Discrete Structures!

CMPSC 102

Statistical Tools

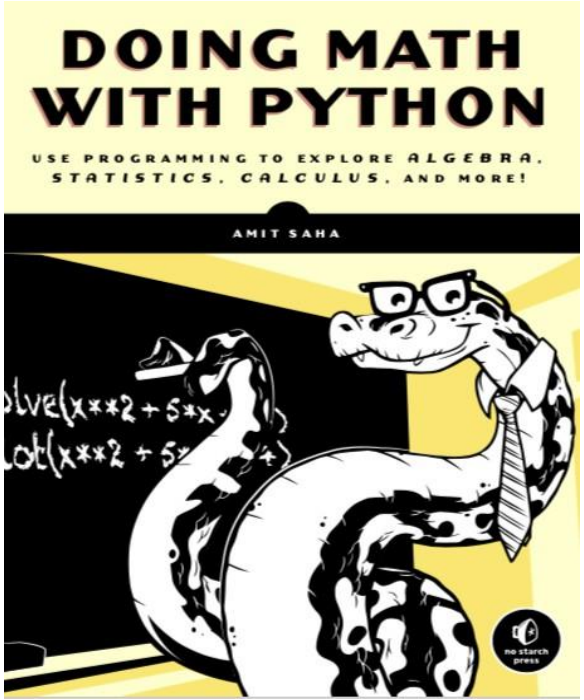
ALLEGHENY COLLEGE

Key Questions and Learning Objectives

- How can I describe data using statistical tools such as correlation, variance, standard deviation and others?

- To remember and understand some concepts about plots, and the code used to make them from matplotlib and other libraries.

Where Are We Now?



Saha, Chapter 3: Describing Data with Statistics

- Basic statistics
- General: Frequencies, Mode, Median, Range, etc
- Complex: Correlation, Quantiles (more to come!)

Setting Up Virtual Environment

- Create a project directory

```
mkdir week14_projects
cd week14_projects
```

- Create virtual environment using Python

```
python3 -m venv myenv
# see the file tree
find . -not -path '*/\.*'
```

- Activate myenv the virtual environment

```
source myenv/bin/activate  # macOS/Linux
myenv\Scripts\activate     # Windows
```

- Install Dependencies

```
pip install matplotlib numpy pandas plotly
```

Basic Stats

Check your sandbox/ for example files!

- See `sandbox/` files to experiment with statistical code
  - Examples: Frequencies, Mode, Range, Plotting
  - Be sure to build your virtual environment first to load the libraries!! (See next slide for help)

Basic Stats

Check your sandbox/freq.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

'''
Frequency table for a list of numbers
'''
from collections import Counter

def frequency_table(numbers):
        table = Counter(numbers)
        print('Number\tFrequency')
        for number in table.most_common():
                # print(f"number : {number}"),
                print('value: {0}\tfreq: {1}'.format(number[0], number[1]))

if __name__=='__main__':
        scores = [7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10]
        print(f"Scores :{scores}")
        frequency_table(scores)
```

Basic Stats

Check your sandbox/mode_commonElements.py

```
'''
Finding the Most Common Elements
'''
from collections import Counter

simplelist = [4, 2, 1, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4] # four is most common int in the list
c = Counter(simplelist)
c.most_common()
print(f"counter : {c}")
print(f"c.most_common() : {c.most_common()}")

# [(4, 10), (2, 1), (1, 1), (3, 1)]
print(f"c.most_common(1) : {c.most_common(1)}")
# c.most_common(1) : [(4, 10)], four with ten counts

print(f"Mode is the following : {c.most_common(1)}")
print(f"Mode is the following : {c.most_common(1)[0][0]}")
```

Basic Stats

Check your sandbox/mode_multipleModes.py (part 1)

```
'''
Calculating the mode when the list of numbers may have multiple modes
'''

from collections import Counter
def calculate_mode(numbers):
        c = Counter(numbers)
        numbers_freq = c.most_common()
        max_count = numbers_freq[0][1]
        modes = []
        for num in numbers_freq:
                if num[1] == max_count:
                        modes.append(num[0])
        return modes

if __name__ == '__main__':
```

Basic Stats

Check your sandbox/mode_multipleModes.py (part 2)

```
scores = [5, 5, 5, 4, 4, 4, 9, 1, 3]

        print(f" score are the following: {scores}")
        modes = calculate_mode(scores)
        print(" The mode(s) of the list of numbers are:")

        counter = 1
        for mode in modes:
                print(f" {counter}: Common number: {mode}")
                counter += 1
```

Basic Stats

Check your sandbox/findRange.py

```
'''
Find the range
'''

def find_range(numbers):
        lowest = min(numbers)
        highest = max(numbers)
        # Find the range
        r = highest-lowest
        return lowest, highest, r

if __name__ == '__main__’:
        donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]
        lowest, highest, r = find_range(donations)
        print('Lowest: {0} Highest: {1} Range: {2}'.format(lowest, highest, r))
```

Basic Stats

Check your sandbox/freq_plot.py (part 1)

```
'''
Frequency table for a list of numbers, make a plot of frequencies
'''
from collections import Counter
import matplotlib.pyplot as plt

def frequency_table(numbers:list) -> list:
        table = Counter(numbers)
        print('Number\tFrequency')
        char_list = [] # characters
        freq_list = [] # freqs of characters
        for number in table.most_common():
                # print(f"number : {number}"),
                print('value: {0}\tfreq: {1}'.format(number[0], number[1]))
                char_list.append(number[0])
                freq_list.append(number[1])
        return char_list, freq_list
# end of frequency_table()
```

Basic Stats

Check your sandbox/freq_plot.py (part 2)

```
def plot_character_frequencies(char_list, freq_list):
                # Prepare data for plotting
                # characters = list(char_freq.keys())
                # frequencies = list(char_freq.values())
                # Plotting
                plt.bar(char_list, freq_list)
                plt.title('Character Frequencies’)
                plt.xlabel('Character’)
                plt.ylabel('Frequency’)
                plt.xticks(rotation=45)
                plt.show()
# end of plot_character_frequencies()

if __name__=='__main__’:
                scores = [7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10]
                print(f"Scores :{scores}")
                char_list, freq_list = frequency_table(scores)
                plot_character_frequencies(char_list, freq_list)
```

Basic Stats

Check your sandbox/mean.py

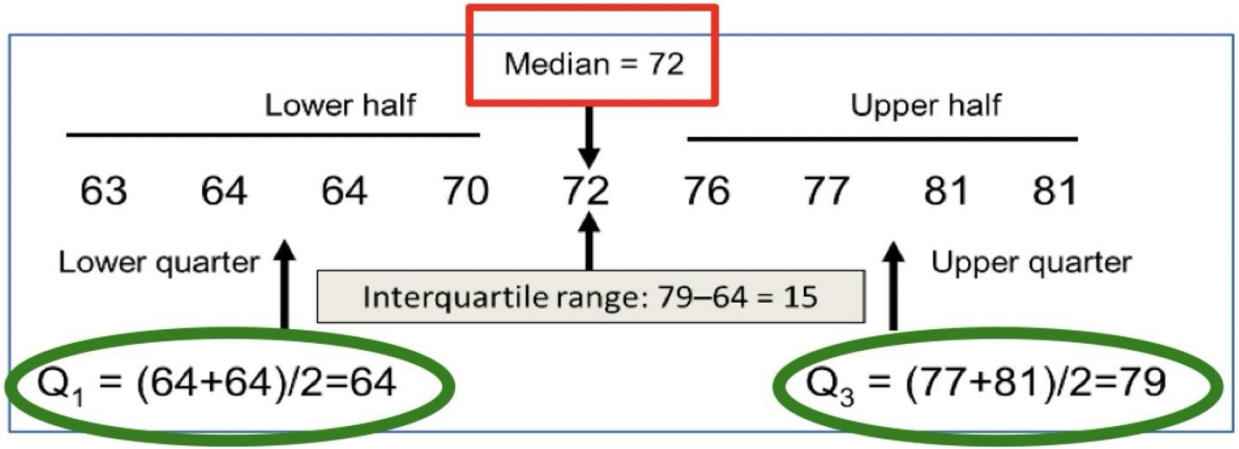
```
'''
Calculating the mean
'''
def calculate_mean(numbers):
        s = sum(numbers)
        N = len(numbers)
        # Calculate the mean
        mean = s/N
        return mean

# calculate_mean
if __name__ == '__main__’:
        donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]
        mean = calculate_mean(donations)
        N = len(donations)
        print('Mean donation over the last {0} days is {1}'.format(N, mean))
```

Quantiles and Medians

Check your sandbox/ median example:"median.py"

- Determine exactly what value "splits" the dataset.
- The Q s denote the four quarters ("Quantiles")
    - Used to determine specify areas of the sorted dataset
    - Q1 is at first 1 position of data, Q3 is the 3 position in data



Quantiles and Medians

Check your sandbox/ median example:"median.py"

- You must know if the dataset has an odd or even number of elements

First, arrange the observations in an ascending order.

If the number of observations (n) is odd: the median is the value at position

$$\left(\frac{n+1}{2}\right)$$

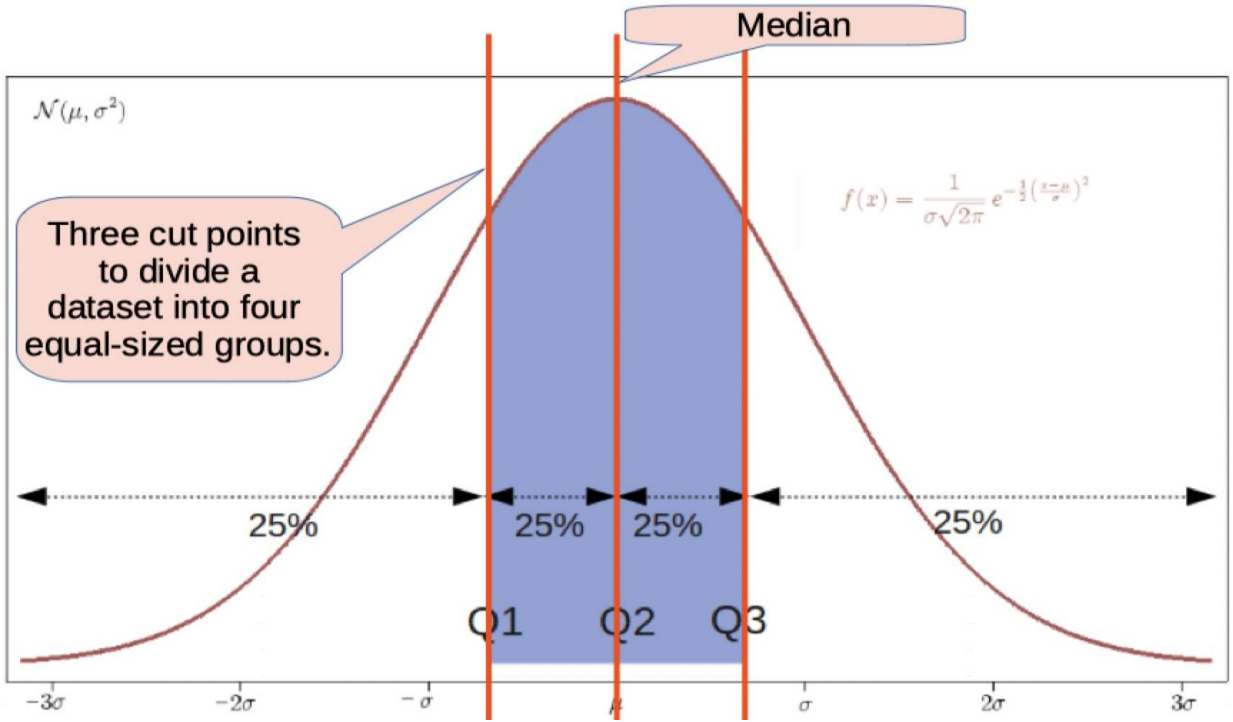
If the number of observations (n) is even:

1. Find the value at position $\left(\frac{n}{2}\right)$
2. Find the value at position $\left(\frac{n+1}{2}\right)$
3. Find the average of the two values to get the median.

Quantiles and Medians

Check your sandbox/ median example:"median.py"

- Determine exactly what value "splits" the dataset.



Quantiles and Medians

Check your sandbox/ median example:“median.py”

```
def calculate_median(numbers):
        N = len(numbers)
        numbers.sort()
        # Find the median
        if N % 2 == 0:
                # if N is even
                m1 = N/2
                m2 = (N/2) + 1
                # Convert to integer, match position
                m1 = int(m1) - 1
                m2 = int(m2) - 1
                median = (numbers[m1] + numbers[m2])/2
        else:
                m = (N+1)/2
                # Convert to integer, match position
                m = int(m) - 1
                median = numbers[m]
        return median

if __name__ == '__main__':
```

Quantiles and Medians

Check your sandbox/ median example:“median.py”

```
donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]
median = calculate_median(donations)
N = len(donations)
print('Median donation over the last {0} days is {1}'.format(N, median))
```

Correlation

Check your sandbox/ for example: correlation.py

Correlation is a statistical measure that describes the strength and direction of a relationship between two variables. A correlation score ranges from -1 to 1, where:

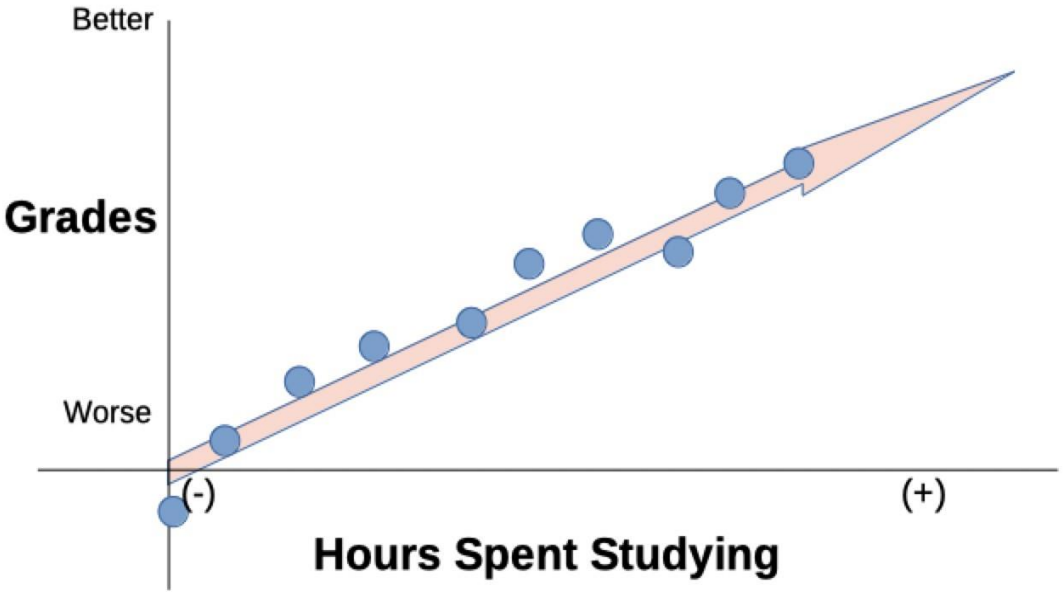
- -1: Perfect negative correlation
- 0: No correlation
- 1: Perfect positive correlation
- Values between -1 and 1 denote the strength of the correlation, as shown in the example below.

Correlation can be calculated using various methods, one of which is the Pearson correlation coefficient.

Correlation

Check your sandbox/ for example: correlation.py

- Positive correlation between x and y

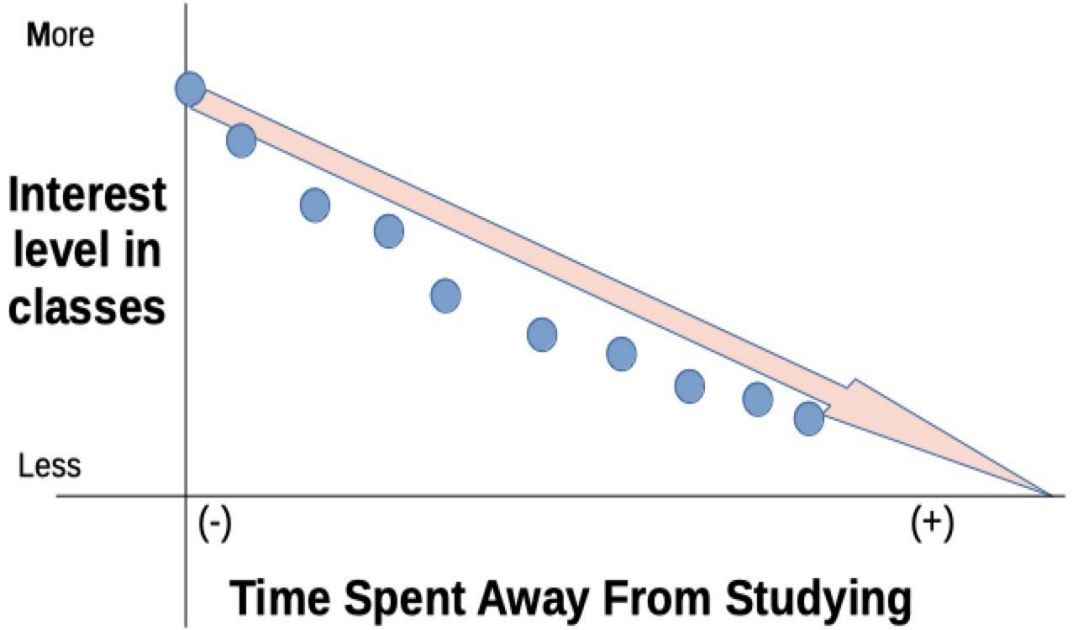


Points lie close to a straight line that has a **positive** gradient.

Correlation

Check your sandbox/ for example: correlation.py

- Negative correlation between x and y

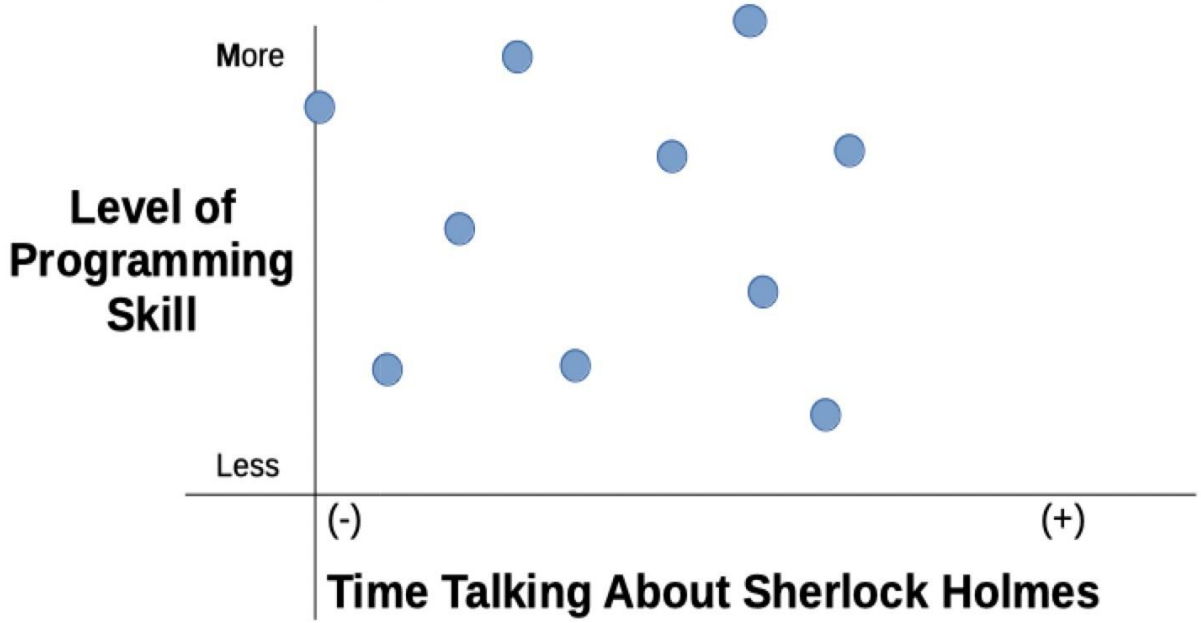


Points lie close to a straight line that has a **negative** gradient.

Correlation

Check your sandbox/ for example: correlation.py

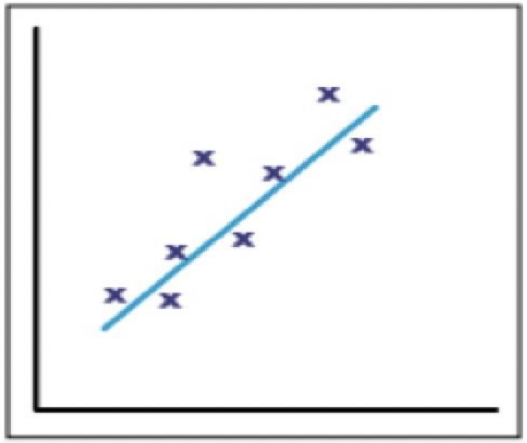
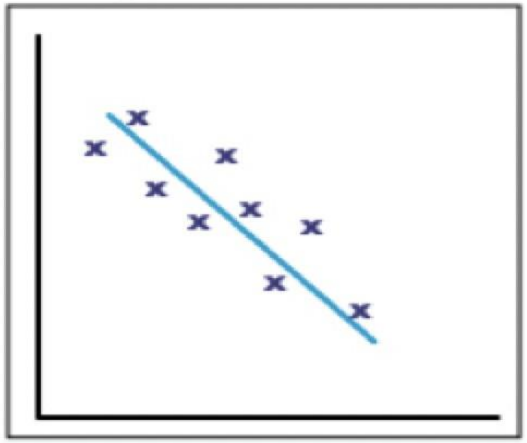
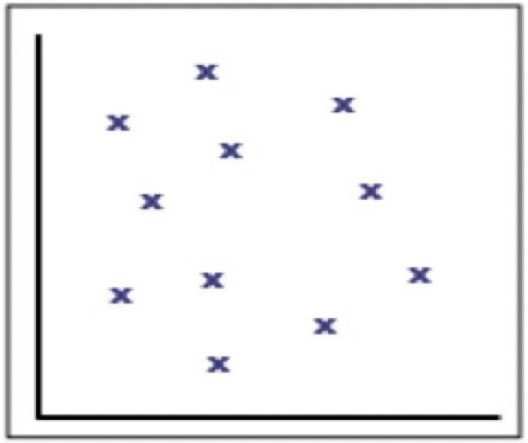
- No correlation between x and y



No pattern exists in the layout of points. :-(

Correlation: Summary

Check your sandbox/ for example: correlation.py

Positive correlation	Negative correlation	No correlation
		
The points lie close to a straight line, which has a positive gradient.	The points lie close to a straight line, which has a negative gradient.	There is no pattern to the points.
This shows that as one variable **increases** the other **increases**.	This shows that as one variable **increases**, the other **decreases**.	This shows that there is **no connection** between the two variables.