Hands-on Activity 6.1 Introduction to Data Analysis and Tools

CPE311 Computational Thinking with Python

Name: Alorro, Jay-ann S.

Section: CPE22S3

Performed on: 03/07/2024

Submitted on: 03/07/2024

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 [4]:
```
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 (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 (https://docs.python.org/3/library/collections.html#collections.Counter))
- Sample varience
- Sample standard deviation

In [17]:
```
# mean
def mean(salaries):
  return sum(salaries) / len(salaries) # the average/mean of the salaries data
```

In [18]:
```
mean(salaries)
```

Out[18]: 585690.0

In [31]:
```
# median
def median(salaries):
    # sort salaries in ascending order
    salaries_result = sorted(salaries)

    # when odd, return the middle item of the list
    if len(salaries) % 2 != 0:
        mid_index = (len(salaries) - 1) // 2
        return salaries_result[mid_index]

    # when even, return the average of the two middle items of the list
    else:
        mid_index_1 = len(salaries) // 2
        mid_index_2 = mid_index_1 - 1
        median_value = (salaries_result[mid_index_1] + salaries_result[mid_ind
        return median_value
```

In [32]:
```
median(salaries)
```

Out[32]: 589000.0

In [41]:
```
# mode
def mode(salaries):
    # counter of occurrences
    counts = {}
    for salary in salaries:
        counts[salary] = counts.get(salary, 0) + 1

    # finding the max value in count
    max_count = max(counts.values())

    # find all items with max count
    modes = [salary for salary, count in counts.items() if count == max_count]

    if len(modes) == len(set(salaries)):
        return "No mode"  # no mode when all items occurred only once
    else:
        return modes  # return list of all possible modes
```

In [20]:
```
mode(salaries)
```

Out[20]:
```
[477000.0]
```

In [36]:
```
# sample varience
def sample_variance(salaries):
  # finding the mean with the function above
  first = mean(salaries)
  # subtracting each item with the mean and squaring it
  second = 0
  for i in salaries:
    second += (i-first)**2
  # main formula and answer
  return second/(len(salaries) - 1)
```

In [37]:
```
sample_variance(salaries)
```

Out[37]:
```
70664054444.44444
```

In [25]:
```
# sample standard deviation
def sample_sd(salaries):
  # finding the mean with the function above
  first = mean(salaries)
  # subtracting each item with the mean and squaring it
  second = 0
  for i in salaries:
    second += (i-first)**2
  # main formula and answer
  return (second/(len(salaries) - 1)) ** 0.5
```

In [26]:
```
sample_sd(salaries)
```

Out[26]:
```
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 [97]:
```
import statistics as stat
```

In [95]:
```
# range
def range_stat(salaries):
  return max(salaries) - min(salaries) # range formula
```

In [96]:
```
range_stat(salaries)
```

Out[96]:
```
995000.0
```

In [98]:
```
# coefficient of variation Interquartile range
def coefficent_of_variation(salaries):
  mean_salary = stat.mean(salaries) # getting the mean
  std_deviation = stat.stdev(salaries) # getting the standard deviation

  return (std_deviation/mean_salary) * 100 # results base on the formula
```

In [100]:
```
coefficent_of_variation(salaries)
```

Out[100]:
```
45.38699889443903
```

In [109]:
```
import numpy as np # numpy is used instead of the statistics module for the pe

# Quartile coefficient of dispersion
def quartile_of_dispersion(salaries):
  q1 = np.percentile(salaries, 25)
  q3 = np.percentile(salaries, 75)

  return (q3 - q1) / (q3 + q1)
```

In [110]:
```
quartile_of_dispersion(salaries)
```

Out[110]:
```
0.338660110633067
```

Exercise 3: Pandas for Data Analysis

Load the diabetes.csv file. COnvert the diabetes.csv file 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 displays "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" that 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 age greater than 19
12. Use numpy to get the average age and glucose value
13. Use numply 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 [42]:
```
filepath = '/content/diabetes.csv' # upload of file
```

In [44]:
```
import pandas as pd

df = pd.read_csv(filepath)
```

In [46]:
```
df
```

Out[46]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

768 rows × 9 columns

In [48]:
```
# 1. Identify the column names
df.columns
```

Out[48]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [51]:
```
# 2. Identify the data types of the data
df.dtypes
```

Out[51]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [52]:
```
# 3. Display the total number of records
len(df)
```

Out[52]:
```
768
```

In [53]:
```
# 4. Display the first 20 records
df[:20]
```

Out[53]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28
5	5	116	74	0	0	25.6	0.20
6	3	78	50	32	88	31.0	0.24
7	10	115	0	0	0	35.3	0.13
8	2	197	70	45	543	30.5	0.15
9	8	125	96	0	0	0.0	0.23
10	4	110	92	0	0	37.6	0.19
11	10	168	74	0	0	38.0	0.53
12	10	139	80	0	0	27.1	1.44
13	1	189	60	23	846	30.1	0.39
14	5	166	72	19	175	25.8	0.58
15	7	100	0	0	0	30.0	0.48
16	0	118	84	47	230	45.8	0.55
17	7	107	74	0	0	29.6	0.25
18	1	103	30	38	83	43.3	0.18
19	1	115	70	30	96	34.6	0.52

In [54]:
```
# 5. Display the last 20 records
df[-20:]
```

Out[54]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
748	3	187	70	22	200	36.4	0.4
749	6	162	62	0	0	24.3	0.1
750	4	136	70	0	0	31.2	1.1
751	1	121	78	39	74	39.0	0.2
752	3	108	62	24	0	26.0	0.2
753	0	181	88	44	510	43.3	0.2
754	8	154	78	32	0	32.4	0.4
755	1	128	88	39	110	36.5	1.0
756	7	137	90	41	0	32.0	0.3
757	0	123	72	0	0	36.3	0.2
758	1	106	76	0	0	37.5	0.1
759	6	190	92	0	0	35.5	0.2
760	2	88	58	26	16	28.4	0.7
761	9	170	74	31	0	44.0	0.4
762	9	89	62	0	0	22.5	0.1
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

In [56]:
```
# 6. Change the Outcome column to Diagnosis
df.rename(columns={'Outcome':'Diagnosis'}, inplace=True)
```

In [57]:
```
df
```

Out[57]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

768 rows × 9 columns

In [59]:
```
# 7. Create a new column Classification that displays "Diabetes" if the value
import numpy as np

df['Classification'] = np.where(df['Diagnosis'] == 1, 'Diabetes', 'No Diabetes
```

In [60]:
```
df
```

Out[60]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

768 rows × 10 columns

In [67]:
```
# 8. Create a new dataframe "withDiabetes" that gathers data with diabetes
withDiabetes = df[df['Classification'] == 'Diabetes'].copy() # using condition
```

In [68]:
```
withDiabetes
```

Out[68]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.6
2	8	183	64	0	0	23.3	0.6
4	0	137	40	35	168	43.1	2.2
6	3	78	50	32	88	31.0	0.2
8	2	197	70	45	543	30.5	0.1
...	...	...	...	...	...	...	
755	1	128	88	39	110	36.5	1.0
757	0	123	72	0	0	36.3	0.2
759	6	190	92	0	0	35.5	0.2
761	9	170	74	31	0	44.0	0.4
766	1	126	60	0	0	30.1	0.3

268 rows × 10 columns

In [69]:
```
# 9. Create a new dataframe "noDiabetes" that gathers data with no diabetes
noDiabetes = df[df['Classification'] == 'No Diabetes'].copy() # using conditio
```

In [70]:
```
noDiabetes
```

Out[70]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
1	1	85	66	29	0	26.6	0.3
3	1	89	66	23	94	28.1	0.1
5	5	116	74	0	0	25.6	0.2
7	10	115	0	0	0	35.3	0.1
10	4	110	92	0	0	37.6	0.1
...	...	...	...	...	...	...	
762	9	89	62	0	0	22.5	0.1
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
767	1	93	70	31	0	30.4	0.3

500 rows × 10 columns

In [71]:
```
# 10. Create a new dataframe "Pedia" that gathers data with age 0 to 19
Pedia = df[df['Age'] <= 19].copy() # conditions on age
```

In [72]:
```
Pedia
```

Out[72]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction

In [73]:
```
# 11. Create a new dataframe "Adult" that gathers data age greater than 19
Adult = df[df['Age'] > 19].copy() # conditions on age
```

In [74]:
```
Adult
```

Out[74]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.6
1	1	85	66	29	0	26.6	0.3
2	8	183	64	0	0	23.3	0.6
3	1	89	66	23	94	28.1	0.1
4	0	137	40	35	168	43.1	2.2
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.1
764	2	122	70	27	0	36.8	0.3
765	5	121	72	23	112	26.2	0.2
766	1	126	60	0	0	30.1	0.3
767	1	93	70	31	0	30.4	0.3

768 rows × 10 columns

In [82]:
```
# 12. Use numpy to get the average age and glucose value

# utilizing the mean function in numpy
age_mean = np.mean(df['Age'])
glucose_mean = np.mean(df['Glucose'])

print('Age:', age_mean)
print('Glucose:', glucose_mean)
```

```
Age: 33.240885416666664
Glucose: 120.89453125
```

In [87]:
```
# 13. Use numply to get the median age and glucose value

# using the median function in numpy
median_age = np.median(df['Age'])
median_glucose = np.median(df['Glucose'])

print("Median Age:", median_age)
print("Median Glucose:", median_glucose)
```

```
Median Age: 29.0
Median Glucose: 117.0
```

In [89]:
```
# 14. Use numpy to get the middle values of glucose and age

# the percentile function is used to find the middle value which is the same t
middle_age = np.percentile(df['Age'], 50)
middle_glucose = np.percentile(df['Glucose'], 50)

print("Middle Value of Age:", middle_age)
print("Middle Value of Glucose:", middle_glucose)
```

```
Middle Value of Age: 29.0
Middle Value of Glucose: 117.0
```

In [91]:
```
# 15. Use numpy to get the standard deviation of the skinthickness

# standard deviation function of numpy is used
std_skin_thickness = np.std(df['SkinThickness'])

print("Standard Deviation of SkinThickness:", std_skin_thickness)
```

```
Standard Deviation of SkinThickness: 15.941828626496939
```

6.4 Conclusion

This activity gave me a broader understanding on how to use Pandas, Numpy, and even the module statistics in data sets. The first part was the trickiest but it was simple enough as long as there is a formula provided. Exercise 2 was quite difficult for me because I'm not familiat with the statistics module so I had to browse the documentation and research on how I can apply it in the 3 different functions created. I had the most fun in exercise 3 where we got to manipulate data and meet certain conditions. It introduced me to creating new columns and dataframes that has conditions in it. Overall, this was a fun activity and I have learned a lot of new things while working on it.