64
1	height_cm	2000 non-null	int64
2	weight_kg	2000 non-null	int64
3	heart_rate	2000 non-null	float64
4	blood_pressure	2000 non-null	float64
5	sleep_hours	1840 non-null	float64
6	nutrition_quality	2000 non-null	float64
7	activity_index	2000 non-null	float64
8	smokes	2000 non-null	object
9	gender	2000 non-null	object
10	is_fit	2000 non-null	int64

dtypes: float64(5), int64(4), object(2)

memory usage: 172.0+ KB

None

```
Index(['age', 'height_cm', 'weight_kg', 'heart_rate', 'blood_pressure',
      'sleep_hours', 'nutrition_quality', 'activity_index', 'smokes',
      'gender', 'is_fit'],
      dtype='object')
```

```
In [3]: # Getting the count of missing values
print(df.isnull().sum())
```

```

age                0
height_cm          0
weight_kg          0
heart_rate         0
blood_pressure     0
sleep_hours       160
nutrition_quality   0
activity_index     0
smokes            0
gender            0
is_fit            0
dtype: int64

```

```

In [5]: # Deleting columns that have 30% missing values
missing_percentage = df.isnull().mean() * 100
threshold = 30

# Filter columns with more than the threshold percentage of missing values
columns_to_drop = missing_percentage[missing_percentage > threshold].index

# Drop the columns
df = df.drop(columns=columns_to_drop)

print(f"Columns dropped: {list(columns_to_drop)}")
print(f"Remaining columns: {df.columns}")

```

```

Columns dropped: []
Remaining columns: Index(['age', 'height_cm', 'weight_kg', 'heart_rate', 'blood_p
ressure',
                        'sleep_hours', 'nutrition_quality', 'activity_index', 'smokes',
                        'gender', 'is_fit'],
                        dtype='object')

```

```

In [6]: from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
for column in df.select_dtypes(include=['object']).columns:
    df[column] = label_encoder.fit_transform(df[column])

print("\nData after label encoding (head):")
print(df.head())

```

```

Data after label encoding (head):
   age  height_cm  weight_kg  heart_rate  blood_pressure  sleep_hours \
0   56        152         65        69.6           117.0         NaN
1   69        186         95        60.8           114.8          7.5
2   46        192        103        61.4           116.4         NaN
3   32        189         83        60.2           130.1          7.0
4   60        175         99        58.1           115.8          8.0

   nutrition_quality  activity_index  smokes  gender  is_fit
0                2.37            3.97      2       0       1
1                8.77            3.19      0       0       1
2                8.20            2.03      0       0       0
3                6.18            3.68      0       1       1
4                9.95            4.83      3       0       1

```

```

In [7]: #Filling missing values based on feature type
# Fill missing values in float64 columns with mean
for column in df.select_dtypes(include=['float64']).columns:
    mean_value = df[column].mean()

```

```

df[column] = df[column].fillna(mean_value)

# Fill missing values in categorical columns (if any remain as object) with mode
for column in df.select_dtypes(include=['object']).columns:
    mode_value = df[column].mode()[0]
    df[column] = df[column].fillna(mode_value)

print("\nMissing values after handling:")
print(df.isnull().sum())

```

Missing values after handling:

```

age                0
height_cm          0
weight_kg          0
heart_rate         0
blood_pressure     0
sleep_hours        0
nutrition_quality  0
activity_index     0
smokes             0
gender             0
is_fit             0
dtype: int64

```

```

In [9]: #Handle outliers
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
numeric_cols_without_target = [c for c in numeric_cols if c != "is_fit"]
print("\nNumeric columns:")
print(numeric_cols)

# Identify outliers using the IQR rule
print("\nPotential outliers detected using IQR rule:")
for col in numeric_cols_without_target:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers_count = ((df[col] < lower_bound) | (df[col] > upper_bound)).sum()
    print(f" - {col}: {outliers_count} potential outliers")

# Visual boxplots for each numeric feature
sns.set(style="whitegrid")
plt.figure(figsize=(16, 10))
for i, col in enumerate(numeric_cols_without_target, start=1):
    plt.subplot((len(numeric_cols_without_target) // 3) + 1, 3, i)
    sns.boxplot(x=df[col])
    plt.title(f"Boxplot of {col}")
plt.tight_layout()
plt.show()

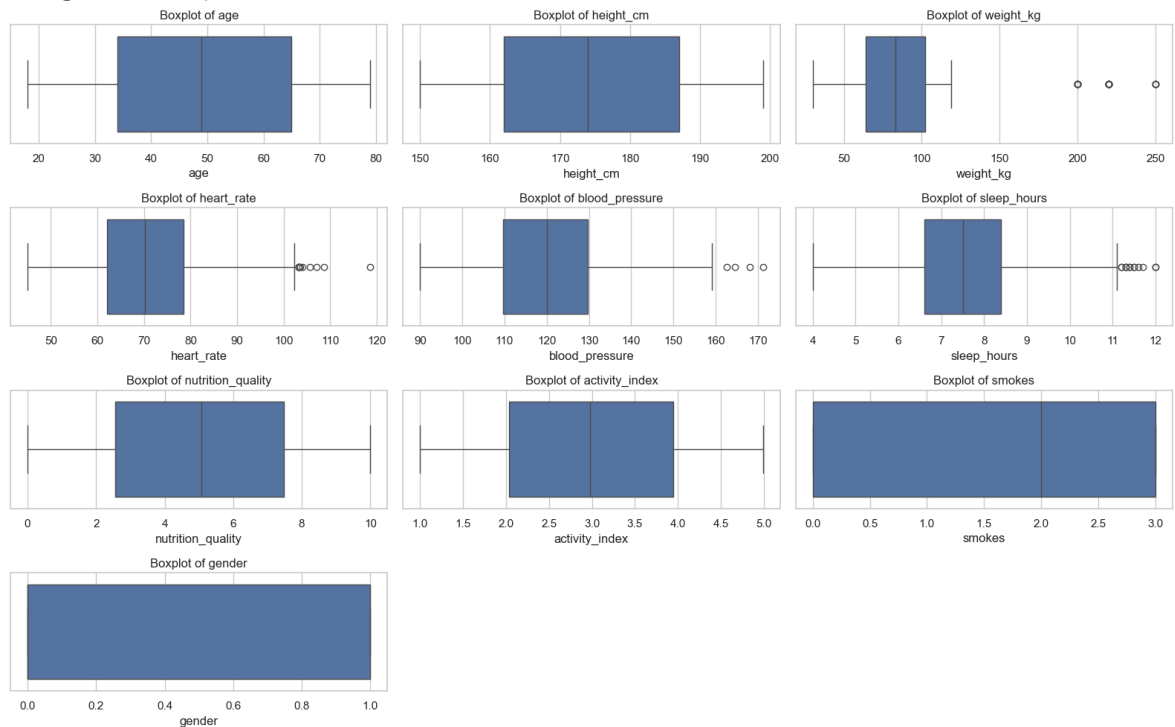
```

Numeric columns:

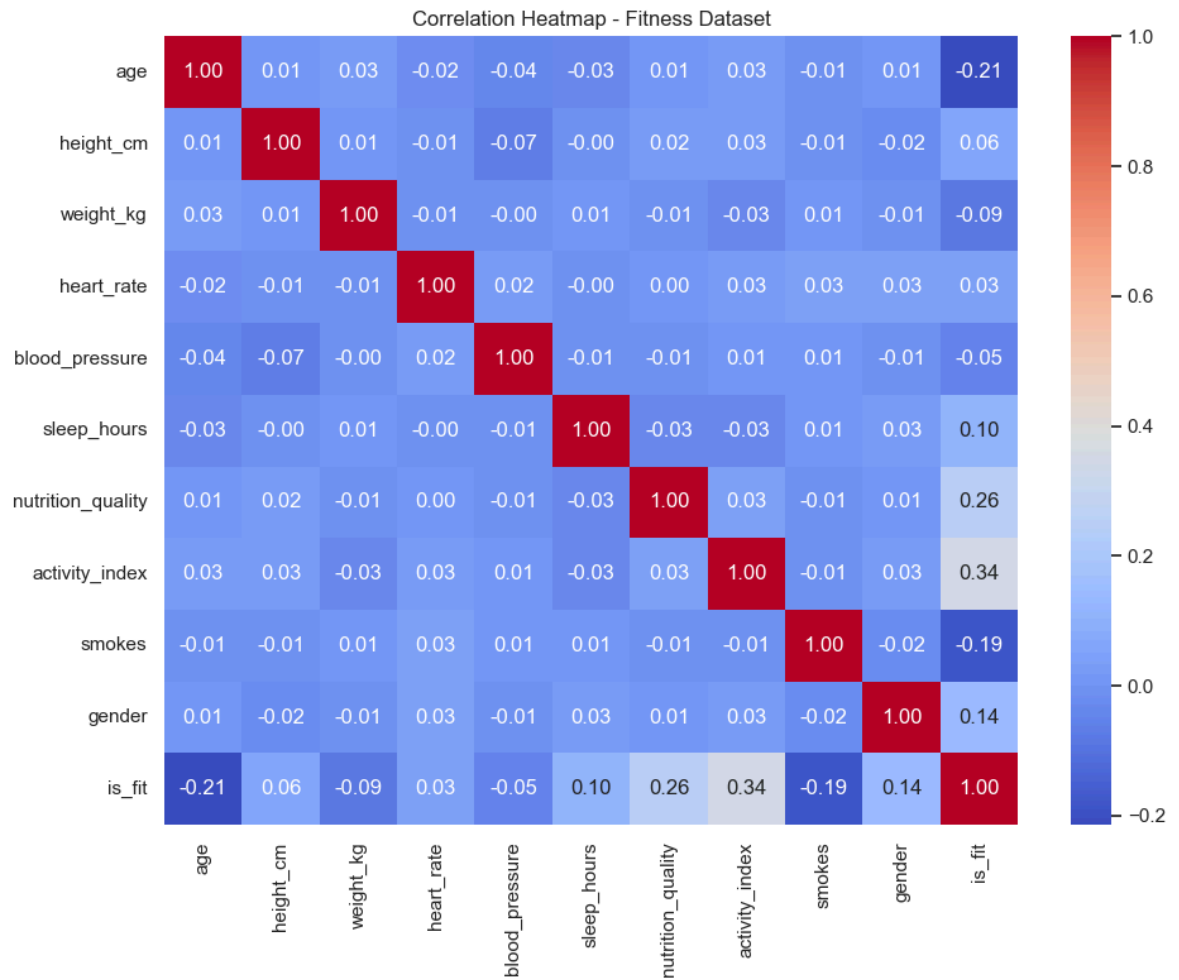
['age', 'height_cm', 'weight_kg', 'heart_rate', 'blood_pressure', 'sleep_hours', 'nutrition_quality', 'activity_index', 'smokes', 'gender', 'is_fit']

Potential outliers detected using IQR rule:

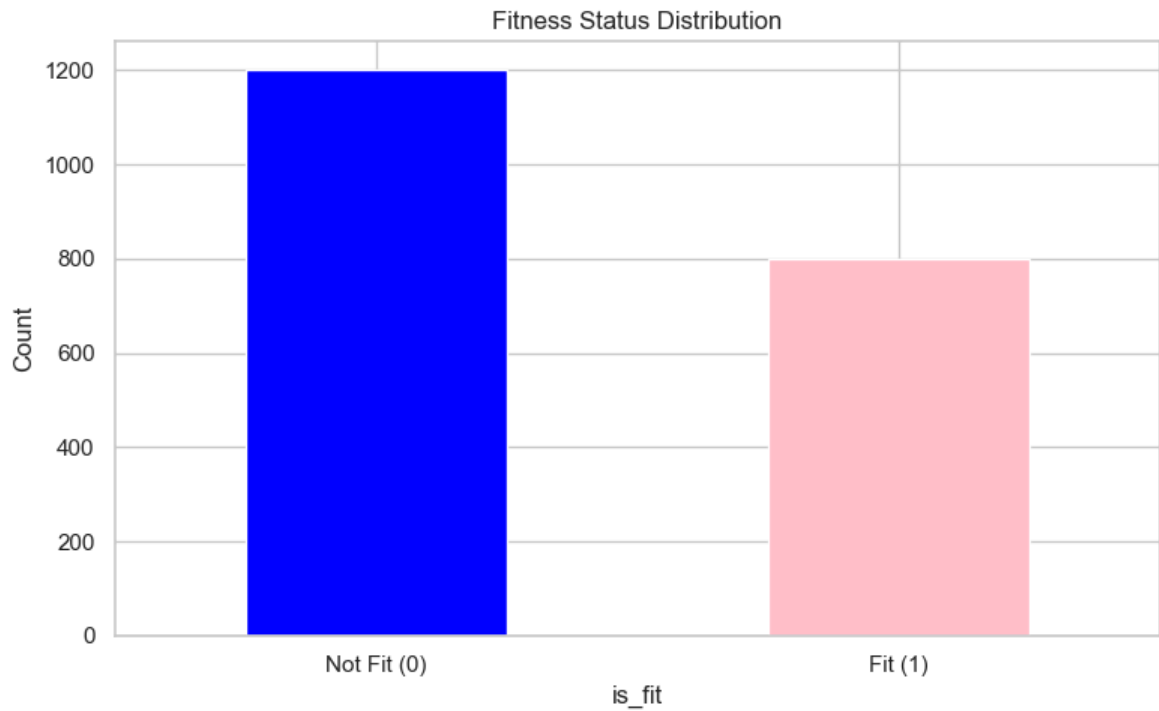
- age: 0 potential outliers
- height_cm: 0 potential outliers
- weight_kg: 21 potential outliers
- heart_rate: 10 potential outliers
- blood_pressure: 4 potential outliers
- sleep_hours: 13 potential outliers
- nutrition_quality: 0 potential outliers
- activity_index: 0 potential outliers
- smokes: 0 potential outliers
- gender: 0 potential outliers



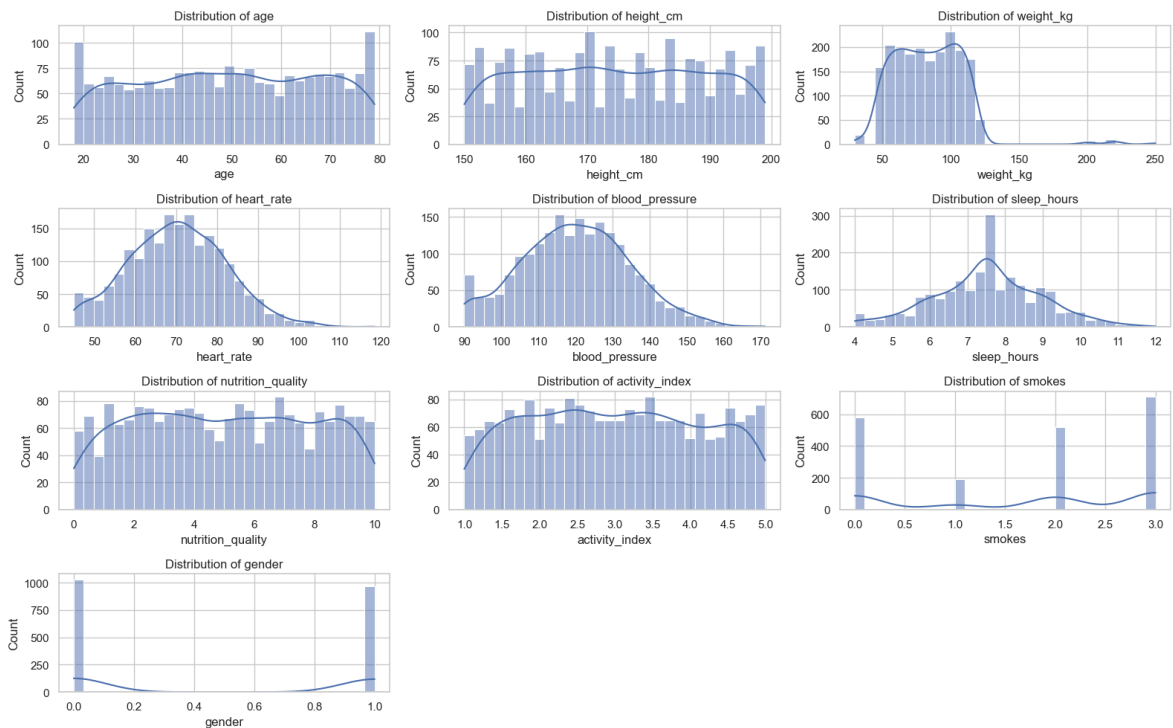
```
In [10]: #Data visualization
# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap - Fitness Dataset")
plt.tight_layout()
plt.show()
```



```
In [11]: #Bar chart of the target (similar to classification bar chart)
plt.figure(figsize=(8, 5))
df["is_fit"].value_counts().plot(kind="bar", color=["blue", "pink"])
plt.xticks([0, 1], ["Not Fit (0)", "Fit (1)"], rotation=0)
plt.title("Fitness Status Distribution")
plt.xlabel("is_fit")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```



```
In [12]: # Histograms for numeric features
plt.figure(figsize=(16, 10))
for i, col in enumerate(numeric_cols_without_target, start=1):
    plt.subplot((len(numeric_cols_without_target) // 3) + 1, 3, i)
    sns.histplot(df[col], bins=30, kde=True)
    plt.title(f"Distribution of {col}")
plt.tight_layout()
plt.show()
```

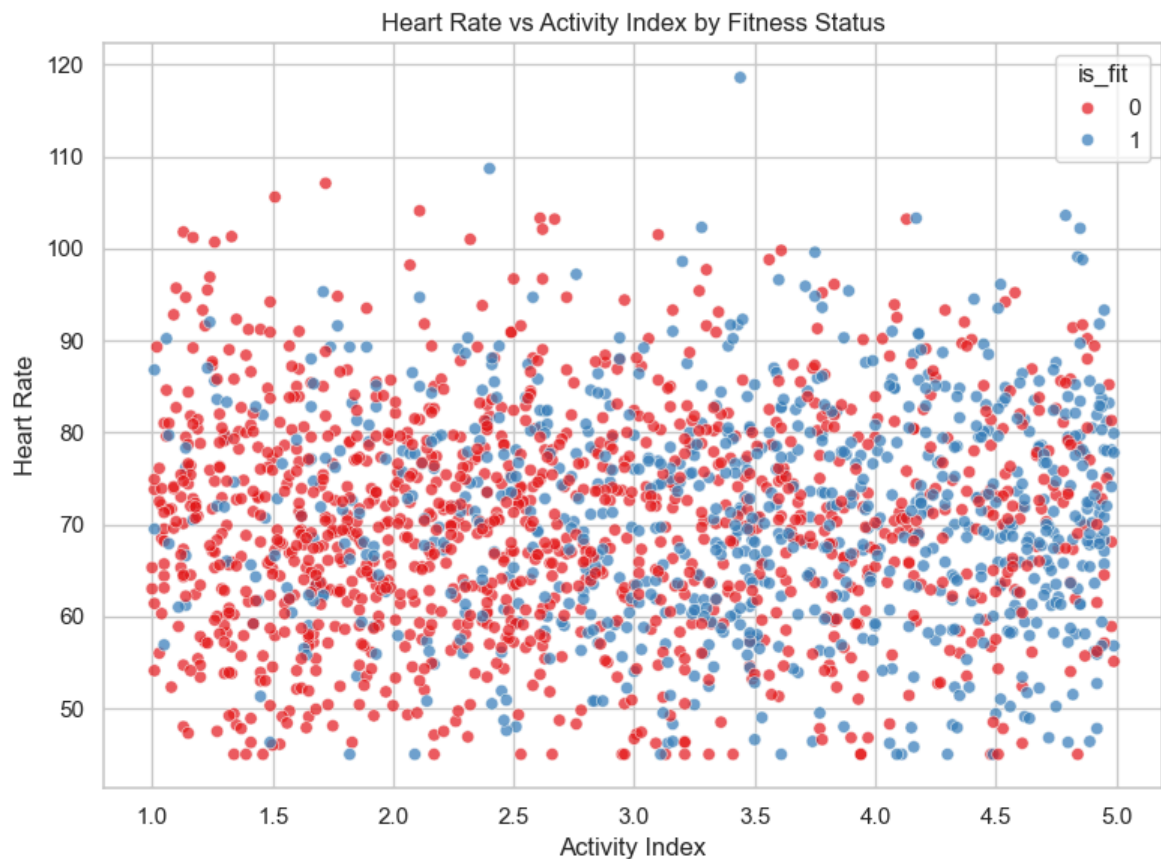


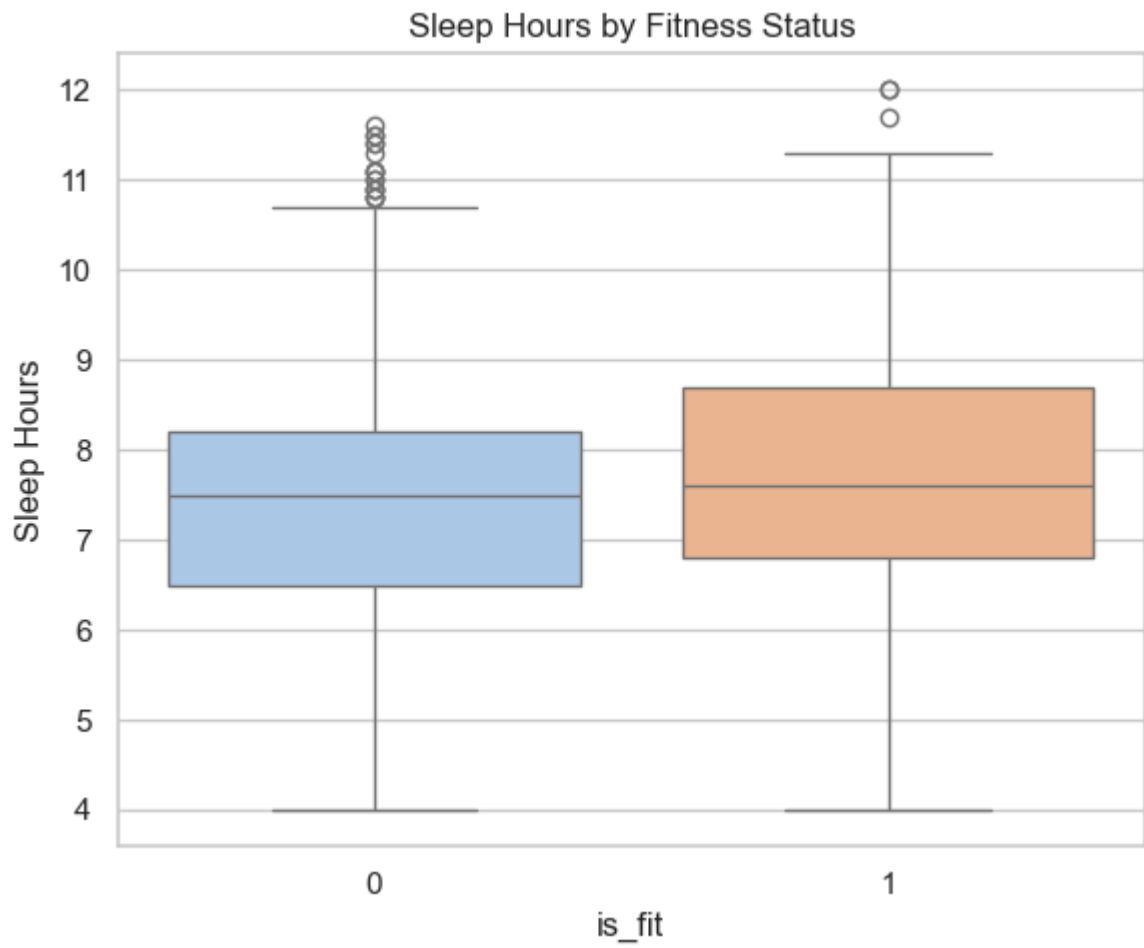
```
In [13]: # Relationship plots

# Scatter: Heart rate vs activity index by fitness status
if {"activity_index", "heart_rate"}.issubset(df.columns):
    plt.figure(figsize=(8, 6))
    sns.scatterplot(
```

```
x="activity_index",
y="heart_rate",
hue="is_fit",
data=df,
palette="Set1",
alpha=0.7,
)
plt.title("Heart Rate vs Activity Index by Fitness Status")
plt.xlabel("Activity Index")
plt.ylabel("Heart Rate")
plt.tight_layout()
plt.show()

# Boxplot: Sleep hours by fitness status
if "sleep_hours" in df.columns:
    plt.figure(figsize=(6, 5))
    sns.boxplot(x="is_fit", y="sleep_hours", data=df, palette="pastel")
    plt.title("Sleep Hours by Fitness Status")
    plt.xlabel("is_fit")
    plt.ylabel("Sleep Hours")
    plt.tight_layout()
    plt.show()
```





In []: