Hands-on Activity 6.1 Introduction to Data Analysis and Tools

CPE311 Computational Thinking with Python

Name: **Villamor, Kurt Russel**

Section: **CPE22S3**

Performed on: **03-06-2024**

Submitted on: **03-06-2024**

Submitted to: **Engr. Roman M. Richard**

6.1 Intended Learning Outcome

- Use pandas and numpy data analysis tools.
- 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

```
def mean(arr):
  return sum(arr) / len(arr)

print(mean(salaries))
```

```
585690.0
```

- Median

```
def median(arr):
  arr.sort()
  if len(arr) % 2 == 0:
    return (arr[int(len(arr)/2)-1] + arr[int(len(arr)/2)]) / 2
  else:
    return arr[int(len(arr)/2)-1]

print(median(salaries))
```

```
589000.0
```

- **Mode**

```
def mode(arr):
  dictDigits = {}
  for i in arr:
    if i not in  dictDigits:
      dictDigits[i] = 1
    else:
      dictDigits[i] += 1
  max_rep = max(x for x in dictDigits.values())
  return [k for k, v in dictDigits.items() if v == max_rep]

print(mode(salaries))
```

```
[477000.0]
```

- **Sample Varience**

```
def sVarience(arr):
  tabs = []
  x_bar = mean(arr)
  for x in arr:
    tabs.append((x - x_bar) ** 2)
  return sum(tabs) / (len(arr) - 1)

print(sVarience(salaries))
```

```
70664054444.44444
```

- **Standard Deviation**

```
def sDeviation(arr):
  return sVarience(arr) ** 0.5

print(sDeviation(salaries))
```

```
265827.11382484
```

Exercise 2

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

- **Range**

```
def sRange(arr):
  return max(arr) - min(arr)

print(sRange(salaries))
```

```
995000.0
```

- **Coefficient of variation Interquartile range**

```
from statistics import quantiles
```

```
def vIQR(arr): # getting the Interquartile Range (using statisatics module)
  arr.sort()
  qrt = quantiles(arr)
  return qrt[2] - qrt[1]

print(vIQR(salaries))
```

```
233250.0
```

- **Quartile coefficient of dispersion**

```
def qCoeficientDisp(arr): # getting the Quartile Of Dispersion (usiung statisatics module)
  arr.sort()
  qrt = quantiles(arr)
  return (qrt[2] - qrt[1]) / (qrt[2] + qrt[1])

print(qCoeficientDisp(salaries))
```

```
0.16527900797165634
```

Exercise 3: Pandas for Data Analysis

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

```
from google.colab import drive
drive.mount('/content/drive') # getting the file from my Drive
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
```

```
import pandas as pd
import numpy as np

file_path = "/content/drive/MyDrive/Datasets/diabetes.csv"
data = pd.read_csv(file_path) # reading the Csv file before
diabetes_df = pd.DataFrame(data) # using data to creation of the DataFrame
```

```
diabetes_df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
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	
...	...	...	...	...	...	...	
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	

768 rows × 9 columns

Next steps: View recommended plots

Perform the following tasks in the diabetes dataframe:

1. Identify the column names

```
1 diabetes_df.columns # Printing the columns in the Diabetes Data Frame
```

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

2. Identify the data types of the data

```
1 diabetes_df.dtypes # Printing Data Types
```

```
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

```
1 len(diabetes_df) # Printing number of records
```

```
768
```

4. Display the first 20 records

```
1 diabetes_df.head(20) # The First 20 using .head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
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	

Next steps: View recommended plots

5. Display the last 20 records

```
1 diabetes_df.tail(20) # the last 20 using .tail()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
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

```
1 diabetes_df.rename(columns = {"Outcome":"Diabetes"}, inplace = True) # renaming the Outcome column to become Diabetes using .rename() and implementing a dictionary to it.
```

```
1 diabetes_df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
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	
...	...	...	...	...	...	...	
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	

768 rows × 9 columns

Next steps: View recommended plots

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

```
1 diabetes_df["Classification"] = (diabetes_df.Diabetes.apply(lambda x: {1:"Diabetes", 0:"No Diabetes"}[x])) # implementing a lambda that returns the string results depending to the \
```

```
1 diabetes_df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
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	
...	...	...	...	...	...	...	
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	

768 rows × 10 columns

Next steps: View recommended plots

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

```
withDiabetes = pd.DataFrame(diabetes_df[diabetes_df["Diabetes"] == 1]) # Review your DBMS WHERE Command, it works like that
```

```
withDiabetes
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
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	
...	...	...	...	...	...	...	
755	1	128	88	39	110	36.5	
757	0	123	72	0	0	36.3	
759	6	190	92	0	0	35.5	
761	9	170	74	31	0	44.0	
766	1	126	60	0	0	30.1	

268 rows × 10 columns

Next steps: View recommended plots

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

```
noDiabetes = pd.DataFrame(diabetes_df[diabetes_df["Diabetes"] == 0]) # DMS where command!
```

```
noDiabetes
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
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	
...	...	...	...	...	...	...	
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	
767	1	93	70	31	0	30.4	

500 rows × 10 columns

Next steps: View recommended plots

10. Create a new dataframe "Pedia" that gathers data with age 0 to 19

```
Pedia = pd.DataFrame(diabetes_df[(diabetes_df["Age"] <= 19) & (diabetes_df["Age"] > 0)]) # also this, but this part refrain for using pythonic operands ("and", "or")
```

```
Pedia
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre

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

```
Adult = pd.DataFrame(diabetes_df[diabetes_df["Age"] > 19]) # anada one WHERE
```

```
Adult
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
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	
763	10	101	76	48	180	32.9	
766	1	126	60	0	0	30.1	

Next View recommended plots

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

```
1 np.mean(diabetes_df["Age"]) # using Numpy for this Mean Self explanatory.
```

```
33.2408854166666664
```

```
1 np.mean(diabetes_df["Glucose"])
```

```
120.89453125
```

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

```
1 np.median(diabetes_df["Age"]) # same also in the Median
```

```
29.0
```

```
1 np.median(diabetes_df["Glucose"])
```

```
117.0
```

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

```
1 np.median(diabetes_df["Age"]) # the middle of a data set is also median ...
```

```
29.0
```

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

```
1 diabetes_df['SkinThickness'].std() # standard Deviation using Pandas
```

```
15.952217567727637
```

```
1 np.std(diabetes_df["SkinThickness"],axis=0) # Standard Deviation using the Numpy (almost the same)
```

```
15.941828626496939
```

6.4 Conclusion

- This Activity opens the opportunity to learn the infamous Libraries for handling datasets, The Numpy and Pandas. Based in what I learned that Pandas run at top of numpy so there are many similarities in both parts. But in this Activity, we used pandas to create data frames of the data sets and Numpy for solving statistical functions such as mean, median and standard deviation.