DATA 690 Homework 2 (50 points - Due on Sunday, September 25, 2022 by 11:00 pm ET)

The output of this assignment for submission should be in PDF format **AND** .py or .ipynb. The name of the file should be as follows: Lastname_Firstname_Homework2.pdf (example: Thomas_Sunela_Homework2.pdf) **AND** Lastname_Firstname_Homework2.ipynb (example: Thomas_Sunela_Assignment2.ipynb. In short, you are submitting the python notebook as well as the pdf of that notebook. Do **NOT** submit .html file, the system will give you an error.

Incorrect file name will cost you points!

Instructions for converting a Jupyter Python notebook to PDF: Go to the menu and choose, File --> Download As --> html. Open that html file and print it to PDF. Submit the PDF file **NOT** the html file.

If you are using Google Colab, remember to review the PDF before submitting to ensure that all cells and answers are displayed in the PDF.

Things to note:

- Each cell should display an output
- Use only the basic Python concepts and methods
- Use both Markdown and code comments in the Jupyter Notebook as needed

We covered a lot of the basics of the Python programming language. Mastering the fundamentals of Python will equip you for more advanced data analysis libraries like the *Pandas* library.

The goal of this activity is to start thinking about data analysis related tasks only using base Python. You will get practice using various data types, data structures, conditionals, and loops, and gain more experience writing functions.

Clinical Trials - Sheep

While working for a pharmaceutical company that specializes in medications for agriculture you were assigned to work on a project assessing the effectiveness of a new drug designed to treat tapeworms in sheep. (If the sheep have tapeworms they can't get as fat as they need to be at slaughter and it is a waste of feed.)

To investigate you obtain a random sample of 25 worm-infected lambs of approximately the same age and health. Then you randomly divided the lambs into two groups. Twelve of the lambs were injected with the drug and the remaining thirteen were left untreated. After 6 months, the lambs were slaughtered and the number of worms were counted. You need to report to your boss whether the new medication has reduced the average number of tapeworms in the lambs. If so, how much?

```
Treated: 18, 43, 28, 50, 16, 32, 13, 35, 38, 33, 6, 7
Untreated: 40, 54, 26, 63, 21, 37, 39, 23, 48, 58, 28, 39, 42
```

Exercise 1: (2 points)

Create two lists (*treated* and *untreated*) containing the data above

In [1]:
```
## Exercise Answer

#Creating Lists
treated = [18, 43, 28, 50, 16, 32, 13, 35, 38, 33, 6, 7]
untreated = [40, 54, 26, 63, 21, 37, 39, 23, 48, 58, 28, 39, 42]
```

In [2]:
```
print(f'Treated List : {treated}')
```

```
Treated List : [18, 43, 28, 50, 16, 32, 13, 35, 38, 33, 6, 7]
```

In [3]:
```
print(f'UnTreated List : {untreated}')
```

```
UnTreated List : [40, 54, 26, 63, 21, 37, 39, 23, 48, 58, 28, 39, 42]
```

Exercise 2: (3 points)

Use an appropriate method to print the number of elements in each list (*treated* and *untreated*). Use fstrings to nicely print the output as a complete sentence.

len() function returns the number of elements in the List

In [4]:
```
## Exercise Answer

num_treated=len(treated)
```

Using fstrings to print the length of the strings

In [5]:
```
print(f'There are {num_treated} elements in the Treated List')
```

```
There are 12 elements in the Treated List
```

In [6]:
```
num_untreated=len(untreated)
```

In [7]:
```
print(f'There are {num_untreated} elements in the UnTreated List')
```

```
There are 13 elements in the UnTreated List
```

Exercise 3: (5 points)

Create two new lists (*trt_sorted*, *untrt_sorted*) using the $copy()$ method that are sorted versions of the original data

In [8]:
```
## Exercise Answer

#Creating new lists using the copy() method and sorting the new list using sort()
trt_sorted = treated.copy()
trt_sorted.sort()

untrt_sorted=untreated.copy()
untrt_sorted.sort()
```

In [46]:
```
print(f'Sorted Treated List: {trt_sorted}')
```

```
Sorted Treated List: [6, 7, 13, 16, 18, 28, 32, 33, 35, 38, 43, 50]
```

In [45]:
```
print(f'Sorted UnTreated List: {untrt_sorted}')
```

```
Sorted UnTreated List: [21, 23, 26, 28, 37, 39, 39, 40, 42, 48, 54, 58, 63]
```

Exercise 4: (5 points)

Create a function called $\geq t_nlar \geq st()$ that takes in a list as input, and an optional parameter *num* indicating how many values to return but with a default value of 3. The function should then return the *num* largest values from the list. Apply this function to *treated* with num = 4 and *untreated* with the default value.

In [29]:
```
## Exercise Answer
def get_nlargest(input_list, num = 3):
    input_list.sort()
    return input_list[-num: ]
```

In [30]:
```
#Applying get_nlargest function to treated list with num = 4
treated = [18, 43, 28, 50, 16, 32, 13, 35, 38, 33, 6, 7]
get_nlargest(treated,4)
```

Out[30]:
```
[35, 38, 43, 50]
```

In [31]:
```
#Applying get_nlargest function to untreated with default value
untreated = [40, 54, 26, 63, 21, 37, 39, 23, 48, 58, 28, 39, 42]
get_nlargest(untreated)
```

Out[31]:
```
[54, 58, 63]
```

Exercise 5: (5 points)

Create a function $\geq t_mean()$ to calculate the mean of a list input called *num_list* and return the mean. Use this function to calculate the average number of tapeworms for each group. Use fstrings and the round function to nicely print the output as a complete sentence. Use two decimal places.

In [32]:
```
## Exercise Answer
# Calculating the mean of the Input List and rounding it to 2 decimals
def get_mean(num_list):
    n = len(num_list)
    mean = sum(num_list) / n
    avg_list = round(mean,2)
    return avg_list
```

we can also use the mean method from statistics package, avg_list = round(mean(num_list),2)

Calculating the Mean or Average number of the Treated Tapeworms

In [33]:
```
treated = [18, 43, 28, 50, 16, 32, 13, 35, 38, 33, 6, 7]
avg_treated_tapeworms = get_mean(treated)
print(f'The Average Number of Treated Tapeworms are {avg_treated_tapeworms}')
```

```
The Average Number of Treated Tapeworms are 26.58
```

Calculating the Mean or Average number of the UnTreated Tapeworms

In [34]:
```
untreated = [40, 54, 26, 63, 21, 37, 39, 23, 48, 58, 28, 39, 42]
avg_untreated_tapeworms = get_mean(untreated)
print(f'The Average Number of UnTreated Tapeworms are {avg_untreated_tapeworms}')
```

```
The Average Number of UnTreated Tapeworms are 39.85
```

Exercise 6: (5 points)

Create a function $\geq t_d \Leftrightarrow erence_means()$ to calculate the difference in the average between two lists: *num_list1* and *num_list2*. Use this function to calculate the difference in the average number of tapeworms for the *treated* and *untreated* sheep. What does this suggest about the efficacy of the tapeworm drug? Does it seem like it may be effective in reducing the number of tapeworms in sheep?

In [17]:
```
## Exercise Answer
treated = [18, 43, 28, 50, 16, 32, 13, 35, 38, 33, 6, 7]
untreated = [40, 54, 26, 63, 21, 37, 39, 23, 48, 58, 28, 39, 42]

def get_mean(num_list):
    n = len(num_list)
    mean = sum(num_list) / n
    avg_list = round(mean,2)
    return avg_list

def get_difference_means(num_list1, num_list2):
    avg_diff = round(get_mean(num_list1) - get_mean(num_list2),2)
    return avg_diff
```

We can also use mean function from statistics avg_diff = mean(num_list1) - mean(num_list2)

calculating the difference in the average number of tapeworms for the treated and untreated

In [18]:
```
mean_diff = get_difference_means(untreated,treated)
print(f'The difference in the average number of tapeworms for the untreated and treated sheep is {mean_diff}')
```

```
The difference in the average number of tapeworms for the untreated and treated sheep is 13.27
```

There is a significant difference between the average number of the tapeworms for the untreated and treated sheep. It seems like drug is effective in reducing the number of tapeworms in sheep.

Exercise 7: (5 points)

Create a function $\geq t_std()$ to calculate the standard deviation in number of tapeworms for each group. The function should have a single parameter called *num_list* and return the standard deviation. This function will require the import of the math package to add the ability to calculate a square root ($math.\sqrt{}$).

The sample standard deviation s of a list of numbers $x_1, x_2, x_3, \ldots, x_n$ with sample mean $\bar{x}$ is defined to be:

$$s = \sqrt{\sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}}$$

Function to calculate the Standard Deviation

In [35]:
```
## Exercise Answer

#Importing math package for using math.sqrt() function
import math

#Function to calculate the Standard Deviation
def get_std(num_list):
    n = len(num_list)
    mean = sum(num_list) / n
    variance = sum((x - mean)**2 for x in num_list) / (n - 1)
    std_dev = round(math.sqrt(variance),2)
    return std_dev
```

calculating the the standard deviation in number of tapeworms for treated group

In [36]:
```
treated = [18, 43, 28, 50, 16, 32, 13, 35, 38, 33, 6, 7]
trtd_stddev= get_std(treated)
print(f'The standard deviation in number of tapeworms for treated group is: {trtd_stddev}')
```

```
The standard deviation in number of tapeworms for treated group is: 14.36
```

calculating the the standard deviation in number of tapeworms for untreated group

In [37]:
```
untreated = [40, 54, 26, 63, 21, 37, 39, 23, 48, 58, 28, 39, 42]
trtd_stddev= get_std(untreated)
print(f'The standard deviation in number of tapeworms for untreated group is: {trtd_stddev}')
```

```
The standard deviation in number of tapeworms for untreated group is: 13.28
```

Exercise 8: (5 points)

Create a function called get_max_min() that takes a single list as input and returns a tuple with the first element being the minimum and the second element being the maximum. Apply this function to treated and untreated. Add a default parameter pretty_print that is by default set to False but if True prints out a complete sentence with min and max as shown below. Note, if you have a list of numbers, max(list) will return the maximum value and min(list) will return the minimum value of the list.

Note, if you have a list of numbers, $\max(list)$ will return the maximum value and $\min(list)$ will return the minimum value of the list.

Test Input:

```
get_min_max(treated, pretty_print = True)
```

Example output:

```
The minimum is 6 and the maximum is 50
(6, 50)
```

Test Input:

```
get_min_max(treated, pretty_print = False)
```

Example output:

```
(6, 50)
```

In [38]:
```
## Exercise Answer

#Function to return min and max values from the List
def get_min_max(treated, pretty_print = False):
    max_min_values = (min(treated),max(treated))
    if pretty_print == True:
        print(f'The minimum is {max_min_values[0]} and the maximum is {max_min_values[1]}')
    return max_min_values
```

Applying get_min_max function to the Treated List

In [23]:
```
treated = [18, 43, 28, 50, 16, 32, 13, 35, 38, 33, 6, 7]

#pretty_print is True, so we get the complete sentence in output
get_min_max(treated, True)
```

```
The minimum is 6 and the maximum is 50
```

Out[23]:
```
(6, 50)
```

In [24]:
```
#pretty_print is False by default, so we dont get the complete sentence in output
get_min_max(treated)
```

Out[24]:
```
(6, 50)
```

Applying get_min_max function to the UnTreated List

In [25]:
```
untreated = [40, 54, 26, 63, 21, 37, 39, 23, 48, 58, 28, 39, 42]

#pretty_print is True, so we get the complete sentence in output
get_min_max(untreated, True)
```

```
The minimum is 21 and the maximum is 63
```

Out[25]:
```
(21, 63)
```

In [26]:
```
#pretty_print is False by default, so we dont get the complete sentence in output
get_min_max(untreated)
```

Out[26]:
```
(21, 63)
```

Exercise 9: (5 points)

Create a dictionary called 'data_dict' containing the sheep experiment data. Use the string literals 'treated' and 'untreated' as the **keys** and use the lists, *treated* and *untreated*, created in Exercise 1 as the **values**.

In [41]:
```
## Exercise Answer

#Creating empty dictionary
data_dict = {}

#Adding the Lists into dictionary
data_dict['treated'] = [18, 43, 28, 50, 16, 32, 13, 35, 38, 33, 6, 7]
data_dict['untreated'] = [40, 54, 26, 63, 21, 37, 39, 23, 48, 58, 28, 39, 42]

data_dict
```

Out[41]:
```
{'treated': [18, 43, 28, 50, 16, 32, 13, 35, 38, 33, 6, 7],
 'untreated': [40, 54, 26, 63, 21, 37, 39, 23, 48, 58, 28, 39, 42]}
```

Exercise 10: (10 points)

Create a function called grader() that accepts a dictionary containing student names as keys and their midterm scores as values. Returns a dictionary with the students names as keys and a dictionary containing midterm score and letter grade as values.

test_scores_dict = {'Chis':88, 'Neal':56, 'Mary':72, 'Sasha':99, 'John':91, 'Mike':87, 'Kayla': 62, 'Caylee':85}

Example return for first student:

test_scores_return = {'Chis':{'midterm':88, 'grade':'B'}}

In [42]:
```
test_scores_dict = {'Chis':88, 'Neal':56, 'Mary':72, 'Sasha':99, 'John':91, 'Mike':87, 'Kayla': 62, 'Caylee':85}

# Funtion which returns a dictionary with the students names as keys and a dictionary containing midterm score and letter grade as values
def grader(input_dict):
    stud_grades = {}

    for x in input_dict.keys():
        midgrades_dict = {}

        midgrades_dict['midterm'] = input_dict[x]

        # Updating grades based on midterm marks
        if input_dict[x] >= 90:
            midgrades_dict['grade'] = 'A'
        elif input_dict[x] >= 80:
            midgrades_dict['grade'] = 'B'
        elif input_dict[x] >= 70:
            midgrades_dict['grade'] = 'C'
        elif input_dict[x] >= 60:
            midgrades_dict['grade'] ='D'
        else:
            midgrades_dict['grade'] = 'Fail'

        stud_grades[x] = midgrades_dict

    return stud_grades

grader(test_scores_dict)
```

Out[42]:
```
{'Chis': {'midterm': 88, 'grade': 'B'},
 'Neal': {'midterm': 56, 'grade': 'Fail'},
 'Mary': {'midterm': 72, 'grade': 'C'},
 'Sasha': {'midterm': 99, 'grade': 'A'},
 'John': {'midterm': 91, 'grade': 'A'},
 'Mike': {'midterm': 87, 'grade': 'B'},
 'Kayla': {'midterm': 62, 'grade': 'D'},
 'Caylee': {'midterm': 85, 'grade': 'B'}}
```