Hands-on Activity 6.1 Introduction to Data Analysis and Tools

CPE311 Computational Thinking with Python

Name: Bulambao, Adrian Justin

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)]
```

Using the data generated above, calculate the following statistics without importing anything from the statistics module in 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

Median

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)

Sample variance

Sample standard deviation

In [2]:
```
# Write a comment per statistical function
print(salaries)
```

```
[844000.0, 758000.0, 421000.0, 259000.0, 511000.0, 405000.0, 784000.0, 303000.0, 477
000.0, 583000.0, 908000.0, 505000.0, 282000.0, 756000.0, 618000.0, 251000.0, 910000.
0, 983000.0, 810000.0, 902000.0, 310000.0, 730000.0, 899000.0, 684000.0, 472000.0, 1
01000.0, 434000.0, 611000.0, 913000.0, 967000.0, 477000.0, 865000.0, 260000.0, 80500
0.0, 549000.0, 14000.0, 720000.0, 399000.0, 825000.0, 668000.0, 1000.0, 494000.0, 86
8000.0, 244000.0, 325000.0, 870000.0, 191000.0, 568000.0, 239000.0, 968000.0, 80300
0.0, 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, 445000.0, 596000.0, 3850
00.0, 576000.0, 290000.0, 189000.0, 187000.0, 613000.0, 657000.0, 477000.0, 90000.0,
758000.0, 877000.0, 923000.0, 842000.0, 898000.0, 923000.0, 541000.0, 391000.0, 7050
00.0, 276000.0, 812000.0, 849000.0, 895000.0, 590000.0, 950000.0, 580000.0, 451000.
0, 660000.0, 996000.0, 917000.0, 793000.0, 82000.0, 613000.0, 486000.0]
```

In [3]:
```
#Mean
#the formula for mean is the sum of all entries divided by the number of entries
mean = sum(salaries)/len(salaries)
print("The mean is: ", mean)
#sum() is a function to get the sum of all entries
#len() is a function to get the number of all entries
```

```
The mean is:  585690.0
```

In [4]:
```
#Median
#the median is the middle values in a set of numbers
# in getting the median, the values must be sorted,
#the sorted() function is used to sort the entries ascendingly
salaries = sorted(salaries)
print(len(salaries))#this just shows the number of entries
```

```
100
```

In [5]:
```
#since the number of entries is 100, the 50th and 51st number is needed
median = (salaries[49] + salaries[50])/2
#in indexing it always starts at 0
#meaning the 50th number is in the 49th index and the 51st number is in the 50th in
```

```
# the total number of entries is even so the middle of the two values is needed
print("The median is: ", median)
```

```
The median is:  589000.0
```

In [6]:
```
#Mode (hint: check out the Counter in the collections module of the standard librar
#https://docs.python.org/3/library/collections.html#collections.Counter)
# this list all of the numbers without duplicates
unique = set(salaries)
#this part shows the number of entries for the entry with the maximum value
max_count = max([salaries.count(x) for x in unique])
# the part of "x for x in unique" puts the number in the set made
# the "if salaries.count(x) == max_count" is a statement if the number of the numbe
#is equal to the max_count made or the number that is the maximum amount of entries
mode = [x for x in unique if salaries.count(x) == max_count]
print("The mode is: ", mode)
```

```
The mode is:  [477000.0]
```

In [7]:
```
#Sample variance
#the formula for sample variance is
#the average of the squared differences between each data point and the sample mean
def s_variance(list):
    n = len(list) # is the number of entries
    mean = sum(list)/n # mean of the whole list
    #this part below gets the squared differences between each entry and mean
    summ = sum((x-mean)**2 for x in list)
    # the average is then taken to get the sample variance
    s_v = summ/(n - 1)
    return s_v
sample_variance = s_variance(salaries)
print("The sample variance is: ", sample_variance)
```

```
The sample variance is:  70664054444.44444
```

In [8]:
```
#Sample standard deviation
#the standard deviation is the square root of the sample variance
def s_deviation(list):
    n = len(list) # is the number of entries
    mean = sum(list)/n # mean of the whole list
    #this part below gets the squared differences between each entry and mean
    summ = sum((x-mean)**2 for x in list)
    # the average is then taken to get the sample variance
    s_v = summ/(n - 1)
    return (s_v) ** 0.5 # ** is a command for exponent a number raised to 0.5 is eq
standard_deviation = s_deviation(salaries)
print("The standard deviation is: ", standard_deviation)
```

```
The standard deviation is:  265827.11382484
```

Exercise 2

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

Range

Coefficient of variation Interquartile range

Quartile coefficient of dispersion

In [9]:
```
import statistics
import numpy as np
```

In [10]:
```
#Range
# the formula for range is the difference between the maximum and minimum values
range = max(salaries) - min(salaries)
print("The range is: ", range)
```

```
The range is:  995000.0
```

In [11]:
```
#Coefficient of variation
# the formula for coefficient of variation is the standard deviation divided by the
# multiplied by 100
sd = statistics.stdev(salaries)
mean = statistics.mean(salaries)
cv = sd/mean *100
print("The coefficient of variation: ", cv)
```

```
The coefficient of variation:  45.38699889443903
```

In [12]:
```
#Interquartile range
#formula: IQR = Q3 - Q1
#in order to get the quartile1 and quartile 3 of a sample we can use a numpy comman
q1, q3 = np.percentile(salaries, [25,75]) #here the percentile command of numpy is
#percentile 25 is equal to quartile 1 and percentile 75 is equal to quartile 3
# the values is then stored to be used in the calculation
IQR = q3 - q1
print("The interquartile range is: ", IQR)
```

```
The interquartile range is:  413250.0
```

In [13]:
```
#Quartile coefficient of dispersion
#formula: QCD = (Q3 - Q1) / (Q3 + Q1)
q1, q3 = np.percentile(salaries, [25,75]) #this part is like the one above
#percentile 25 is equal to quartile 1 and percentile 75 is equal to quartile 3
# the values is then stored to be used in the calculation
QCD = (q3 - q1) / (q3 + q1)
print("The quartile coefficient of dispersion is: ",QCD)
```

```
The quartile coefficient of dispersion is:  0.338660110633067
```

Exercise 3: Pandas for Data Analysis

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

Perform the following tasks in the diabetes dataframe:

1. Identify the column names
2. Identify the data types of the data

3. Display the total number of records
4. Display the first 20 records
5. Display the last 20 records
6. Change the Outcome column to Diagnosis
7. Create a new column Classification that display "Diabetes" if the value of outcome is 1 , otherwise "No Diabetes"
8. Create a new dataframe "withDiabetes" that gathers data with diabetes
9. Create a new dataframe "noDiabetes" thats gathers data with no diabetes
10. Create a new dataframe "Pedia" that gathers data with age 0 to 19
11. Create a new dataframe "Adult" that gathers data with age greater than 19
12. Use numpy to get the average age and glucose value.
13. Use numpy to get the median age and glucose value.
14. Use numpy to get the middle values of glucose and age.
15. Use numpy to get the standard deviation of the skinthickness.

In [14]:
```
import pandas as pd # used for data manipulation
import numpy as np # used for computing
diabetes = pd.read_csv('diabetes.csv') # a pandas command to read a csv
diabetes.head(1)
```

Out[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	(

In [15]:
```
#1. Identify the column names
diabetes.columns #the columns command is used to output the column names of the dat
```

Out[15]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [16]:
```
# 2. Identify the data types of the data
diabetes.dtypes
#the dtypes command is used to output the data types of each column of the datafram
```

Out[16]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

In [17]:
```
# 3. Display the total number of records
diabetes.shape[0]
```

```
#shape returns the attributes of the dataframe into a tuple in the format (rows,col
#the [0] is used to access only the first one which is the number of rows
```

Out[17]: 768

In [18]:
```
# 4. Display the first 20 records
diabetes.head(20)
#the head() function is used to access the first rows of a dataframe
#it is in 5 rows by default, but you can put a number to access certain numbers of
# in this case head(20) shows the first 20 in the dataframe
```

Out[18]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFur
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	

In [19]:
```
# 5. Display the last 20 records
diabetes.tail(20)
#the tail() function is the same as the head() but accesses the last rows
# it is also in 5 rows by default, and can put a number to access certain numbers o
# in this case tail(20) shows the last 20 in the dataframe
```

Out[19]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
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	

In [20]:

```
# 6. Change the Outcome column to Diagnosis
diabetes = diabetes.rename(columns = {'Outcome':'Diagnosis'})
#.rename is used to rename in a dataframe, and what to rename, in this case
#a column is to be renamed, and the {'Outcome':'Diagnosis'} indicates what to renam
diabetes.head() # this accesses the dataframe to see if the change hase been made
```

Out[20]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
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	2

In [21]:

```
# 7. Create a new column Classification that display "Diabetes"
#     if the value of outcome is 1 , otherwise "No Diabetes"
diabetes['Classification'] = diabetes['Diagnosis'].apply(lambda x: 'Diabetes' if x
# to create a new columns you used the command (dataframe name)['(column name)']
#the apply method is used to apply a function in the rows or columns of a dataFrame
# the function used is the "lambda x: 'Diabetes' if x == 1 else 'No Diabetes'"
# to check specific entries of a column and what entry should be made
diabetes.head()
```

Out[21]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
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	2

In [22]:

```
# 8. Create a new dataframe "withDiabetes" that gathers data with diabetes
withDiabetes = diabetes[diabetes['Diagnosis'] == 1]
# this gets the entries where it is 1 in the column Diagnosis or with diabetes
withDiabetes.head()
```

Out[22]:

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

In [23]:

```
# 9. Create a new dataframe "noDiabetes" thats gathers data with no diabetes
noDiabetes = diabetes[diabetes['Diagnosis'] == 0]
```

```
#this works the same as the one above it but takes those with a 0 values in the Dia
noDiabetes.head()
```

Out[23]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFur
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	

In [24]:

```
# 10. Create a new dataframe "Pedia" that gathers data with age 0 to 19
Pedia = diabetes[diabetes['Age'] <=19]
#this uses a function "<=19" to get only those with the value of 19 below in the Ag
Pedia.head()
```

Out[24]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct

In [25]:

```
# 11. Create a new dataframe "Adult" that gathers data with age greater than 19
Adult = diabetes[diabetes['Age'] > 19]
#this uses a function "> 19" to get only those with the value of above 19 in the Ag
Adult.head()
```

Out[25]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
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	2

In [26]:

```
# 12. Use numpy to get the average age and glucose value.
ave_age = np.mean(diabetes['Age'])
ave_gluc = np.mean(diabetes['Glucose'])
# numpy has a command to get the mean of a sample
# in this case it is used in a filtered sample
#which is the diabetes['Age'] and diabetes['Glucose']
# which only takes those columns separately
# the mean function is then used to get the mean of those columns
print("The average age is: ",ave_age)
print("The average glucose value is: ",ave_gluc)
```

```
The average age is:  33.240885416666664
The average glucose value is:  120.89453125
```

In [27]:

```
# 13. Use numpy to get the median age and glucose value.
median_age = np.median(diabetes['Age'])
median_gluc = np.median(diabetes['Glucose'])
# numpy has a command to get the median of a sample
# in this case it is used in a filtered sample
#which is the diabetes['Age'] and diabetes['Glucose']
# which only takes those columns separately
# the mean function is then used to get the median of those columns
print("The median of the age sample is: ", median_age)
print("The median of the glucose value sample is: ",median_gluc)
```

```
The median of the age sample is:  29.0
The median of the glucose value sample is:  117.0
```

In [28]:

```
# 14. Use numpy to get the middle values of glucose and age.
mid_age = np.percentile(diabetes['Age'],50)
mid_gluc = np.percentile(diabetes['Glucose'],50)
#numpy has a command percentile to get the percentile of a sample
#percentile 50 is equal to the middle of a sample
# the value is then stored to be used in the calculation
print("The middle value of the age sample is: ",mid_age)
print("The middle value of the glucose value sample is: ",mid_gluc)
```

```
The middle value of the age sample is:  29.0
The middle value of the glucose value sample is:  117.0
```

In [29]:

```
# 15. Use numpy to get the standard deviation of the skinthickness.
std_skinthicc = np.std(diabetes['SkinThickness'])
#numpy also has a standard deviation command known as std
#this automatically calculates the standard deviation of a sample
print("The standard deviation of the skinthickness column is:", std_skinthicc)
```

```
The standard deviation of the skinthickness column is: 15.941828626496978
```

6.4 Conclusion

In this activity I was able to learn about some ways to analyze data and how get specific data like mean, median, mode standard deviation, without using any modules, though, it is somehow hard, because I must know the formula in order to get the result I need, using modules lessens the intensity of this task, since the calculations are done by machine when specific commands are used. And I don't need to memorize it on my own. Modules like numpy and statistics help very well in this situation. For the dataframe analysis, I was able to use many commands in modifying the dataframe, and in getting specific data that was needed in this activity.

End

In [ ]: