Hands-on Activity 6.1 Introduction to Data Analysis and Tools

CPE311 Computational Thinking with Python

Name: De los Reyes, Jann Moises Nyll B. *Section: * CPE22S3

Performed on : 03/06/2024

Submitted on : 03/06/2024

Submitted to : 03/06/2024

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

```
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

```
#Write a comment per statistical function

def mean(dataset):
  # Calculate mean by the sum of all dataset
  #divided by the number of the data
  return sum(dataset)/len(dataset)

#To test
mean(salaries)
```

```
585690.0
```

```
def median(dataset):
  #Sort the dataset in increasing order
  dataset.sort()
  #Get the number of data
  length = len(dataset)
  # if the number of data is odd, Median = [(n + 1)/2] th term
  if length % 2 == 1:
    return dataset[int(length/2)]
  #If the number of data is even,
  #Median = [ (n/2) th term + (n/2 + 1) th term]/2.
  else:
    x = dataset[int((length/2)-1)]
    y =dataset[int((length/2))]
    ave= (x+y)/2
    return ave

#To test
print(median(salaries))
```

```
589000.0
```

```
import collections

def mode(dataset):
  #create a dictionary-like object that count the occurence of each unique element in the dataset.
  # the .most_common() method sort the element by frequency in desceding order and return them as a list of tuple.
  modes = collections.Counter(dataset).most_common()

  #the indexing retrieves the most common element(mode) from the sorted list
  return modes[0][0]

#To test
mode(salaries)
```

```
477000.0
```

```
def sample_variance(dataset):
    n = len(dataset)
    mean = sum(dataset) / n

    # Calculate the squared deviations
    squared_deviations = [(x - mean) ** 2 for x in dataset]

    # Sum up the squared deviations
    sum_squared_deviations = sum(squared_deviations)

    # Compute sample variance
    sample_variance = sum_squared_deviations / (n - 1)
    return sample_variance

#To test:
var = sample_variance(salaries)
#Display the sample variance  with  6 decimal places
print(f"{var:.6f}")
```

```
70664054444.444443
```

```
def sample_std(dataset):
  n = len(dataset)
  mean = sum(dataset)/n

  # Calculate the squared deviations
  squared_deviations = [(x - mean) ** 2 for x in dataset]

  # Sum up the squared deviations
  sum_squared_deviations = sum(squared_deviations)

  # Compute sample variance
  sample_std = (sum_squared_deviations / (n - 1))**0.5
  return sample_std

#To test:
std =sample_std(salaries)
#Display the sample standard deviation with  6 decimal places
print(f"{std:.6f}")
```

```
265827.113825
```

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

```
#Write a comment per statistical function

import statistics

def calc_range(dataset):
  #To check if the data set contains at least two values
  if len(dataset)< 2:
    raise ValueError("The data must contain at least two values.")

  #Compute the range by subtracting the maximum and minimum value of the dataset.
  return max(dataset)- min(dataset)

range_value = calc_range(salaries)
print(f" The range value is {range_value:.2f}")
```

```
 The range value is 995000.00
```

```
from statistics import stdev,mean

def calc_COV(dataset):
  #Compute the  coefficient variation by calculating
  # the standard deviation over the mean of the dataset
  COV = stdev(dataset)/mean(dataset)

  #Convert the COV in percentage
  P_COV = COV  * 100

  calc_COV = print(f" Coefficient of Variation: {COV}\n COV in Percentage : {P_COV:.2f}")
  return calc_COV

#To test:
calc_COV(salaries)
```

```
 Coefficient of Variation: 0.45386998894439035
 COV in Percentage : 45.39 %
```

```
from statistics import quantiles
# Computation for interquartile range
# 1st step
def calc_quartiles(dataset):

  #use the .quantiles() module in which .quantiles(data, n=4 )
  #Divide data into n continuous intervals with equal probability.
  #Returns a list of n - 1 cut points separating the intervals.
  quartiles = quantiles(dataset, n = 4)

  return quartiles

#To test:
calc_quartiles(salaries)


```

```
[400500.0, 589000.0, 822250.0]
```

```
# 2nd Step
def calc_IQR(dataset):
  #Assign a  variable  for the quartile list.
  Qlist = quantiles(dataset)

  #Compute for Interquartile Range where IQR = Q3 -Q1
  calc_IQR = Qlist[-1]- Qlist[0]
  return calc_IQR

#To test:
calc_IQR(salaries)
```

```
421750.0
```

```
def calc_QCD(dataset):

  #Assign a variable for the quartile list.
  Qlist = quantiles(dataset)

  #compute for  Quartile Coefficient of Dispersion where
  #QCD =  (Q3-Q1)/(Q3+Q1) * 100
  #use the calc_IQR for the numerator
  QCD = calc_IQR(dataset)/(Qlist[-1]+ Qlist[0])

  #Convert the QCD in percentage
  P_QCD = QCD * 100

  #Display the value of QCD
  calc_QCD = print(f"Quartile Coefficient of Dispersion: {QCD} \nQCD in Percentage: {P_QCD:.6f} %")

  return calc_QCD

calc_QCD(salaries)

```

```
Quartile Coefficient of Dispersion: 0.34491923941934166
QCD in Percentage: 34.491924 %
```

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.

```
# Indicate which item you're answering with a comment
```

```
#Uploading  diabetes.csv file and converting dataframe
filepath = '/content/diabetes.csv'
import numpy as np
import pandas as pd
data = pd.read_csv(filepath)

data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Next steps: View recommended plots

```
#converting the data as dataframe
exercise_df = pd.DataFrame(data)
exercise_df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Next steps: View recommended plots

```
#1. Identify the column names
exercise_df = pd.DataFrame(data)
exercise_df.columns
```

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

```
#2. Identify the data types of the data
exercise_df = pd.DataFrame(data)
exercise_df.dtypes
```

```
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
exercise_df = pd.DataFrame(data)
len(exercise_df)
```

```
768
```

```
#4 Display the first 20 records
exercise_df.iloc[:20]
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1
10	4	110	92	0	0	37.6	0.191	30	0
11	10	168	74	0	0	38.0	0.537	34	1
12	10	139	80	0	0	27.1	1.441	57	0
13	1	189	60	23	846	30.1	0.398	59	1
14	5	166	72	19	175	25.8	0.587	51	1
15	7	100	0	0	0	30.0	0.484	32	1
16	0	118	84	47	230	45.8	0.551	31	1
17	7	107	74	0	0	29.6	0.254	31	1
18	1	103	30	38	83	43.3	0.183	33	0
19	1	115	70	30	96	34.6	0.529	32	1

```
#5.Display the last 20 records
exercise_df.iloc[-20:]
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
748	3	187	70	22	200	36.4	0.408	36	1
749	6	162	62	0	0	24.3	0.178	50	1
750	4	136	70	0	0	31.2	1.182	22	1
751	1	121	78	39	74	39.0	0.261	28	0
752	3	108	62	24	0	26.0	0.223	25	0
753	0	181	88	44	510	43.3	0.222	26	1
754	8	154	78	32	0	32.4	0.443	45	1
755	1	128	88	39	110	36.5	1.057	37	1
756	7	137	90	41	0	32.0	0.391	39	0
757	0	123	72	0	0	36.3	0.258	52	1
758	1	106	76	0	0	37.5	0.197	26	0
759	6	190	92	0	0	35.5	0.278	66	1
760	2	88	58	26	16	28.4	0.766	22	0
761	9	170	74	31	0	44.0	0.403	43	1
762	9	89	62	0	0	22.5	0.142	33	0
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

```
#6. Change the Outcome column to Diagnosis
exercise_df.rename(columns={"Outcome":"Diagnosis"})
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Diagnosis
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

```
#7. Create a new column Classification that display "Diabetes" if the value of outcome is 1 , otherwise "No Diabetes"
exercise_df['Classification'] = exercise_df['Outcome'].apply(lambda x: 'Diabetes' if x ==1 else 'No Diabetes')
exercise_df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 10 columns

Next steps: View recommended plots

```
#8. Create a new dataframe "withDiabetes" that gathers data with diabetes
withDiabetes = exercise_df[exercise_df['Classification']=='Diabetes']
withDiabetes
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
2	8	183	64	0	0	23.3	0.672	32	1
4	0	137	40	35	168	43.1	2.288	33	1
6	3	78	50	32	88	31.0	0.248	26	1
8	2	197	70	45	543	30.5	0.158	53	1
...	...	...	...	...	...	...	...	...	...
755	1	128	88	39	110	36.5	1.057	37	1
757	0	123	72	0	0	36.3	0.258	52	1
759	6	190	92	0	0	35.5	0.278	66	1
761	9	170	74	31	0	44.0	0.403	43	1
766	1	126	60	0	0	30.1	0.349	47	1

268 rows × 10 columns

Next steps: View recommended plots

```
#9. Create a new dataframe "noDiabetes" thats gathers data with no diabetes
noDiabetes = exercise_df[exercise_df['Classification']=='No Diabetes']
noDiabetes
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	1	85	66	29	0	26.6	0.351	31	0
3	1	89	66	23	94	28.1	0.167	21	0
5	5	116	74	0	0	25.6	0.201	30	0
7	10	115	0	0	0	35.3	0.134	29	0
10	4	110	92	0	0	37.6	0.191	30	0
...	...	...	...	...	...	...	...	...	...
762	9	89	62	0	0	22.5	0.142	33	0
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
767	1	93	70	31	0	30.4	0.315	23	0

500 rows × 10 columns

Next steps: View recommended plots

```
#10. Create a new dataframe "Pedia" that gathers data with age 0 to 19
Pedia = exercise_df[exercise_df['Age'] <= 19]
Pedia
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Clas

```
#11. Create a new dataframe "Adult" that gathers data with age greater than 19
Adult = exercise_df[exercise_df['Age'] > 19]
Adult
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Next steps: View recommended plots

```
#12. Use numpy to get the average age and glucose value.
#Extraction of ages  and glucose value in dataset
ages= np.array(exercise_df['Age'])
glucose_value =np.array(exercise_df['Glucose'])

#computing the average value of age and glucose using numpy
average_age = np.average(ages)
average_glucose =np.average(glucose_value)

#Display the average value of age as a whole number and
#glucose average value into 2 decimal places
print(f"Average age:{average_age:.0f} years old")
print(f"Average Glucose value: {average_glucose:.2f}")
```

```
Average age:33 years old
Average Glucose value: 120.89
```

```
#13.Use numpy to get the median age and glucose value
# Using the extraction above we can get the median

#Compute the median of age and value using numpy
median_age = np.median(ages)
median_glucose = np.median(glucose_value)

#Display the median
print(f"Median age:{median_age} years old")
print(f"Median Glucose value: {median_glucose:.2f}")
```

```
Median age:29.0 years old
Median Glucose value: 117.00
```

```
#14.Use numpy to get the middle values of glucose and age
#Using the extraction above we can get the middle value
#middle value is the value of a given dataset that is not arranged in order.
index = int(len(ages)/2)
midage = ages[index]
midglucose = glucose_value[index]
print("Middle Age: " + str(midage) + " years old")
print("Middle Glucose: " + str(midglucose))

```

```
Middle Age: 25 years old
Middle Glucose: 125
```

```
#15.Use numpy to get the standard deviation of the skinthickness.
skinthickness = np.array(exercise_df['SkinThickness'])

#compute for the std
std_skinthickness = np.std(skinthickness)

#Display the STD with 6 decimal places
print(f"Standard Deviation of SkinThickness: {std_skinthickness:.6f}")
```

```
Standard Deviation of SkinThickness: 15.941829
```

6.4 Conclusion

In this activity, I learn how to make a import a csv file and manipulate data using numpy and pandas. I also implement some data analysis functions without using statistics module. The things I loved while doing the activity is the preparation,manipulation and analyzing the data. Learning numpy and pandas is a valuable skill for anyone who wants to work with data analysis, machine learning, or scientific computing.