22510064 PARSHWA HERWADE

TY CSE T4

Assignment 3: Univariate Outlier Detection and Analysis

a. Identify the top 50 female heights in the distributions generated in Assignment 1, and increase the height of these female samples by 10 cm each.

b. Observe the changes in sample mean and standard deviation after altering the heights.

c. Run the classification algorithms developed in Assignment 1.c on this altered dataset, and note the change in classification accuracy for each case.

d. Design strategies to detect outliers in the female sample set:

- **Visual Methods:**
    1. Plot the data histogram and observe any gaps, elbows, or unusual patterns.
    2. Create a box-and-whisker plot and use the whiskers to identify potential outliers.
- **Parametric Methods:**
    1. Convert the heights into z-scores.
    2. Experiment with z-score cutoffs (e.g., 2 and 3 on both sides).
- **Non-Parametric Methods:**
    1. Detect and remove outliers based on the interquartile range (IQR).
    2. Detect outliers using the Median Absolute Deviation (MAD).
    3. Experiment with different cutoff values (e.g., 1.5, 2, 3 on both sides).

e. Remove data labeled as outliers using the z-score, IQR, or MAD methods.

f. Run the classification methods from Assignment 1.c again and document the impact on the mean, standard deviation, and classification accuracy.

g. **Data Trimming:**
Drop the lower and upper k% of data (varying k from 1% to 15% in increments of 1%) from the dataset generated in part (a), and run the classification algorithms. Observe the impact on accuracy using a scatter plot.

CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm

# Generate synthetic height data using a normal distribution.
def generate_heights(mean, std_dev, size, label):
    heights = np.random.normal(mean, std_dev, size)
    labels = [label] * size
    return pd.DataFrame({'height': heights, 'label': labels})

# Plot overlaid histograms for female and male data.
def plot_histograms(female_heights, male_heights, bins=50, title_suffix=""):
    plt.figure()
    plt.hist([female_heights, male_heights], bins=bins, label=['Female', 'Male'],
             alpha=0.7, color=['purple', 'green'])
    plt.title(f'Height Distributions {title_suffix}')
    plt.xlabel('Height (cm)')
    plt.ylabel('Frequency')
    plt.legend(loc='upper right')
    plt.show()

# Simple threshold-based classifier.
def threshold_classifier(female_data, male_data, threshold):
    combined = list(female_data) + list(male_data)
    predictions = ['F' if x < threshold else 'M' for x in combined]
```

```
    actual = ['F'] * len(female_data) + ['M'] * len(male_data)
    return actual, predictions

# Probability classifier using the normal PDF.
def probability_classifier(female_data, male_data, female_mean, female_sd, male_mean, male_sd):
    def classify(height):
        female_prob = norm.pdf(height, female_mean, female_sd)
        male_prob = norm.pdf(height, male_mean, male_sd)
        return 'F' if female_prob > male_prob else 'M'
    combined = list(female_data) + list(male_data)
    predictions = [classify(x) for x in combined]
    actual = ['F'] * len(female_data) + ['M'] * len(male_data)
    return actual, predictions

# Quantized classifier that groups data into intervals using integer division.
def quantized_classifier(female_data, male_data, interval_len):
    def quantize(data):
        return [int(x // interval_len) for x in data]
    female_intervals = quantize(female_data)
    male_intervals = quantize(male_data)
    all_intervals = sorted(set(female_intervals + male_intervals))
    predictions = []
    actual = []
    for interval in all_intervals:
        female_count = female_intervals.count(interval)
        male_count = male_intervals.count(interval)
        majority_label = 'F' if female_count >= male_count else 'M'
```

```
        predictions.extend([majority_label] * (female_count + male_count))
        actual.extend(['F'] * female_count + ['M'] * male_count)
    return actual, predictions

# Evaluate classifier performance by computing accuracy.
def evaluate_classifier(actual, predictions, description=""):
    accuracy = sum(1 for a, p in zip(actual, predictions) if a == p) / len(actual)
    return accuracy

# Detect outliers using the z-score method.
def detect_outliers_zscore(data, cutoff=2):
    mean_val = np.mean(data)
    std_val = np.std(data)
    z_scores = (data - mean_val) / std_val
    return [i for i, z in enumerate(z_scores) if abs(z) > cutoff]

# Detect outliers using the IQR method with a simple percentile calculation.
def detect_outliers_iqr(data, factor=1.5):
    sdata = sorted(data)
    n = len(sdata)
    Q1 = sdata[int(0.25 * n)]
    Q3 = sdata[int(0.75 * n)]
    IQR = Q3 - Q1
    lower_bound = Q1 - factor * IQR
    upper_bound = Q3 + factor * IQR
    return [i for i, x in enumerate(data) if x < lower_bound or x > upper_bound]

# Detect outliers using the MAD method (implemented with basic loops).
```

```
def detect_outliers_mad(data, cutoff=3):
    sdata = sorted(data)
    median_val = sdata[len(sdata) // 2]
    deviations = [abs(x - median_val) for x in data]
    sdev = sorted(deviations)
    mad = sdev[len(sdev) // 2]
    if mad == 0:
        return []
    modified_z_scores = [0.6745 * (x - median_val) / mad for x in data]
    return [i for i, z in enumerate(modified_z_scores) if abs(z) > cutoff]

# Main function orchestrating the workflow.
def run_code():
    female_mean = 152
    male_mean = 166
    sd = 7.5
    sample_size = 1000

    # Generate female and male data.
    df_female = generate_heights(female_mean, sd, sample_size, 'F')
    df_male = generate_heights(male_mean, sd, sample_size, 'M')
    female_data = df_female['height'].values
    male_data = df_male['height'].values

    # --- Part (a) & (b): Original vs. Altered Data ---
    fig, axes = plt.subplots(1, 2, figsize=(12, 6))
    axes[0].hist([female_data, male_data], bins=30, label=['Female', 'Male'],
                 alpha=0.7, color=['purple', 'green'])
```

```
    axes[0].set_title("Original Data")
    axes[0].set_xlabel("Height (cm)")
    axes[0].set_ylabel("Frequency")
    axes[0].legend()

    # Increase the top 50 female heights by 10 cm.
    indices_top50 = sorted(range(len(female_data)), key=lambda i: female_data[i])[-50:]
    female_data_altered = female_data.copy()
    for i in indices_top50:
        female_data_altered[i] += 10

    axes[1].hist([female_data_altered, male_data], bins=30, label=['Female', 'Male'],
                 alpha=0.7, color=['purple', 'green'])
    axes[1].set_title("Altered Data")
    axes[1].set_xlabel("Height (cm)")
    axes[1].set_ylabel("Frequency")
    axes[1].legend()
    plt.tight_layout()
    plt.show()

    print("=== Female Data Statistics ===")
    print("Before alteration: Mean = {:.2f}, SD = {:.2f}".format(np.mean(female_data),
np.std(female_data)))
    print("After alteration:  Mean = {:.2f}, SD = {:.2f}".format(np.mean(female_data_altered),
np.std(female_data_altered)))

    # --- Box Plot Comparison: Original vs. Altered Data ---
    df_original = pd.concat([df_female, df_male], ignore_index=True)
    df_female_altered = df_female.copy()
```

```
    indices_top50 = sorted(range(len(df_female_altered['height'])), key=lambda i:
df_female_altered['height'][i])[-50:]
    for i in indices_top50:
        df_female_altered.at[i, 'height'] += 10
    df_altered = pd.concat([df_female_altered, df_male], ignore_index=True)


    fig, axes = plt.subplots(1, 2, figsize=(12, 6))
    sns.boxplot(x='label', y='height', hue='label', data=df_original,
                palette={'F':'purple','M':'green'}, dodge=False, ax=axes[0])
    axes[0].set_title("Original Data Box Plot")
    axes[0].set_xlabel("Gender")
    axes[0].set_ylabel("Height (cm)")
    if axes[0].get_legend() is not None:
        axes[0].get_legend().remove()

    sns.boxplot(x='label', y='height', hue='label', data=df_altered,
                palette={'F':'purple','M':'green'}, dodge=False, ax=axes[1])
    axes[1].set_title("Altered Data Box Plot")
    axes[1].set_xlabel("Gender")
    axes[1].set_ylabel("Height (cm)")
    if axes[1].get_legend() is not None:
        axes[1].get_legend().remove()

    plt.tight_layout()
    plt.show()

    # --- Part (d): Outlier Detection in Altered Female Data ---
    plt.figure()
```

```
plt.hist(female_data_altered, bins=30, color='purple', edgecolor='black', alpha=0.7)
plt.title("Visual Outlier Detection: Histogram of Altered Female Heights")
plt.xlabel("Height (cm)")
plt.ylabel("Frequency")
plt.show()

plt.figure()
plt.boxplot(female_data_altered, patch_artist=True, boxprops=dict(facecolor='purple'))
plt.title("Visual Outlier Detection: Boxplot of Altered Female Heights")
plt.xlabel("Altered Female Data")
plt.show()

print("\n=== Parametric Outlier Detection (z-score) ===")
for cutoff in [2, 3]:
    outliers_z = detect_outliers_zscore(female_data_altered, cutoff)
    print(f"Z-score cutoff {cutoff}: {len(outliers_z)} outliers detected")

print("\n=== Non-Parametric Outlier Detection (IQR) ===")
for factor in [1.5, 2, 3]:
    outliers_iqr = detect_outliers_iqr(female_data_altered, factor)
    print(f"IQR factor {factor}: {len(outliers_iqr)} outliers detected")

print("\n=== Non-Parametric Outlier Detection (MAD) ===")
for cutoff in [1.5, 2, 3]:
    outliers_mad = detect_outliers_mad(female_data_altered, cutoff)
    print(f"MAD cutoff {cutoff}: {len(outliers_mad)} outliers detected")

# Remove outliers using a z-score cutoff of 3.
```

```
    outlier_indices = detect_outliers_zscore(female_data_altered, cutoff=3)
    female_data_clean = [x for i, x in enumerate(female_data_altered) if i not in
outlier_indices]
    print("\nAfter removing outliers (z-score cutoff = 3):")
    print("Clean Female Data: Mean = {:.2f}, SD = {:.2f}".format(np.mean(female_data_clean),
np.std(female_data_clean)))


    plt.figure()
    plt.hist(female_data_clean, bins=30, color='purple', edgecolor='black', alpha=0.7)
    plt.title("Histogram of Cleaned Female Heights (Outliers Removed)")
    plt.xlabel("Height (cm)")
    plt.ylabel("Frequency")
    plt.show()


    plt.figure()
    plt.boxplot(female_data_clean, patch_artist=True, boxprops=dict(facecolor='purple'))
    plt.title("Boxplot of Cleaned Female Heights (Outliers Removed)")
    plt.xlabel("Cleaned Female Data")
    plt.show()


    # --- Part (c) & (f): Run Classification on Altered and Cleaned Data ---
    threshold_val = (np.mean(female_data_altered) + np.mean(male_data)) / 2
    actual_thr, predictions_thr = threshold_classifier(female_data_altered, male_data,
threshold_val)
    acc_thr = evaluate_classifier(actual_thr, predictions_thr)
    actual_prob, predictions_prob = probability_classifier(female_data_altered, male_data,
                                        np.mean(female_data_altered),
np.std(female_data_altered),
                                        np.mean(male_data), np.std(male_data))
```

```
    acc_prob = evaluate_classifier(actual_prob, predictions_prob)
    actual_quant, predictions_quant = quantized_classifier(female_data_altered, male_data, interval_len=1)
    acc_quant = evaluate_classifier(actual_quant, predictions_quant)
    print("\nClassification Results on Altered Data:")
    print("Threshold Classifier Accuracy: {:.2f}%".format(acc_thr * 100))
    print("Probability Classifier Accuracy: {:.2f}%".format(acc_prob * 100))
    print("Quantized Classifier Accuracy: {:.2f}%".format(acc_quant * 100))

    threshold_clean = (np.mean(female_data_clean) + np.mean(male_data)) / 2
    actual_thr_clean, predictions_thr_clean = threshold_classifier(female_data_clean, male_data, threshold_clean)
    acc_thr_clean = evaluate_classifier(actual_thr_clean, predictions_thr_clean)
    actual_prob_clean, predictions_prob_clean = probability_classifier(female_data_clean, male_data,
                                                np.mean(female_data_clean), np.std(female_data_clean),
                                                np.mean(male_data), np.std(male_data))
    acc_prob_clean = evaluate_classifier(actual_prob_clean, predictions_prob_clean)
    actual_quant_clean, predictions_quant_clean = quantized_classifier(female_data_clean, male_data, interval_len=1)
    acc_quant_clean = evaluate_classifier(actual_quant_clean, predictions_quant_clean)
    print("\nClassification Results on Cleaned Data (Outliers Removed):")
    print("Threshold Classifier Accuracy: {:.2f}%".format(acc_thr_clean * 100))
    print("Probability Classifier Accuracy: {:.2f}%".format(acc_prob_clean * 100))
    print("Quantized Classifier Accuracy: {:.2f}%".format(acc_quant_clean * 100))

    # --- Part (g): Data Trimming Experiment ---
    combined_heights = list(female_data_altered) + list(male_data)
```

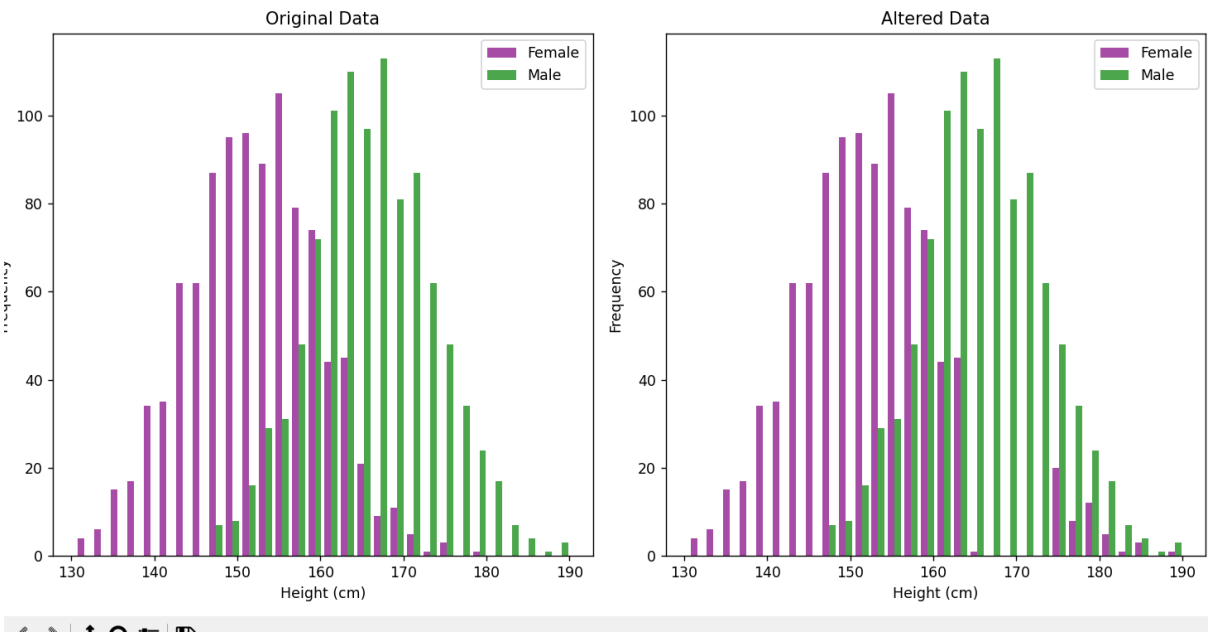
```
    combined_labels = ['F'] * len(female_data_altered) + ['M'] * len(male_data)
    df_altered_combined = pd.DataFrame({'height': combined_heights, 'label':
combined_labels})
    df_sorted = df_altered_combined.sort_values(by='height').reset_index(drop=True)
    n_total = len(df_sorted)
    k_values = list(range(1, 16))
    acc_thr_list, acc_prob_list, acc_quant_list = [], [], []


    for k in k_values:
        k_lower = int(n_total * k / 100)
        k_upper = int(n_total * (1 - k / 100))
        df_trimmed = df_sorted.iloc[k_lower:k_upper].copy()
        trimmed_female = df_trimmed[df_trimmed['label'] == 'F']['height'].values
        trimmed_male = df_trimmed[df_trimmed['label'] == 'M']['height'].values
        if len(trimmed_female) == 0 or len(trimmed_male) == 0:
            continue
        threshold_trim = (np.mean(trimmed_female) + np.mean(trimmed_male)) / 2
        actual_thr_trim, predictions_thr_trim = threshold_classifier(trimmed_female,
trimmed_male, threshold_trim)
        acc_thr_trim = evaluate_classifier(actual_thr_trim, predictions_thr_trim)
        acc_thr_list.append(acc_thr_trim)
        actual_prob_trim, predictions_prob_trim = probability_classifier(trimmed_female,
trimmed_male,
                                                         np.mean(trimmed_female),
np.std(trimmed_female),
                                                         np.mean(trimmed_male), np.std(trimmed_male))
        acc_prob_trim = evaluate_classifier(actual_prob_trim, predictions_prob_trim)
        acc_prob_list.append(acc_prob_trim)
```

```
        actual_quant_trim, predictions_quant_trim = quantized_classifier(trimmed_female,
trimmed_male, interval_len=1)
        acc_quant_trim = evaluate_classifier(actual_quant_trim, predictions_quant_trim)
        acc_quant_list.append(acc_quant_trim)


    plt.figure(figsize=(10, 6))
    plt.plot(k_values[:len(acc_thr_list)], [acc * 100 for acc in acc_thr_list],
             marker='o', linestyle='-', color='red', label='Threshold Classifier')
    plt.plot(k_values[:len(acc_prob_list)], [acc * 100 for acc in acc_prob_list],
             marker='s', linestyle='-', color='blue', label='Probability Classifier')
    plt.plot(k_values[:len(acc_quant_list)], [acc * 100 for acc in acc_quant_list],
             marker='^', linestyle='-', color='orange', label='Quantized Classifier')
    plt.xlabel("Trimming Percentage (k%)")
    plt.ylabel("Classification Accuracy (%)")
    plt.title("Impact of Data Trimming on Classification Accuracy")
    plt.legend()
    plt.grid(True)
    plt.show()

# Directly call the main function.
run_code()
```


Original Data
Female
Male
Frequency
Height (cm)
Altered Data
Female
Male
Frequency
Height (cm)

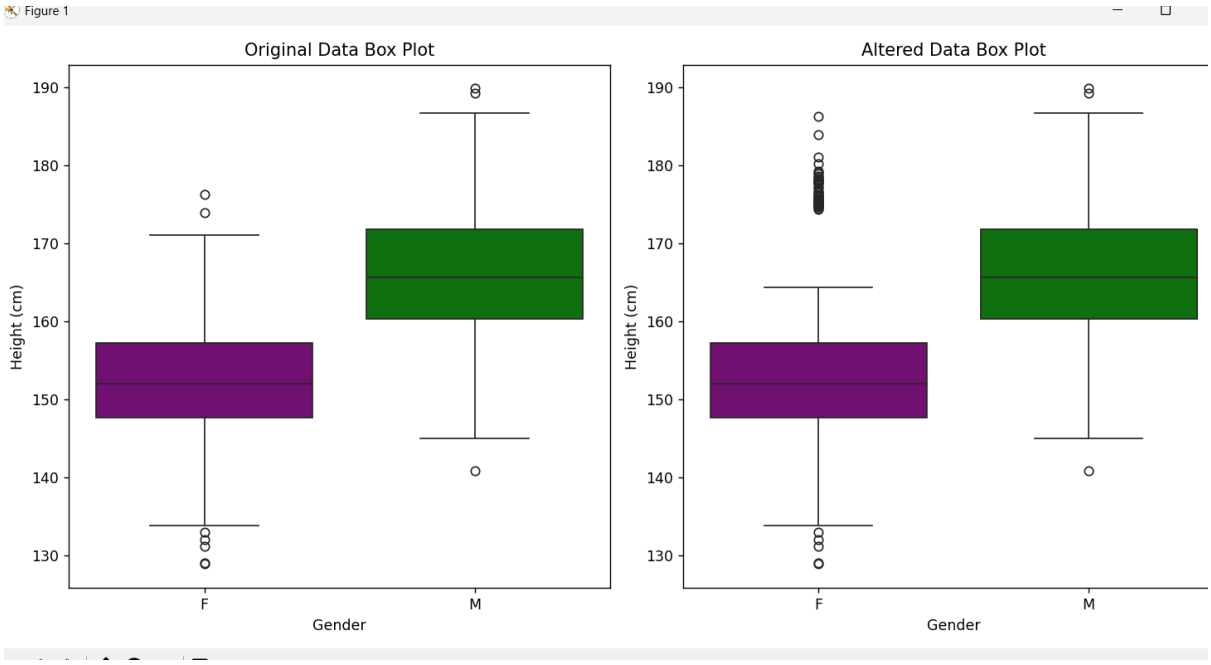
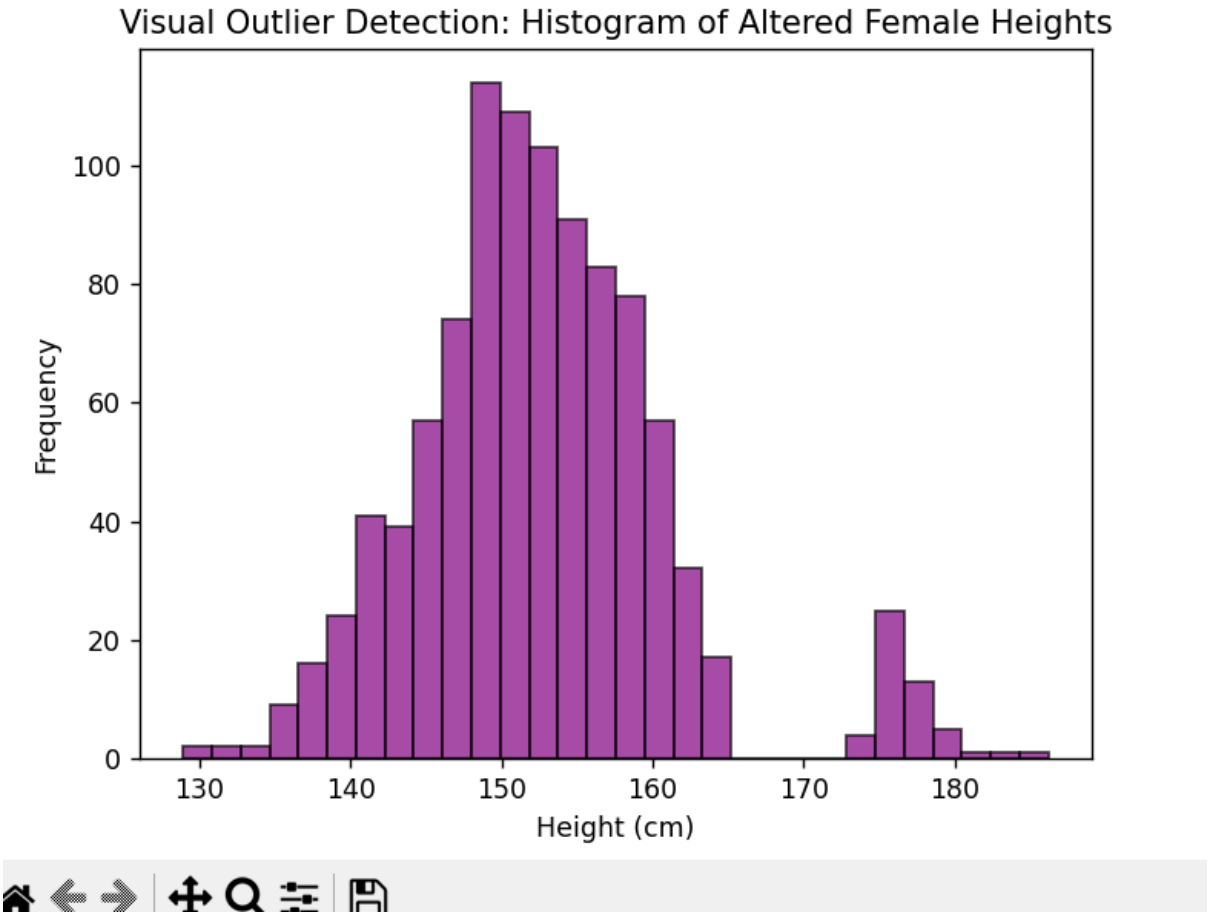
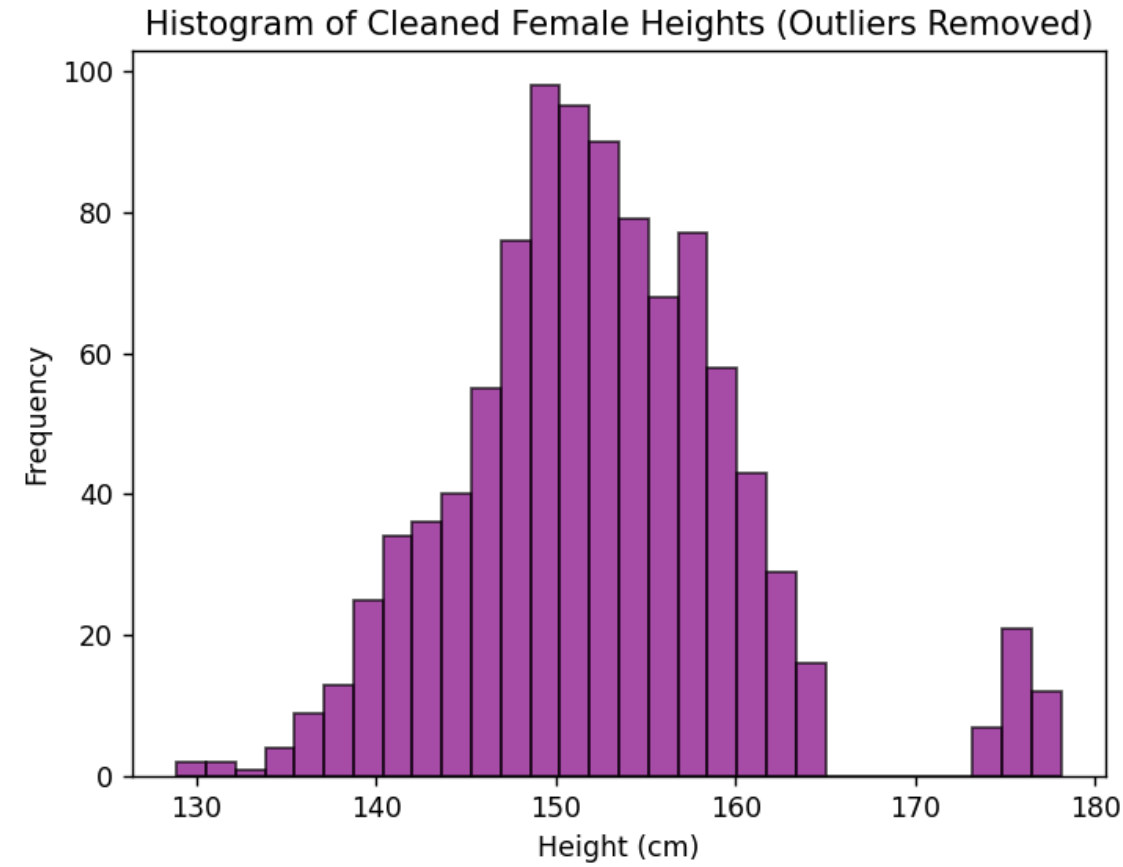
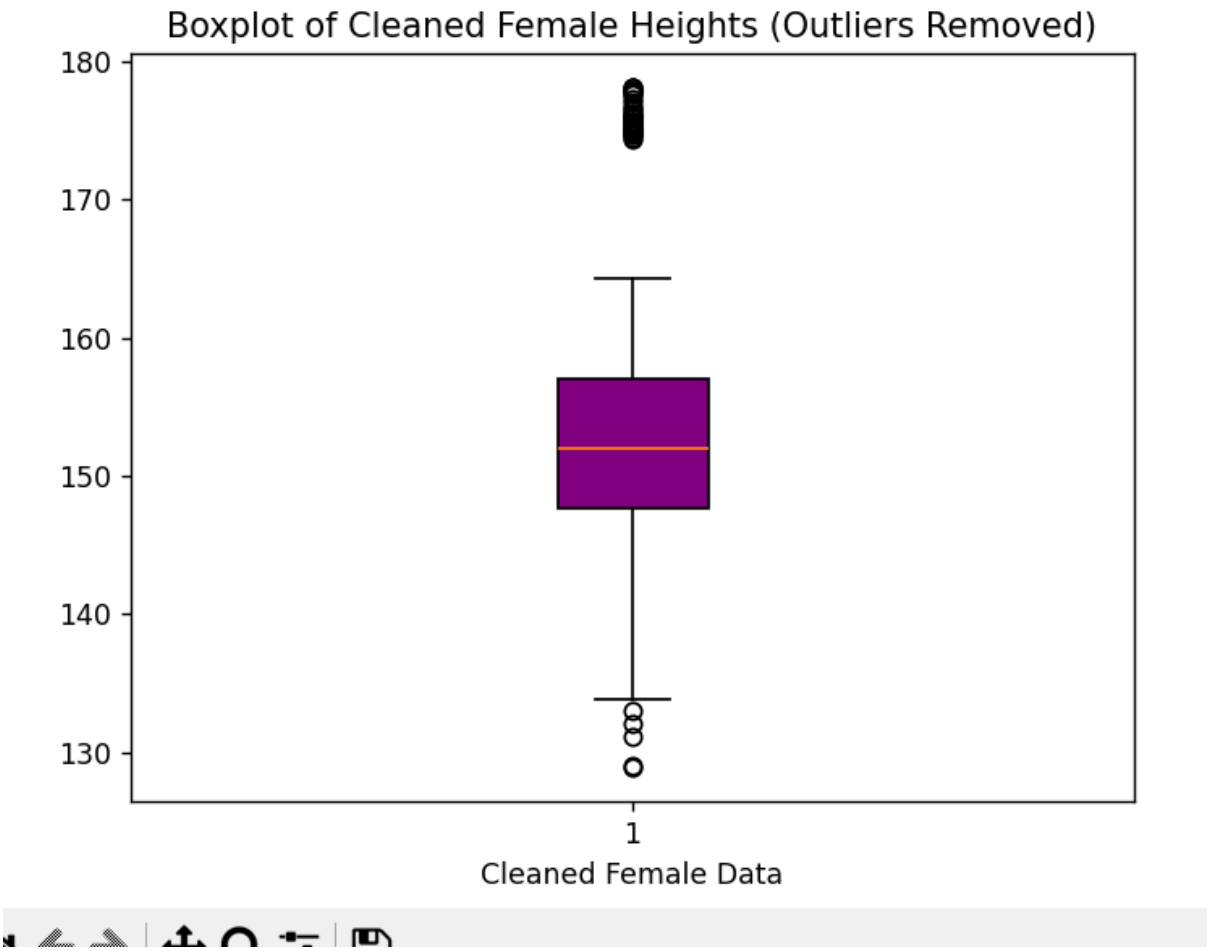
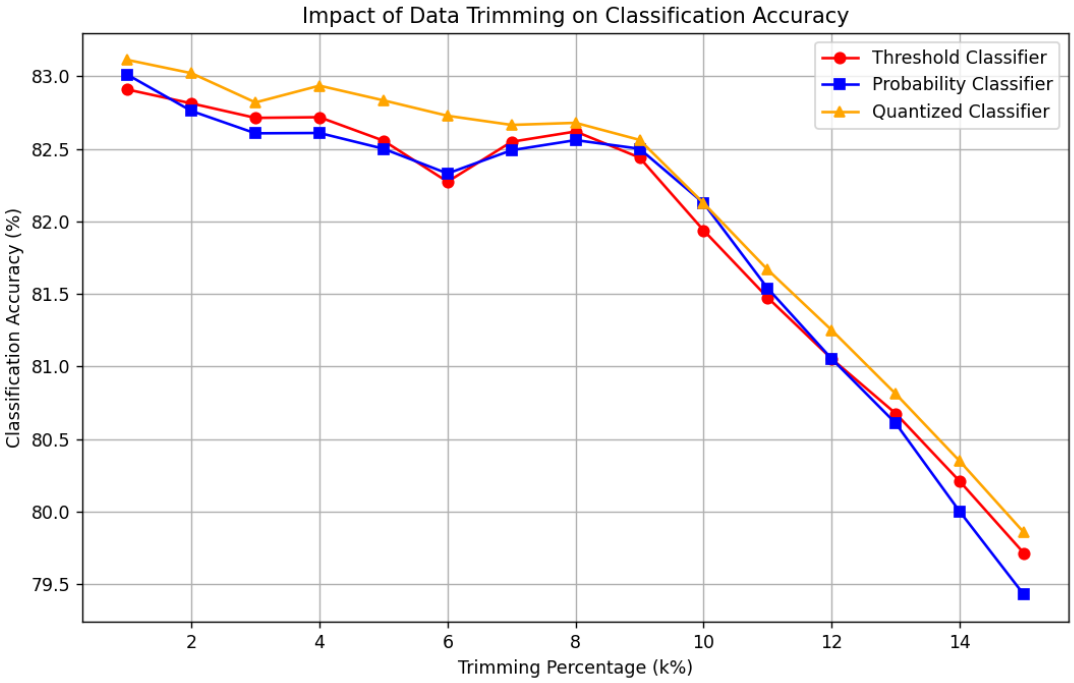


Figure 1
Original Data Box Plot
Height (cm)
Gender
F
M
Altered Data Box Plot
Height (cm)
Gender
F
M


Visual Outlier Detection: Histogram of Altered Female Heights
Frequency
0
20
40
60
80
100
130
140
150
160
170
180
Height (cm)


Histogram of Cleaned Female Heights (Outliers Removed)
Frequency
Height (cm)
0
20
40
60
80
100
130
140
150
160
170
180





OBSERVATIONS:

1. generate_heights(mean, std_dev, size, label)

Observations:

Our graphs (histograms or boxplots) reveal a bell-shaped distribution whose center and spread directly reflect the mean and standard deviation we specified.

Parameter Effects:

Mean: Shifts the entire graph left or right.

Standard Deviation: Widens or narrows the distribution.

Sample Size: Larger samples yield smoother, more stable graphs.

Label: Alters grouping or color in multi-class plots without changing the distribution shape.

2. plot_histograms(female_heights, male_heights, bins=50, title_suffix="")

Observations:

Our overlaid histograms display distinct peaks for each group, allowing us to clearly compare the distributions.

Parameter Effects:

Bins: Fewer bins give a coarser view; more bins reveal finer details.

Title Suffix: Only changes the graph's label, not its overall structure.

3. threshold_classifier(female_data, male_data, threshold)

Observations:

Our visualizations often include a vertical decision line that divides the data into two classes based on the threshold.

Parameter Effects:

Threshold: Shifting this value left or right alters the proportions of each class displayed on the graph.

4. probability_classifier(female_data, male_data, female_mean, female_sd, male_mean, male_sd)

Observations:

When we plot the probability density curves for each class, we see two overlapping curves that indicate the likelihood of a value belonging to either group.

Parameter Effects:

Mean & Standard Deviation: Adjusting these parameters shifts and reshapes the curves, thereby changing their overlap and the resulting decision boundary.

5. quantized_classifier(female_data, male_data, interval_len)

Observations:

Our bar plots show the count of data points within each interval, effectively segmenting the distribution into discrete bins.

Parameter Effects:

Interval Length: Smaller intervals create more, finer bars; larger intervals produce fewer, broader bars that smooth out the details.

6. evaluate_classifier(actual, predictions, description="")

Observations:

We obtain numerical performance outputs (such as accuracy and confusion matrices), which we often use to generate performance plots.

Parameter Effects:

Description: This only affects the labeling of our outputs; the performance plots change in response to any modifications in predictions made earlier.

7. detect_outliers_zscore(data, cutoff=2)

Observations:

Our boxplots or scatter plots highlight outliers as individual points that lie outside the main data cluster.

Parameter Effects:

Cutoff: A lower cutoff increases sensitivity (flagging more outliers), while a higher cutoff reduces sensitivity (flagging fewer outliers).

8. detect_outliers_iqr(data, factor=1.5)

Observations:

In our boxplots, outliers appear as dots beyond the whiskers, clearly marking values that fall outside the typical range.

Parameter Effects:

Factor: A lower factor shortens the whiskers and flags more outliers; a higher factor lengthens the whiskers, reducing the number of outliers.

9. detect_outliers_mad(data, cutoff=3)

Observations:

Our plots show outliers as isolated points when using the MAD method, similar to the z-score approach.

Parameter Effects:

Cutoff: Lowering the cutoff increases sensitivity (more outliers appear), while raising it decreases sensitivity (fewer outliers are marked).

10. main_assignment3()

Observations:

Our comprehensive set of graphs includes:

Histograms and boxplots that compare original and altered data, revealing shifts in peaks, increased spread, and the presence of outliers.

Outlier-focused plots that emphasize extra high values.

Cleaned data graphs where the distributions appear more symmetric.

A trimming experiment plot that shows how accuracy trends change with different percentages of data removal.

Parameter Effects:

Data Generation (mean, SD, sample size): Changes here shift the center and spread of our graphs.

Alteration: Increasing the number or magnitude of altered points creates longer tails and more visible outliers.

Outlier Detection/Cleaning: Adjusting cutoffs or factors changes how many outliers are removed, resulting in a cleaner, more “normal” graph.

Trimming: Different trimming percentages affect the accuracy curve—moderate trimming often helps performance, while excessive trimming may hurt it.