

Hands-on Activity 6.1 Introduction to Data Analysis and Tools

CPE311 Computational Thinking with Python

Name: Pascual, Ken Leonard

Section: CPE22S3

Performed on: 04/05/2025

Submitted on: 04/05/2025

Submitted to: Engr. Roman M. Richard

6.1 Intended Learning Outcome

1. Use pandas and numpy data analysis tools
2. Demonstrate how to analyze data using numpy and pandas

6.2 Resources

- Personal Computer
- Jupyter Notebook
- Internet Connection

6.3 Supplementary Activities

Exercise 1

Run the given code below for exercises 1 and 2, perform the given tasks without using any Python modules.

```
In [1]: import random
random.seed(0)
salaries = [round(random.random()*1000000, -3) for _ in range(100)]
print(salaries)
```

```
[844000.0, 758000.0, 421000.0, 259000.0, 511000.0, 405000.0, 784000.0, 303000.0,
477000.0, 583000.0, 908000.0, 505000.0, 282000.0, 756000.0, 618000.0, 251000.0, 9
10000.0, 983000.0, 810000.0, 902000.0, 310000.0, 730000.0, 899000.0, 684000.0, 47
2000.0, 101000.0, 434000.0, 611000.0, 913000.0, 967000.0, 477000.0, 865000.0, 260
000.0, 805000.0, 549000.0, 14000.0, 720000.0, 399000.0, 825000.0, 668000.0, 1000.
0, 494000.0, 868000.0, 244000.0, 325000.0, 870000.0, 191000.0, 568000.0, 239000.
0, 968000.0, 803000.0, 448000.0, 80000.0, 320000.0, 508000.0, 933000.0, 109000.0,
551000.0, 707000.0, 547000.0, 814000.0, 540000.0, 964000.0, 603000.0, 588000.0, 4
45000.0, 596000.0, 385000.0, 576000.0, 290000.0, 189000.0, 187000.0, 613000.0, 65
7000.0, 477000.0, 90000.0, 758000.0, 877000.0, 923000.0, 842000.0, 898000.0, 9230
00.0, 541000.0, 391000.0, 705000.0, 276000.0, 812000.0, 849000.0, 895000.0, 59000
0.0, 950000.0, 580000.0, 451000.0, 660000.0, 996000.0, 917000.0, 793000.0, 82000.
0, 613000.0, 486000.0]
```

Using the data generated above, calculate the following statistics without importing anything from the statistics module i the standard library.

(<https://docs.python.org/3/library/statistics.html>) and then confirm your results match up to those that are obtained when using the statistics module (where possible):

* Mean

Finding the mean is equal to sum of the terms divided by the number of terms so, without a statistics module, the mean can be computed by doing the following:

```
In [2]: mean_no_stat = sum(salaries) / len(salaries)
        print(mean_no_stat)
```

585690.0

```
In [3]: # check if output matches that of using statistics modules in the standard libra
        from statistics import mean
        mean_wstat = mean(salaries)
        print(mean_wstat)
```

585690.0

* Median

There are two ways to find the median, depending on whether the number of observations is odd or even. For this, I used [this](#) so that the program will decide what formula to use.

```
In [4]: salaries.sort()
n = len(salaries)

if len(salaries) % 2 == 0: # the number of terms is even
    # find the two middle values
    middle1 = salaries[n//2-1]
    middle2 = salaries[n//2]
    # find the average of the two middle values
    median_no_stat = (middle1 + middle2) / 2

else: # the number of terms is odd
    median_no_stat = salaries[n // 2]
    # use the result of n//2 as an index to find what element within salaries li

print(median_no_stat)
```

589000.0

```
In [5]: # check if output matches that of using statistics modules in the standard library
from statistics import median
salaries.sort()
median(salaries)
```

Out[5]: 589000.0

* Mode

(hint: check out the Counter in the collections module of the standard library at <https://docs.python.org/3/library/collections.html#collections.Counter>)

Reference I used for this method

```
In [6]: from collections import Counter
def mode(value):
    #the Counter counts how many times an element appears in a set of data
    count_em = Counter(value)
    # output of this one is the value of the mode
    mode_no_stat , _ = count_em.most_common(1)[0]
    # output of this one is how many times the mode appeared
    _ , mode_value = count_em.most_common(1)[0]
    """
    count_em.most_common(1)[0] picks the pair of values that indicate (the value
    for example, the highest value is 477000, and it appeared 3 times. In count_
    So we pluck that out and take it as the output.
    """
    print(f"The value that appears the most number of times is {mode_no_stat}.\n")

    # use the function for the salaries list
    mode(salaries)
```

The value that appears the most number of times is 477000.0.
It appeared 3 times.

```
In [7]: # check if output matches that of using statistics modules in the standard library
from statistics import mode
mode(salaries)
```

Out[7]: 477000.0

* Sample variance

The sample variance is equal to the sum of the squared deviations divided by the number of observations minus one.

We have to make separate calculations for each of these two parts.

```
In [8]: # sum of the squared deviations
# since we need the sum of all squared deviations, we have to run (x - the mean)
SSD = sum((x - mean_no_stat) **2 for x in salaries)
# number of observations - 1
denominator = len(salaries) - 1

# formula for sample variance
variance_no_stat = SSD / denominator
print(variance_no_stat)
```

70664054444.44444

```
In [9]: # check if output matches that of using statistics modules in the standard library
from statistics import variance
variance(salaries)
```

Out[9]: 70664054444.44444

* Sample standard deviation

the sample standard deviation is just the square root of the sample variance. So all that's needed here is to get the output of the computation for the sample variance and get the square root of it.

```
In [10]: import math
print (math.sqrt(variance_no_stat))
```

265827.11382484

```
In [11]: # check if output matches that of using statistics modules in the standard library
from statistics import stdev
stdev(salaries)
```

Out[11]: 265827.11382484

Exercise 2

Using the same data, calculate the following statistics using the functions in the statistics module where appropriate:

- Range

Range = highest value - lowest value

We need to get the highest and lowest values first

[Reference](#)

```
In [12]: #function to get the highest and lowest values in the list
def minmax(val_list):
    min_val = min(val_list)
    max_val = max(val_list)

    return (min_val, max_val)
low, _ = minmax(salaries) # lowest value
print("lowest value:", low)

_, high = minmax(salaries) # highest value
print("highest value:", high)

#compute for range
value_range = high - low
print(value_range)
```

```
lowest value: 1000.0
highest value: 996000.0
995000.0
```

- Coefficient of variation (CV)

CV = population standard deviation / population mean

```
In [13]: from statistics import pstdev
# compute for population standard deviation
pop_std_dev = pstdev(salaries)
```

```
In [14]: print(mean_wstat)
# this has been computed earlier, so let's just call that again
```

```
585690.0
```

```
In [15]: CV = pop_std_dev / mean_wstat
print(round(CV,10))
```

```
0.4515949371
```

- interquartile range

To find the interquartile range, we need to find the 1st and 3rd quartiles first

```
In [16]: from statistics import quantiles
# this returns a list of the quartiles (n=4) for the salaries list
q = quantiles(salaries, n=4)

# extract the first and third quartiles
quart_1 = q[0] # first quartile
print("First Quartile:", quart_1)

quart_3 = q[2] # third quartile
print("Third Quartile:", quart_3)

interquartile_r = quart_3 - quart_1
print("Interquartile Range:", interquartile_r)
```

First Quartile: 400500.0
 Third Quartile: 822250.0
 Interquartile Range: 421750.0

- Quartile coefficient of dispersion

Reference

Basically, this is $\text{interquartile range} / (\text{quartile1} + \text{quartile3})$

We just computed the interquartile range. We only need to find the sum between the two quartiles.

```
In [17]: sum_quarts = quart_3 + quart_1

QCD = interquartile_r / sum_quarts
print("QCD =", round(QCD, 11))
```

QCD = 0.34491923942

Exercise 3: Pandas for Data Analysis

Load the diabetes.csv file. Convert the diabetes.csv file into dataframe.

```
In [31]: import pandas as pd
import numpy as np
diabetes = pd.read_csv('Datasets/diabetes.csv')
```

Perform the following tasks in the diabetes dataframe:

1. Identify the column name

```
In [20]: diabetes.columns
```

```
Out[20]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
               'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
              dtype='object')
```

2. Identify the data types of the data.

```
In [21]: diabetes.dtypes
```

```
Out[21]: Pregnancies      int64
Glucose      int64
BloodPressure  int64
SkinThickness  int64
Insulin      int64
BMI          float64
DiabetesPedigreeFunction float64
Age          int64
Outcome      int64
dtype: object
```

3. Display the total number of records.

```
In [22]: diabetes.count()
```


```
Out[22]: Pregnancies      768  
         Glucose          768  
         BloodPressure    768  
         SkinThickness    768  
         Insulin          768  
         BMI              768  
         DiabetesPedigreeFunction 768  
         Age              768  
         Outcome          768  
         dtype: int64
```

4. Display the first 20 records

```
In [23]: diabetes.head(20)
```

Out[23]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
5	5	116	74	0	0	25.6	
6	3	78	50	32	88	31.0	
7	10	115	0	0	0	35.3	
8	2	197	70	45	543	30.5	
9	8	125	96	0	0	0.0	
10	4	110	92	0	0	37.6	
11	10	168	74	0	0	38.0	
12	10	139	80	0	0	27.1	
13	1	189	60	23	846	30.1	
14	5	166	72	19	175	25.8	
15	7	100	0	0	0	30.0	
16	0	118	84	47	230	45.8	
17	7	107	74	0	0	29.6	
18	1	103	30	38	83	43.3	
19	1	115	70	30	96	34.6	




5. Display the last 20 records

In [24]: `diabetes.tail(20)`

Out[24]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
748	3	187	70	22	200	36.4	
749	6	162	62	0	0	24.3	
750	4	136	70	0	0	31.2	
751	1	121	78	39	74	39.0	
752	3	108	62	24	0	26.0	
753	0	181	88	44	510	43.3	
754	8	154	78	32	0	32.4	
755	1	128	88	39	110	36.5	
756	7	137	90	41	0	32.0	
757	0	123	72	0	0	36.3	
758	1	106	76	0	0	37.5	
759	6	190	92	0	0	35.5	
760	2	88	58	26	16	28.4	
761	9	170	74	31	0	44.0	
762	9	89	62	0	0	22.5	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	




6. Change the Outcome column to Diagnosis.

```
In [34]: diabetes = diabetes.rename(columns = {'Outcome':'Diagnosis'})
```

```
In [35]: diabetes.head()
```

Out[35]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	



7. Create a new column Classification that display "Diabetes" if the value of outcome is 1, otherwise "No Diabetes".

```
In [37]: diabetes['Diagnosis'].value_counts()
```

```
Out[37]: Diagnosis
0      500
1      268
Name: count, dtype: int64
```

```
In [39]: diabetes['Classification'] = diabetes['Diagnosis'].apply(lambda x: 'Diabetes' if
diabetes.head()
```

```
Out[39]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

8. Create a new dataframe "withDiabetes" that gathers data with diabetes

```
In [43]: withDiabetes = diabetes.query('Classification == "Diabetes"')
withDiabetes.head()
```

```
Out[43]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
0	6	148	72	35	0	33.6	
2	8	183	64	0	0	23.3	
4	0	137	40	35	168	43.1	
6	3	78	50	32	88	31.0	
8	2	197	70	45	543	30.5	

9. Create a new dataframe "noDiabetes" that gathers data with no diabetes

```
In [46]: noDiabetes = diabetes.query('Classification == "No Diabetes"')
noDiabetes.head()
```

Out[46]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
--	-------------	---------	---------------	---------------	---------	-----	------------------

1	1	85	66	29	0	26.6	
3	1	89	66	23	94	28.1	
5	5	116	74	0	0	25.6	
7	10	115	0	0	0	35.3	
10	4	110	92	0	0	37.6	



10. Create a new dataframe "Pedia" that gathers data with age 0 to 1

```
In [51]: Pedia = diabetes.query('Age >= 0 and Age <= 1')
Pedia.head()
```

Out[51]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
--	-------------	---------	---------------	---------------	---------	-----	--------------------



11. Create a new dataframe "Adult" that gathers data with age greater than 19

```
In [48]: Adult = diabetes.query('Age >= 19')
Adult.head()
```

Out[48]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
--	-------------	---------	---------------	---------------	---------	-----	--------------------

0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	



12. Use numpy to get the average age and glucose value.

```
In [55]: import numpy as np

age = diabetes['Age'].values
mean_age = np.mean(age)
print("Average age:", round(mean_age, 4))

glucose = diabetes['Glucose'].values
mean_glucose = np.mean(glucose)
print("Average glucose value:", round(mean_glucose, 4))
```

Average age: 33.2409

Average glucose value: 120.8945

13. Use numpy to get the median age and glucose value.

```
In [58]: import numpy as np

diabetes.sort_values(by = 'Age')
age = diabetes['Age'].values
median_age = np.median(age)
print("Median age:", round(median_age, 4))
```

Median age: 29.0

```
In [59]: diabetes.sort_values(by = 'Glucose')
glucose = diabetes['Glucose'].values
median_glucose = np.median(glucose)
print("Median glucose value:", round(median_glucose, 4))
```

Median glucose value: 117.0

14. Use numpy to get the middle values of glucose and age.

```
In [60]: import numpy as np

diabetes.sort_values(by = 'Age')
age = diabetes['Age'].values
median_age = np.median(age)
print("Median age:", round(median_age, 4))

diabetes.sort_values(by = 'Glucose')
glucose = diabetes['Glucose'].values
median_glucose = np.median(glucose)
print("Median glucose value:", round(median_glucose, 4))
```

Median age: 29.0

Median glucose value: 117.0

15. Use numpy to get the standard deviation of the skinthickness.

```
In [61]: skintthicc = diabetes['SkinThickness'].values
stdev_skinthicc = np.std(skinthicc)
print("Standard deviation of skin thickness:", round(stdev_skinthicc,4))
```

Standard deviation of skin thickness: 15.9418

6.4 Conclusion

In this activity I was introduced and familiarized with the data analysis tools used in Python. I observed that importing libraries for data analysis greatly helped me in performing the tasks provided, and was much easier to utilize instead of figuring out a formula for each statistical term. For instance, I don't need to think about the formula for getting the median value since I can just call median(). However, being knowledgeable about the statistical formulae helps in understanding the idea behind those imported functions.

Furthermore, I was able to review about utilizing pandas to analyze datasets and dataframes. I was able to discover ways to filter the dataset based on different criteria, as well as perform basic statistical analysis using numpy. I was used in using the pandas libraries to perform statistical analysis, but using numpy has allowed me to be a bit more familiarized in the process of analyzing data statistically.

Overall, this activity helped enhance my knowledge in using basic data analysis tools and also allowed me to review concepts I have learned in my past courses.