# DATA 690 Homework 1 (50 points - Due on Sunday, February 12, 2023 by 11:59 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_Homework1.pdf (example: Thomas_Sunela_Homework1.pdf) **AND** Lastname_Firstname_Homework1.ipynb (example: Thomas_Sunela_Assignment1.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
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]

#printing
print(f'List of Treated tapeworms: {Treated}')
print(f'List of Untreated tapeworms: {Untreated}')

#confirming
print(f'\nConfirming the Type of Treated list : {type(Treated)}')
print(f'Confirming the Type of Untreated list : {type(Untreated)}')
```

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

Confirming the Type of Treated list : <class 'list'>
Confirming the Type of Untreated list : <class 'list'>
```

Created ***Treated*** and ***Untreated*** lists and also confirmed their type using the **type()** function.

## 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.

In [2]:

```
## Exercise Answer

print(f"\n'Treated' list has a total of {len(Treated)} elements and 'Untreated' list ha
```

```
'Treated' list has a total of 12 elements and 'Untreated' list has a total of 13 elemen
ts.
```

### Explanation:

1. Here print statement uses an f-string, as explained in our class.
2. Used ***len()*** function is to determine the number of elements in each list, and the values are then inserted into the string using the {} syntax.

## Exercise 3: (5 points)

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

In [3]:

```
## Exercise Answer

#Copying treated list
trt_sorted = Treated.copy()
trt_sorted.sort()

#Copying untreated list
untrt_sorted = Untreated.copy()
untrt_sorted.sort()

#printing
print(f' Sorted Treated list: {trt_sorted}')
print(f' Sorted Untreated list: {untrt_sorted}')
```

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

### Explanation:

- Used the ***copy()*** method to create new lists.

- Then used ***sort()*** function to sort the elements in each list in ascending order.
- Finally, the sorted lists are printed using f-strings.

## 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. Make sure output of both lists are shown.

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

#defining the function
def get_nlargest(x, num=3):
    x.sort()
    return x[-num: ]

#printing results
print(f'\n Four largest numbers in the Treated list : {get_nlargest(Treated, 4)}')
print(f'\n Largest numbers in the Untreated list : {get_nlargest(Untreated)}')
```

```
Four largest numbers in the Treated list : [35, 38, 43, 50]

Largest numbers in the Untreated list : [54, 58, 63]
```

### Explanation:

- Function `get_nlargest` takes two arguments: ***x*** and ***num***, where
  - ***x*** = input list and
  - ***num*** = optional parameter with a default value of 3.
- `get_nlargest` sorts the elements in x in ascending order using the ***sort()*** method. Then, it returns the last num elements of the list using slicing, with the syntax ***x[-num:]***

## Exercise 5: (5 points)

Create a function to calculate the median of a list. The function should have a single parameter called *input_list* and return a single float. However, the calculation of the median is slightly different depending if there are an even number of elements in the list or not. If there are an odd number of elements in the list, then the median is the middle number. If there are an even number of elements in the list, then the median is the average of the middle two numbers. Use this function to calculate the median number of tapeworms for **each group** (*treated* and *untreated*). Display the output in a good format

In [6]:
```
## Exercise Answer

def median(input_list):
    half = len(input_list) // 2
    input_list.sort()
    if not len(input_list) % 2:
        return float((input_list[half - 1] + input_list[half]) / 2.0)
    return float(input_list[half])


print(f' The median is: {median(Treated)} in the Treated list of tapeworms.')
print(f' The median is: {median(Untreated)} in the Untreated list of tapeworms.')
```

```
The median is: 30.0 in the Treated list of tapeworms.
The median is: 39.0 in the Untreated list of tapeworms.
```

### Explanation:

- The function `median` takes a list ***l*** as input and returns the median value of the list.
- First, I calculated the middle index of the list ***l*** and then sorted in ascending order using the ***sort()*** method.
- Then determined if ***l*** is even or odd. If:
  - ***l*** is even, then median = average of the two middle values.
  - ***l*** is odd, then median = the middle value of the list l, which is accessed using the index half.
- Finally, the function is applied to the Treated and Untreated lists and the results are printed using f-strings.

## Exercise 6: (5 points)

Create a function $\geq t_s td()$ 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 ($math.\sqrt{}$). Remember to display std of both lists in a clear format in 2 decimal places.

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:

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

In [7]:

```python
## Exercise Answer

import math

def get_mean(li):
    return sum(li) / len(li)


def get_std_dev(li):
    n = len(li)
    mean = get_mean(li)
    square_list = []
    for item in li:
        square_list.append((item - mean) ** 2)
    return math.sqrt(sum(square_list) / n)


print(f' The Standard Deviation is {get_std_dev(Treated):.2f} for the Treated list.')
print(f' The Standard Deviation is {get_std_dev(Untreated):.2f} for the Untreated list.
```

```
The Standard Deviation is 13.75 for the Treated list.
The Standard Deviation is 12.76 for the Untreated list.
```

### Explanation:

In this code,

- ***n*** = length of the list num_list, and ***mean*** = mean of the list.

- Calculated a `square_list` of the input list using a ***for*** loop.
- Finally, the standard deviation is calculated as the square root of the `square_list` divided by ***n***.
- The function returns the standard deviation.
- The print() statements are used to display the standard deviation of the treated and untreated lists in a clear format with 2 decimal places.

## Exercise 7: (10 points)

Create a function called $\geq t_quarti \leq s()$ to calculate the first and third quartiles of a list. The function should accept a list called *num_list* and return a tuple with (Q1, Q3) and 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 Q1 and Q3.

To calculate the quartiles, use the following algorithm:

1. Use the median to divide the ordered data set into two halves.
   - If there are an odd number of data points in the original ordered data set, include the median (the central value in the ordered list) in both halves.
   - If there are an even number of data points in the original ordered data set, split this data set exactly in half.
2. The lower quartile value is the median of the lower half of the data. The upper quartile value is the median of the upper half of the data.

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

def get_quartiles(temp_list, pretty_print=False):
    temp_list.sort()
    n = len(temp_list)
    if n % 2 == 0:
        Q1 = (temp_list[n//4-1] + temp_list[n//4]) / 2
        Q3 = (temp_list[3*n//4-1] + temp_list[3*n//4]) / 2
    else:
        Q1 = temp_list[n//4]
        Q3 = temp_list[3*n//4]

    if pretty_print:
        return f"\tThe first quartile (Q1) is {Q1} and the third quartile (Q3) is {Q3}.
    else:
        return (Q1, Q3)


print(f' Non-Pretty print example : {get_quartiles(Treated)}\n')

print(f' Pretty print example : \n{get_quartiles(Untreated,True)}')
```

```
 Non-Pretty print example : (14.5, 36.5)

 Pretty print example :
        The first quartile (Q1) is 28 and the third quartile (Q3) is 48.
```

### Explanation:

The `get_quartiles` function takes a list , `temp_list` in this case, and `pretty_print` which is set to default-False as inputs.

- Calculated the length of the temp_list and stored in the variable n.
- If ***n*** is even, then

    - ***Q1*** = average of the two middle numbers in the lower half of the list and
    - ***Q3*** = average of the two middle numbers in the upper half of the list.
- If ***n*** is odd, then
    - ***Q1*** = middle number in the lower half of the list and
    - ***Q3*** = middle number in the upper half of the list.

If `pretty_print` is set to ***True***, then the output will be printed in a complete sentence, else it will by default be a tuple. Used a second `if/else` statement is for that reason.

## Exercise 8: (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**. Display the dictionary

In [9]:
```
## Exercise Answer

#Creating a dictionary
data_dict = {

    '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]
}

#printing
data_dict
```

Out[9]:
```
{'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 [10]:
```
keys = data_dict.keys() #finding the keys of the dictionary
print("The keys in the dictionary are: {}".format(', '.join(keys))) #printing keys
```

```
The keys in the dictionary are: Treated, Untreated
```

## Exercise 9: (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 = {'Chris':88, 'Neal':56, 'Mary':72, 'Sasha':99, 'John':91, 'Mike':87, 'Kayla': 62, 'Caylee':85}

Example return for first student:

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

In [11]:
```
## Exercise Answer

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

```
}

test_scores_dict
```

Out[11]:
```
{'Chris': 88,
 'Neal': 56,
 'Mary': 72,
 'Sasha': 99,
 'John': 91,
 'Mike': 87,
 'Kayla': 62,
 'Caylee': 85}
```

In [12]:
```
#defining the function

def grader(dictionary):
    temp_dict = {} #temporary dictionary
    for key, value in dictionary.items():

        #assigning grades
        if value>=90:
            grade = 'A'

        elif value>=80:
            grade = 'B'

        elif value>=70:
            grade = 'C'

        elif value>=60:
            grade = 'D'
        else:
            grade = 'F'
        temp_dict[key]= {'Midterm': value, 'Grade':grade} #storing results in temporary
    return temp_dict

test_scores_return = grader(test_scores_dict) #calling the function
test_scores_return
```

Out[12]:
```
{'Chris': {'Midterm': 88, 'Grade': 'B'},
 'Neal': {'Midterm': 56, 'Grade': 'F'},
 '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'}}
```

## Explanation:

- The function `grader` takes dictionary as input and returns `temp_dict` .
- I've used `if/elif` statements for different range of values, as you taught in our second class, to assign grades for students. And saved them to a temporary dictionary, which contains the names of students as `key` .
- Finally, I called the `grader` function with `test_scores_dict` as an input, and assigned it to the final `test_scores_return` dictionary.