Generic Elective – Computer Science

Data Analysis and Visualisation using Python

Semester – II

2023-24

PRACTICAL FILE

Lucky Lao

B.Sc. (Hons.) Mathematics

23BMAT019

Q1. Write programs in Python using NumPy library to do the following:

a. Compute the mean, standard deviation, and variance of a two dimensional random integer array along the second axis.

Code:

```
import numpy as np

m = int(input("Enter m (rows): "))
n = int(input("Enter n (columns): "))

arr = np.random.randint(10, size=(m, n))
print("\nOriginal Array:")
print(arr)

print("\nMean of the Array along the second axis:", np.mean(arr, axis=1))
print("Standard Deviation of Array along the second axis:", np.std(arr, axis=1))
print("Variance of Array along the second axis:", np.var(arr, axis=1))
```

Output:

```
Enter m (rows):  3
Enter n (columns):  4

Original Array:
[[5 4 6 6]
 [6 3 0 3]
 [1 5 3 6]]

Mean of the Array along the second axis: [5.25 3.   3.75]
Standard Deviation of Array along the second axis: [0.8291562  2.12132034 1.92028644]
Variance of Array along the second axis: [0.6875 4.5    3.6875]
```

b. Create a 2-dimensional array of size m x n integer elements, also print the shape, type and data type of the array and then reshape it into an n x m array, where n and m are user inputs given at the run time.

Code:

```
import numpy as np

m = int(input("Enter m (rows): "))
n = int(input("Enter n (columns): "))

arr = np.random.randint(10, size=(m,n))
print("\nOriginal Array:")
print(arr)

print("Shape of Array:", arr.shape)
print("Dimension of Array:", arr.ndim)
print("Data type of Array:", arr.dtype)

# Re-shaping the array
arr_new = np.reshape(arr, (n, m))
print("\nRe-shaped Array:")
print(arr_new)
```

Output:

```
Enter m (rows):  3
Enter n (columns):  4

Original Array:
[[9 9 6 8]
 [3 7 9 5]
 [3 5 3 8]]
Shape of Array: (3, 4)
Dimension of Array: 2
Data type of Array: int32

Re-shaped Array:
[[9 9 6]
 [8 3 7]
 [9 5 3]
 [5 3 8]]
```

c. Test whether the elements of a given 1D array are zero, non-zero and NaN. Record the indices of these elements in three separate arrays.

Code:

```
import numpy as np

# Creating an array containing zero, non-zero and NaN elements
ar = np.zeros(15)
print(f'Original array is:\n{ar}')
ar[3:6] = 100
ar[7:9] = float('Nan')
ar[12:14] = float('Nan')
print(f'\nNew array is:\n{ar}')

# Creating a boolean array representing indices with 0 as the element
bool_zero = (ar == 0)
print(f'\nBoolean Array with zero-indicators:\n{bool_zero}')

# Creating a boolean array representing indices with a non-zero element
bool_nonzero = (ar!=0 & np.isnan(ar))
print(f'\nBoolean Array with non-zero-indicators:\n{bool_nonzero}')

# Creating a boolean array representing indices with a null/NaN element
bool_nan = np.isnan(ar)
print(f'\nBoolean Array with null-indicators:\n{bool_nan}')

# Creating an index array representing the indices of 'ar'
index=np.arange(15)
print(f'\nIndex Array:\n{index}')

# Using boolean indexing to retrieve index arrays - zero, non-zero and nan elements of ar.
zero_index = index[bool_zero == True]
print(f'\nArray containing indices of zero entries:\n{zero_index}')

non_zero_index = index[bool_nonzero == True]
print(f'\nArray containing indices of non-zero entries:\n{non_zero_index}')

nan_index = index[bool_nan == True]
print(f'\nArray containing indices of NaN enries:\n{nan_index}')
```

(Output on Next Page)

Output:

```
Original array is:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

New array is:
[  0.   0.   0. 100. 100. 100.   0.  nan  nan   0.   0.   0.  nan  nan
   0.]

Boolean Array with zero-indicators:
[ True  True  True False False False  True False False  True  True  True
 False False  True]

Boolean Array with non-zero-indicators:
[False False False  True  True  True False  True  True False False False
  True  True False]

Boolean Array with null-indicators:
[False False False False False False False  True  True False False False
  True  True False]

Index Array:
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]

Array containing indices of zero entries:
[ 0  1  2  6  9 10 11 14]

Array containing indices of non-zero entries:
[ 3  4  5  7  8 12 13]

Array containing indices of NaN enries:
[ 7  8 12 13]
```

d. Create three random arrays of the same size: Array1, Array2 and Array3. Subtract Array 2 from Array3 and store in Array4. Create another array Array5 having two times the values in Array1. Find Co-variance and Correlation of Array1 with Array4 and Array5 respectively.

Code:

```
import numpy as np

Array1 = np.random.random((3, 3))
print("Array1:")
print(Array1)

Array2 = np.random.random((3, 3))
print("\nArray2:")
print(Array2)

Array3 = np.random.random((3, 3))
print("\nArray3:")
print(Array3)

Array4 = Array2 - Array3
print("\nArray4 (Array2 - Array3):")
print(Array4)

Array5 = 2 * Array1
print("\nArray4 (2 * Array1):")
print(Array5)

print("\nCo-variance between Array1 and Array4:")
print(np.cov(Array4, Array1))

print("\nCorrelation between Array1 and Array5:")
print(np.corrcoef(Array5, Array1))
```

(Output on Next Page)

Output:

```
Array1:
[[0.25388684 0.09025092 0.31568552]
 [0.89595299 0.36696187 0.26172833]
 [0.30998816 0.62019591 0.90459939]]

Array2:
[[0.29110749 0.11162578 0.88486029]
 [0.51512098 0.48241184 0.90833106]
 [0.97358125 0.06691973 0.01388881]]

Array3:
[[0.22997612 0.48743758 0.63734098]
 [0.99849213 0.46402001 0.00191505]
 [0.17207097 0.1624605  0.25289828]]

Array4 (Array2 - Array3):
[[ 0.06113137 -0.3758118   0.2475193 ]
 [-0.48337115  0.01839183  0.90641601]
 [ 0.80151028 -0.09554077 -0.23900947]]

Array4 (2 * Array1):
[[0.50777367 0.18050184 0.63137104]
 [1.79190599 0.73392374 0.52345667]
 [0.61997631 1.24039182 1.80919878]]

Co-variance between Array1 and Array4:
[[ 0.10236691  0.09888846  0.01809858  0.03725642  0.00788865  0.02542712]
 [ 0.09888846  0.49531023 -0.33726861  0.03399532 -0.20671923  0.20576517]
 [ 0.01809858 -0.33726861  0.31799421  0.00835737  0.19159218 -0.15629665]
 [ 0.03725642  0.03399532  0.00835737  0.01356943  0.00394074  0.00834991]
 [ 0.00788865 -0.20671923  0.19159218  0.00394074  0.11552444 -0.09519051]
 [ 0.02542712  0.20576517 -0.15629665  0.00834991 -0.09519051  0.08844612]]

Correlation between Array1 and Array5:
[[ 1.          0.09953132  0.2410247   1.          0.09953132  0.2410247 ]
 [ 0.09953132  1.         -0.94171029  0.09953132  1.         -0.94171029]
 [ 0.2410247  -0.94171029  1.          0.2410247  -0.94171029  1.        ]
 [ 1.          0.09953132  0.2410247   1.          0.09953132  0.2410247 ]
 [ 0.09953132  1.         -0.94171029  0.09953132  1.         -0.94171029]
 [ 0.2410247  -0.94171029  1.          0.2410247  -0.94171029  1.        ]]
```

e. Create two random arrays of the same size 10: Array1, and Array2. Find the sum of the first half of both the arrays and product of the second half of both the arrays.

Code:

```
import numpy as np

Array1 = np.random.random(size=10)
print("Array1:")
print(Array1)

Array2 = np.random.random(size=10)
print("\nArray2:")
print(Array2)

print("\nSum of the first half of both arrays:")
print(Array1[:5] + Array2[:5])

print("\nProduct of the second half of both arrays:")
print(Array1[5:] * Array2[5:])
```

Output:

```
Array1:
[0.03582014 0.08651176 0.18055458 0.04456098 0.26991844 0.31331365
 0.65042008 0.87145045 0.15883223 0.88449827]

Array2:
[0.5640922  0.97819022 0.1556042  0.06088445 0.50372476 0.00971907
 0.87853942 0.67353239 0.55051318 0.9528223 ]

Sum of the first half of both arrays:
[0.59991234 1.06470198 0.33615877 0.10544543 0.7736432 ]

Product of the second half of both arrays:
[0.00304512 0.57141968 0.5869501  0.08743924 0.84276968]
```

Q2. Do the following using PANDAS Series:

a. Create a series with 5 elements. Display the series sorted on index and also sorted on values separately

Code:

```
import pandas as pd

s1 = pd.Series([7, 8, 9, 4, 5], index=['a', 'b', 'c', 'd', 'e'])
print(f"The original series is:\n{s1}")

x = s1.sort_index()
print(f"\nThe series sorted on the basis of index is:\n{x}")

y = s1.sort_values()
print(f"\nThe series sorted on the basis of values is:\n{y}")
```

Output:

```
The original series is:
a    7
b    8
c    9
d    4
e    5
dtype: int64

The series sorted on the basis of index is:
a    7
b    8
c    9
d    4
e    5
dtype: int64

The series sorted on the basis of values is:
d    4
e    5
a    7
b    8
c    9
dtype: int64
```

b. Create a series with N elements with some duplicate values. Find the minimum and maximum ranks assigned to the values using 'first' and 'max' methods

Code:

```
import pandas as pd
s = pd.Series([2, 10, 5, 7, 10, 3, 2, 9, 4, 8], index=list("abcdefghij"))
print(f"Ranks through 'first' method are:\n{s.rank(method='first')}")
print(f"\nRanks through 'max' method are:\n{s.rank(method='max')}")
```

Output:

```
Ranks through 'first' method are:
a     1.0
b     9.0
c     5.0
d     6.0
e    10.0
f     3.0
g     2.0
h     8.0
i     4.0
j     7.0
dtype: float64

Ranks through 'max' method are:
a     2.0
b    10.0
c     5.0
d     6.0
e    10.0
f     3.0
g     2.0
h     8.0
i     4.0
j     7.0
dtype: float64
```

c. Display the index value of the minimum and maximum element of a Series

Code:

```
import numpy as np
import pandas as pd

obj1 = pd.Series(np.arange(10), index=list("abcdefghij"))
print(f"The series is:\n{obj1}")

print(f"\nIndex value for the minimum element of the series is: {obj1.idxmin()}")
print(f"\nIndex value for the maximum element of the series is: {obj1.idxmax()}")
```

Output:

```
The series is:
a    0
b    1
c    2
d    3
e    4
f    5
g    6
h    7
i    8
j    9
dtype: int32

Index value for the minimum element of the series is: a

Index value for the maximum element of the series is: j
```

Q3. Create a data frame having at least 3 columns and 50 rows to store numeric data generated using a random function. Replace 10% of the values by null values whose index positions are generated using random function.

Code: (Creating the required data frame)

```
import numpy as np
import pandas as pd

data = np.random.randint(0, 500, size=(50,3))
df = pd.DataFrame(data)
print(f"The original dataframe is:\n{df}")

for i in range(15):
    a = np.random.randint(0, 50)
    b = np.random.randint(0, 3)
    df.iloc[a,b] = np.nan
print(f"The updated dataframe is:\n{df}")
```

(Output on Next Page)

Output:

```
The original dataframe is:
      0    1    2
0    12  358  248
1    10  379   90
2   287   57  460
3   113  228  497
4   438  390  159
5    69   70   35
6    78  404  326
7   395  130  410
8    35   79  128
9    33  172  353
10  119  338  104
11   69  498  370
12  265  114  156
13  423  190  347
14  453  424  280
15   25  155  169
16  408  443   49
17  303  404  119
18   32   10  355
19   77  438   65
20  179  379   38
21  456  171  423
22   58  336   24
23   35  466  272
24  199  400   59
25  409  207  149
26  283  177  338
27   90  374  309
28  118  489  327
29  199  180   63
30  224  374  311
31  491   65  164
32  483  274  120
33  216  387  292
34  402  188  103
35   58  185  468
36  433  283  237
37  473  354  448
38  474  276  345
39  225   84  423
40  437  116  326
41  213  279  286
42  396  226  258
43   10  331  136
44  432    1  306
45  410  203   72
46  335  202  140
47  402  455  433
48  163  411  173
49  399  470  256
```

```
The updated dataframe is:
        0      1      2
0    12.0  358.0  248.0
1    10.0  379.0   90.0
2   287.0   57.0  460.0
3   113.0  228.0  497.0
4   438.0  390.0  159.0
5    69.0   70.0   35.0
6    78.0  404.0    NaN
7   395.0  130.0  410.0
8     NaN   79.0  128.0
9    33.0  172.0  353.0
10  119.0  338.0  104.0
11   69.0  498.0  370.0
12  265.0  114.0  156.0
13  423.0  190.0  347.0
14  453.0  424.0  280.0
15   25.0  155.0    NaN
16  408.0  443.0   49.0
17  303.0  404.0  119.0
18   32.0   10.0  355.0
19   77.0  438.0   65.0
20    NaN  379.0   38.0
21    NaN  171.0    NaN
22    NaN  336.0   24.0
23   35.0  466.0    NaN
24  199.0  400.0   59.0
25  409.0  207.0  149.0
26  283.0  177.0  338.0
27   90.0    NaN  309.0
28  118.0  489.0  327.0
29  199.0  180.0   63.0
30  224.0  374.0    NaN
31  491.0   65.0  164.0
32  483.0  274.0  120.0
33  216.0    NaN  292.0
34  402.0  188.0  103.0
35   58.0  185.0  468.0
36  433.0  283.0  237.0
37  473.0  354.0  448.0
38  474.0  276.0  345.0
39  225.0   84.0    NaN
40    NaN  116.0  326.0
41  213.0  279.0  286.0
42  396.0  226.0    NaN
43   10.0  331.0  136.0
44  432.0    1.0  306.0
45  410.0  203.0    NaN
46  335.0  202.0  140.0
47  402.0  455.0  433.0
48  163.0  411.0  173.0
49  399.0  470.0  256.0
```

Do the following:

a. Identify and count missing values in a data frame.

Code:

```
print(df.isnull().sum())
print(df.isnull().sum().sum())
```

Output:

```
0    5
1    2
2    8
dtype: int64
15
```

b. Drop the column having more than 5 null values.

Code:

```
print(df.dropna(axis=1, thresh=45))
```

(Output on Next Page)

Output:

```
        0      1
0    12.0  358.0
1    10.0  379.0
2   287.0   57.0
3   113.0  228.0
4   438.0  390.0
5    69.0   70.0
6    78.0  404.0
7   395.0  130.0
8     NaN   79.0
9    33.0  172.0
10  119.0  338.0
11   69.0  498.0
12  265.0  114.0
13  423.0  190.0
14  453.0  424.0
15   25.0  155.0
16  408.0  443.0
17  303.0  404.0
18   32.0   10.0
19   77.0  438.0
20    NaN  379.0
21    NaN  171.0
22    NaN  336.0
23   35.0  466.0
24  199.0  400.0
25  409.0  207.0
26  283.0  177.0
27   90.0    NaN
28  118.0  489.0
29  199.0  180.0
30  224.0  374.0
31  491.0   65.0
32  483.0  274.0
33  216.0    NaN
34  402.0  188.0
35   58.0  185.0
36  433.0  283.0
37  473.0  354.0
38  474.0  276.0
39  225.0   84.0
40    NaN  116.0
41  213.0  279.0
42  396.0  226.0
43   10.0  331.0
44  432.0    1.0
45  410.0  203.0
46  335.0  202.0
47  402.0  455.0
48  163.0  411.0
49  399.0  470.0
```

c. Identify the row label having maximum of the sum of all values in a row and drop that row.

Code:

```
max_row = df.sum(axis=1).idxmax()
print("The row with maximum sum value is:", max_row)
print(df.drop(max_row))
```

(Output on Next Page)

Output:

```
The row with maximum sum value is: 47
        0      1      2
0    12.0  358.0  248.0
1    10.0  379.0   90.0
2   287.0   57.0  460.0
3   113.0  228.0  497.0
4   438.0  390.0  159.0
5    69.0   70.0   35.0
6    78.0  404.0    NaN
7   395.0  130.0  410.0
8     NaN   79.0  128.0
9    33.0  172.0  353.0
10  119.0  338.0  104.0
11   69.0  498.0  370.0
12  265.0  114.0  156.0
13  423.0  190.0  347.0
14  453.0  424.0  280.0
15   25.0  155.0    NaN
16  408.0  443.0   49.0
17  303.0  404.0  119.0
18   32.0   10.0  355.0
19   77.0  438.0   65.0
20    NaN  379.0   38.0
21    NaN  171.0    NaN
22    NaN  336.0   24.0
23   35.0  466.0    NaN
24  199.0  400.0   59.0
25  409.0  207.0  149.0
26  283.0  177.0  338.0
27   90.0    NaN  309.0
28  118.0  489.0  327.0
29  199.0  180.0   63.0
30  224.0  374.0    NaN
31  491.0   65.0  164.0
32  483.0  274.0  120.0
33  216.0    NaN  292.0
34  402.0  188.0  103.0
35   58.0  185.0  468.0
36  433.0  283.0  237.0
37  473.0  354.0  448.0
38  474.0  276.0  345.0
39  225.0   84.0    NaN
40    NaN  116.0  326.0
41  213.0  279.0  286.0
42  396.0  226.0    NaN
43   10.0  331.0  136.0
44  432.0    1.0  306.0
45  410.0  203.0    NaN
46  335.0  202.0  140.0
48  163.0  411.0  173.0
49  399.0  470.0  256.0
```

d. Sort the data frame on the basis of the first column.

Code:

```
print(df.sort_values([0]))
```

(Output on Next Page)

Output:

```
        0      1      2
1    10.0  379.0   90.0
43   10.0  331.0  136.0
0    12.0  358.0  248.0
15   25.0  155.0    NaN
18   32.0   10.0  355.0
9    33.0  172.0  353.0
23   35.0  466.0    NaN
35   58.0  185.0  468.0
5    69.0   70.0   35.0
11   69.0  498.0  370.0
19   77.0  438.0   65.0
6    78.0  404.0    NaN
27   90.0    NaN  309.0
3   113.0  228.0  497.0
28  118.0  489.0  327.0
10  119.0  338.0  104.0
48  163.0  411.0  173.0
29  199.0  180.0   63.0
24  199.0  400.0   59.0
41  213.0  279.0  286.0
33  216.0    NaN  292.0
30  224.0  374.0    NaN
39  225.0   84.0    NaN
12  265.0  114.0  156.0
26  283.0  177.0  338.0
2   287.0   57.0  460.0
17  303.0  404.0  119.0
46  335.0  202.0  140.0
7   395.0  130.0  410.0
42  396.0  226.0    NaN
49  399.0  470.0  256.0
47  402.0  455.0  433.0
34  402.0  188.0  103.0
16  408.0  443.0   49.0
25  409.0  207.0  149.0
45  410.0  203.0    NaN
13  423.0  190.0  347.0
44  432.0    1.0  306.0
36  433.0  283.0  237.0
4   438.0  390.0  159.0
14  453.0  424.0  280.0
37  473.0  354.0  448.0
38  474.0  276.0  345.0
32  483.0  274.0  120.0
31  491.0   65.0  164.0
8     NaN   79.0  128.0
20    NaN  379.0   38.0
21    NaN  171.0    NaN
22    NaN  336.0   24.0
40    NaN  116.0  326.0
```

e. Remove all duplicates from the first column.

Code:

```
print(df.drop_duplicates([0]))
```

Output:

```
        0      1      2
0    12.0  358.0  248.0
1    10.0  379.0   90.0
2   287.0   57.0  460.0
3   113.0  228.0  497.0
4   438.0  390.0  159.0
5    69.0   70.0   35.0
6    78.0  404.0    NaN
7   395.0  130.0  410.0
8     NaN   79.0  128.0
9    33.0  172.0  353.0
10  119.0  338.0  104.0
12  265.0  114.0  156.0
13  423.0  190.0  347.0
14  453.0  424.0  280.0
15   25.0  155.0    NaN
16  408.0  443.0   49.0
17  303.0  404.0  119.0
18   32.0   10.0  355.0
19   77.0  438.0   65.0
23   35.0  466.0    NaN
24  199.0  400.0   59.0
25  409.0  207.0  149.0
26  283.0  177.0  338.0
27   90.0    NaN  309.0
28  118.0  489.0  327.0
30  224.0  374.0    NaN
31  491.0   65.0  164.0
32  483.0  274.0  120.0
33  216.0    NaN  292.0
34  402.0  188.0  103.0
35   58.0  185.0  468.0
36  433.0  283.0  237.0
37  473.0  354.0  448.0
38  474.0  276.0  345.0
39  225.0   84.0    NaN
41  213.0  279.0  286.0
42  396.0  226.0    NaN
44  432.0    1.0  306.0
45  410.0  203.0    NaN
46  335.0  202.0  140.0
48  163.0  411.0  173.0
49  399.0  470.0  256.0
```

f. Find the correlation between first and second column and covariance between second and third column.

Code:

```
print(df[0].corr(df[1]))
print(df[1].corr(df[2]))
```

Output:

```
-0.10775273312299381
-0.17241754130534206
```

g. Discretize the second column and create 5 bins.

Code:

```
bins = [0, 100, 200, 300, 400, 500]
pd.cut(df[1], bins)
```

(Output on Next Page)

Output:

```
0      (300.0, 400.0]
1      (300.0, 400.0]
2        (0.0, 100.0]
3      (200.0, 300.0]
4      (300.0, 400.0]
5        (0.0, 100.0]
6      (400.0, 500.0]
7      (100.0, 200.0]
8        (0.0, 100.0]
9      (100.0, 200.0]
10     (300.0, 400.0]
11     (400.0, 500.0]
12     (100.0, 200.0]
13     (100.0, 200.0]
14     (400.0, 500.0]
15     (100.0, 200.0]
16     (400.0, 500.0]
17     (400.0, 500.0]
18       (0.0, 100.0]
19     (400.0, 500.0]
20     (300.0, 400.0]
21     (100.0, 200.0]
22     (300.0, 400.0]
23     (400.0, 500.0]
24     (300.0, 400.0]
25     (200.0, 300.0]
26     (100.0, 200.0]
27                NaN
28     (400.0, 500.0]
29     (100.0, 200.0]
30     (300.0, 400.0]
31       (0.0, 100.0]
32     (200.0, 300.0]
33                NaN
34     (100.0, 200.0]
35     (100.0, 200.0]
36     (200.0, 300.0]
37     (300.0, 400.0]
38     (200.0, 300.0]
39       (0.0, 100.0]
40     (100.0, 200.0]
41     (200.0, 300.0]
42     (200.0, 300.0]
43     (300.0, 400.0]
44       (0.0, 100.0]
45     (200.0, 300.0]
46     (200.0, 300.0]
47     (400.0, 500.0]
48     (400.0, 500.0]
49     (400.0, 500.0]
Name: 1, dtype: category
Categories (5, interval[int64, right]): [(0, 100] < (100, 200] < (200, 300] < (300, 400] <
(400, 500]]
```

Q4. Consider two excel files having attendance of two workshops. Each file has three fields 'Name', 'Date', 'Duration (in minutes)' where names are unique within a file. Note that duration may take one of three values (30, 40, 50) only. Import the data into two data frames.

<u>File contents:</u> (**`work1.csv`** and **`work2.csv`** respectively)

	A	B	C
1	Name	Date	Duration
2	Anya	01-07-2023	30
3	Amit	01-07-2023	40
4	Geeti	01-07-2023	50
5	Shikha	01-07-2023	30

	A	B	C
1	Name	Date	Duration
2	Shikha	02-07-2023	50
3	Amit	02-07-2023	50
4	Arun	02-07-2023	30

<u>Code:</u> (Importing the data into data frames)

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

work1_df = pd.read_csv("C:\\Users\\Student_2\\Desktop\\work1.csv")
work2_df = pd.read_csv("C:\\Users\\Student_2\\Desktop\\work2.csv")
print(work1_df)
print(work2_df)
```

<u>Output:</u>

```
     Name        Date  Duration
0    Anya  01-07-2023        30
1    Amit  01-07-2023        40
2   Geeti  01-07-2023        50
3  Shikha  01-07-2023        30
     Name        Date  Duration
0  Shikha  02-07-2023        50
1    Amit  02-07-2023        50
2    Arun  02-07-2023        30
```

Now, do the following:

a. Perform merging of the two data frames to find the names of students who had attended both workshops.

Code:

```
print("Students who have attended both the workshops are...\n",
      pd.merge(work1_df, work2_df, on="Name"))
```

Output:

```
Students who have attended both the workshops are...
      Name      Date_x  Duration_x      Date_y  Duration_y
0     Amit  01-07-2023          40  02-07-2023          50
1   Shikha  01-07-2023          30  02-07-2023          50
```

b. Find names of all students who have attended a single workshop only.

Code:

```
print("Students who have attended a single workshop are...\n",
      pd.concat([work1_df, work2_df]).drop_duplicates("Name", keep=False))
```

Output:

```
Students who have attended a single workshop are...
     Name        Date  Duration
0    Anya  01-07-2023        30
2   Geeti  01-07-2023        50
2    Arun  02-07-2023        30
```

c. Merge two data frames row-wise and find the total number of records in the data frames.

Code:

```
mer_row = pd.concat([work1_df, work2_df]).reset_index(drop=True)
print("Merging data frames row wise\n", mer_row)
print("Total number of records in the data frames:", mer_row.shape[0])
```

Output:

```
Merging data frames row wise
      Name        Date  Duration
0    Anya  01-07-2023        30
1    Amit  01-07-2023        40
2   Geeti  01-07-2023        50
3  Shikha  01-07-2023        30
4  Shikha  02-07-2023        50
5    Amit  02-07-2023        50
6    Arun  02-07-2023        30
Total number of records in the data frames: 7
```

d. Merge two data frames row-wise and use two columns viz. names and dates as multi-row indexes.

Code:

```
multi_index = pd.concat([work1_df.set_index(['Name', 'Date']),
                         work2_df.set_index(['Name', 'Date'])])
print(multi_index)
```

Output:

```
                   Duration
Name   Date
Anya   01-07-2023        30
Amit   01-07-2023        40
Geeti  01-07-2023        50
Shikha 01-07-2023        30
       02-07-2023        50
Amit   02-07-2023        50
Arun   02-07-2023        30
```

Generate descriptive statistics for this hierarchical data frame.

Code:

```
des_stats = multi_index.describe()
print(des_stats)
```

Output:

```
       Duration
count       7.0
mean       40.0
std        10.0
min        30.0
25%        30.0
50%        40.0
75%        50.0
max        50.0
```

Q5. Using Iris data, plot the following with proper legend and axis labels: (Download IRIS data from: https://archive.ics.uci.edu/ml/datasets/iris or import it from sklearn datasets)

Code:

(Importing necessary libraries)

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from scipy import stats
```

(Loading IRIS data)

```
iris = load_iris()
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
target = iris.target
target_names = iris.target_names
print(data)
```

Output:

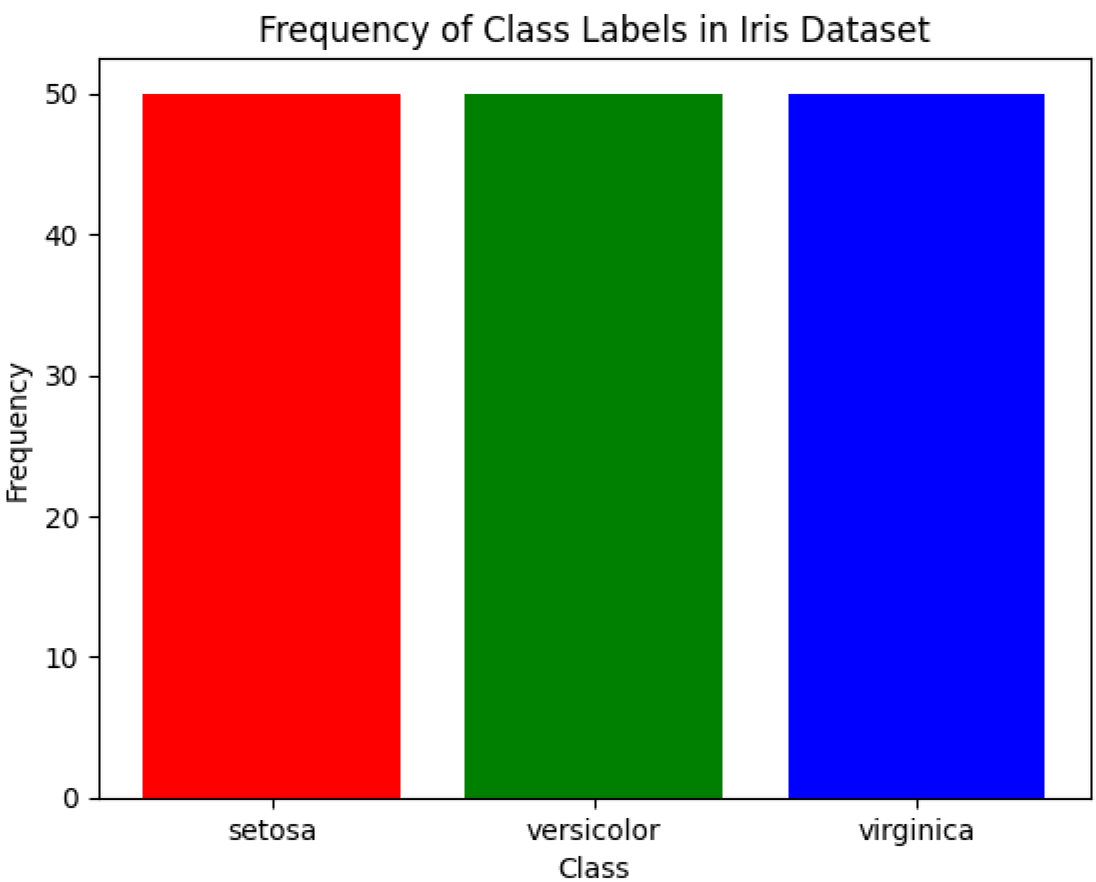
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

[150 rows x 4 columns]
```

a. Plot bar chart to show the frequency of each class label in the data.

<u>Code:</u>

```
class_counts = data.groupby(target_names[target]).size()
plt.bar(target_names, class_counts, color=["red","green","blue"])
plt.xlabel("Class")
plt.ylabel("Frequency")
plt.title("Frequency of Class Labels in Iris Dataset")
plt.show()
```

<u>Output:</u>

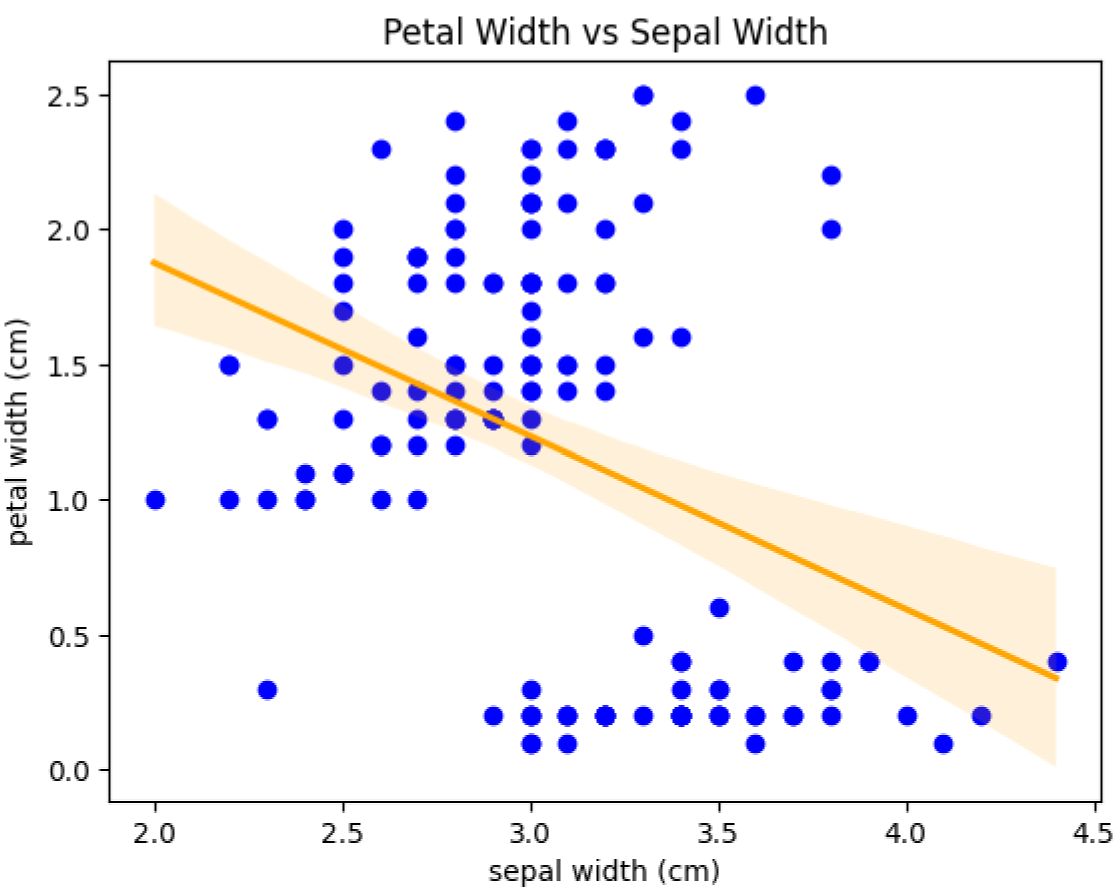


b. Draw a scatter plot for Petal width vs sepal width and fit a regression line

Code:

```
plt.scatter(data["sepal width (cm)"], data["petal width (cm)"], color="blue")
plt.xlabel("Sepal Width (cm)")
plt.ylabel("Petal Width (cm)")
plt.title("Petal Width vs Sepal Width")
sns.regplot(x="sepal width (cm)", y="petal width (cm)", data=data, scatter=False, color="orange")
plt.show()
```

Output:

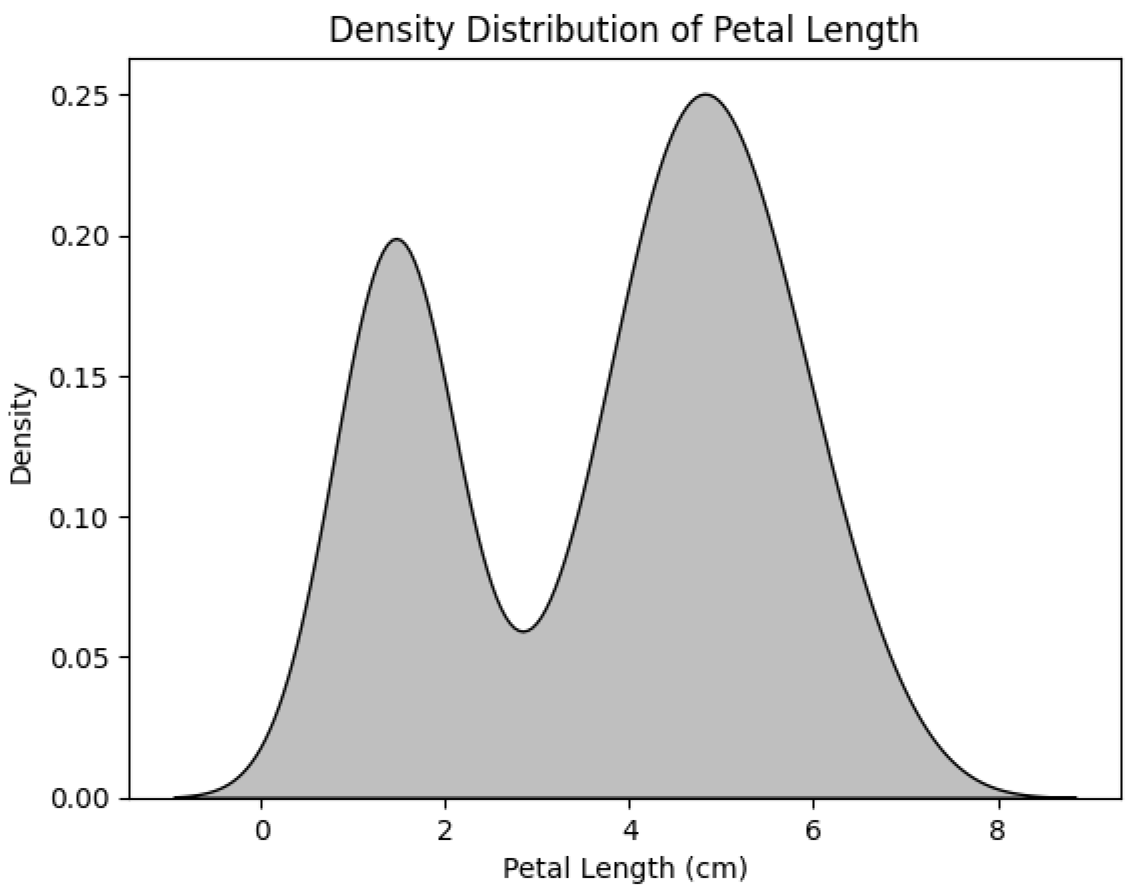


c. Plot density distribution for feature petal length.

Code:

```
sns.kdeplot(data["petal length (cm)"], fill=True, color="black")
plt.xlabel("Petal Length (cm)")
plt.ylabel("Density")
plt.title("Density Distribution of Petal Length")
plt.show()
```

Output:

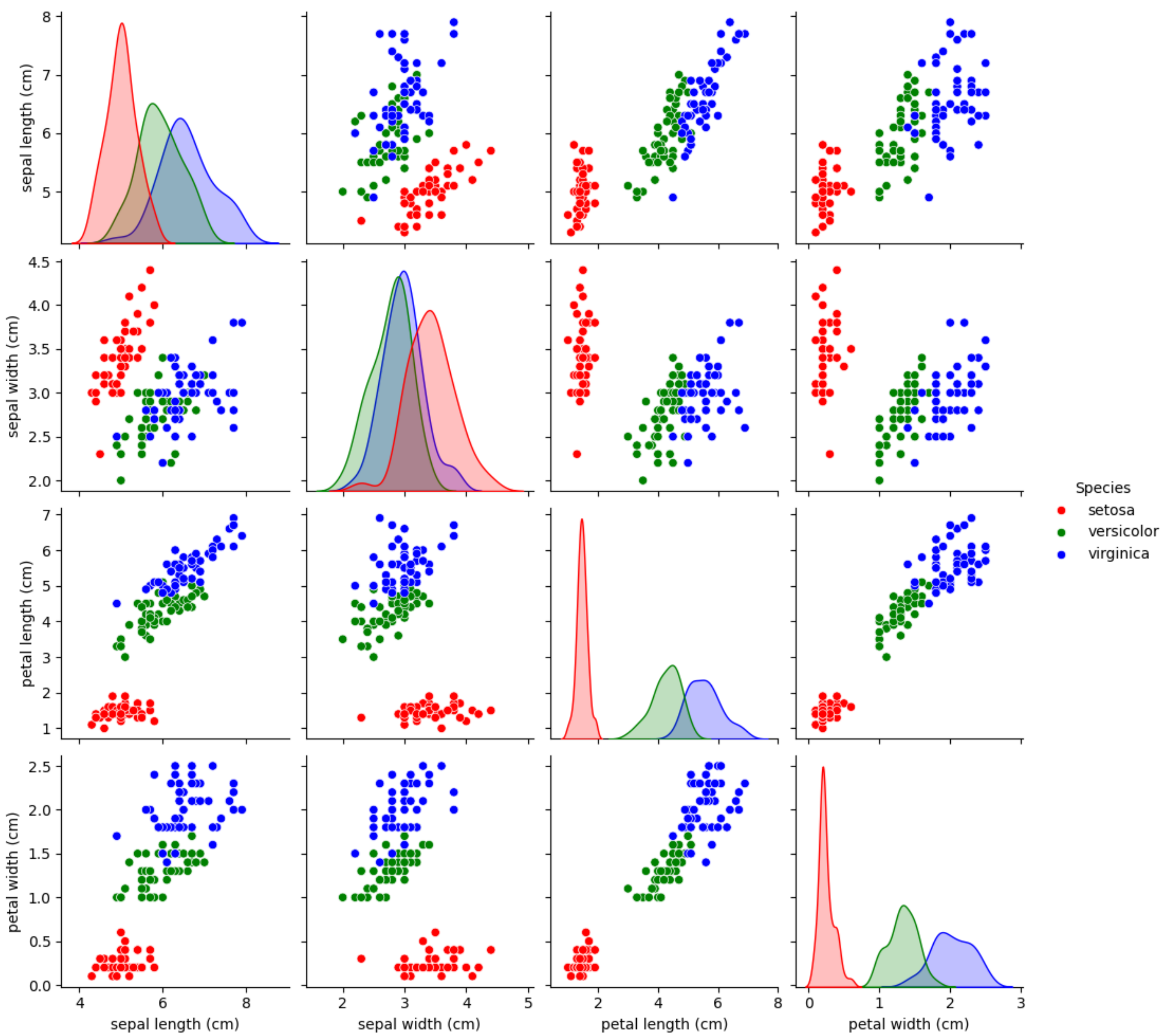


d. Use a pair plot to show pairwise bivariate distribution in the Iris Dataset.

Code:

```
classes = {0:"setosa", 1:"versicolor", 2:"virginica"}
data["Target"] = iris.target
data["Species"] = data["Target"].map(classes)
data.drop("Target", axis=1, inplace=True)
sns.pairplot(data=data, hue="Species", palette=["Red", "Green", "Blue"])
plt.show()
```

Output:

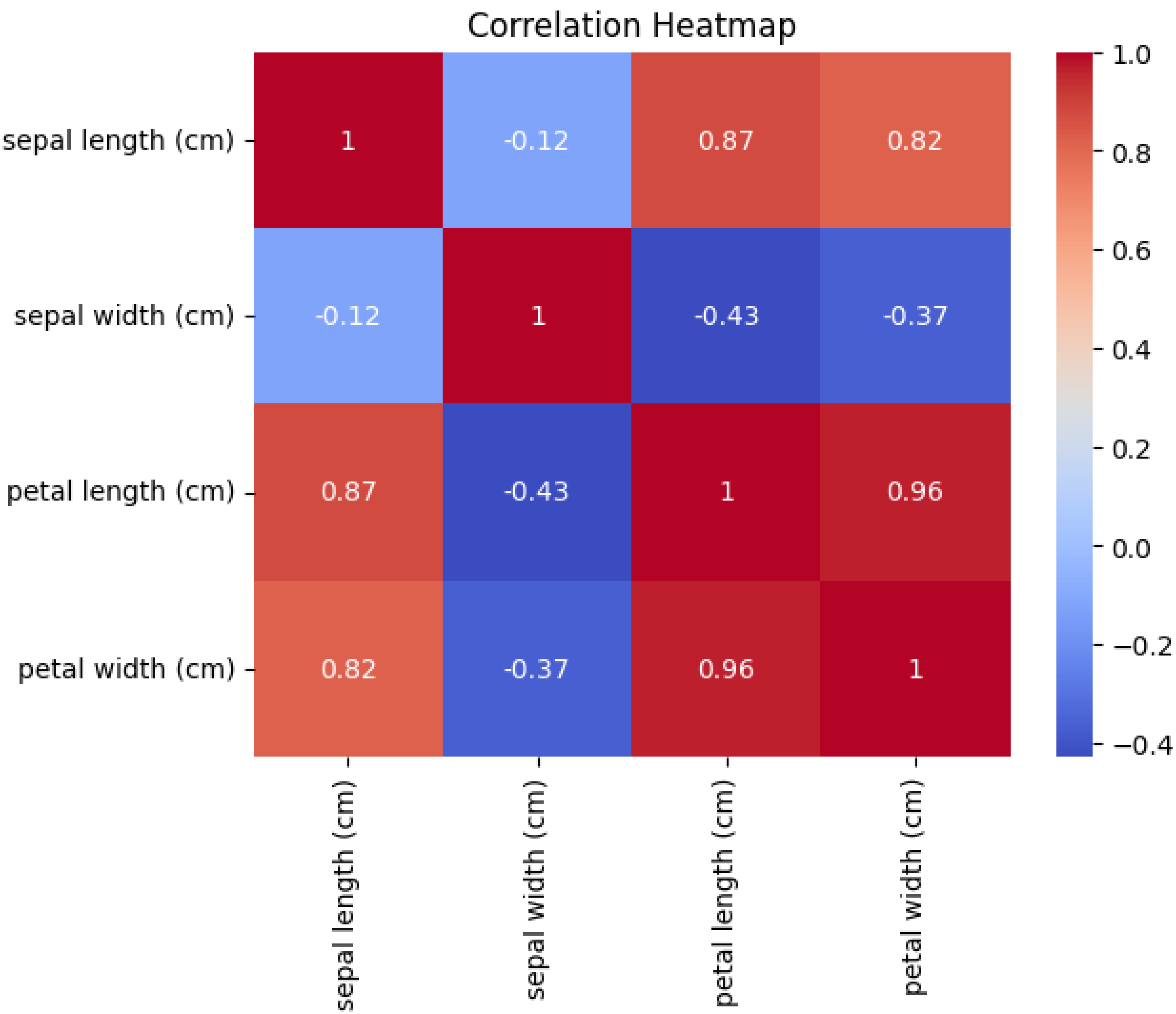


e. Draw heatmap for the four numeric attributes

Code:

```
data.drop("Species", axis=1, inplace=True)
sns.heatmap(data.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```

Output:



f. Compute mean, mode, median, standard deviation, confidence interval and standard error for each feature

Code:

```
feature_stats = data.describe().loc[["mean", "std", "50%"]]
feature_stats = feature_stats.append(data.mode().iloc[0], ignore_index=True)
conf_int = stats.t.interval(0.95, len(data)-1, loc=data.mean(), scale=stats.sem(data))
feature_stats = feature_stats.append(pd.Series(conf_int, index=["95% CI Min", "95% CI Max"]),
                                     ignore_index=True)
feature_stats.index = ["Mean", "Standard Deviation", "Media", "Mode", "CI"]
print(feature_stats)
```

Output:

```
                    sepal length (cm)  sepal width (cm)  petal length (cm)  \
Mean                         5.843333          3.057333           3.758000
Standard Deviation           0.828066          0.435866           1.765298
Median                       5.800000          3.000000           4.350000
Mode                         5.000000          3.000000           1.400000
CI                                NaN               NaN                NaN

                    petal width (cm)  \
Mean                        1.199333
Standard Deviation          0.762238
Median                      1.300000
Mode                        0.200000
CI                               NaN

                                                           95% CI Min  \
Mean                                                              NaN
Standard Deviation                                                NaN
Median                                                            NaN
Mode                                                              NaN
CI                  [5.709732481507367, 2.9870103180785437, 3.4731...

                                                           95% CI Max
Mean                                                              NaN
Standard Deviation                                                NaN
Median                                                            NaN
Mode                                                              NaN
CI                  [5.976934185159302, 3.1276563485881246, 4.0428...
```

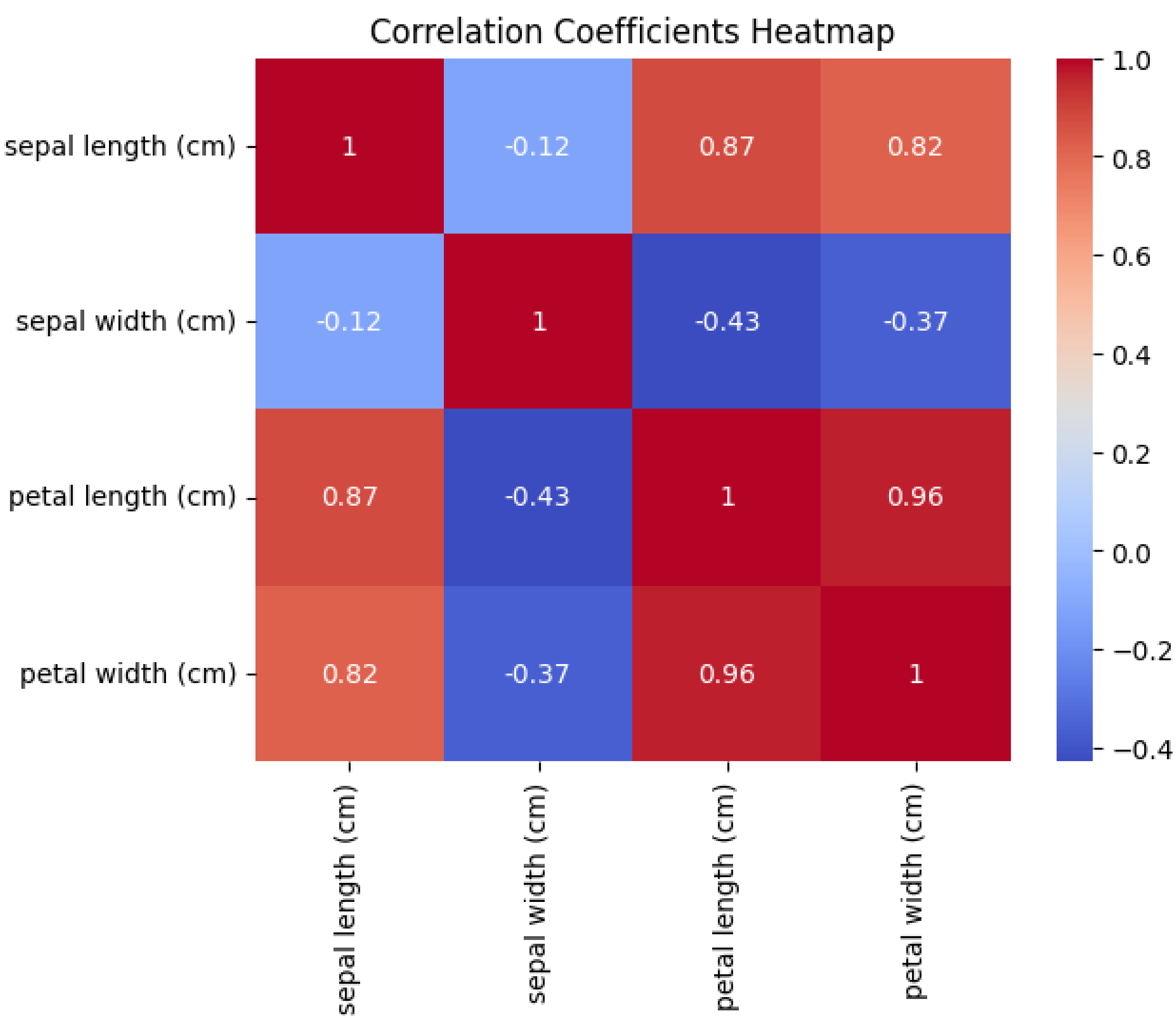
g. Compute correlation coefficients between each pair of features and plot heatmap

Code:

```
correlation_matrix = data.corr()
print(correlation_matrix)
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title("Correlation Coefficients Heatmap")
plt.show()
```

Output:

```
                   sepal length (cm)  sepal width (cm)  petal length (cm)  \
sepal length (cm)           1.000000         -0.117570           0.871754
sepal width (cm)           -0.117570          1.000000          -0.428440
petal length (cm)           0.871754         -0.428440           1.000000
petal width (cm)            0.817941         -0.366126           0.962865

                   petal width (cm)
sepal length (cm)          0.817941
sepal width (cm)          -0.366126
petal length (cm)          0.962865
petal width (cm)           1.000000
```



Q6. Consider the following data frame containing a family name, gender of the family member and her/his monthly income in each record.

Name	Gender	MonthlyIncome (Rs.)
Shah	Male	114000.00
Vats	Male	65000.00
Vats	Female	43150.00
Kumar	Female	69500.00
Vats	Female	155000.00
Kumar	Male	103000.00
Shah	Male	55000.00
Shah	Female	112400.00
Kumar	Female	81030.00
Vats	Male	71900.00

Code: (Creating the required data frame)

```
import numpy as np
import pandas as pd
df = pd.read_csv("family.csv")
print(df)
```

Output:

```
    Name  Gender  MonthlyIncome
0   Shah    Male       114000.0
1   Vats    Male        65000.0
2   Vats  Female        43150.0
3  Kumar  Female        69500.0
4   Vats  Female       155000.0
5  Kumar    Male       103000.0
6   Shah    Male        55000.0
7   Shah  Female       112400.0
8  Kumar  Female        81030.0
9   Vats    Male        71900.0
```

Write a program in Python using Pandas to perform the following:

a. Calculate and display familywise gross monthly income.

Code:

```
print("The familywise gross monthly income is:")
df.groupby(['Name'])['MonthlyIncome'].sum()
```

Output:

```
The familywise gross monthly income is:
Name
Kumar    253530.0
Shah     281400.0
Vats     335050.0
Name: MonthlyIncome, dtype: float64
```

b. Calculate and display the member with the highest monthly income.

Code:

```
print("Highest monthly income in each family:")
df.groupby(['Name'])['MonthlyIncome'].max()
```

Output:

```
Highest monthly income in each family:
Name
Kumar    103000.0
Shah     114000.0
Vats     155000.0
Name: MonthlyIncome, dtype: float64
```

c. Calculate and display monthly income of all members with income greater than Rs. 60000.00.

Code:

```
print("Members with monthly income more than Rs. 60000:")
df[df["MonthlyIncome"]>60000]
```

Output:

```
Members with monthly income more than Rs. 60000:
```

	Name	Gender	MonthlyIncome
0	Shah	Male	114000.0
1	Vats	Male	65000.0
3	Kumar	Female	69500.0
4	Vats	Female	155000.0
5	Kumar	Male	103000.0
7	Shah	Female	112400.0
8	Kumar	Female	81030.0
9	Vats	Male	71900.0

d. Calculate and display the average monthly income of the female members

Code:

```
print("The average monthly income of the female members is Rs.",
      df[df["Gender"]=="Female"]["MonthlyIncome"].mean())
```

Output:

```
The average monthly income of the female members is Rs. 92216.0
```

Q7. Using Titanic dataset, to do the following:

Code: (Importing **seaborn** and loading TITANIC data)

```
import seaborn as sns
sns.set_style("whitegrid")
titanic = sns.load_dataset("titanic")
titanic
```

Output:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns

a. Find total number of passengers with age less than 30

Code:

```
print("There are", sum(titanic.age < 30), "passengers under the age of 30.")
```

Output:

```
There are 384 passengers under the age of 30.
```

b. Find total fare paid by passengers of first class

Code:

```
print("Total fare paid by first class passengers:", titanic[titanic.pclass == 1].fare.sum())
```

Output:

```
Total fare paid by first class passengers: 18177.4125
```

c. Compare number of survivors of each passenger class

Code:

```
print("Number of survivors of each passenger class")
titanic.groupby("pclass").survived.sum()
```

Output:

```
Number of survivors of each passenger class
pclass
1    136
2     87
3    119
Name: survived, dtype: int64
```

d. Compute descriptive statistics for any numeric attribute genderwise

Code:

```
print("Genderwise age descriptive statistics:")
titanic.groupby("sex").age.describe()
```

Output:

```
Genderwise age descriptive statistics:
```

sex	count	mean	std	min	25%	50%	75%	max
female	261.0	27.915709	14.110146	0.75	18.0	27.0	37.0	63.0
male	453.0	30.726645	14.678201	0.42	21.0	29.0	39.0	80.0