San Francisco Bay University
MATH208 - Probability and Statistics
2023 Fall Homework #2

Kritika Regmi(19702)

1. Write the program in any computer language to read-in the data from the attached file "*original_diabetes.xlsx*" partially coming from Pima Indian Diabetes in the National Institute of Diabetes and Digestive and Kidney Diseases[1]. Find the means, variances, standard deviations, z scores and the values of Q_1, $\tilde{x}$(median) and Q_3 of "*Glucose*" and "*BloodPressure*" by user-defined functions rather than calling existing functions in the libraries. After that, please plot the boxplots of both variables in one frame.

Step1: Importing necessary libraries:

```
[2] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
```

Step2: CODE

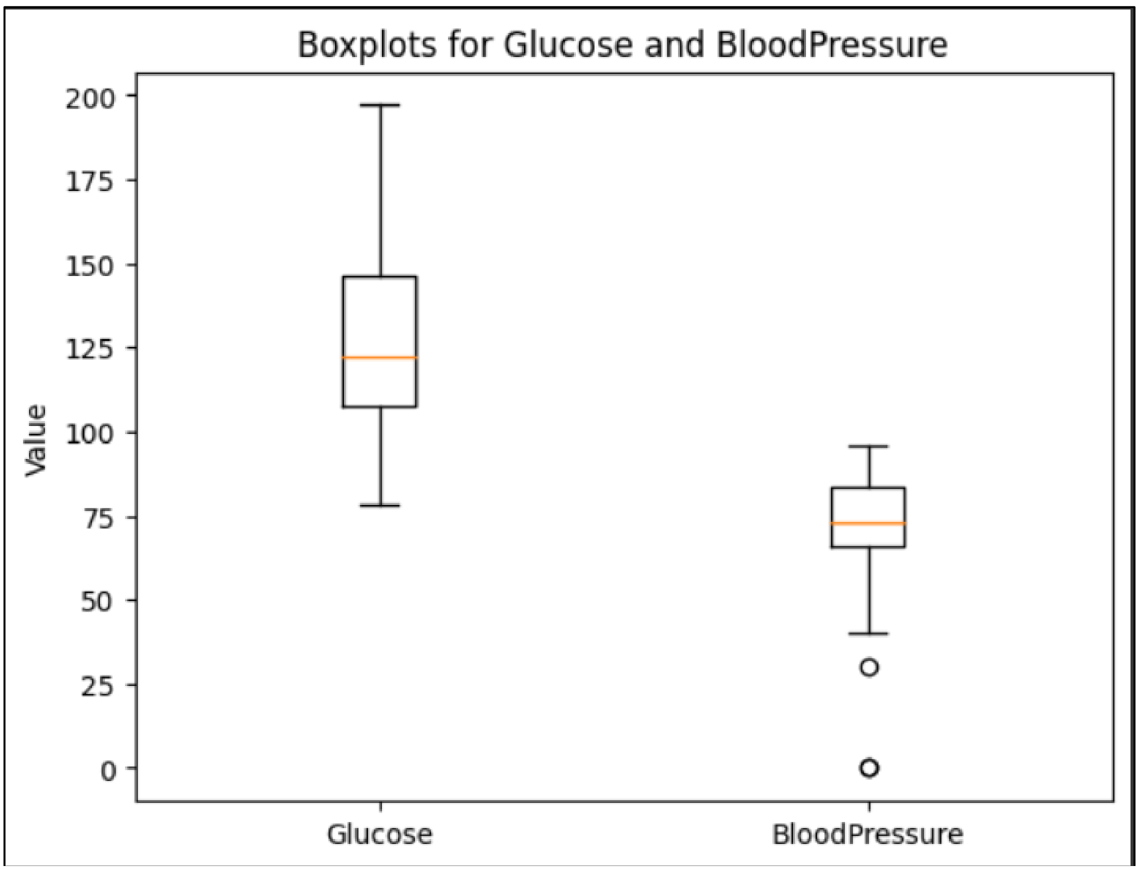
```
#MATH208_Week2_Question1_19702_Kritika_Regmi
def m(diabetes):
    return sum(diabetes) / len(diabetes)

def variance(diabetes):
    mean= m(diabetes)
    return sum((x - mean) ** 2 for x in diabetes) / (len(diabetes) - 1)

def standard_deviation(diabetes):
    return variance(diabetes) ** 0.5

def z_score(diabetes):
    mean = m(diabetes)
    std_dev = standard_deviation(diabetes)
    z_score = [(x - mean) / std_dev for x in diabetes]
    return z_score

def quartiles(diabetes):
    q1 = np.percentile(diabetes, 25)
    median = np.percentile(diabetes, 50)
    q3 = np.percentile(diabetes, 75)
    return q1, median, q3

diabetes = pd.read_excel("original_diabetes.xlsx")
```

```python
glucose_data = diabetes["Glucose"].dropna()
blood_pressure_data = diabetes["BloodPressure"].dropna()

# Calculate statistics for "Glucose"
glucose_mean = m(glucose_data)
glucose_variance = variance(glucose_data)
glucose_std_dev = standard_deviation(glucose_data)
glucose_z_scores = z_score(glucose_data)
glucose_q1, glucose_median, glucose_q3 = quartiles(glucose_data)

# Calculate statistics for "BloodPressure"
blood_pressure_mean = m(blood_pressure_data)
blood_pressure_variance = variance(blood_pressure_data)
blood_pressure_std_dev = standard_deviation(blood_pressure_data)
blood_pressure_z_scores = z_score(blood_pressure_data)
blood_pressure_q1, blood_pressure_median, blood_pressure_q3 = quartiles(blood_pressure_data)
```

```python
print("Statistics for Glucose:")
print(f"Mean: {glucose_mean}")
print(f"Variance: {glucose_variance}")
print(f"Standard Deviation: {glucose_std_dev}")
print(f"Z-Scores: {glucose_z_scores}")
print(f"Q1: {glucose_q1}")
print(f"Median: {glucose_median}")
print(f"Q3: {glucose_q3}")

print("\nStatistics for BloodPressure:")
print(f"Mean: {blood_pressure_mean}")
print(f"Variance: {blood_pressure_variance}")
print(f"Standard Deviation: {blood_pressure_std_dev}")
print(f"Z-Scores: {blood_pressure_z_scores}")
print(f"Q1: {blood_pressure_q1}")
print(f"Median: {blood_pressure_median}")
print(f"Q3: {blood_pressure_q3}")

plt.boxplot([glucose_data, blood_pressure_data], labels=["Glucose", "BloodPressure"])
plt.title("Boxplots for Glucose and BloodPressure")
plt.xticks([1, 2], ["Glucose", "BloodPressure"])
plt.ylabel("Value")
plt.show()
```

Step3: OUTPUT

```
Statistics for Glucose:
Mean: 130.06666666666666
Variance: 1075.0298850574711
Standard Deviation: 32.787648361196496
Z-Scores: [0.5469539363047783, -1.3745013417919332, 1.6144290908029513, -1.252504181
Q1: 107.75
Median: 122.0
Q3: 146.5

Statistics for BloodPressure:
Mean: 68.53333333333333
Variance: 572.119540229885
Standard Deviation: 23.91902046969911
Z-Scores: [0.14493347129571138, -0.10591292133148125, -0.18952838554054546, -0.10591
Q1: 66.0
Median: 73.0
Q3: 83.5
```



2. Write the program to verify Chebyshev's inequity as follows by *50* random numbers generated from a normal distribution with mean $\mu = 10$ and standard deviation $\sigma = 0.5$.

$$P(-k\sigma < X - \mu < k\sigma) \geq \left(1 - \frac{1}{k^2}\right) or\ P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

```python
#MATH208_Week2_Question2_19702_Kritika_Regmi

mean_val = 10
std_val = 0.5
random_numbers_new = np.random.normal(mean_val, std_val, 50)

def Chebyshev_inequality(lst, k):

    mean_val_new = np.mean(lst)
    std_val_new = np.std(lst)

    lower_bound = mean_val - k * std_val
    upper_bound = mean_val + k * std_val

    total_num_data = len([x for x in lst if lower_bound <= x <= upper_bound])

    prob = total_num_data / len(lst)

    return prob

ks = [1, 2**0.5, 1.5, 2, 3]

for k in ks:
    prob = Chebyshev_inequality(random_numbers_new, k)
    print(f"When k = {k}, P(|X-mean_val| < {k}*std_val) = {prob}; 1 - 1/{k**2} = {1 - 1/(k**2)}")
    print(f"When k = {k}, P(|X-mean_val| < {k}*std_val) >= 1 - 1/{k**2} is {prob >= 1 - 1/(k**2)}\n")
```

OUTPUT:

```
When k = 1, P(|X-mean_val| < 1*std_val) = 0.58; 1 - 1/1 = 0.0
When k = 1, P(|X-mean_val| < 1*std_val) >= 1 - 1/1 is True

When k = 1.4142135623730951, P(|X-mean_val| < 1.4142135623730951*std_val) = 0.8; 1 - 1/2.0000000000000004 = 0.5000000000000001
When k = 1.4142135623730951, P(|X-mean_val| < 1.4142135623730951*std_val) >= 1 - 1/2.0000000000000004 is True

When k = 1.5, P(|X-mean_val| < 1.5*std_val) = 0.82; 1 - 1/2.25 = 0.5555555555555556
When k = 1.5, P(|X-mean_val| < 1.5*std_val) >= 1 - 1/2.25 is True

When k = 2, P(|X-mean_val| < 2*std_val) = 0.9; 1 - 1/4 = 0.75
When k = 2, P(|X-mean_val| < 2*std_val) >= 1 - 1/4 is True

When k = 3, P(|X-mean_val| < 3*std_val) = 1.0; 1 - 1/9 = 0.8888888888888888
When k = 3, P(|X-mean_val| < 3*std_val) >= 1 - 1/9 is True
```

3. Given the following dataset, write the program to fit it by linear regression showing the values of b_1, b_0 and coefficient of linear correlation r. After that, please plot the curve of X vs Y and straight fitting line. Can we draw the conclusion that linear model is good for the dataset if the value of r is very close to $+1$? Suggest which fitting model should be better than linear based on the data visualization of the given dataset.

X	Y
2	30
3	25
4	95
5	115
6	265
7	325
8	570
9	700
10	1085
11	1300

```
#MATH208_Week2_Question3_19702_Kritika_Regmi

a = np.array([2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
b = np.array([30, 25, 95, 115, 265, 325, 570, 700, 1085, 1300])

n = len(a)
b1 = (n * np.sum(a * b) - np.sum(a) * np.sum(b)) / (n * np.sum(a**2) - (np.sum(a))**2)
b0 = (np.sum(b) - b1 * np.sum(a)) / n

r = np.corrcoef(a, b)[0, 1]

print(f"b1: {b1}")
print(f"b0: {b0}")
print(f"Coefficient of Linear Correlation (r): {r}")

regression_line = b0 + b1 * a

plt.scatter(a, b, label='Data Points', color='b')
plt.plot(a, regression_line, label='Linear Regression', color='r')
plt.xlabel('a')
plt.ylabel('b')
plt.legend()
plt.show()
```

OUTPUT:

```
b1: 141.21212121212122
b0: -466.878787878788
Coefficient of Linear Correlation (r): 0.9435795518902779
```