Name (Last, First):	Student ID:
Kunjangada Arun Sushma	002473132

Assignment 3

NEU_COE_INFO6105_Fall2024

Instructions:

1. For answering **programming questions**, please use Adobe Acrobat to edit the pdf file in two steps **[See Appendix: Example Question and Answer]**:
   a. Copy and paste your R or python code as text in the box provided (so that your teaching team can run your code);
   b. Screenshot your R or python console outputs, save them as a .PNG image file, and paste/insert them in the box provided.
   c. Show all work - credit will not be given for code without showing the code in action by including the screenshot of R or python console outputs.
2. To answer **non-programming questions**, please type or handwrite your final answers clearly in the boxes. Show all work - credit will not be given for numerical solutions that appear without explanation in the space above the boxes. **You're encouraged to use R or python to graph/plot the data and produce numerical summaries; please append your code and screenshot of the outputs at the end of your pdf submission.**
3. **[Total 123 pts = 12 + 48 + 48 pts + 15 Extra Credit pts]**

Grading Rubric

Each question is worth 3 points and will be graded as follows:

3 points: Correct answer with work shown

2 points: Incorrect answer but attempt shows some understanding (work shown)

1 point: Incorrect answer but an attempt was made (work shown), or **correct answer without explanation (work not shown)**

0 points: Left blank or made little to no effort/work not shown

Reflective Journal [3 pts]

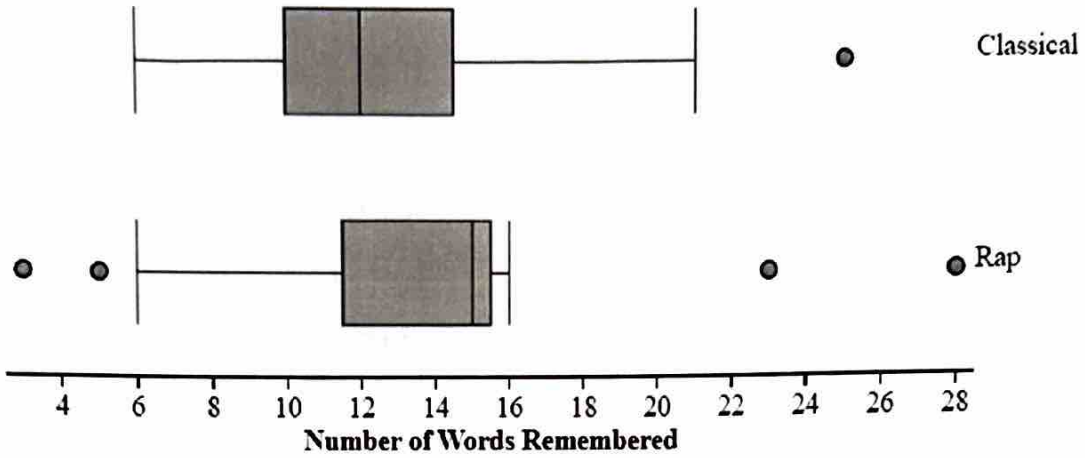
(Copy and paste the link to your live Google doc in the box below)

https://docs.google.com/document/d/1ptEhnYHniNtT1yxDPcvXK7LpJaPGzi80BCSGZZhom7Y/edit?usp=sharing

Part I. Exploring One Variable Data: SOCS and Comparing (12 pts)

Amy and Bob want to know if listening to different types of music while studying will help you remember the material better. They randomly assigned a group of students to two different groups: one group studied a list of words while listening to classical music while the second group studied the same list of words while listening to rap music. There was a total of 30 words on the list and each group studied the list while listening to the music for 5 minutes. They were

then asked to write down as many words as they could remember. The data is displayed in the boxplots below.



a) Approximate the interquartile range for each set of data. Why is this the appropriate measure of spread to use for these two data sets? **(3 pts)**

Answer:

Classical music:
$Q1 = 10$ $Q3 = 15$
$IQR = Q3 - Q1 = 15 - 10 = 5$ words

Rap Music:
$Q1 = 12$ $Q3 = 16$
$IQR = 16 - 12 = 4$ words

The IQR is an appropriate measure of spread because it's resistant to outliers and focuses on the middle 50% of the data giving a good sense of how spread the data is.

b) Write **three** sentences comparing the number of words remembered between each of the two groups. **(9 pts)**

Answer:

- The median number of words remembered for rap music is approximately 15 while the classical music is 12. This indicates average students listening to rap music recalled more words.
- The IQR of classical music is 5 while rap music IQR is 4. This shows classical music group had a slightly wider spread of scores, more variability in word recalled compared to rap group.
- In the classical music group there is one outlier at 25. The rap music group has multiple outliers: 23 and 28 as high outliers & 3 and 5 as low outliers, these outliers highlight greater variability in the rap group with both very high & very low performers.

Part II. Exploring One Variable Data: The Empirical Rule and Z-Scores (48 pts)

1. Three landmarks of baseball achievement are Ty Cobb's batting average of 0.420 in 1911, Ted Williams's 0.406 in 1941, and George Brett's 0.390 in 1980. These batting averages cannot be compared directly because the distribution of major league batting averages has changed over the years. The distributions are quite symmetric, except for outliers such as Cobb, Williams, and Brett. While the mean batting average has been held roughly constant by rule changes and the balance between hitting and pitching, the standard deviation has dropped over time. Here are the facts:

Decade	Mean	Standard Deviation
1910s	0.266	0.0371
1940s	0.267	0.0326
1970s	0.261	0.0317

Find the standardized scores for Cobb, Williams, and Brett. Who was the best hitter? **(12 pts)**

Answer:

Z Score = $z = \frac{x - \mu}{\sigma}$

x - individual score (batting average)
μ - mean of the decade
σ - Standard deviation of the decade

For Cobb:

$x = 0.420 \quad \mu = 0.266 \quad \sigma = 0.0371$

$z = \frac{0.420 - 0.266}{0.0371} = \frac{0.154}{0.0371} \simeq 4.15$ (Rounding to 2 decimal)

For Williams:

$x = 0.406 \quad \mu = 0.267 \quad \sigma = 0.0326$

$z = \frac{0.406 - 0.267}{0.0326} = \frac{0.139}{0.0326} \simeq 4.26$

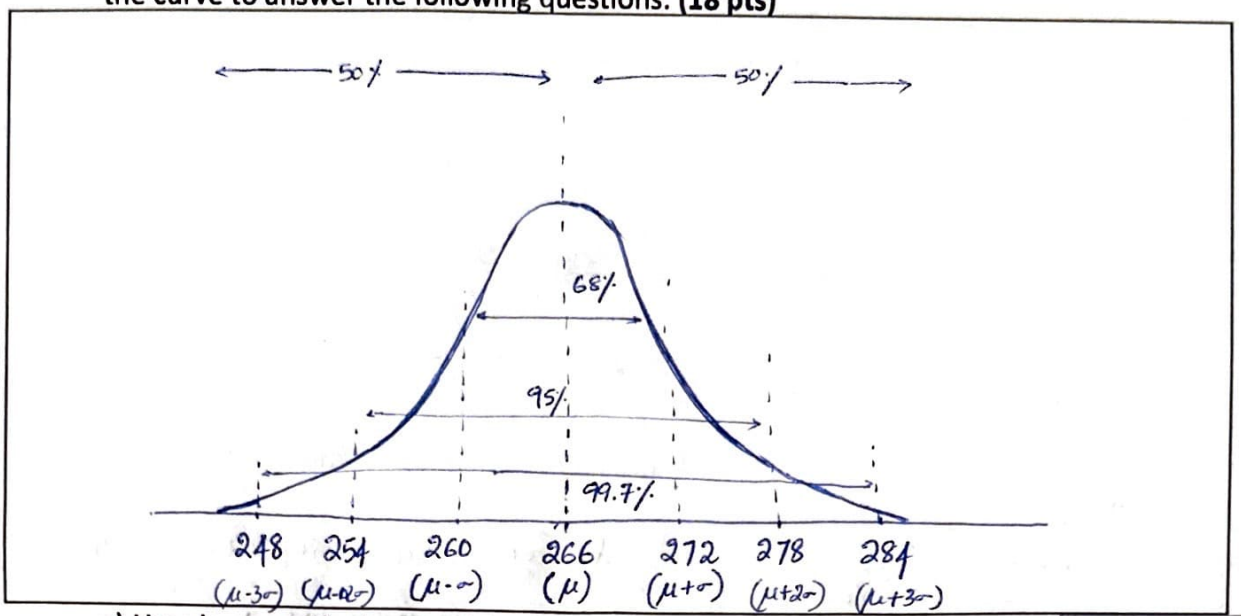
For Brett:

$x = 0.390 \quad \mu = 0.261 \quad \sigma = 0.0317$

$z = \frac{0.390 - 0.261}{0.0317} = \frac{0.129}{0.0317} \simeq 4.07$

Jed William's was the best hitter with highest z-score (4.26) relative to his era.

2. The length of human pregnancies from conception to birth varies on an approximate normal distribution with mean 266 days and a standard deviation of 6 days. Draw a normal curve below and mark three standard deviations out on either side. Then, use the curve to answer the following questions. **(18 pts)**



a) How long are the longest 16% of pregnancies?

For Longest 16% we look at the right tail's top 16% of pregnancies

$z = 1$

$\therefore$ Longest 16% of pregnancies (pregnancies longer than 84% of population) $\geq 266 + 1*6$

≥ 272

= 272 days or more

b) How short are the shortest 16% of pregnancies?

For Shortest 16% of pregnancies we are looking for pregnancies shorter than 16% of the population. ie.,

$\leq 266 - 1*6$

≤ 260

= 260 days or fewer

c) Between how many days does the middle 95% of all pregnancies fall?

The middle 95% of pregnancies fall between $266 \pm 2\sigma$ i.e., $\mu - 2\sigma \leq X \leq \mu + 2\sigma$

$266 - 12 \leq X \leq 266 + 12 = 254 \leq X \leq 278$

Between 254 and 278 days.

d) Between how many days does the middle 99.7% of all pregnancies fall?

The middle 99.7% of pregnancies fall between $266 \pm 3\sigma$

$\mu - 3\sigma \leq X \leq \mu + 3\sigma$

$266 - 18 \leq X \leq 266 + 18 = 248 \leq X \leq 284$

Between 248 and 284 days.

e) A pregnancy of 254 days corresponds to what percentile of pregnancies?

$$z = \frac{254 - 266}{6} = \frac{-12}{6} = -2$$

A z score of -2 corresponds to approximately $\Rightarrow$ 2.5th percentile

ie, A pregnancy of 254 days is in the 2.5th percentile, meaning it is shorter than 97.5% of all pregnancies.

3. The scores of an introductory statistics class at Simmons University is normally distributed with a mean of 72 and a standard deviation of 10. **(6 pts)**

(a) If a student has a z-score of 1.4, what score does she have?

(b) If a student has a z-score of -1.23, what score does she have?

Answer:

a. $z = \frac{X-\mu}{\sigma} \Rightarrow X = z\times\sigma+\mu$

$z = 1.4 \quad \sigma = 10 \quad \mu = 72$

$X = 1.4 \times 10 + 72 = 14 + 72 = 86$

A student with z score 1.4 has a score 86.

b. $z = X-\mu \Rightarrow X = z\times\sigma+\mu$

$z = -1.23 \quad \sigma = 10 \quad \mu = 72$

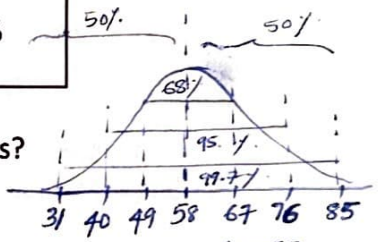
$X = -1.23\times10+72$

$= -12.3+72 = 59.7$

A student with a z-score of -1.23 has a score 59.7

4. A college professor plans to give a test and to curve the results. The raw test scores are normally distributed with a mean of 58 and a standard deviation of 9. She wants the grades to be divided up as follows: **(12 pts)**

D	C	B	A
Bottom 25%	50th percentile or lower before a D	Between an A and a C	Top 10%



What will be the raw score averages she gives to each of the letter grades?

Answer:

D (Bottom 25%)

From z score table 25%

z score = -0.674

$X = z\times\sigma+\mu$

$= -0.674\times9+58$

$= -6.066+58 = 51.934$

C =

c. z score is 0 (50th percentile or 20)

$X = z\times\sigma+\mu$

$= 0\times9+58$

$= 58$

Raw test score for C is

$58 \le C > 51.93$

B (Between C and A)

z score for 90 percentile is 1.282 from z score table

$X = z\times\sigma+\mu$

$= 1.282\times9+58$

$= 11.538+58 = 69.538$

$58 < B < 69.54$

A.

D (Bottom 25%)	C	B	A
51.93	Between 51.93 and 58 or 58.	Between 58 and 69.54	More than 69.54

A. z score for 90% = 1.282

$X = 1.282\times9+58$

$= 11.538+58 = 69.538 \simeq 69.54$

Top 10% Student 70 or higher.

Part IV. Statistical Programming (60 = 48 + 12 extra credit pts)

The following sample data shows the scores of the students in an exam:

68, 25, 87, 89, 91, 79, 99, 80, 62, 74

Do the following using R or python code **with only a single expression (one-liner)** for each question. The solutions should be generic and work for any data size **(meaning no hard coding)**. You can assume there will be an even number of values in the given data.

1. **Assigning and Accessing Data (15 pts):**
   a. Assign the above data as a *vector* in the same order to the variable ***scores***. Use this variable for the remaining problems.
   b. Using the *length* function, compute how many students took the exam? Store the expression in the variable ***n***.
   c. Using indexing, write the expression for accessing the first two items. Store the expression in the variable ***first_and_second.***
   d. Using indexing, write the expression for accessing the first and last items. Store the expression in the variable ***first_and_last.***
   e. Using indexing, write the expression for accessing the middle two items. Store the expression in the variable ***middle_two.***

Sample output:

```
[1] 10
[1] 68 25
[1] 68 74
[1] 91 79
```

2. **Median (15 pts):**
   a. Use *median(scores)* to compute the median of the data. Store the expression in the variable **median_score**.
   b. Using comparison operators, write the R or python expression for scores less than or equal to the median of the data. Store the expression in the variable **below_median.**
   c. Using comparison operators, write the R or python expression for scores greater than the median of the data. Store the expression in the variable **above_median.**
   d. Using the **sum** function, write the R or python expression for the number of scores less than or equal to the median of the data. Store the expression in the variable **count_below_median.**
   e. Using the **sum** function, write the R or python expression for the number of scores greater than the median of the data. Store the expression in the variable **count_above_median.**

Sample output:

[1] 79.5
[1] TRUE TRUE FALSE FALSE FALSE TRUE FALSE FALSE TRUE TRUE
[1] FALSE FALSE TRUE TRUE TRUE FALSE TRUE TRUE FALSE FALSE
[1] 5
[1] 5

3. **Indexing and subsetting (18 pts):**
    a. Using logical indexing and the results from Q2), write the R or python expression for all the scores that are less than or equal to the median value of the data. Store the expression in the variable **scores_below_median.**
    b. Similarly, write the R or python expression for all the scores that are greater than the median. Store the expression in the variable **scores_above_median.**

Sample output:
[1] 68 25 79 62 74
[1] 87 89 91 99 80

Use the **seq** function to generate the numeric indices for (c) and (d) below.

    c. Using numeric indexing, write the R or python expression for the odd indexed values from the scores. Store the expression in the variable **odd_index_values**.
    d. Similarly, write the R or python expression for the even indexed values from the scores. Store the expression in the variable **even_index_values**.

Sample output:
[1] 68 87 91 99 62
[1] 25 89 79 80 74

    e. Using the **paste** function with LETTERS, write the expression for the following output. Store the expression in the variable **format_scores_version1**. You can assume there are no more than 26 values.

Sample Output:
[1] "A=68" "B=25" "C=87" "D=89" "E=91" "F=79" "G=99" "H=80" "I=62" "J=74"

    f. Similarly, using the **paste** function with LETTERS, write the expression for the following output. Store the expression in the variable **format_scores_version2**.

Sample output:
[1] "J=68" "I=25" "H=87" "G=89" "F=91" "E=79" "D=99" "C=80" "B=62" "A=74"

4. **Extra Credit: Matrix and named matrix (12 pts):**
    a. Create a matrix with two rows using the ***scores*** data. The first half of the values belong to the first row of the matrix. Store the expression in the variable **scores_matrix.**

The code should work for any size input data.
You can assume that there are an even number of values in scores.

Sample output:

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   68   25   87   89   91
[2,]   79   99   80   62   74
```

b. Write the expression for displaying the first and last columns of the above matrix. The code should work for any size matrix. Store the expression in the variable **first_and_last_version1.**

Sample output:

```
     [,1] [,2]
[1,]   68   91
[2,]   79   74
```

c. Copy **scores_matrix** to the variable **named_matrix**. Assign column names for the *named_matrix* as Student_1, Student_2,... and row names as Quiz_1, Quiz_2, ... The code should work for any size matrix, i.e., for any number of columns in the matrix and any number of rows.

Sample output:

```
       Student_1 Student_2 Student_3 Student_4 Student_5
Quiz_1        68        25        87        89        91
Quiz_2        79        99        80        62        74
```

d. Show the result for displaying the first and last columns of the *named_matrix.* The code should work for any size matrix. Store the expression in the variable **first_and_last_version2.**

Sample output:

```
       Student_1 Student_5
Quiz_1        68        91
Quiz_2        79        74
```

Answer: Copy and paste your R or python code in the box below (not an image but the text).

1.

```
# a. Assign the data to the variable scores
scores = [68, 25, 87, 89, 91, 79, 99, 80, 62, 74]

# b. Compute how many students took the exam
n = len(scores)

# c. Access the first two items
first_and_second = scores[:2]

# d. Access the first and last items
first_and_last = [scores[0], scores[-1]]

# e. Access the middle two items
middle_two = scores[len(scores)//2 - 1 : len(scores)//2 + 1]

# Output the results
print("Number of students (n):", n)
print("First and second items:", first_and_second)
print("First and last items:", first_and_last)
print("Middle two items:", middle_two)
```

```
Number of students (n): 10
First and second items: [68, 25]
First and last items: [68, 74]
Middle two items: [91, 79]
```

Answer: Copy and paste your R or python code in the box below (not an image but the text).

2.

```
import numpy as np

# a. Compute the median of the data
scores = [68, 25, 87, 89, 91, 79, 99, 80, 62, 74]
median_score = np.median(scores)

# b. True/False for scores less than or equal to the median
below_median = [score <= median_score for score in scores]

# c. True/False for scores greater than the median
above_median = [score > median_score for score in scores]

# d. Count the number of scores less than or equal to the median
count_below_median = sum(below_median)

# e. Count the number of scores greater than the median
count_above_median = sum(above_median)

# Output the results
print("Median score:", median_score)
print("True/False for scores less than or equal to the median:", below_median)
print("True/False for scores greater than the median:", above_median)
print("Count of scores less than or equal to the median:", count_below_median)
print("Count of scores greater than the median:", count_above_median)
```

```
Median score: 79.5
True/False for scores less than or equal to the median: [True, True, False, False, False, True, False, False, True, True]
True/False for scores greater than the median: [False, False, True, True, True, False, True, True, False, False]
Count of scores less than or equal to the median: 5
Count of scores greater than the median: 5
```

Screenshot of your R or python console outputs and paste the image in the box below

3.

```
import numpy as np
import string

# Assign the data to the variable scores
scores = [68, 25, 87, 89, 91, 79, 99, 80, 62, 74]

# a. Get all scores that are less than or equal to the median
median_score = np.median(scores)
scores_below_median = [score for score in scores if score <= median_score]

# b. Get all scores that are greater than the median
scores_above_median = [score for score in scores if score > median_score]

# c. Get the odd indexed values from the scores (odd index means 1st, 3rd, 5th, etc.)
odd_index_values = [scores[i] for i in range(len(scores)) if i % 2 == 0]  # Python uses 0-based indexing, so 1st is index 0

# d. Get the even indexed values from the scores (even index means 2nd, 4th, etc.)
even_index_values = [scores[i] for i in range(len(scores)) if i % 2 != 0]  # Python uses 0-based indexing, so 2nd is index 1

# e. Format as "A=score", "B=score", ... in alphabetical order
format_scores_version1 = [f"{letter}={score}" for letter, score in zip(string.ascii_uppercase, scores)]

# f. Format as "J=score", "I=score", ... in reverse alphabetical order starting from "J"
format_scores_version2 = [f"{letter}={score}" for letter, score in zip(string.ascii_uppercase[9::-1], scores)]  # From J to A

# Output the results
print("Scores below median:", scores_below_median)
print("Scores above median:", scores_above_median)
print("Odd indexed values:", odd_index_values)
print("Even indexed values:", even_index_values)
print("Formatted scores (version 1):", format_scores_version1)
print("Formatted scores (version 2):", format_scores_version2)
```

```
Scores below median: [68, 25, 79, 62, 74]
Scores above median: [87, 89, 91, 99, 80]
Odd indexed values: [68, 87, 91, 99, 62]
Even indexed values: [25, 89, 79, 80, 74]
Formatted scores (version 1): ['A=68', 'B=25', 'C=87', 'D=89', 'E=91', 'F=79', 'G=99', 'H=80', 'I=62', 'J=74']
Formatted scores (version 2): ['J=68', 'I=25', 'H=87', 'G=89', 'F=91', 'E=79', 'D=99', 'C=80', 'B=62', 'A=74']
```

Screenshot of your R or python console outputs and paste the image in the box below

4.

```
import numpy as np
import pandas as pd

# Assign the data to the variable scores
scores = [68, 25, 87, 89, 91, 79, 99, 80, 62, 74]

# a. Create a matrix with two rows
# Reshape the scores into a 2-row matrix
scores_matrix = np.array(scores).reshape(2, -1)  # -1 automatically calculates the number of columns

# b. Display the first and last columns of the scores_matrix
first_and_last_version1 = scores_matrix[:, [0, -1]]  # Selecting first and last columns

# c. Copy scores_matrix to named_matrix and assign row and column names
num_students = scores_matrix.shape[1]  # Get number of columns (students)
named_matrix = pd.DataFrame(scores_matrix,
                            index=[f'Quiz_{i+1}' for i in range(2)],  # Row names
                            columns=[f'Student_{i+1}' for i in range(num_students)])  # Column names

# d. Display the first and last columns of the named_matrix
first_and_last_version2 = named_matrix.iloc[:, [0, -1]]  # Selecting first and last columns

# Output the results
print("Scores Matrix:\n", scores_matrix)
print("First and Last Columns (Version 1):\n", first_and_last_version1)
print("Named Matrix:\n", named_matrix)
print("First and Last Columns (Version 2):\n", first_and_last_version2)
```

```
Scores Matrix:
 [[68 25 87 89 91]
 [79 99 80 62 74]]
First and Last Columns (Version 1):
 [[68 91]
 [79 74]]
Named Matrix:
         Student_1  Student_2  Student_3  Student_4  Student_5
Quiz_1         68         25         87         89         91
Quiz_2         79         99         80         62         74
First and Last Columns (Version 2):
         Student_1  Student_5
Quiz_1         68         91
Quiz_2         79         74
```

Appendix: Example Question and Answer for R or python programming questions:

Calculate the sum $\sum_{j=0}^{n} r^j$, where r has been assigned the value 1.08, and compare with $(1 - r^{n+1})/(1 - r)$, for $n = 10, 20, 30, 40$.

Answer: Copy and paste your R or python code in the box below (not an image but the text).

```
r <- 1.08
n <- c(10, 20, 30, 40)
sum1 <- c()
for(i in n){
  x <- 0:i
  sum1 <- c(sum1, sum(r^x))
}
sum1   # This gives the calculated sums for n = 10, 20, 30, 40.

sum2 <- (1 - r^(n + 1)) / (1 - r)
sum2

sum2 - sum1   # The formula works.
```

Screenshot of your R or python console outputs and paste/insert the image in the box below

```
> r <- 1.08
> n <- c(10, 20, 30, 40)
> sum1 <- c()
> for(i in n){
+   x <- 0:i
+   sum1 <- c(sum1, sum(r^x))
+ }
> sum1   # This gives the calculated sums for n = 10, 20, 30, 40.
[1]  16.64549  50.42292 123.34587 280.78104
> sum2 <- (1 - r^(n + 1)) / (1 - r)
> sum2
[1]  16.64549  50.42292 123.34587 280.78104
> sum2 - sum1   # The formula works.
[1] 0 0 0 0
```

THE END