Hands-on Activity 6.1 Introduction to Data Analysis and Tools

CPE311 Computational Thinking with Python

Name: Dejoras, Dylan James N.
Section: CPE22S3

Performed on: 03/07/2024
Submitted on: 03/07/2024
Submitted to: Engr. Roman M. Richard

6.1 Intended Learning Outcomes:

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

Instructions:

1. Download the given file: diabetes.csv
2. Create your own Jupyter Notebook and accomplish all the items in the pdf file: Hands-on Activity 6.1 Introduction to Data Analysis and Tools.pdf Download Hands-on Activity 6.1 Introduction to Data Analysis and Tools.pdf
3. Submit the .pdf file of the accomplished activity.

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

Mean

```
# the formula of mean is the sum of the given divided by the total
def calculate_mean(data):
  mean = sum(data) / len(data)
  return mean
```

```
calculate_mean(salaries)
```

```
585690.0
```

```
from statistics import mean
mean(salaries)
```

```
585690.0
```

Median

```
# we need to sort the list first from lowest to highest
sorted_salaries = sorted(salaries)
```

```
# if length of list is even, get the two values in the middle then divide them by two
def calculate_median(data):
  if len(data) % 2 == 0:
    l_median = data[len(data) // 2-1]
    r_median = data[len(data) // 2]
    median = (l_median + r_median) / 2
  # if length of list is odd, simmply return the value in the middle
  else:
    median = data[len(data // 2)]
  return median
```

```
calculate_median(sorted_salaries)
```

```
589000.0
```

```
from statistics import median
median(salaries)
```

```
589000.0
```

Mode

```
from collections import Counter

# get all modes and their frequencies using Counter
modes = (Counter(salaries).most_common())

mode_count = max((Counter(salaries).most_common()))
def calculate_mode(data):
  # check if there is more than one mode
  if len(modes) > 1 and modes[0][1] == modes[1][1]:
      # get modes with same count
      all_modes = [mode[0] for mode in modes if mode[1] == modes[0][1]]
      return "There are multiple modes:" , all_modes, "with a count of", mode_count
  else:
      # since in this case there is only one mode, use modes[0] and set the variable derived by .most_common()
      # to mode_value and mode_count
      mode_value, mode_count = modes[0]
      return "The mode is: ", mode_value, "with a count of", mode_count
```

```
calculate_mode(salaries)
```

```
('The mode is: ', 477000.0, 'with a count of', 3)
```

```
from statistics import mode
mode(salaries)
```

```
477000.0
```

Sample Variance

```
# formula for sample variance is total of (value of all nth elements - mean)^2 / size of the given - 1
sumz = []
def calculate_samplevariance(data):
  for n in data: # using for loop for traversing through the data
    sumz.append((n - calculate_mean(data))**2) # code for total of (value of all nth elements - mean)^2
  totalz = sum(sumz)
  s_variance = totalz / (len(data) - 1) #implementation of the formula
  return s_variance
```

```
calculate_samplevariance(salaries)
```

```
70664054444.44444
```

```
from statistics import variance
variance(salaries)
```

```
70664054444.44444
```

Sample Standard Deviation

```
# the formula for standard deviation is basically the square root of the variance
# the square root could also be done by using 1/2 as an exponent
def calculate_standardDeviation(data):
  s_standardDeviation = (calculate_samplevariance(data) ** (1/2))
  return s_standardDeviation
```

```
calculate_standardDeviation(salaries)
```

```
265827.11382484
```

```
from statistics import stdev
stdev(salaries)
```

```
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

Range

```
def calculate_range(data):
  range = sorted_salaries[-1] - sorted_salaries[0] #reuse sorted list then subtract maximum value to minimum value
  return range
```

```
calculate_range(salaries)
```

```
995000.0
```

Coefficient of variation

```
from statistics import stdev, mean
def calculate_Cov(data):
  Cov = stdev(salaries) / mean(salaries) # the formula for coefficient of variation is standard deviation / mean
  P_Cov= round(Cov * 100, 2) # multiply the result by 100 to get percentage
  output = print("Coefficient of variation: ", Cov, "\nCoefficient of variation as percentage: " , P_Cov) # implement a displayable result
  return output
```

```
calculate_Cov(salaries)
```

```
Coefficient of variation:  0.45386998894439035
Coefficient of variation as percentage:  45.39
```

Interquartile range

```
from statistics import quantiles
def calculate_quartiles(data):
  quartiles = quantiles(data, n=4) # .quantiles(list, and n = 4 to get the quartiles, n=10 for deciles)
  return quartiles
```

```
calculate_quartiles(salaries)
```

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

```
def calculate_IQR(data):
  quartile_list = calculate_quartiles(salaries) # assign a variable to the list earlier
  IQR = quartile_list[-1] - quartile_list [0] # formula for IQR is Q3 - Q1
  return IQR
```

```
calculate_IQR(salaries)
```

```
421750.0
```

Quartile coefficient of dispersion

```
def calculate_QCoD(data):
  quartile_list = calculate_quartiles(salaries) # assign a variable to the list earlier
  QCoD = calculate_IQR(salaries) / (quartile_list[-1] + quartile_list[0]) # formula for QCoD is Q3 - Q1 / Q3 + Q1
  P_QCoD = round(QCoD * 100, 2) # multiply the result by 100 to get percentage
  output = print("Quartile coefficient of dispersion: ", QCoD, "\nQuartile coefficient of dispersion as percentage: " , P_QCoD) # implement
  return output
```

```
calculate_QCoD(salaries)
```

```
Quartile coefficient of dispersion:  0.34491923941934166
Quartile coefficient of dispersion as percentage:  34.49
```

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.

```
filepath = '/content/diabetes.csv'
```

```
import pandas as pd
import numpy as np
```

```
data = pd.read_csv(filepath)
data
```

	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	
...	...	...	...	...	...	...	
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

```
# 1. Identify the column names as well
for col in data.columns:
  print(col)
```

```
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
```

```
# 2. Identify the data types of the data
data.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
len(data)
```

```
768
```

```
# 4. Display the first 20 records
data.head(20)
```

	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	
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
data.tail(20)
```

	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
data.rename(columns={'Outcome': 'Diagnosis'}, inplace=True) #.rename() for changing
```

```
data
```

	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	
...	...	...	...	...	...	...	
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

```
# 7. Create a new column Classification that display "Diabetes" if the value of outcome is 1 , otherwise "No Diabetes"
data['Classification'] = np.where(data['Diagnosis'] == 1, 'Diabetes', 'No Diabetes') # np.where() for selecting certain values
                                                                                      # in a certain column
```

```
data
```

	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	
...	...	...	...	...	...	...	
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

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

column = 'Diagnosis'
value = [1]

checker = data[column].isin(value) # .isin() is also a function for checking if a column
                                   # has a certain value

withDiabetes = data[checker]

withDiabetes
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
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

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

column = 'Diagnosis'
value = [0]

checker = data[column].isin(value) # .isin() is also a function for checking if a column
                                   # has a certain value

noDiabetes = data[checker]

noDiabetes
```

	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	
...	...	...	...	...	...	...	
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

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

column = 'Age'
min_value = 0
max_value = 19

checker = data[column].between(min_value, max_value) # .between() function for checking between two certain integers

Pedia = data[checker]
# there is no one under the age of 20 so it will return an empty dataframe
Pedia
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc

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

column = 'Age'

condition = data[column] > 20 # condition for ages above 20

Adult = data[condition]

Adult
```

	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	
...	...	...	...	...	...	...	
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

```
# 12. Use numpy to get the average age and glucose value.
print("Average age:", np.mean(data['Age'])) # np.mean() for average
print("Average glucose value:", np.mean(data['Glucose']))
```

```
Average age: 33.240885416666664
Average glucose value: 120.89453125
```

```
# 13. Use numpy to get the median age and glucose value.
print("Median of age:", np.median(data['Age'])) # np.median() for median
print("Median of glucose value:", np.median(data['Glucose']))
```

```
Median of age: 29.0
Median of glucose value: 117.0
```

```
# 14. Use numpy to get the middle values of glucose and age.
print("Middle value of glucose:", np.median(data['Glucose'])) # np.median() for median
print("Middle value of age:", np.median(data['Age']))
```

```
Middle value of glucose: 117.0
Middle value of age: 29.0
```

```
# 15 Use numpy to get the standard deviation of the skinthickness.
print("Standard deviation of skin thickness:", np.std(data['SkinThickness'])) # np.std() for standard deviation
```

```
Standard deviation of skin thickness: 15.941828626496939
```

6.4 Conclusion

This Hands-on Activity introduced us to commonly used Python modules that are considered data analysis tools. These modules are numpy and pandas. "NumPy" stands for Numerical Python which serves as a numerical computing library in Python. It provides a wide range of mathematical functions to provide convenience and save time. As for "Pandas", it is a data manipulation and analysis library for Python. It efficiently stores and manipulates large datasets. This activity demonstrated how to analyze data using numpy and pandas. This is a good introduction to data analysis and taking advantage of existing tools. One could say that after doing this activity, it is understandable why numpy and pandas are used together for an all-out and finesse data analysis.