Module 6: Introduction to Data Analysis Tools in Python

PSMDSRC103 – Programming Review
Submitted by: BASAL, RAFFY
Performed on: October 8, 2024
Submitted on: October 8, 2024

Submitted to: Engr. Roman M. Richard

Assignment 6.1 Introduction to Data Analysis

6.2 Supplementary Activity

Run the given code and generate a list of 100 values.

```
import random
random.seed(0)
salaries = [round(random.random()*1000000, -3) for _ in range(100)]
```

Exercise

Using the generated data, calculate the following statistics without importing anything from the**statistics** module in the standard library (and without using pandas/numpy).

Once you have computed for each, compare your obtained output with the output when using the **statistics** module.

WITHOUT STATISTICS MODULE

MEAN

```
def cal_mean(data):
    return sum(data) / len(data)

print(f"Mean", cal_mean(salaries))
```

✓ 0.0s

```
Mean 585690.0
```

MEDIAN

```
def cal_median(data):
    sort_data = sorted(data)
    n = len(sort_data)
    mid = n // 2

    if n % 2 == 0:
        return (sort_data[mid - 1] + sort_data[mid]) / 2
    else:
        return sort_data[mid]

print(f"Median:", cal_median(salaries))
```

✓ 0.0s

```
Median: 589000.0
```

MODE

```
def mode(data):
    frequency = {}
    for item in data:
        frequency[item] = frequency.get(item, 0) + 1

    mode = max(frequency, key=frequency.get)
    return mode

print(f"Mode:", mode(salaries))
```

✓ 0.0s

```
Mode: 477000.0
```

SAMPLE VARIANCE

```
def variance(data):
    n = len(data)
    squared_diffs = [(x - cal_mean(data)) ** 2 for x in data]
    variance = sum(squared_diffs) / (n - 1)  # Step 3: Divide b
    return variance

print(f"Sample Variance:", variance(salaries))
```

✓ 0.0s

```
Sample Variance: 70664054444.44444
```

STANDARD DEVIATION

```
def standard_deviation(data):
    var = variance(data)  # Step 1: Find the variance
    std_deviation = var ** 0.5  # Step 2: Take the square root of th
    return std_deviation

print(f"Standard Deviation:", standard_deviation(salaries))
```

```
[13] ✓ 0.0s
Standard Deviation: 265827.11382484
```

IMPORTED STATICTICS

```
import statistics

print(f"Mean:", statistics.mean(salaries))
print(f"Median:", statistics.median(salaries))
print(f"Mode:", statistics.mode(salaries))
print(f"Sample Variance:", statistics.variance(salaries))
print(f"Standart Deviation:", statistics.stdev(salaries))
```

```
[20] ✓ 0.0s
Mean: 585690.0
Median: 589000.0
Mode: 477000.0
Sample Variance: 70664054444.44444
Standart Deviation: 265827.11382484
```

It shows same result and its much faster when importing STATISTICS, there is no need to manually code the formula for each method. Importing libraries like this can lessen the time of coding for the programmers, but this will be an added knowledge for a beginners like me.

EXERCISE 2

Using the previously generated data, calculate the following statistics using the functions in the **statistics** module where appropriate.

RANGE

```
def range(data):
    return max(data) - min(data)

print(f"Range:",range(salaries))
```

```
2] ✓ 0.0s
Range: 995000.0
```

COEFFECIENT OF VARIATION

```
def coefficient_of_variation(data):
    mean = cal_mean(data)
    stdev = standard_deviation(data)
    return (stdev / mean) * 100

print(f"Coefficient of Variation:", coefficient_of_variation(salaries))
```

```
✓ 0.0s
Coefficient of Variation: 45.38699889443903
```

INTERQUARTILE RANGE

```
def interquartile_range(data):
    sort_data = sorted(data)
    q1 = cal_median(sort_data[:len(sort_data) // 2])  # Lower quartile (Q1)
    q3 = cal_median(sort_data[len(sort_data) // 2 + (1 if len(sort_data) % 2 == 1 else 0):])
    return q3 - q1
print(f"Interquartile Range:", interquartile_range(salaries))
```

```
✓ 0.0s
Interquartile Range: 417500.0
```

QUARTILE COEFFECIENT OF DISPERSION

```
def quartile_coefficient_of_dispersion(data):
    sorted_data = sorted(data)
    q1 = cal_median(sorted_data[:len(sorted_data) // 2])  # Lower quartile (Q1)
    q3 = cal_median(sorted_data[len(sorted_data) // 2 + (1 if len(sorted_data) % 2 == 1 else 0):])
    return (q3 - q1) / (q3 + q1)

print(f"Quartile Coefficient of Dispersion:", quartile_coefficient_of_dispersion(salaries))
```

```
✓ 0.0s
Quartile Coefficient of Dispersion: 0.3417928776094965
```

EXERCISE 3

1. Identify Column names
2. Identify the Data types of Data
3. Display the total number of records

```
# Identify Column names
print(f"Column Names:", data.columns)
print()

# Identify the Data types of Data
print(f"Data Types:", data.dtypes)
print()

# Display the total number of records
print(f"Total Number of Records:", len(data))
print(f"Rows & Colums:", data.shape)
```

[5] ✓ 0.0s

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

Data Types: Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object

Total Number of Records: 768
Rows & Colums: (768, 9)
```

4. Display the First 20 records

```
# Display the First 20 records
print(f"First 20 Records:", data.head(20))
```

✓ 0.0s

```
First 20 Records:     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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

    DiabetesPedigreeFunction  Age  Outcome
0                      0.627   50        1
1                      0.351   31        0
...
16                     0.551   31        1
17                     0.254   31        1
18                     0.183   33        0
19                     0.529   32        1
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

5. Display the Last 20 Records

```
# Display the Last 20 Records
print(f"Last 20 Records:", data.tail(20))
```

✓ 0.0s

```
Last 20 Records:      Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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

     DiabetesPedigreeFunction  Age  Outcome
748                     0.408   36        1
749                     0.178   50        1
...
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

6. Change the Outcome column to Diagnosis

```
# Change the Outcome column to Diagnosis
data.rename(columns={'Outcome':'Diagnosis'}, inplace = True)
print(data.head())
```

[8] ✓ 0.0s

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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

   DiabetesPedigreeFunction  Age  Diagnosis
0                     0.627   50          1
1                     0.351   31          0
2                     0.672   32          1
3                     0.167   21          0
4                     2.288   33          1
```

7. Create new columns classification that display "Diabetes" if the value of diagnosis is 1, otherwise "No Diabetes".

```
# Create new columns classification that display "Diabetes" if the value of diagnosis is 1, otherwise "No Diabetes".

data['Classification'] = data['Diagnosis'].apply(lambda x: 'Diabetes' if x == 1 else 'No Diabetes')
# .apply () function applies lamba function to each value to check values of if x == 1, otherwise the else condition

print(data.head())

[10] ✓ 0.0s
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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

   DiabetesPedigreeFunction  Age  Diagnosis Classification
0                     0.627   50          1       Diabetes
1                     0.351   31          0    No Diabetes
2                     0.672   32          1       Diabetes
3                     0.167   21          0    No Diabetes
4                     2.288   33          1       Diabetes
```

8. Create new dataFrame "withDiabetes" that gathers data with diabetes.

```
# Create new dataFrame "withDiabetes" that gathers data with diabetes

with_diabetes = data[data['Classification'] == 'Diabetes']
print(with_diabetes.head())

[13] ✓ 0.0s
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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

   DiabetesPedigreeFunction  Age  Diagnosis Classification
0                     0.627   50          1       Diabetes
2                     0.672   32          1       Diabetes
4                     2.288   33          1       Diabetes
6                     0.248   26          1       Diabetes
8                     0.158   53          1       Diabetes
```

9. Create new dataFrame "noDiabetes" that gathers data with no diabetes.

```
# Create new dataFrame "noDiabetes" that gathers data with no diabetes

no_diabetes = data[data['Classification'] == 'No Diabetes']
print(no_diabetes.head())

[14] ✓ 0.0s
```

```
    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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

    DiabetesPedigreeFunction  Age  Diagnosis Classification
1                      0.351   31          0    No Diabetes
3                      0.167   21          0    No Diabetes
5                      0.201   30          0    No Diabetes
7                      0.134   29          0    No Diabetes
10                     0.191   30          0    No Diabetes
```

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

```
# Create new dataFrame "Pedia" that gathers data with age 0 to 19

pedia = data[data['Age'] <= 19]
print(pedia.head())

# Returns empty cell due to nothing from the data with the age <= 19.
```

[17] ✓ 0.0s

```
Empty DataFrame
Columns: [Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age, Diagnosis, Classification]
Index: []
```

11. Create new dataFrame "Pedia" that gathers data with age greater than 19.

```
# Create new dataFrame "Pedia" that gathers data with age greater than 19
pedia = data[data['Age'] > 19]
print(pedia.head())
```

[18] ✓ 0.0s

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Diagnosis	Classification
0	6	148	72	35	0	33.6	0.627	50	1	Diabetes
1	1	85	66	29	0	26.6	0.351	31	0	No Diabetes
2	8	183	64	0	0	23.3	0.672	32	1	Diabetes
3	1	89	66	23	94	28.1	0.167	21	0	No Diabetes
4	0	137	40	35	168	43.1	2.288	33	1	Diabetes

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

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

print()
ave_glucose = np.mean(data['Glucose'])
print(f"Average Glucose:", ave_glucose)
```

[23] ✓ 0.0s

```
Average Age: 33.240885416666664

Average Glucose: 120.89453125
```

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

```
# Use numpy to get the median age and glucose value
median_age = np.median(data['Age'])
print(f"Median Age:", median_age)

print()
median_glucose = np.median(data['Glucose'])
print(f"Median Glucose:", median_glucose)
```

[24] ✓ 0.0s

```
Median Age: 29.0

Median Glucose: 117.0
```

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

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

print()
middle_age = np.median(data['Age'])
print(f"Middle value for Age:", middle_age)
```

✓ 0.0s

```
Middle value for Glucose: 117.0

Middle value for Age: 29.0
```

Activity #14 and #13 is the same thought. This two are looking for the Median value of the data set.

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

```
# Use numpy to get the standard deviation of the skinthickness
std_skinthickness = np.std(data['SkinThickness'])
print(f"Standard Deviation for Skinthickness:", std_skinthickness)

# Import Statics and Numpy has different attribute for Standard Deviation
```

[30] ✓ 0.0s

```
Standard Deviation for Skinthickness: 15.941828626496939
```

6.3 Conclusion

In this learning exercise, we explored various statistical concepts using Python and libraries like NumPy and pandas. We learned how to compute basic statistics such as the mean (average), median (middle value), and mode (most frequent value) without using advanced libraries. We also explored more complex statistical measures like standard deviation and variance, which help us understand how spread-out data is. Through hands-on coding, we saw how to add new columns, filter data, and extract useful information such as creating a new classification column for diabetes and generating subsets of data with specific conditions.

In conclusion, mastering these basic statistical tools allows us to better understand and analyze data. We also saw that Python provides multiple ways to manipulate and work with data efficiently, from manually calculating statistics to using built-in functions in NumPy and pandas, and in data cleaning. This hands-on experience reinforces the importance of knowing when to use specific statistical measures and how to apply them to real-world datasets. By learning how to apply these techniques, we gain valuable skills in data analysis, which are essential in many fields today.