practical

December 10, 2023

[ ]:

1 Practical Question No. 1

Q1: Given below is a dictionary having two keys 'Boys' and 'Girls' and having two lists of heights of five Boys and Five Girls respectively as values associated with these keys, Original dictionary of lists: {'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65, 69, 62, 61]}
From the given dictionary of lists create the following list of dictionaries:
[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls': 65}, {'Boys': 70, 'Girls': 69}, {'Boys': 69, 'Girls': 62}, {'Boys':74, 'Girls':61]

[1]:
```
dict = {'Boys':[72,68,70,69,74],'Girls':[63,65,69,62,61]}
new_list=[]
for b,g in zip( dict['Boys'], dict['Girls']):
    new_list.append({'Boys':b, 'Girls':g})
print(new_list)
```

```
[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls': 65}, {'Boys': 70, 'Girls':
69}, {'Boys': 69, 'Girls': 62}, {'Boys': 74, 'Girls': 61}]
```

[2]:
```
#Another Method
dict = {'Boys':[72,68,70,69,74],'Girls':[63,65,69,62,61]}
a=dict['Boys']
c=dict['Girls']
l=[]
for i in range(0,len(c)):
    l.append({'Boys':a[i],'Girls':c[i]})
print(l)
```

```
[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls': 65}, {'Boys': 70, 'Girls':
69}, {'Boys': 69, 'Girls': 62}, {'Boys': 74, 'Girls': 61}]
```

[ ]:

[ ]:

2 Practical Question No. 2

Q2: 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.
b. Get the indices of the sorted elements of given array. a. B = [56, 48, 22, 41, 78, 91, 24, 46, 8, 33]
c. 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 nx m array, n and m are user inputs given at the run time.
d. Test whether the elements of a given array are zero, non-zero and NaN. Record the indices of these elements in three separate arrays.

[3]:
```
import numpy as np
```

[4]:
```
#np.random.seed(123)
arr= np.random.randint(1,20,16).reshape(4,4)
print(arr)
print(arr.shape)
```

```
[[17  2 11 15]
 [12  6 18  9]
 [10  3  4  4]
 [ 4 17  7 16]]
(4, 4)
```

[5]:
```
# a) calculate Mean, Variance and standard deviation
print(f"Mean: {arr.mean(axis = 1)}")# 1 for column,but select row wise
print(arr.sum()/16)
print(arr.std(axis = 1))# 1 for column,but select row wise
print(arr.var(axis=1))
```

```
Mean: [11.25 11.25  5.25 11.  ]
9.6875
[5.76085931 4.43705984 2.77263413 5.61248608]
[33.1875 19.6875  7.6875 31.5   ]
```

[6]:
```
# b) get indices of stored element in array B.
arr2 = np.array([56,48,22,41,78,91,24,46,8,33])
print(arr2)
print(arr2.argsort())
```

```
[56 48 22 41 78 91 24 46  8 33]
[8 2 6 9 3 7 1 0 4 5]
```

[7]:
```
# c) create 2D array by user and perform operations.
m= int(input("Enter m: "))
n= int(input("Enter n: "))
```

```
print(f"Enter element of {m},{n} matrix: ")
arr3= []
for i in range(0 , m ):
    a=[]
    for j in range(0 , n):
        a.append(int(input()))
    arr3.append(a)
arr3 = np.array(arr3)
```

```
Enter m: 3
Enter n: 2
Enter element of 3,2 matrix:
1
2
3
4
5
6
```

[8]:
```
arr3= np.array(arr3)
print(arr3)
print(arr3.shape)
print(arr3.astype)
print(arr3.dtype)
print(arr3.reshape(n,m))
print(arr3.reshape(n,m).shape)
```

```
[[1 2]
 [3 4]
 [5 6]]
(3, 2)
<built-in method astype of numpy.ndarray object at 0x000001703D2CA130>
int32
[[1 2 3]
 [4 5 6]]
(2, 3)
```

[9]:
```
# d) Test and record indices of the elements of array are zero, non-zero and␣
  ↪NaN.
array = np.array([1,2,3,0,45,0,np.NaN,34,np.NaN,0,90])
indices = np.arange(array.size)
zero_array = np.where(array ==0)
print(zero_array)
NaN_array = np.where(np.isnan(array))
print(NaN_array)
Nonzero_array = np.where((array!=0) & (~np.isnan(array)))
print(Nonzero_array)
```

```
(array([3, 5, 9], dtype=int64),)
(array([6, 8], dtype=int64),)
(array([ 0,  1,  2,  4,  7, 10], dtype=int64),)
```

[10]:
```
# 2-D array indexes
arr3= np.array([[1,2,3],[4,0,0],[np.nan,np.nan,5]])
indices= np.arange(array.size)
zero_indices = np.where(arr3 == 0)
nonzero_indices = np.where(arr3 != 0)
nan_indices = np.where(np.isnan(arr3))
print("Original array: ",arr3)
print("Index of Zero element: ",zero_indices)
print("Index of Non-Zero element: ",nonzero_indices)
print("Index of Nan element: ",nan_indices)
```

```
Original array:  [[ 1.  2.  3.]
 [ 4.  0.  0.]
 [nan nan  5.]]
Index of Zero element:  (array([1, 1], dtype=int64), array([1, 2], dtype=int64))
Index of Non-Zero element:  (array([0, 0, 0, 1, 2, 2, 2], dtype=int64),
array([0, 1, 2, 0, 0, 1, 2], dtype=int64))
Index of Nan element:  (array([2, 2], dtype=int64), array([0, 1], dtype=int64))
```

[ ]:

[ ]:

3 Practical Question No.3

Q3: Create a dataframe 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. Do the following:

a. Identify and count missing values in a dataframe.

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

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

d. Sort the dataframe on the basis of the first column.

e. Remove all duplicates from the first column.

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

g. Detect the outliers and remove the rows having outliers.

h. Discretize second column and create 5 bins

[11]:
```
import numpy as np
import pandas as pd
```

[12]:
```
#np.random.seed(0)
data = np.random.randn(50,3)
df= pd.DataFrame(data, columns=["col_1","col_2","col_3"])
print(df)
```

```
       col_1     col_2     col_3
0  -0.750709  0.208758 -0.125483
1   0.459216 -0.240304 -0.104679
2  -0.858515 -0.513321 -0.074659
3   0.096007  1.192015 -0.866090
4   0.446627 -0.455601  0.097443
5  -0.762498  0.151789 -0.639029
6  -1.956194 -0.260461 -0.169400
7   0.789727  0.649513  0.632190
8  -0.047023 -0.406263  0.197463
9  -0.160148 -1.366876  2.324615
10 -0.710671  0.894266  2.080256
11 -0.715697  1.238808  0.166821
12  0.301475 -0.722795  0.284083
13 -0.359478 -0.732963 -0.153790
14 -0.248535  0.327528  1.232972
15 -2.150820  1.574903  0.916579
16 -1.147021 -0.891992  1.186074
17  1.327575  2.891202  0.546870
18 -0.420230 -1.067328 -1.151118
19 -0.650511  1.461089 -0.008172
20 -0.430567 -0.379547 -1.432214
21  0.053915 -1.642026 -0.205310
22 -2.476917  1.705954  0.434852
23  2.587359 -0.329640  1.006869
24 -0.488339 -0.370721  0.124189
25  1.980738  0.441037  2.031571
26 -1.404242  0.072790  1.060882
27 -0.461829 -2.592565 -0.335051
28  0.789692 -1.438032 -0.170011
29  0.160528 -0.392204 -0.113666
30  0.495617  0.829839  0.078042
31  0.708170  0.314424 -0.023001
32  0.188483  0.549842  1.119011
33 -1.159368  0.321084 -0.952141
34 -1.770784 -1.335341  0.703457
35 -0.676012 -0.112984  1.832119
36  0.925124  1.524849 -1.264252
```

```
37 -0.436553 -1.254383 -0.804968
38 -0.084453  0.587786 -0.374940
39  0.037131 -0.765057 -0.607043
40  0.335446  0.404526  1.143217
41  0.701059 -0.451315  1.327836
42 -0.474245  0.734476 -0.501736
43  0.051953 -1.462614 -2.608745
44  1.252329 -0.089442 -1.473156
45 -0.210475  0.776516  1.241877
46 -0.665587  0.552967  0.237734
47  1.090242  0.516434 -0.746425
48 -2.123700 -1.138817 -1.004661
49  0.158125  2.923560  0.828388
```

[13]:
```
# create a dataframe with 10 percent missing values
null_indices = np.random.choice(50 * 3, size=int(0.1 * 50 * 3))
df.values.ravel()[null_indices]=np.nan
print(df)
```

```
       col_1     col_2     col_3
0  -0.750709  0.208758 -0.125483
1   0.459216 -0.240304 -0.104679
2  -0.858515 -0.513321 -0.074659
3   0.096007  1.192015 -0.866090
4   0.446627 -0.455601  0.097443
5  -0.762498  0.151789 -0.639029
6  -1.956194 -0.260461       NaN
7   0.789727  0.649513  0.632190
8  -0.047023 -0.406263  0.197463
9  -0.160148 -1.366876  2.324615
10 -0.710671  0.894266  2.080256
11 -0.715697  1.238808  0.166821
12       NaN -0.722795  0.284083
13 -0.359478 -0.732963 -0.153790
14 -0.248535       NaN  1.232972
15 -2.150820  1.574903       NaN
16 -1.147021 -0.891992  1.186074
17  1.327575  2.891202  0.546870
18 -0.420230 -1.067328 -1.151118
19 -0.650511  1.461089 -0.008172
20 -0.430567 -0.379547 -1.432214
21  0.053915 -1.642026 -0.205310
22       NaN  1.705954  0.434852
23  2.587359 -0.329640  1.006869
24 -0.488339 -0.370721  0.124189
25  1.980738  0.441037  2.031571
26 -1.404242  0.072790  1.060882
27 -0.461829 -2.592565 -0.335051
```

```
28       NaN -1.438032 -0.170011
29  0.160528       NaN -0.113666
30  0.495617       NaN  0.078042
31  0.708170  0.314424       NaN
32  0.188483  0.549842  1.119011
33 -1.159368  0.321084 -0.952141
34 -1.770784 -1.335341       NaN
35 -0.676012       NaN  1.832119
36  0.925124  1.524849 -1.264252
37 -0.436553 -1.254383 -0.804968
38 -0.084453  0.587786 -0.374940
39       NaN -0.765057 -0.607043
40       NaN  0.404526  1.143217
41  0.701059 -0.451315  1.327836
42 -0.474245  0.734476 -0.501736
43  0.051953 -1.462614 -2.608745
44  1.252329 -0.089442 -1.473156
45 -0.210475  0.776516  1.241877
46       NaN  0.552967  0.237734
47  1.090242  0.516434 -0.746425
48 -2.123700 -1.138817 -1.004661
49       NaN  2.923560  0.828388
```

[14]:
```
# a) print count of missing values
miss_df= df.isnull()
print(miss_df)


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

```
    col_1  col_2  col_3
0   False  False  False
1   False  False  False
2   False  False  False
3   False  False  False
4   False  False  False
5   False  False  False
6   False  False   True
7   False  False  False
8   False  False  False
9   False  False  False
10  False  False  False
11  False  False  False
12   True  False  False
13  False  False  False
14  False   True  False
15  False  False   True
```

```
16  False  False  False
17  False  False  False
18  False  False  False
19  False  False  False
20  False  False  False
21  False  False  False
22   True  False  False
23  False  False  False
24  False  False  False
25  False  False  False
26  False  False  False
27  False  False  False
28   True  False  False
29  False   True  False
30  False   True  False
31  False  False   True
32  False  False  False
33  False  False  False
34  False  False   True
35  False   True  False
36  False  False  False
37  False  False  False
38  False  False  False
39   True  False  False
40   True  False  False
41  False  False  False
42  False  False  False
43  False  False  False
44  False  False  False
45  False  False  False
46   True  False  False
47  False  False  False
48  False  False  False
49   True  False  False
col_1    7
col_2    4
col_3    4
dtype: int64
```

[15]:
```
# b) drop columns haveing greate than 5 missing values
df2 = df.drop(df.columns[df.isnull().sum() > 5], axis = 1)
print(df2)
```

```
      col_2     col_3
0   0.208758 -0.125483
1  -0.240304 -0.104679
2  -0.513321 -0.074659
3   1.192015 -0.866090
```

```
4  -0.455601  0.097443
5   0.151789 -0.639029
6  -0.260461       NaN
7   0.649513  0.632190
8  -0.406263  0.197463
9  -1.366876  2.324615
10  0.894266  2.080256
11  1.238808  0.166821
12 -0.722795  0.284083
13 -0.732963 -0.153790
14       NaN  1.232972
15  1.574903       NaN
16 -0.891992  1.186074
17  2.891202  0.546870
18 -1.067328 -1.151118
19  1.461089 -0.008172
20 -0.379547 -1.432214
21 -1.642026 -0.205310
22  1.705954  0.434852
23 -0.329640  1.006869
24 -0.370721  0.124189
25  0.441037  2.031571
26  0.072790  1.060882
27 -2.592565 -0.335051
28 -1.438032 -0.170011
29       NaN -0.113666
30       NaN  0.078042
31  0.314424       NaN
32  0.549842  1.119011
33  0.321084 -0.952141
34 -1.335341       NaN
35       NaN  1.832119
36  1.524849 -1.264252
37 -1.254383 -0.804968
38  0.587786 -0.374940
39 -0.765057 -0.607043
40  0.404526  1.143217
41 -0.451315  1.327836
42  0.734476 -0.501736
43 -1.462614 -2.608745
44 -0.089442 -1.473156
45  0.776516  1.241877
46  0.552967  0.237734
47  0.516434 -0.746425
48 -1.138817 -1.004661
49  2.923560  0.828388
```

[16]:
```
# c) Identify the row label having maximum of the sum of all values in a row␣
  ↪and drop that row.
print(df.sum(axis = 1))
print(f"Maximum Sum (row-wise): {max(df.sum(axis = 1))}   in row no.= {df.
  ↪sum(axis=1).idxmax()}\n")

# to drop the row having max sum
print(df.drop([(df.sum(axis=1)).idxmax()], axis = 0))
```

```
0    -0.667435
1     0.114233
2    -1.446495
3     0.421932
4     0.088468
5    -1.249738
6    -2.216655
7     2.071430
8    -0.255823
9     0.797592
10    2.263851
11    0.689931
12   -0.438712
13   -1.246231
14    0.984436
15   -0.575917
16   -0.852938
17    4.765646
18   -2.638676
19    0.802406
20   -2.242328
21   -1.793421
22    2.140806
23    3.264588
24   -0.734871
25    4.453346
26   -0.270569
27   -3.389445
28   -1.608044
29    0.046862
30    0.573659
31    1.022594
32    1.857337
33   -1.790426
34   -3.106125
35    1.156107
36    1.185721
37   -2.495904
```

```
38    0.128393
39   -1.372100
40    1.547743
41    1.577580
42   -0.241505
43   -4.019406
44   -0.310268
45    1.807918
46    0.790702
47    0.860251
48   -4.267178
49    3.751948
dtype: float64
Maximum Sum (row-wise): 4.765646343142654   in row no.= 17

       col_1     col_2     col_3
0  -0.750709  0.208758 -0.125483
1   0.459216 -0.240304 -0.104679
2  -0.858515 -0.513321 -0.074659
3   0.096007  1.192015 -0.866090
4   0.446627 -0.455601  0.097443
5  -0.762498  0.151789 -0.639029
6  -1.956194 -0.260461       NaN
7   0.789727  0.649513  0.632190
8  -0.047023 -0.406263  0.197463
9  -0.160148 -1.366876  2.324615
10 -0.710671  0.894266  2.080256
11 -0.715697  1.238808  0.166821
12       NaN -0.722795  0.284083
13 -0.359478 -0.732963 -0.153790
14 -0.248535       NaN  1.232972
15 -2.150820  1.574903       NaN
16 -1.147021 -0.891992  1.186074
18 -0.420230 -1.067328 -1.151118
19 -0.650511  1.461089 -0.008172
20 -0.430567 -0.379547 -1.432214
21  0.053915 -1.642026 -0.205310
22       NaN  1.705954  0.434852
23  2.587359 -0.329640  1.006869
24 -0.488339 -0.370721  0.124189
25  1.980738  0.441037  2.031571
26 -1.404242  0.072790  1.060882
27 -0.461829 -2.592565 -0.335051
28       NaN -1.438032 -0.170011
29  0.160528       NaN -0.113666
30  0.495617       NaN  0.078042
31  0.708170  0.314424       NaN
32  0.188483  0.549842  1.119011
```

```
33 -1.159368  0.321084 -0.952141
34 -1.770784 -1.335341       NaN
35 -0.676012       NaN  1.832119
36  0.925124  1.524849 -1.264252
37 -0.436553 -1.254383 -0.804968
38 -0.084453  0.587786 -0.374940
39       NaN -0.765057 -0.607043
40       NaN  0.404526  1.143217
41  0.701059 -0.451315  1.327836
42 -0.474245  0.734476 -0.501736
43  0.051953 -1.462614 -2.608745
44  1.252329 -0.089442 -1.473156
45 -0.210475  0.776516  1.241877
46       NaN  0.552967  0.237734
47  1.090242  0.516434 -0.746425
48 -2.123700 -1.138817 -1.004661
49       NaN  2.923560  0.828388
```

[17]:
```
# d) sort the datafram in asc order based on column 1
df_new = df.sort_values(by = 'col_1', ascending = True)
print(df_new)
```

```
       col_1     col_2     col_3
15 -2.150820  1.574903       NaN
48 -2.123700 -1.138817 -1.004661
6  -1.956194 -0.260461       NaN
34 -1.770784 -1.335341       NaN
26 -1.404242  0.072790  1.060882
33 -1.159368  0.321084 -0.952141
16 -1.147021 -0.891992  1.186074
2  -0.858515 -0.513321 -0.074659
5  -0.762498  0.151789 -0.639029
0  -0.750709  0.208758 -0.125483
11 -0.715697  1.238808  0.166821
10 -0.710671  0.894266  2.080256
35 -0.676012       NaN  1.832119
19 -0.650511  1.461089 -0.008172
24 -0.488339 -0.370721  0.124189
42 -0.474245  0.734476 -0.501736
27 -0.461829 -2.592565 -0.335051
37 -0.436553 -1.254383 -0.804968
20 -0.430567 -0.379547 -1.432214
18 -0.420230 -1.067328 -1.151118
13 -0.359478 -0.732963 -0.153790
14 -0.248535       NaN  1.232972
45 -0.210475  0.776516  1.241877
9  -0.160148 -1.366876  2.324615
38 -0.084453  0.587786 -0.374940
```

```
8  -0.047023 -0.406263  0.197463
43  0.051953 -1.462614 -2.608745
21  0.053915 -1.642026 -0.205310
3   0.096007  1.192015 -0.866090
29  0.160528       NaN -0.113666
32  0.188483  0.549842  1.119011
4   0.446627 -0.455601  0.097443
1   0.459216 -0.240304 -0.104679
30  0.495617       NaN  0.078042
41  0.701059 -0.451315  1.327836
31  0.708170  0.314424       NaN
7   0.789727  0.649513  0.632190
36  0.925124  1.524849 -1.264252
47  1.090242  0.516434 -0.746425
44  1.252329 -0.089442 -1.473156
17  1.327575  2.891202  0.546870
25  1.980738  0.441037  2.031571
23  2.587359 -0.329640  1.006869
12       NaN -0.722795  0.284083
22       NaN  1.705954  0.434852
28       NaN -1.438032 -0.170011
39       NaN -0.765057 -0.607043
40       NaN  0.404526  1.143217
46       NaN  0.552967  0.237734
49       NaN  2.923560  0.828388
```

[18]:
```
# e) Remove duplicate from 1st column.
df.drop_duplicates(subset = 'col_1')
```

[18]:
```
       col_1     col_2     col_3
0  -0.750709  0.208758 -0.125483
1   0.459216 -0.240304 -0.104679
2  -0.858515 -0.513321 -0.074659
3   0.096007  1.192015 -0.866090
4   0.446627 -0.455601  0.097443
5  -0.762498  0.151789 -0.639029
6  -1.956194 -0.260461       NaN
7   0.789727  0.649513  0.632190
8  -0.047023 -0.406263  0.197463
9  -0.160148 -1.366876  2.324615
10 -0.710671  0.894266  2.080256
11 -0.715697  1.238808  0.166821
12       NaN -0.722795  0.284083
13 -0.359478 -0.732963 -0.153790
14 -0.248535       NaN  1.232972
15 -2.150820  1.574903       NaN
16 -1.147021 -0.891992  1.186074
```

```
17  1.327575  2.891202  0.546870
18 -0.420230 -1.067328 -1.151118
19 -0.650511  1.461089 -0.008172
20 -0.430567 -0.379547 -1.432214
21  0.053915 -1.642026 -0.205310
23  2.587359 -0.329640  1.006869
24 -0.488339 -0.370721  0.124189
25  1.980738  0.441037  2.031571
26 -1.404242  0.072790  1.060882
27 -0.461829 -2.592565 -0.335051
29  0.160528       NaN -0.113666
30  0.495617       NaN  0.078042
31  0.708170  0.314424       NaN
32  0.188483  0.549842  1.119011
33 -1.159368  0.321084 -0.952141
34 -1.770784 -1.335341       NaN
35 -0.676012       NaN  1.832119
36  0.925124  1.524849 -1.264252
37 -0.436553 -1.254383 -0.804968
38 -0.084453  0.587786 -0.374940
41  0.701059 -0.451315  1.327836
42 -0.474245  0.734476 -0.501736
43  0.051953 -1.462614 -2.608745
44  1.252329 -0.089442 -1.473156
45 -0.210475  0.776516  1.241877
47  1.090242  0.516434 -0.746425
48 -2.123700 -1.138817 -1.004661
```

[19]:
```
# f) find covariance between first column and second column
correlation = df['col_1'].corr(df['col_2'])
covariance = df['col_2'].cov(df['col_3'])
print(correlation)
print(covariance)
```

```
0.19377266926119407
0.2519017829652975
```

[20]:
```
# g) detect and remove outliers
data=df[~(np.abs(data) > 3).any(axis=1)]
print(data)
```

```
      col_1     col_2     col_3
0  -0.750709  0.208758 -0.125483
1   0.459216 -0.240304 -0.104679
2  -0.858515 -0.513321 -0.074659
3   0.096007  1.192015 -0.866090
4   0.446627 -0.455601  0.097443
5  -0.762498  0.151789 -0.639029
```

```
6  -1.956194 -0.260461       NaN
7   0.789727  0.649513  0.632190
8  -0.047023 -0.406263  0.197463
9  -0.160148 -1.366876  2.324615
10 -0.710671  0.894266  2.080256
11 -0.715697  1.238808  0.166821
12       NaN -0.722795  0.284083
13 -0.359478 -0.732963 -0.153790
14 -0.248535       NaN  1.232972
15 -2.150820  1.574903       NaN
16 -1.147021 -0.891992  1.186074
17  1.327575  2.891202  0.546870
18 -0.420230 -1.067328 -1.151118
19 -0.650511  1.461089 -0.008172
20 -0.430567 -0.379547 -1.432214
21  0.053915 -1.642026 -0.205310
22       NaN  1.705954  0.434852
23  2.587359 -0.329640  1.006869
24 -0.488339 -0.370721  0.124189
25  1.980738  0.441037  2.031571
26 -1.404242  0.072790  1.060882
27 -0.461829 -2.592565 -0.335051
28       NaN -1.438032 -0.170011
29  0.160528       NaN -0.113666
30  0.495617       NaN  0.078042
31  0.708170  0.314424       NaN
32  0.188483  0.549842  1.119011
33 -1.159368  0.321084 -0.952141
34 -1.770784 -1.335341       NaN
35 -0.676012       NaN  1.832119
36  0.925124  1.524849 -1.264252
37 -0.436553 -1.254383 -0.804968
38 -0.084453  0.587786 -0.374940
39       NaN -0.765057 -0.607043
40       NaN  0.404526  1.143217
41  0.701059 -0.451315  1.327836
42 -0.474245  0.734476 -0.501736
43  0.051953 -1.462614 -2.608745
44  1.252329 -0.089442 -1.473156
45 -0.210475  0.776516  1.241877
46       NaN  0.552967  0.237734
47  1.090242  0.516434 -0.746425
48 -2.123700 -1.138817 -1.004661
49       NaN  2.923560  0.828388
```

[21]:
```
## Another method
z_scores = (df - df_new.mean()) / df.std()
```

```
print(z_scores)
df_new = df[(z_scores.abs() < 3).all(axis=1)]
print(df)
```

```
       col_1     col_2     col_3
0  -0.572494  0.148562 -0.235900
1   0.621938 -0.243786 -0.215869
2  -0.678919 -0.482322 -0.186965
3   0.263380  1.007639 -0.948983
4   0.609510 -0.431893 -0.021259
5  -0.584132  0.098788 -0.730360
6  -1.762542 -0.261398       NaN
7   0.948217  0.533653  0.493614
8   0.122182 -0.388785  0.075044
9   0.010506 -1.228078  2.123142
10 -0.532968  0.747494  1.887864
11 -0.537930  1.048523  0.045540
12       NaN -0.665342  0.158445
13 -0.186272 -0.674225 -0.263155
14 -0.076750       NaN  1.072069
15 -1.954676  1.342171       NaN
16 -0.963730 -0.813170  1.026914
17  1.479177  2.492229  0.411465
18 -0.246247 -0.966361 -1.223418
19 -0.473578  1.242731 -0.122949
20 -0.256451 -0.365443 -1.494068
21  0.221827 -1.468479 -0.312760
22       NaN  1.456671  0.303611
23  2.722830 -0.321840  0.854369
24 -0.313483 -0.357732  0.004493
25  2.123977  0.351505  1.840989
26 -1.217658  0.029766  0.906375
27 -0.287312 -2.298971 -0.437680
28       NaN -1.290248 -0.278774
29  0.327075       NaN -0.224522
30  0.657873       NaN -0.039939
31  0.867704  0.240883       NaN
32  0.354672  0.446569  0.962344
33 -0.975920  0.246702 -1.031836
34 -1.579506 -1.200527       NaN
35 -0.498753       NaN  1.648949
36  1.081880  1.298438 -1.332347
37 -0.262361 -1.129793 -0.890132
38  0.085231  0.479721 -0.476087
39       NaN -0.702266 -0.699563
40       NaN  0.319606  0.985650
41  0.860684 -0.428148  1.163407
```

```
42 -0.299570  0.607885 -0.598170
43  0.219890 -1.311725 -2.626873
44  1.404895 -0.111977 -1.533488
45 -0.039177  0.644616  1.080643
46       NaN  0.449300  0.113819
47  1.244883  0.417380 -0.833765
48 -1.927903 -1.028822 -1.082404
49       NaN  2.520500  0.682521
       col_1     col_2     col_3
0  -0.750709  0.208758 -0.125483
1   0.459216 -0.240304 -0.104679
2  -0.858515 -0.513321 -0.074659
3   0.096007  1.192015 -0.866090
4   0.446627 -0.455601  0.097443
5  -0.762498  0.151789 -0.639029
6  -1.956194 -0.260461       NaN
7   0.789727  0.649513  0.632190
8  -0.047023 -0.406263  0.197463
9  -0.160148 -1.366876  2.324615
10 -0.710671  0.894266  2.080256
11 -0.715697  1.238808  0.166821
12       NaN -0.722795  0.284083
13 -0.359478 -0.732963 -0.153790
14 -0.248535       NaN  1.232972
15 -2.150820  1.574903       NaN
16 -1.147021 -0.891992  1.186074
17  1.327575  2.891202  0.546870
18 -0.420230 -1.067328 -1.151118
19 -0.650511  1.461089 -0.008172
20 -0.430567 -0.379547 -1.432214
21  0.053915 -1.642026 -0.205310
22       NaN  1.705954  0.434852
23  2.587359 -0.329640  1.006869
24 -0.488339 -0.370721  0.124189
25  1.980738  0.441037  2.031571
26 -1.404242  0.072790  1.060882
27 -0.461829 -2.592565 -0.335051
28       NaN -1.438032 -0.170011
29  0.160528       NaN -0.113666
30  0.495617       NaN  0.078042
31  0.708170  0.314424       NaN
32  0.188483  0.549842  1.119011
33 -1.159368  0.321084 -0.952141
34 -1.770784 -1.335341       NaN
35 -0.676012       NaN  1.832119
36  0.925124  1.524849 -1.264252
37 -0.436553 -1.254383 -0.804968
38 -0.084453  0.587786 -0.374940
```

```
39       NaN -0.765057 -0.607043
40       NaN  0.404526  1.143217
41  0.701059 -0.451315  1.327836
42 -0.474245  0.734476 -0.501736
43  0.051953 -1.462614 -2.608745
44  1.252329 -0.089442 -1.473156
45 -0.210475  0.776516  1.241877
46       NaN  0.552967  0.237734
47  1.090242  0.516434 -0.746425
48 -2.123700 -1.138817 -1.004661
49       NaN  2.923560  0.828388
```

[22]:
```
# h) Discretize second column and create 5 bins
# df1= df.dropna()
df1= df
df1['col_2_Bin'] = pd.cut(df1['col_2'], bins=5)
print(df1)
```

```
       col_1     col_2     col_3         col_2_Bin
0  -0.750709  0.208758 -0.125483   (-0.386, 0.717]
1   0.459216 -0.240304 -0.104679   (-0.386, 0.717]
2  -0.858515 -0.513321 -0.074659  (-1.489, -0.386]
3   0.096007  1.192015 -0.866090     (0.717, 1.82]
4   0.446627 -0.455601  0.097443  (-1.489, -0.386]
5  -0.762498  0.151789 -0.639029   (-0.386, 0.717]
6  -1.956194 -0.260461       NaN   (-0.386, 0.717]
7   0.789727  0.649513  0.632190   (-0.386, 0.717]
8  -0.047023 -0.406263  0.197463  (-1.489, -0.386]
9  -0.160148 -1.366876  2.324615  (-1.489, -0.386]
10 -0.710671  0.894266  2.080256     (0.717, 1.82]
11 -0.715697  1.238808  0.166821     (0.717, 1.82]
12       NaN -0.722795  0.284083  (-1.489, -0.386]
13 -0.359478 -0.732963 -0.153790  (-1.489, -0.386]
14 -0.248535       NaN  1.232972               NaN
15 -2.150820  1.574903       NaN     (0.717, 1.82]
16 -1.147021 -0.891992  1.186074  (-1.489, -0.386]
17  1.327575  2.891202  0.546870     (1.82, 2.924]
18 -0.420230 -1.067328 -1.151118  (-1.489, -0.386]
19 -0.650511  1.461089 -0.008172     (0.717, 1.82]
20 -0.430567 -0.379547 -1.432214   (-0.386, 0.717]
21  0.053915 -1.642026 -0.205310  (-2.598, -1.489]
22       NaN  1.705954  0.434852     (0.717, 1.82]
23  2.587359 -0.329640  1.006869   (-0.386, 0.717]
24 -0.488339 -0.370721  0.124189   (-0.386, 0.717]
25  1.980738  0.441037  2.031571   (-0.386, 0.717]
26 -1.404242  0.072790  1.060882   (-0.386, 0.717]
27 -0.461829 -2.592565 -0.335051  (-2.598, -1.489]
28       NaN -1.438032 -0.170011  (-1.489, -0.386]
```

```
29  0.160528       NaN -0.113666               NaN
30  0.495617       NaN  0.078042               NaN
31  0.708170  0.314424       NaN   (-0.386, 0.717]
32  0.188483  0.549842  1.119011   (-0.386, 0.717]
33 -1.159368  0.321084 -0.952141   (-0.386, 0.717]
34 -1.770784 -1.335341       NaN  (-1.489, -0.386]
35 -0.676012       NaN  1.832119               NaN
36  0.925124  1.524849 -1.264252     (0.717, 1.82]
37 -0.436553 -1.254383 -0.804968  (-1.489, -0.386]
38 -0.084453  0.587786 -0.374940   (-0.386, 0.717]
39       NaN -0.765057 -0.607043  (-1.489, -0.386]
40       NaN  0.404526  1.143217   (-0.386, 0.717]
41  0.701059 -0.451315  1.327836  (-1.489, -0.386]
42 -0.474245  0.734476 -0.501736     (0.717, 1.82]
43  0.051953 -1.462614 -2.608745  (-1.489, -0.386]
44  1.252329 -0.089442 -1.473156   (-0.386, 0.717]
45 -0.210475  0.776516  1.241877     (0.717, 1.82]
46       NaN  0.552967  0.237734   (-0.386, 0.717]
47  1.090242  0.516434 -0.746425   (-0.386, 0.717]
48 -2.123700 -1.138817 -1.004661  (-1.489, -0.386]
49       NaN  2.923560  0.828388     (1.82, 2.924]
```

[ ]:

Practical Question No. 4

Q4: Consider two excel files having attendance of a workshop's participants for two days. Each file has three fields 'Name', 'Time of joining', 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 dataframes and do the following:

a. Perform merging of the two dataframes to find the names of students who had attended the workshop on both days.

b. Find names of all students who have attended workshop on either of the days.

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

d. Merge two data frames and use two columns names and duration as multi-row indexes. Generate descriptive statistics for this multi-index.

[23]:
```
import numpy as np
import pandas as pd
```

[24]:
```
df1 = pd.read_excel('attendance.xlsx', "Day_1")
df2 = pd.read_excel('attendance.xlsx', "Day_2")
```

[25]:
```
print(df1,"\n")
print(df2)
```

```
             Name Time of Joining  Duration
0          Hemant         13:30:00        50
1    Gautam Meena         13:45:15        40
2   Rishabh Kumar         13:40:00        40
3           Rohan         13:11:11        50
4          Chirag         13:30:00        30
5       Dev Dubey         13:30:00        30
6    Girish Goyal         13:14:59        40
7   Manish Sharma         13:30:00        50
8    Himank Singh         13:30:00        40
9    Absalom Maxy         17:29:59        30
```

```
             Name Time of Joining  Duration
0          Hemant         13:35:00        50
1          Gautam         14:00:00        30
2   Rishabh Kumar         13:55:00        40
3         Anirban         13:30:00        50
4          Chirag         13:30:00        50
5       Dev Dubey         13:30:00        50
6    Girish Goyal         13:25:00        40
7   Manish Sharma         13:30:00        50
8    Himank Singh         13:30:00        40
```

[26]:
```
# a) merge the two dataframes to find out names of participants who attended␣
 ↪both workshops.
pd.merge(df1, df2, how="inner", on="Name")
```

[26]:
```
             Name Time of Joining_x  Duration_x Time of Joining_y  Duration_y
0          Hemant          13:30:00          50          13:35:00          50
1   Rishabh Kumar          13:40:00          40          13:55:00          40
2          Chirag          13:30:00          30          13:30:00          50
3       Dev Dubey          13:30:00          30          13:30:00          50
4    Girish Goyal          13:14:59          40          13:25:00          40
5   Manish Sharma          13:30:00          50          13:30:00          50
6    Himank Singh          13:30:00          40          13:30:00          40
```

[27]:
```
# b) Find names of all students who have attended workshop on either of the␣
 ↪days.
either_day = pd.merge(df1,df2,how='outer',on='Name')
either_day['Name']
```

[27]:
```
0            Hemant
1      Gautam Meena
2     Rishabh Kumar
3             Rohan
4            Chirag
5         Dev Dubey
```

```
6      Girish Goyal
7     Manish Sharma
8      Himank Singh
9      Absalom Maxy
10           Gautam
11          Anirban
Name: Name, dtype: object
```

[28]:
```
# c) Merge two data frames row-wise and find the total number of records in the data frame.
print(either_day)
print(f"Total No. of Records: {either_day['Name'].count()}")
```

```
             Name Time of Joining_x  Duration_x Time of Joining_y  Duration_y
0          Hemant           13:30:00        50.0           13:35:00        50.0
1     Gautam Meena           13:45:15        40.0                NaN         NaN
2    Rishabh Kumar           13:40:00        40.0           13:55:00        40.0
3           Rohan            13:11:11        50.0                NaN         NaN
4          Chirag           13:30:00        30.0           13:30:00        50.0
5       Dev Dubey           13:30:00        30.0           13:30:00        50.0
6     Girish Goyal           13:14:59        40.0           13:25:00        40.0
7    Manish Sharma           13:30:00        50.0           13:30:00        50.0
8     Himank Singh           13:30:00        40.0           13:30:00        40.0
9     Absalom Maxy           17:29:59        30.0                NaN         NaN
10          Gautam                NaN         NaN           14:00:00        30.0
11         Anirban                NaN         NaN           13:30:00        50.0
Total No. of Records: 12
```

[29]:
```
# d) Merge two data frames and use two columns names and duration as multi-row indexes.
#    Generate descriptive statistics for this multi-index

multi_merge = pd.merge(df1, df2, on=['Name', 'Duration'])
stats= multi_merge.groupby(['Name', 'Duration']).describe()
stats
```

[29]:

		Time of Joining_x				\
		count	unique	top	freq	
Name	Duration					
Girish Goyal	40	1	1	13:14:59	1	
Hemant	50	1	1	13:30:00	1	
Himank Singh	40	1	1	13:30:00	1	
Manish Sharma	50	1	1	13:30:00	1	
Rishabh Kumar	40	1	1	13:40:00	1	

		Time of Joining_y			
		count	unique	top	freq

```
Name          Duration
Girish Goyal  40                      1       1  13:25:00    1
Hemant        50                      1       1  13:35:00    1
Himank Singh  40                      1       1  13:30:00    1
Manish Sharma 50                      1       1  13:30:00    1
Rishabh Kumar 40                      1       1  13:55:00    1
```

[30]:
```
# also
both_days = pd.merge(df1,df2,how='outer',on=['Name','Duration']).copy() # copy␣
 ↪of an existing list

both_days.fillna(value='-',inplace=True) # to fill out the missing values in␣
 ↪the given series object

both_days.set_index(['Name','Duration']) # a method to set a List as index of␣
 ↪DataFrame
```

[30]:
```
                        Time of Joining_x Time of Joining_y
Name          Duration
Hemant        50                13:30:00          13:35:00
Gautam Meena  40                13:45:15                 -
Rishabh Kumar 40                13:40:00          13:55:00
Rohan         50                13:11:11                 -
Chirag        30                13:30:00                 -
Dev Dubey     30                13:30:00                 -
Girish Goyal  40                13:14:59          13:25:00
Manish Sharma 50                13:30:00          13:30:00
Himank Singh  40                13:30:00          13:30:00
Absalom Maxy  30                17:29:59                 -
Gautam        30                       -          14:00:00
Anirban       50                       -          13:30:00
Chirag        50                       -          13:30:00
Dev Dubey     50                       -          13:30:00
```

[31]:
```
both_days.describe
```

[31]:
```
<bound method NDFrame.describe of              Name Time of Joining_x  Duration
Time of Joining_y
0           Hemant          13:30:00        50          13:35:00
1     Gautam Meena          13:45:15        40                 -
2    Rishabh Kumar          13:40:00        40          13:55:00
3            Rohan          13:11:11        50                 -
4           Chirag          13:30:00        30                 -
5        Dev Dubey          13:30:00        30                 -
6     Girish Goyal          13:14:59        40          13:25:00
7    Manish Sharma          13:30:00        50          13:30:00
8     Himank Singh          13:30:00        40          13:30:00
```

```
9     Absalom Maxy          17:29:59        30                -
10          Gautam                 -        30         14:00:00
11         Anirban                 -        50         13:30:00
12          Chirag                 -        50         13:30:00
13       Dev Dubey                 -        50         13:30:00>
```

[ ]:

[ ]:

Practical Question No. 5

Q5: Taking 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)
a. Plot bar chart to show the frequency of each class label in the data.
b. Draw a scatter plot for Petal width vs sepal width.
c. Plot density distribution for feature petal length.
d. Use a pair plot to show pairwise bivariate distribution in the Iris Dataset.

[32]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
iris = datasets.load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
target_names = {i: name for i, name in enumerate(iris.target_names)}
iris_df['target_names'] = iris_df['target'].map(target_names)
iris_df
```

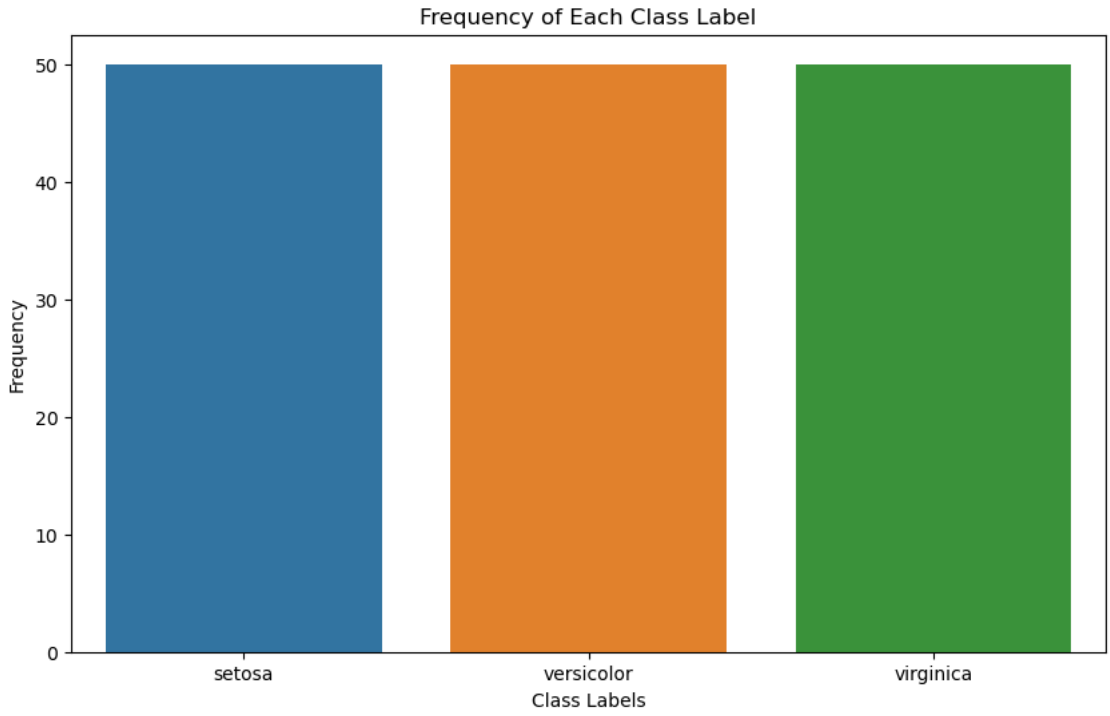
[32]:

	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	

	target	target_names
0	0	setosa
1	0	setosa
2	0	setosa
3	0	setosa

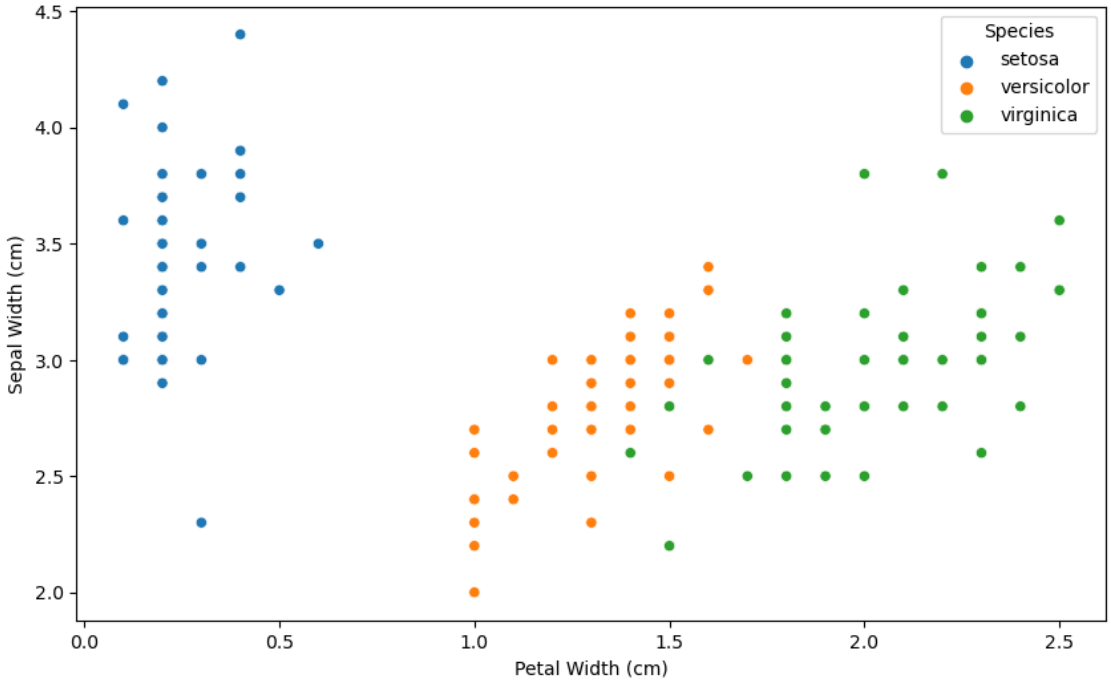
```
4          0       setosa
..       …           …
145        2    virginica
146        2    virginica
147        2    virginica
148        2    virginica
149        2    virginica

[150 rows x 6 columns]
```

[33]:
```
# a) plot a bar chart to show the frequency of each class label in the data
plt.figure(figsize=(10, 6))
sns.countplot(x='target_names', data=iris_df)
plt.xlabel('Class Labels')
plt.ylabel('Frequency')
plt.title('Frequency of Each Class Label')
plt.show()
```

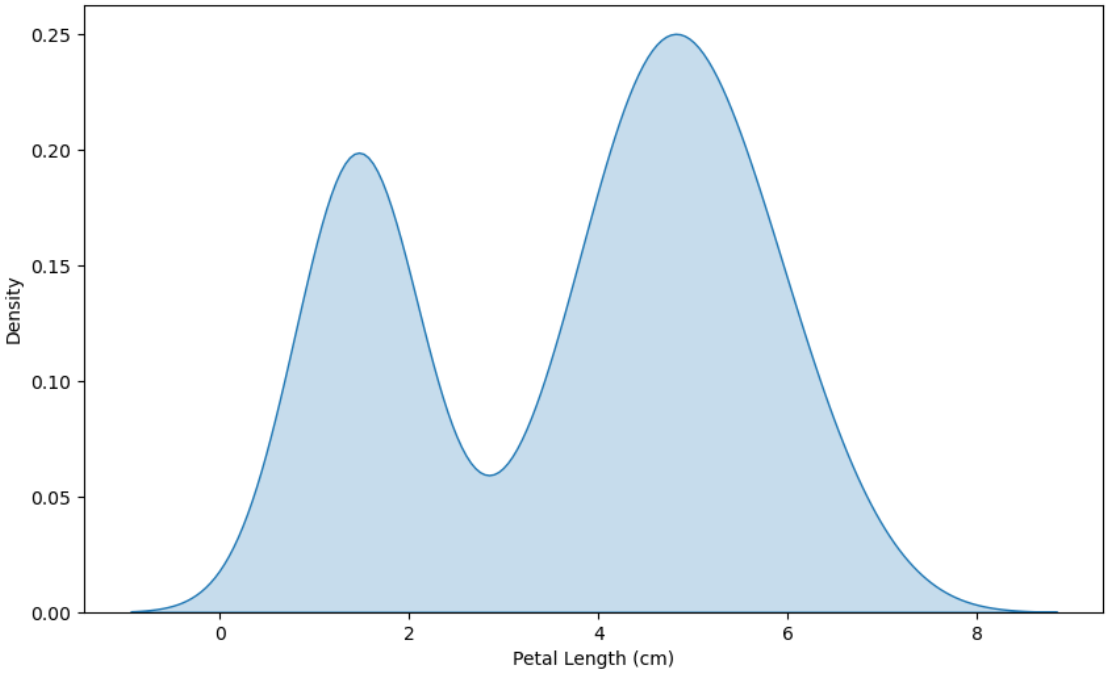


[34]:
```
# b) Draw a scatter plot for Petal width vs sepal width.
plt.figure(figsize=(10, 6))
sns.scatterplot(x='petal width (cm)', y='sepal width (cm)', hue='target_names',␣
↪data=iris_df)
plt.xlabel('Petal Width (cm)')
plt.ylabel('Sepal Width (cm)')
```

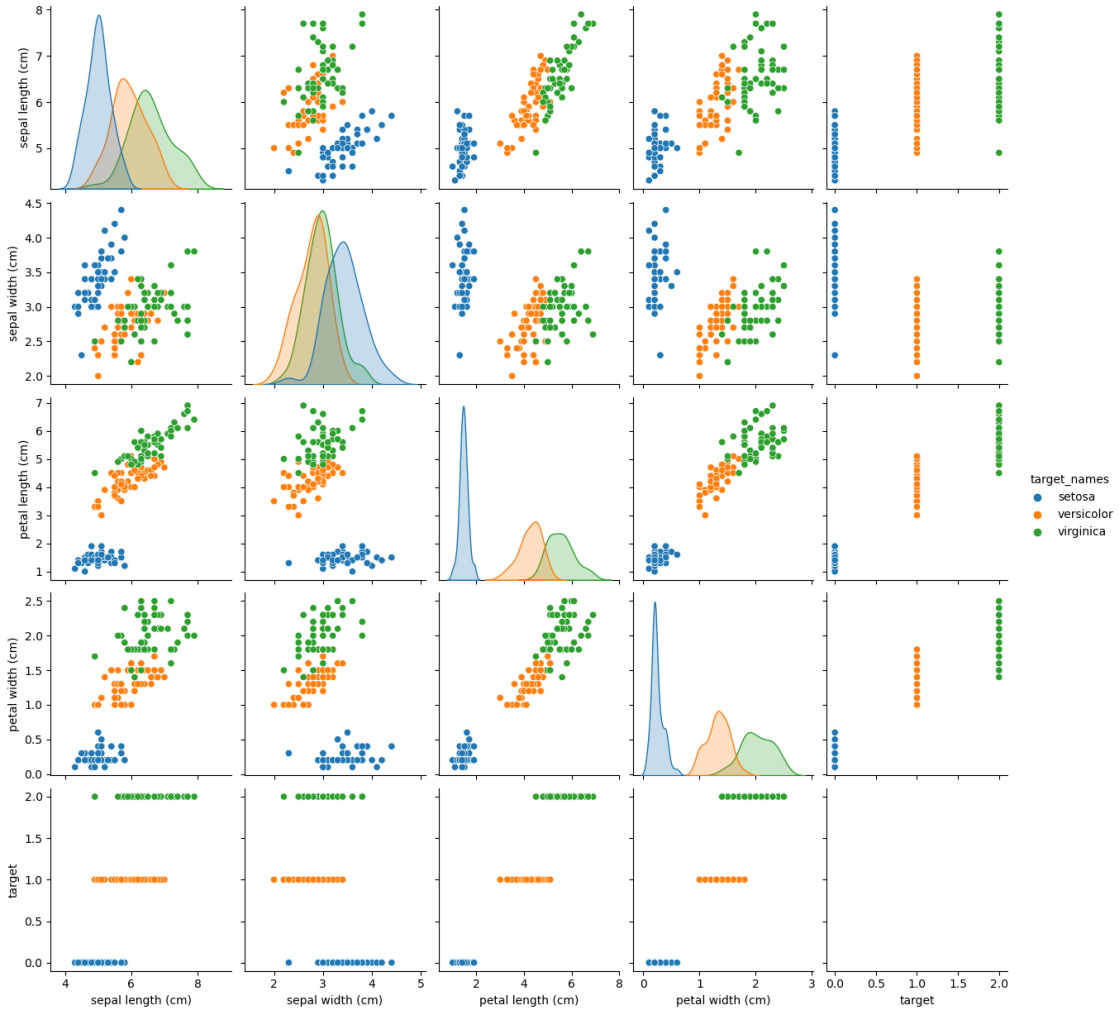
```
plt.legend(title='Species')
plt.show()
```



[35]:
```
# c) plot density distribution for feature petal length
plt.figure(figsize=(10, 6))
sns.kdeplot(iris_df['petal length (cm)'],fill=True)
plt.xlabel('Petal Length (cm)')
plt.ylabel('Density')
plt.show()
```



[36]:
```
# d) Use a pair plot to show pairwise bivariate distribution in the IRIS␣
  ↪dataset.
sns.pairplot(iris_df, hue='target_names')
plt.show()
```



[ ]:

[ ]:

Practical Question No. 6

Q6: Consider any sales training/ weather forecasting dataset
a. Compute mean of a series grouped by another series
b. Fill an intermittent time series to replace all missing dates with values of previous non-missing date.
c. Perform appropriate year-month string to dates conversion.
d. Split a dataset to group by two columns and then sort the aggregated results within the groups.
e. Split a given dataframe into groups with bin counts.

[37]:
```
import pandas as pd
import numpy as np
```

```
data = {
    'Date': ['2023-01-01', '2023-01-02','2023-01-03', '2023-01-04',␣
 ↪'2023-01-06', '2023-01-07'],
    'Sales': [100, 150, 300, 200, 180, 220],
    'Region': ['North', 'South', 'East','East', 'West', 'North'],
    'Product': ['A', 'B', 'C', 'A', 'C', 'B']
}
# Convert 'Date' column to datetime format
sales_df = pd.DataFrame(data)
sales_df['Date'] = pd.to_datetime(sales_df['Date'])
sales_df
```

[37]:

	Date	Sales	Region	Product
0	2023-01-01	100	North	A
1	2023-01-02	150	South	B
2	2023-01-03	300	East	C
3	2023-01-04	200	East	A
4	2023-01-06	180	West	C
5	2023-01-07	220	North	B

[38]:
```
# a) Compute mean of a series grouped by another series
mean_sales = sales_df.groupby('Region')['Sales'].mean()
print(mean_sales)
```

```
Region
East     250.0
North    160.0
South    150.0
West     180.0
Name: Sales, dtype: float64
```

[39]:
```
# b) fill an intermittent time series to replace all missing dates with values␣
 ↪of previous non-missing date.
sales_df1 = sales_df.set_index('Date').asfreq('D', method='ffill')
sales_df1
```

[39]:

Date	Sales	Region	Product
2023-01-01	100	North	A
2023-01-02	150	South	B
2023-01-03	300	East	C
2023-01-04	200	East	A
2023-01-05	200	East	A
2023-01-06	180	West	C
2023-01-07	220	North	B

[40]:
```
# c) Perfrom appropriate year-month string to dates conversion.
sales_df['year_month']=['2023-08','2023-09','2023-11','2023-10','2023-11','2023-07']
sales_df['year_month'] = pd.to_datetime(sales_df['year_month'])
sales_df
```

[40]:

	Date	Sales	Region	Product	year_month
0	2023-01-01	100	North	A	2023-08-01
1	2023-01-02	150	South	B	2023-09-01
2	2023-01-03	300	East	C	2023-11-01
3	2023-01-04	200	East	A	2023-10-01
4	2023-01-06	180	West	C	2023-11-01
5	2023-01-07	220	North	B	2023-07-01

[41]:
```
# d) Split a dataset to group by two columns and then sort the aggregated␣
 ↪results within the groups.
sorted = sales_df.groupby(['Region', 'Product'])['Sales'].sum().sort_values()
print(sorted)
```

```
Region  Product
North   A          100
South   B          150
West    C          180
East    A          200
North   B          220
East    C          300
Name: Sales, dtype: int64
```

[42]:
```
# e) Split a given dataframe into groups with bin counts.
sales_df['Bins'] = pd.cut(sales_df['Sales'], bins=4)
sales_groups = sales_df.groupby('Bins')
for key, group in sales_groups:
    print(key)
    print(group)
```

```
(99.8, 150.0]
        Date  Sales Region Product year_month           Bins
0 2023-01-01    100  North       A 2023-08-01  (99.8, 150.0]
1 2023-01-02    150  South       B 2023-09-01  (99.8, 150.0]
(150.0, 200.0]
        Date  Sales Region Product year_month           Bins
3 2023-01-04    200   East       A 2023-10-01  (150.0, 200.0]
4 2023-01-06    180   West       C 2023-11-01  (150.0, 200.0]
(200.0, 250.0]
        Date  Sales Region Product year_month           Bins
5 2023-01-07    220  North       B 2023-07-01  (200.0, 250.0]
(250.0, 300.0]
        Date  Sales Region Product year_month           Bins
2 2023-01-03    300   East       C 2023-11-01  (250.0, 300.0]
```

[ ]:

[ ]:

Practical Question No. 7

Q7: Consider a data frame containing data about students i.e. name, gender and passing division:

```
           Name        Birth_Month  Gender  Pass_Division
0       Mudit Chauhan   December    M   III
1       Seema Chopra    January         F   II
2   Rani Gupta  March           F   I
3   Aditya Narayan  October     M   I
4   Sanjeev Sahni   February    M   II
5   Prakash Kumar   December    M   III
6   Ritu Agarwal    September   F   I
7   Akshay Goel August          M   I
8   Meeta Kulkarni  July        F   II
9   Preeti Ahuja    November    F   II
10  Sunil Das Gupta April           M   III
11  Sonali Sapre    January         F   I
12  Rashmi Talwar   June            F   III
13  Ashish Dubey    May         M   II
14  Kiran Sharma    February    F   II
15  Sameer Bansal   October         M   I
```

a. Perform one hot encoding of the last two columns of categorical data using the get_dummies() function.

b. Sort this data frame on the "Birth Month" column (i.e. January to December). Hint: Convert Month to Categorical.

[43]:
```
import numpy as np
import pandas as pd

data = {
    'Name': ['Mudit Chauhan', 'Seema Chopra', 'Rani Gupta', 'Aditya Narayan',␣
 ↪'Sanjeev Sahni',
             'Prakash Kumar', 'Ritu Agarwal', 'Akshay Goel', 'Meeta Kulkarni',␣
 ↪'Preeti Ahuja',
             'Sunil Das Gupta', 'Sonali Sapre', 'Rashmi Talwar', 'Ashish␣
 ↪Dubey', 'Kiran Sharma',
             'Sameer Bansal'],
   'Birth_Month': ['December', 'January', 'March', 'October', 'February',␣
 ↪'December', 'September',
                   'August', 'July', 'November', 'April', 'January', 'June',␣
 ↪'May', 'February',
                   'October'],
```

```
    'Gender': ['M', 'F', 'F', 'M', 'M', 'M', 'F', 'M', 'F', 'F', 'M', 'F', 'F',␣
 ↪'M', 'F', 'M'],
    'Pass_Division': ['III', 'II', 'I', 'I', 'II', 'III', 'I', 'I', 'II', 'II',␣
 ↪'III', 'I', 'III',
                      'II', 'II', 'I']
}
df = pd.DataFrame(data)

# or raad and enter data into excel file.
#df = pd.read_excel('Students.xlsx')
df
```

[43]:

	Name	Birth_Month	Gender	Pass_Division
0	Mudit Chauhan	December	M	III
1	Seema Chopra	January	F	II
2	Rani Gupta	March	F	I
3	Aditya Narayan	October	M	I
4	Sanjeev Sahni	February	M	II
5	Prakash Kumar	December	M	III
6	Ritu Agarwal	September	F	I
7	Akshay Goel	August	M	I
8	Meeta Kulkarni	July	F	II
9	Preeti Ahuja	November	F	II
10	Sunil Das Gupta	April	M	III
11	Sonali Sapre	January	F	I
12	Rashmi Talwar	June	F	III
13	Ashish Dubey	May	M	II
14	Kiran Sharma	February	F	II
15	Sameer Bansal	October	M	I

[44]:
```
# a) Perform one hot encoding of the last two columns of categorical data using␣
 ↪the get_dummies() function.
one_hot = pd.get_dummies(df[['Gender', 'Pass_Division']])
df_encode = pd.concat([df, one_hot], axis=1)
print(df_encode)
```

	Name	Birth_Month	Gender	Pass_Division	Gender_F	Gender_M	\
0	Mudit Chauhan	December	M	III	0	1	
1	Seema Chopra	January	F	II	1	0	
2	Rani Gupta	March	F	I	1	0	
3	Aditya Narayan	October	M	I	0	1	
4	Sanjeev Sahni	February	M	II	0	1	
5	Prakash Kumar	December	M	III	0	1	
6	Ritu Agarwal	September	F	I	1	0	
7	Akshay Goel	August	M	I	0	1	
8	Meeta Kulkarni	July	F	II	1	0	
9	Preeti Ahuja	November	F	II	1	0	

```
10  Sunil Das Gupta       April      M           III         0         1
11     Sonali Sapre     January      F             I         1         0
12    Rashmi Talwar        June      F           III         1         0
13     Ashish Dubey         May      M            II         0         1
14     Kiran Sharma    February      F            II         1         0
15    Sameer Bansal     October      M             I         0         1

    Pass_Division_I  Pass_Division_II  Pass_Division_III
0                 0                 0                  1
1                 0                 1                  0
2                 1                 0                  0
3                 1                 0                  0
4                 0                 1                  0
5                 0                 0                  1
6                 1                 0                  0
7                 1                 0                  0
8                 0                 1                  0
9                 0                 1                  0
10                0                 0                  1
11                1                 0                  0
12                0                 0                  1
13                0                 1                  0
14                0                 1                  0
15                1                 0                  0
```

[45]:
```
# b) Sort this data frame on the "Birth Month" column (i.e. January to␣
 ↪December). Hint: Convert Month to Categorical.
month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July',␣
 ↪'August', 'September', 'October', 'November', 'December']
df['Birth_Month'] = pd.Categorical(df['Birth_Month'], categories=month_order,␣
 ↪ordered=True)
df_sorted = df.sort_values('Birth_Month')
print(df_sorted)
```

```
               Name Birth_Month Gender Pass_Division
1      Seema Chopra     January      F            II
11     Sonali Sapre     January      F             I
4     Sanjeev Sahni    February      M            II
14     Kiran Sharma    February      F            II
2        Rani Gupta       March      F             I
10  Sunil Das Gupta       April      M           III
13     Ashish Dubey         May      M            II
12    Rashmi Talwar        June      F           III
8    Meeta Kulkarni        July      F            II
7       Akshay Goel      August      M             I
6      Ritu Agarwal   September      F             I
3    Aditya Narayan     October      M             I
```

```
15    Sameer Bansal     October      M            I
9      Preeti Ahuja    November      F           II
0     Mudit Chauhan    December      M          III
5     Prakash Kumar    December      M          III
```

[ ]:

[ ]:

Practical Question No. 8

Q8: 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
```

Write a program in Python using Pandas to perform the following:
a. Calculate and display familywise gross monthly income.
b. Calculate and display the member with the highest monthly income in a family.
c. Calculate and display monthly income of all members with income greater than Rs. 60000.00.
d. Calculate and display the average monthly income of the female members in the Shah family.

[46]:
```
import numpy as np
import pandas as pd
```

[47]:
```
family_df = {'Name':['Shah', 'Vats', 'Vats', 'Kumar', 'Vats', 'Kumar','Shah',␣
 ↪'Shah', 'Kumar', 'Vats'],
             'Gender':['Male','Male','Female', 'Female',␣
 ↪'Female','Male','Male','Female','Female','Male'],
             'MonthlyIncome':[114000.00, 65000.00,43150.00, 69500.00, 155000.
 ↪00, 103000.00, 55000.00, 112400.00, 81030.00, 71900.00]
            }

df = pd.DataFrame(family_df, columns = ['Name', 'Gender', 'MonthlyIncome'])
print(df)
```

```
    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
```

[48]:
```
# a. Calculate and display familywise gross monthly income
df.groupby('Name')['MonthlyIncome'].sum()
```

[48]:
```
Name
Kumar    253530.0
Shah     281400.0
Vats     335050.0
Name: MonthlyIncome, dtype: float64
```

[49]:
```
# b. Calculate and display the member with the highest monthly income in a␣
  ↪family
member = df.iloc[df.groupby('Name')['MonthlyIncome'].idxmax()]
member
```

[49]:
```
    Name  Gender  MonthlyIncome
5  Kumar    Male       103000.0
0   Shah    Male       114000.0
4   Vats  Female       155000.0
```

[50]:
```
# c. Calculate and display monthly income of all members with inome greater␣
  ↪than Rs.60000.00.
df[df['MonthlyIncome']>60000.00]
```

[50]:
```
    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
```

[51]:
```
# d. Calculate and display the average monthly income of the female member in␣
  ↪the Shah family.
df[(df['Name'] == 'Shah') & (df['Gender'] == 'Female')]['MonthlyIncome'].mean()
```

[51]:
```
112400.0
```

[ ]:

[ ]:

[ ]: