Correction

Change 1:

For zero division problem of Question 7, to prevent a division by zero error in the calculation of the following code:

```
male_high_income_percentage = (
    data[(data['Gender'] == 'Male') & (data['Income'] > 75000)].shape[0] /
    data[data['Gender'] == 'Male'].shape[0]
)
```

We added an if-else statement to control that the calculation only proceeds if the condition total_male_customers > 0 is met. This ensures the denominator is not zero before attempting to compute the percentage of male customers with an income greater than $75,000.

This is our final code for question7 after correction:

```
# 7.What percentage of Male customers have an income greater than $75,000?

### BEGIN SOLUTION
# First, calculate the total number of male customers
total_male_customers = data[data['Gender'] == 'Male'].shape[0]

if total_male_customers > 0:
    # If there are male customers, calculate the number of male customers with an income greater than $75,000
    male_high_income_count = data[(data['Gender'] == 'Male') & (data['Income'] > 75000)].shape[0]
    # Calculate the percentage
    male_high_income_percentage = male_high_income_count / total_male_customers
    print("Percentage of Male with Income > $75,000:", male_high_income_percentage)
else:
    # If there are no male customers, output 0 or a corresponding message
    print("No male customers available for analysis.")
### END SOLUTION
```

Change 2:

For zero division problem of Question 8, to prevent a division by zero error in the calculation of the following code:

```
older_than_50_percentage = data[data['Age'] > 50].shape[0] / data.shape[0]
```

We also added an if-else statement to ensure that the calculation only proceeds if the condition data.shape[0] > 0 is met. This ensures that the denominator is not zero before attempting to compute the percentage of customers older than 50.

This is our code for question7 after correction:

```
# 8.What percentage of customers are older than 50 years (in float, e.g., 0.15)?

### BEGIN SOLUTION
# 检查数据集是否为空
if data.shape[0] > 0:
    # 计算年龄大于50岁的顾客占比
    older_than_50_percentage = data[data['Age'] > 50].shape[0] / data.shape[0]
    # 格式化输出百分比，保留两位小数
    print(f"Percentage of Customers Older than 50: {older_than_50_percentage*100:.2f}%")
else:
    # 如果数据集为空，输出提示信息
    print("No data available.")
### END SOLUTION
```

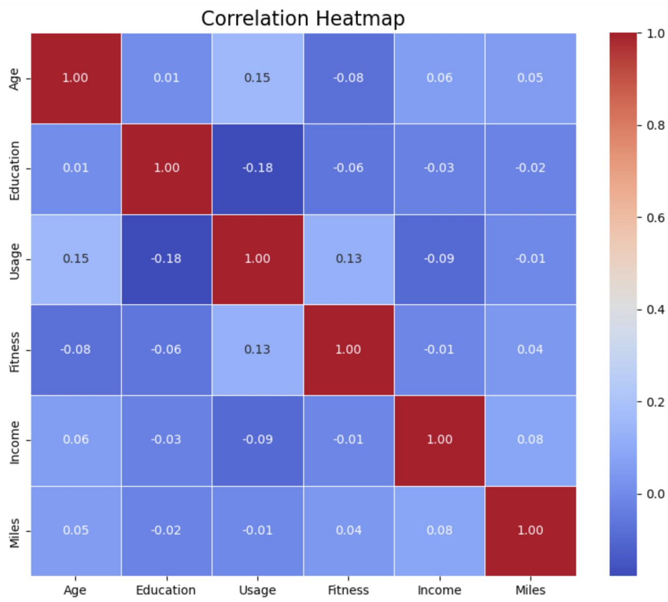
Change 3:

In visualization part, we change our Heatmap from covariance to correlation:

This is our code after correction:

```
# 3. Draw heat map for all variables.

### BEGIN SOLUTION
# Select numeric features
numeric_columns = data.select_dtypes(include=["float64", "int64"]).columns

# Calculate the covariance matrix
correlation_matrix = data[numeric_columns].corr()

# Draw a covariance heat map
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

# Set the title
plt.title("Correlation Heatmap", fontsize=16)

plt.show()
### END SOLUTION
```

This is the heatmap after correction:



From the correlation heatmap, we can conclude that:

1) Strong positive correlation: Usage and Age (0.15).
2) Strong negative correlation: Education and Usage (-0.18).
3) Negligible correlation: Age and Education (0.01) & Fitness and Income (-0.01).

Most variables show weak or no linear relationships, aiding further analysis and modeling.

Change 4:

For Question 11 and 12 optional questions:

To further validate the performance and accuracy of the decision tree model, we split the training data and calculate the specific accuracy.

1. Dataset Splitting

The dataset is split into training and validation sets in an 8:2 ratio:

```
# Split the dataset into training and validation sets (8:2 ratio)
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)
```

This ensures the training and validation datasets are separated to evaluate model performance.

2. Model Evaluation

The model is evaluated on the validation set, and its accuracy is calculated:

```
# Evaluate model performance using the validation set
y_valid_pred = clf.predict(X_valid)
accuracy = accuracy_score(y_valid, y_valid_pred)

# Output the model accuracy on the validation set
print(f"Model accuracy on the validation set: {accuracy:.2f}")
```

The accuracy_score function is used to calculate the model's performance on the validation data.