DATA ANALYSIS AND VISUALISATION FILE

SUBMITTED BY
- VIVEK SHARMA

COURSE – B.SC(H) CS

YEAR – 3RD

SEMESTER – 5TH

ROLL NO. – 21013570104

Clg ROLL NO. – 2K21/CS/111

SUBMITTED TO
–GEETIKA MA'AM

1.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:

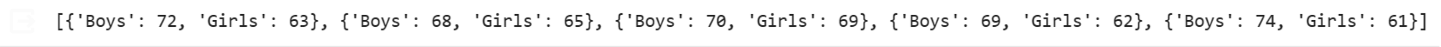
{'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65, 69, 62, 61]}

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]

[ ]

```
original_dictionary = {'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65, 69, 62, 61]}
result_list = []
boy_list = original_dictionary['Boys']
girl_list = original_dictionary['Girls']
for i in range(len(boy_list)):
    boy = boy_list[i]
    girl = girl_list[i]
    result_list.append({'Boys': boy, 'Girls': girl})
print(result_list)
```



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

2.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.

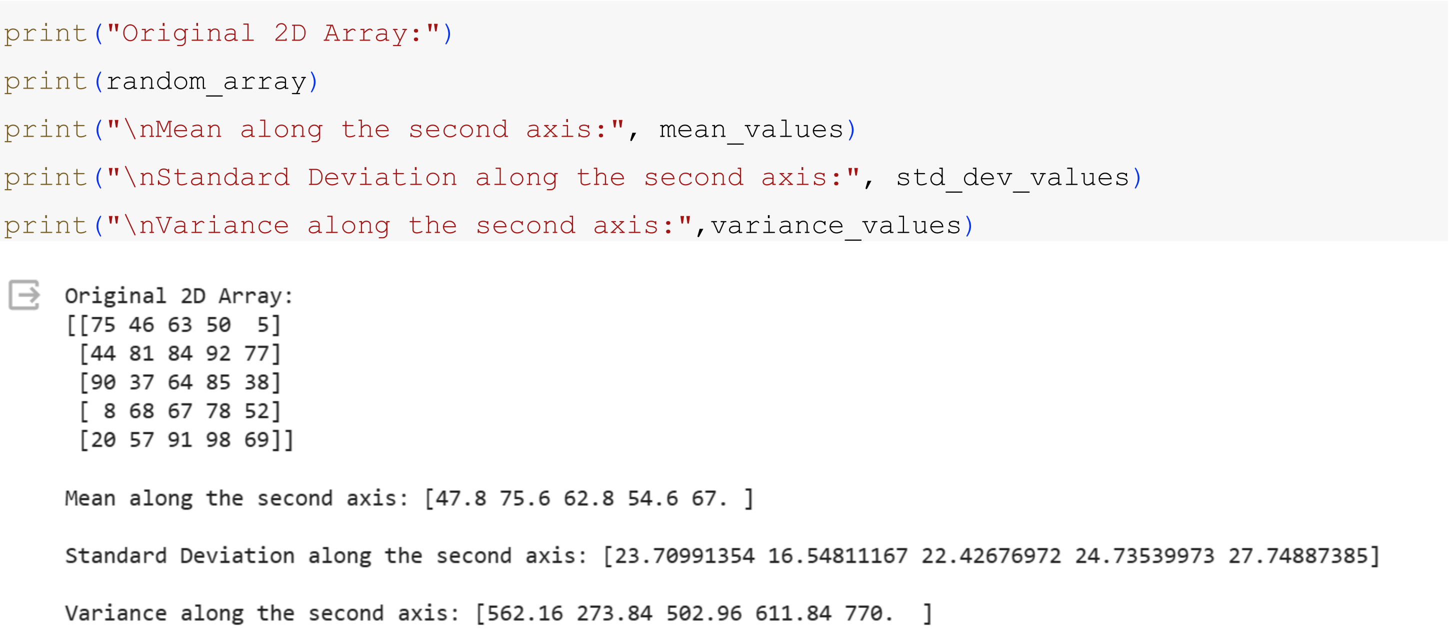
b. Get the indices of the sorted elements of a given array. a. B = [56, 48, 22, 41, 78, 91, 24, 46, 8, 33]

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.

[ ]

```
#a
import numpy as np
#random 2D array
random_array = np.random.randint(1, 100, size=(5, 5))
#mean along the second axis (axis=1)
mean_values = np.mean(random_array, axis=1)
# standard deviation along the second axis (axis=1)
std_dev_values = np.std(random_array, axis=1)
#variance along the second axis (axis=1)
variance_values = np.var(random_array, axis=1)
```

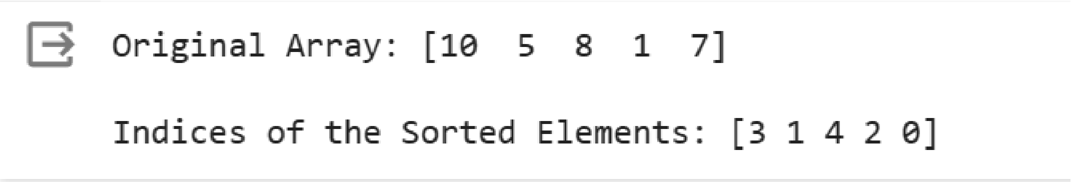


```
print("Original 2D Array:")
print(random_array)
print("\nMean along the second axis:", mean_values)
print("\nStandard Deviation along the second axis:", std_dev_values)
print("\nVariance along the second axis:",variance_values)
```

```
Original 2D Array:
[[75 46 63 50  5]
 [44 81 84 92 77]
 [90 37 64 85 38]
 [ 8 68 67 78 52]
 [20 57 91 98 69]]

Mean along the second axis: [47.8 75.6 62.8 54.6 67. ]

Standard Deviation along the second axis: [23.70991354 16.54811167 22.42676972 24.73539973 27.74887385]

Variance along the second axis: [562.16 273.84 502.96 611.84 770.  ]
```

[ ]

```
#b
import numpy as np

#random array
original_array = np.array([10, 5, 8, 1, 7])

print("Original Array:",original_array)
print()
print("Indices of the Sorted Elements:",np.argsort(original_array))#indices of the sorted elements
```

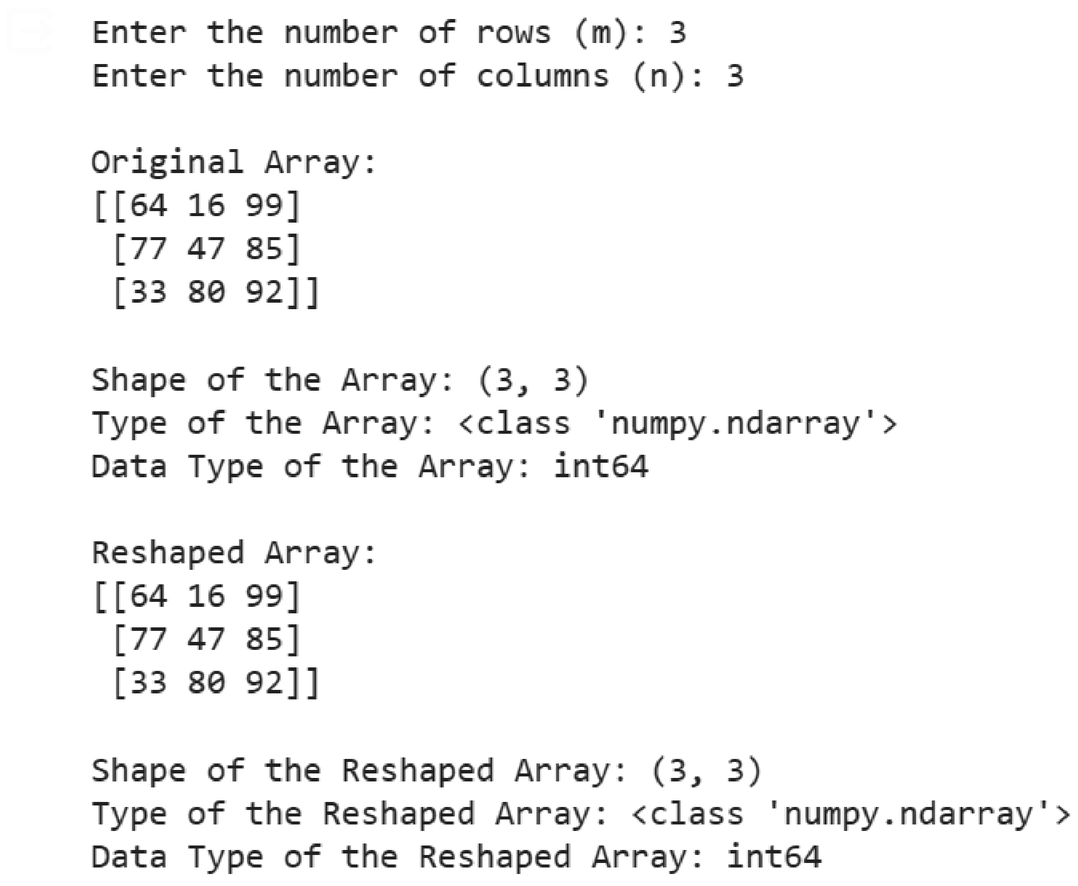


```
Original Array: [10  5  8  1  7]

Indices of the Sorted Elements: [3 1 4 2 0]
```

[ ]

```
#c
import numpy as np

# user inputs for m and n
m = int(input("Enter the number of rows (m): "))
n = int(input("Enter the number of columns (n): "))

#2D array of size m x n with random integer elements
original_array = np.random.randint(1, 100, size=(m, n))

print("\nOriginal Array:")
print(original_array)

print("\nShape of the Array:", original_array.shape)
```

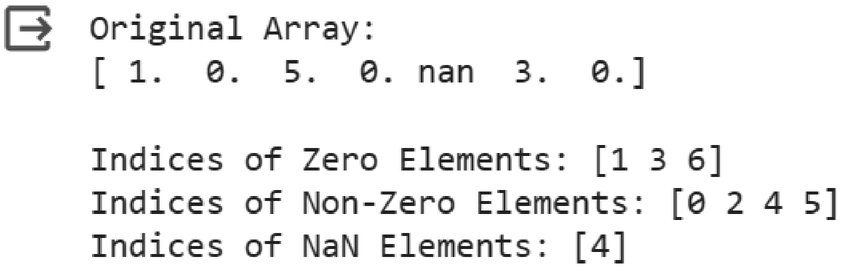
```python
print("Type of the Array:", type(original_array))
print("Data Type of the Array:", original_array.dtype)

# Reshape the array into an nxm array
reshaped_array = original_array.reshape((n, m))

print("\nReshaped Array:")
print(reshaped_array)

print("\nShape of the Reshaped Array:", reshaped_array.shape)
print("Type of the Reshaped Array:", type(reshaped_array))
print("Data Type of the Reshaped Array:", reshaped_array.dtype)
```



```
Enter the number of rows (m): 3
Enter the number of columns (n): 3

Original Array:
[[64 16 99]
 [77 47 85]
 [33 80 92]]

Shape of the Array: (3, 3)
Type of the Array: <class 'numpy.ndarray'>
Data Type of the Array: int64

Reshaped Array:
[[64 16 99]
 [77 47 85]
 [33 80 92]]

Shape of the Reshaped Array: (3, 3)
Type of the Reshaped Array: <class 'numpy.ndarray'>
Data Type of the Reshaped Array: int64
```

```python
[]
#d
import numpy as np
#array with some zeros, non-zeros, and NaN values
sample_array = np.array([1, 0, 5, 0, np.nan, 3, 0])
# Test whether the elements are zero, non-zero, or NaN
zero_indices = np.where(sample_array == 0)[0]
non_zero_indices = np.where(sample_array != 0)[0]
nan_indices = np.where(np.isnan(sample_array))[0]
# original array
```

```
print("Original Array:")
print(sample_array)
# indices of zero, non-zero, and NaN elements
print("\nIndices of Zero Elements:", zero_indices)
print("Indices of Non-Zero Elements:", non_zero_indices)
print("Indices of NaN Elements:", nan_indices)
```



```
Original Array:
[ 1.  0.  5.  0. nan  3.  0.]

Indices of Zero Elements: [1 3 6]
Indices of Non-Zero Elements: [0 2 4 5]
Indices of NaN Elements: [4]
```

3. 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.

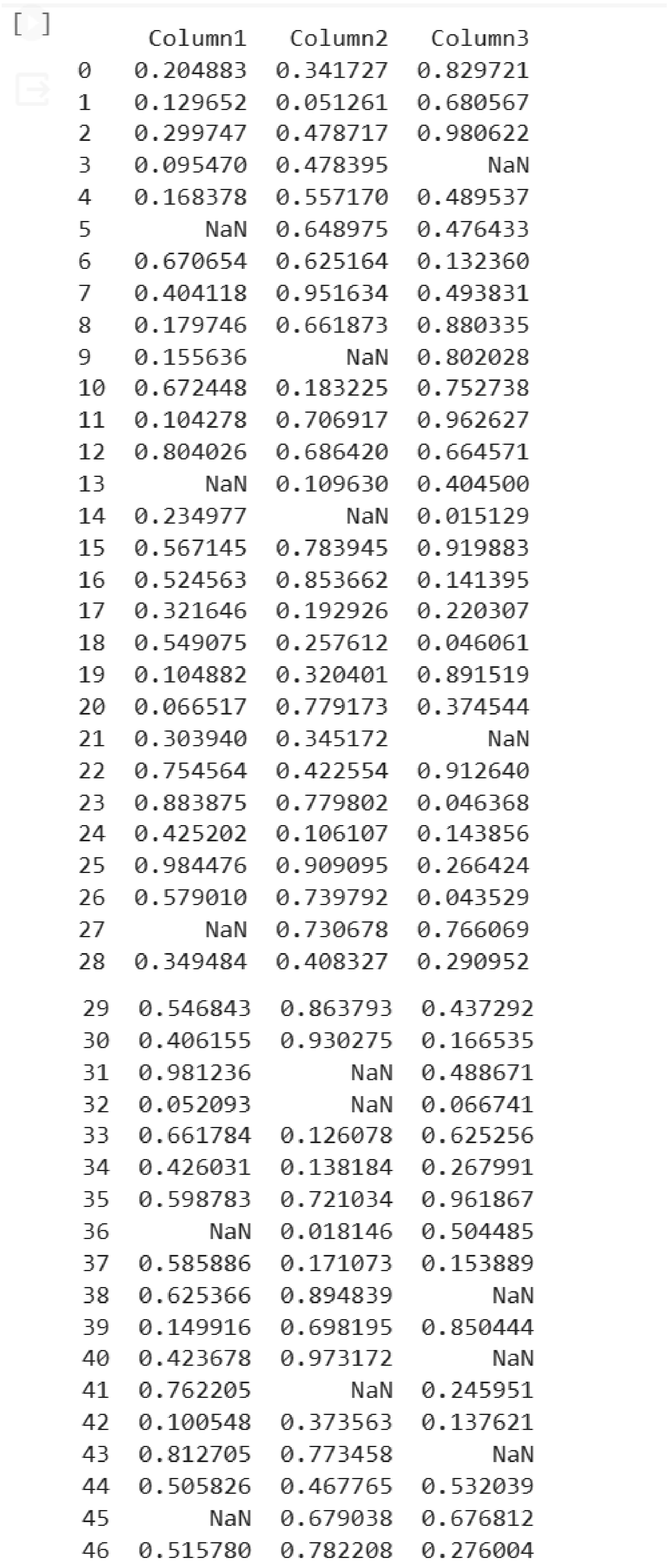
f. Find the correlation between first and second column and covariance between second and third column.

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

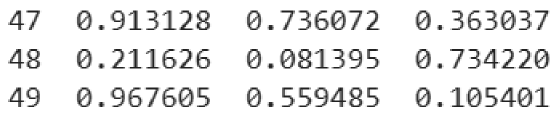
h. Discretize second column and create 5 bins

[ ]

```
import pandas as pd
import numpy as np

# Create a dataframe with random numeric data
data = {'Column1': np.random.rand(50), 'Column2': np.random.rand(50), 'Column3': np.random.rand(50)}
df = pd.DataFrame(data)

#nan values
nan_positions = np.random.choice(df.size, size=int(0.1 * df.size), replace=False)
df.values.flat[nan_positions] = np.nan
print(df)
```



```
     Column1   Column2   Column3
0   0.204883  0.341727  0.829721
1   0.129652  0.051261  0.680567
2   0.299747  0.478717  0.980622
3   0.095470  0.478395       NaN
4   0.168378  0.557170  0.489537
5        NaN  0.648975  0.476433
6   0.670654  0.625164  0.132360
7   0.404118  0.951634  0.493831
8   0.179746  0.661873  0.880335
9   0.155636       NaN  0.802028
10  0.672448  0.183225  0.752738
11  0.104278  0.706917  0.962627
12  0.804026  0.686420  0.664571
13       NaN  0.109630  0.404500
14  0.234977       NaN  0.015129
15  0.567145  0.783945  0.919883
16  0.524563  0.853662  0.141395
17  0.321646  0.192926  0.220307
18  0.549075  0.257612  0.046061
19  0.104882  0.320401  0.891519
20  0.066517  0.779173  0.374544
21  0.303940  0.345172       NaN
22  0.754564  0.422554  0.912640
23  0.883875  0.779802  0.046368
24  0.425202  0.106107  0.143856
25  0.984476  0.909095  0.266424
26  0.579010  0.739792  0.043529
27       NaN  0.730678  0.766069
28  0.349484  0.408327  0.290952

29  0.546843  0.863793  0.437292
30  0.406155  0.930275  0.166535
31  0.981236       NaN  0.488671
32  0.052093       NaN  0.066741
33  0.661784  0.126078  0.625256
34  0.426031  0.138184  0.267991
35  0.598783  0.721034  0.961867
36       NaN  0.018146  0.504485
37  0.585886  0.171073  0.153889
38  0.625366  0.894839       NaN
39  0.149916  0.698195  0.850444
40  0.423678  0.973172       NaN
41  0.762205       NaN  0.245951
42  0.100548  0.373563  0.137621
43  0.812705  0.773458       NaN
44  0.505826  0.467765  0.532039
45       NaN  0.679038  0.676812
46  0.515780  0.782208  0.276004
```



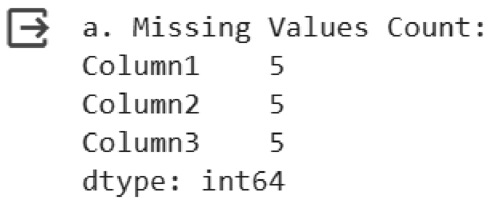
```
47  0.913128  0.736072  0.363037
48  0.211626  0.081395  0.734220
49  0.967605  0.559485  0.105401
```

[ ]

```
#a
missing_values_count = df.isnull().sum()
print("a. Missing Values Count:")
print(missing_values_count)
```

output

```
a. Missing Values Count:
Column1   5
Column2   5
Column3   5
dtype: int64
```

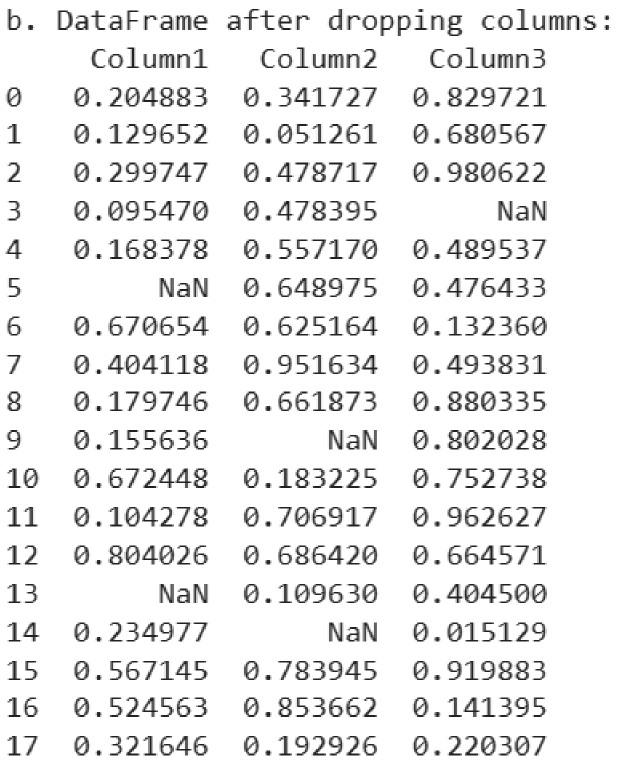


```
a. Missing Values Count:
Column1    5
Column2    5
Column3    5
dtype: int64
```

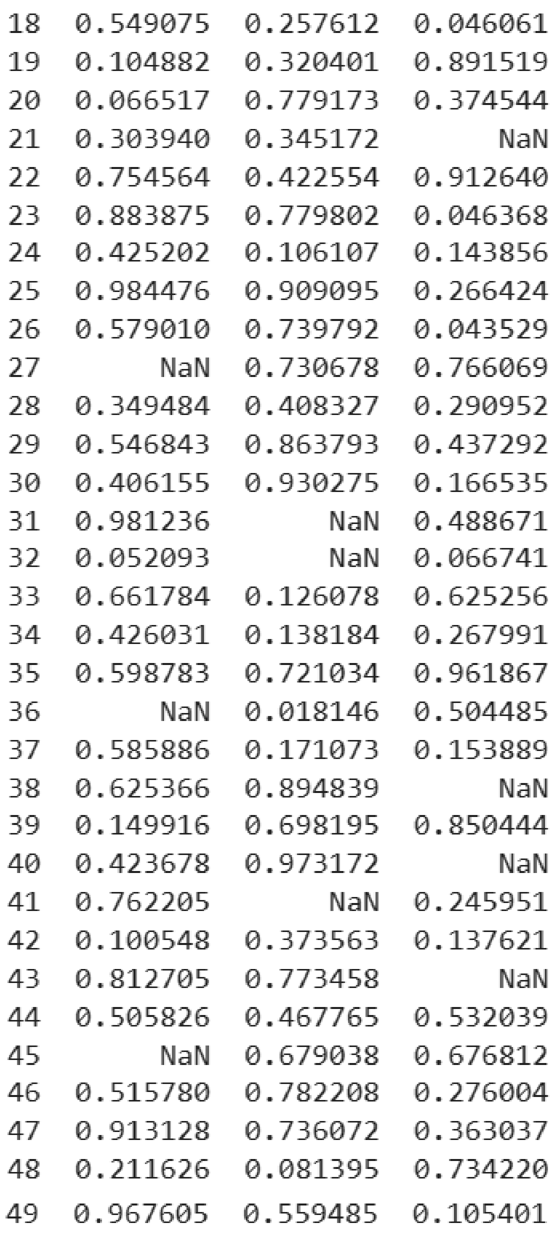
[ ]

```
#b
df = df.dropna(axis=1, thresh=df.shape[0] - 5)
print("\nb. DataFrame after dropping columns:")
print(df)
```

output



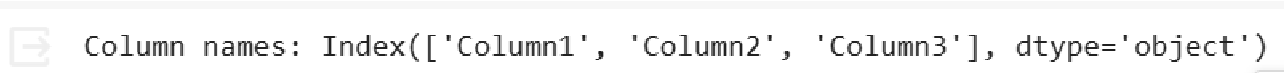
```
b. DataFrame after dropping columns:
     Column1   Column2   Column3
0   0.204883  0.341727  0.829721
1   0.129652  0.051261  0.680567
2   0.299747  0.478717  0.980622
3   0.095470  0.478395       NaN
4   0.168378  0.557170  0.489537
5        NaN  0.648975  0.476433
6   0.670654  0.625164  0.132360
7   0.404118  0.951634  0.493831
8   0.179746  0.661873  0.880335
9   0.155636       NaN  0.802028
10  0.672448  0.183225  0.752738
11  0.104278  0.706917  0.962627
12  0.804026  0.686420  0.664571
13       NaN  0.109630  0.404500
14  0.234977       NaN  0.015129
15  0.567145  0.783945  0.919883
16  0.524563  0.853662  0.141395
17  0.321646  0.192926  0.220307
```



```
18  0.549075  0.257612  0.046061
19  0.104882  0.320401  0.891519
20  0.066517  0.779173  0.374544
21  0.303940  0.345172       NaN
22  0.754564  0.422554  0.912640
23  0.883875  0.779802  0.046368
24  0.425202  0.106107  0.143856
25  0.984476  0.909095  0.266424
26  0.579010  0.739792  0.043529
27       NaN  0.730678  0.766069
28  0.349484  0.408327  0.290952
29  0.546843  0.863793  0.437292
30  0.406155  0.930275  0.166535
31  0.981236       NaN  0.488671
32  0.052093       NaN  0.066741
33  0.661784  0.126078  0.625256
34  0.426031  0.138184  0.267991
35  0.598783  0.721034  0.961867
36       NaN  0.018146  0.504485
37  0.585886  0.171073  0.153889
38  0.625366  0.894839       NaN
39  0.149916  0.698195  0.850444
40  0.423678  0.973172       NaN
41  0.762205       NaN  0.245951
42  0.100548  0.373563  0.137621
43  0.812705  0.773458       NaN
44  0.505826  0.467765  0.532039
45       NaN  0.679038  0.676812
46  0.515780  0.782208  0.276004
47  0.913128  0.736072  0.363037
48  0.211626  0.081395  0.734220
49  0.967605  0.559485  0.105401
```

[ ]

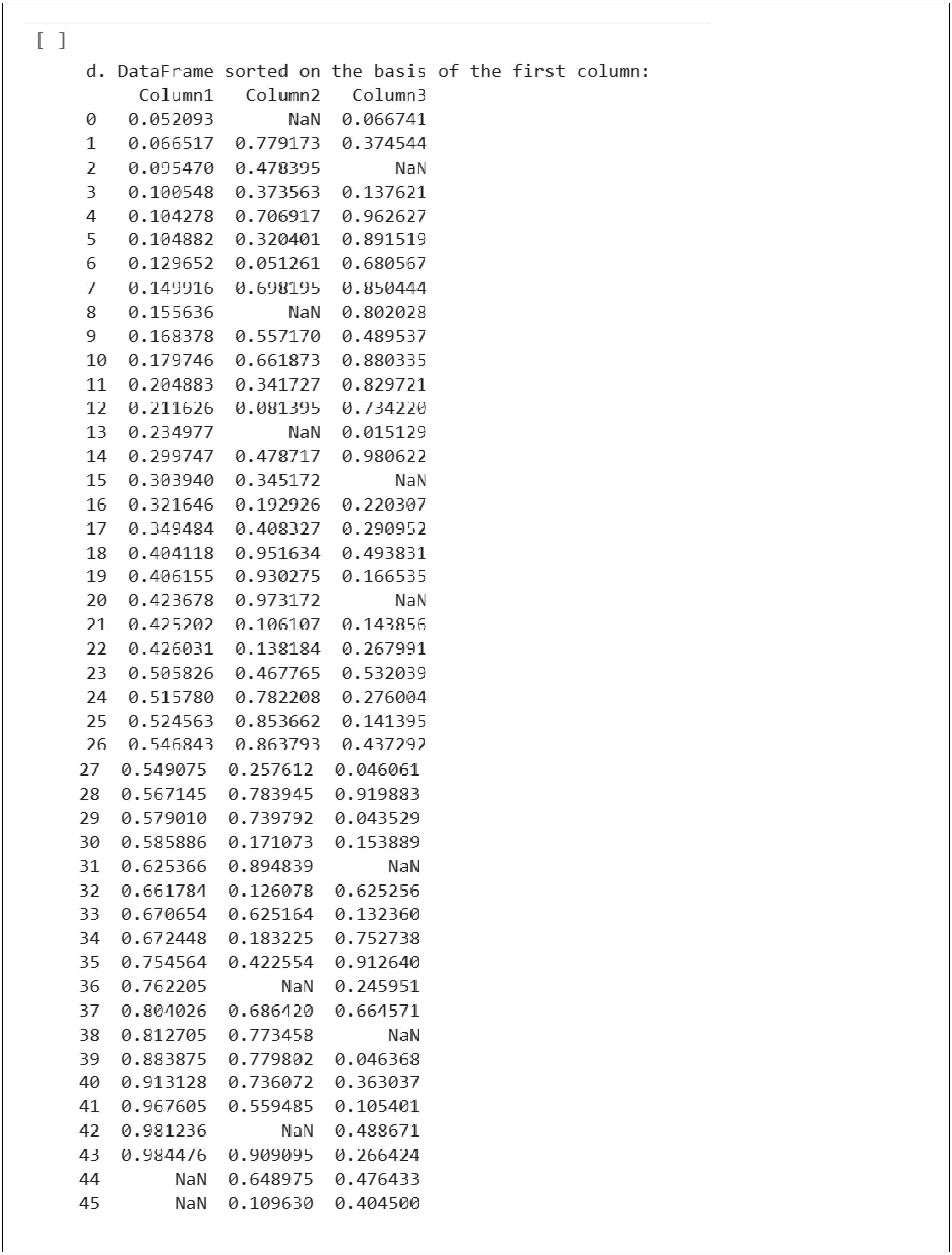
```
#c
max_sum_row_label = df.sum(axis=1).idxmax()
df = df.drop(index=max_sum_row_label)
print("Column names:", df.columns)
```



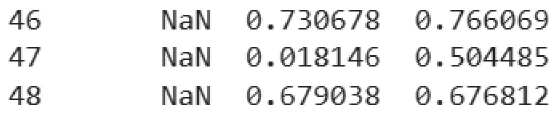
```
Column names: Index(['Column1', 'Column2', 'Column3'], dtype='object')
```

[ ]

```
#d
df = df.sort_values(by=df.columns[0]).reset_index(drop=True)
print("\nd. DataFrame sorted on the basis of the first column:")
print(df)
```



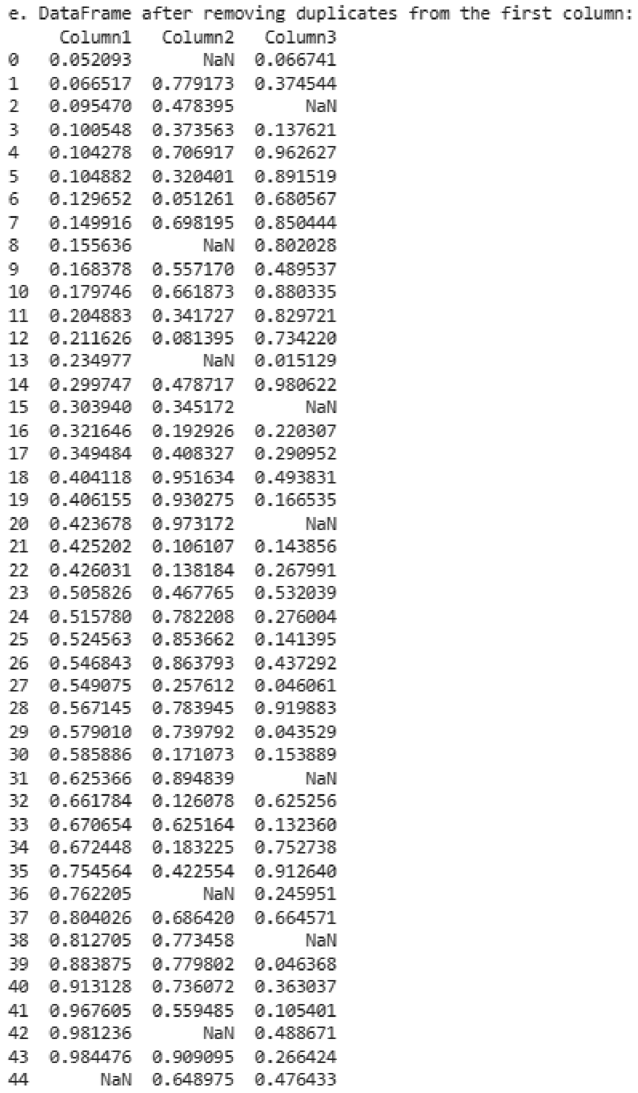
```
[ ]
    d. DataFrame sorted on the basis of the first column:
         Column1   Column2   Column3
    0   0.052093       NaN  0.066741
    1   0.066517  0.779173  0.374544
    2   0.095470  0.478395       NaN
    3   0.100548  0.373563  0.137621
    4   0.104278  0.706917  0.962627
    5   0.104882  0.320401  0.891519
    6   0.129652  0.051261  0.680567
    7   0.149916  0.698195  0.850444
    8   0.155636       NaN  0.802028
    9   0.168378  0.557170  0.489537
    10  0.179746  0.661873  0.880335
    11  0.204883  0.341727  0.829721
    12  0.211626  0.081395  0.734220
    13  0.234977       NaN  0.015129
    14  0.299747  0.478717  0.980622
    15  0.303940  0.345172       NaN
    16  0.321646  0.192926  0.220307
    17  0.349484  0.408327  0.290952
    18  0.404118  0.951634  0.493831
    19  0.406155  0.930275  0.166535
    20  0.423678  0.973172       NaN
    21  0.425202  0.106107  0.143856
    22  0.426031  0.138184  0.267991
    23  0.505826  0.467765  0.532039
    24  0.515780  0.782208  0.276004
    25  0.524563  0.853662  0.141395
    26  0.546843  0.863793  0.437292
    27  0.549075  0.257612  0.046061
    28  0.567145  0.783945  0.919883
    29  0.579010  0.739792  0.043529
    30  0.585886  0.171073  0.153889
    31  0.625366  0.894839       NaN
    32  0.661784  0.126078  0.625256
    33  0.670654  0.625164  0.132360
    34  0.672448  0.183225  0.752738
    35  0.754564  0.422554  0.912640
    36  0.762205       NaN  0.245951
    37  0.804026  0.686420  0.664571
    38  0.812705  0.773458       NaN
    39  0.883875  0.779802  0.046368
    40  0.913128  0.736072  0.363037
    41  0.967605  0.559485  0.105401
    42  0.981236       NaN  0.488671
    43  0.984476  0.909095  0.266424
    44       NaN  0.648975  0.476433
    45       NaN  0.109630  0.404500
```



```
46        NaN  0.730678  0.766069
47        NaN  0.018146  0.504485
48        NaN  0.679038  0.676812
```

[ ]

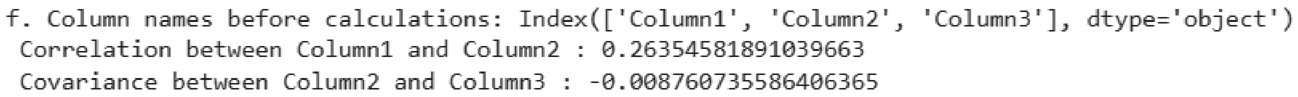
```
#e
df = df.drop_duplicates(subset=df.columns[0], ignore_index=True)
print("\ne. DataFrame after removing duplicates from the first column:")
print(df)
```



```
e. DataFrame after removing duplicates from the first column:
     Column1   Column2   Column3
0   0.052093       NaN  0.066741
1   0.066517  0.779173  0.374544
2   0.095470  0.478395       NaN
3   0.100548  0.373563  0.137621
4   0.104278  0.706917  0.962627
5   0.104882  0.320401  0.891519
6   0.129652  0.051261  0.680567
7   0.149916  0.698195  0.850444
8   0.155636       NaN  0.802028
9   0.168378  0.557170  0.489537
10  0.179746  0.661873  0.880335
11  0.204883  0.341727  0.829721
12  0.211626  0.081395  0.734220
13  0.234977       NaN  0.015129
14  0.299747  0.478717  0.980622
15  0.303940  0.345172       NaN
16  0.321646  0.192926  0.220307
17  0.349484  0.408327  0.290952
18  0.404118  0.951634  0.493831
19  0.406155  0.930275  0.166535
20  0.423678  0.973172       NaN
21  0.425202  0.106107  0.143856
22  0.426031  0.138184  0.267991
23  0.505826  0.467765  0.532039
24  0.515780  0.782208  0.276004
25  0.524563  0.853662  0.141395
26  0.546843  0.863793  0.437292
27  0.549075  0.257612  0.046061
28  0.567145  0.783945  0.919883
29  0.579010  0.739792  0.043529
30  0.585886  0.171073  0.153889
31  0.625366  0.894839       NaN
32  0.661784  0.126078  0.625256
33  0.670654  0.625164  0.132360
34  0.672448  0.183225  0.752738
35  0.754564  0.422554  0.912640
36  0.762205       NaN  0.245951
37  0.804026  0.686420  0.664571
38  0.812705  0.773458       NaN
39  0.883875  0.779802  0.046368
40  0.913128  0.736072  0.363037
41  0.967605  0.559485  0.105401
42  0.981236       NaN  0.488671
43  0.984476  0.909095  0.266424
44       NaN  0.648975  0.476433
```

[ ]

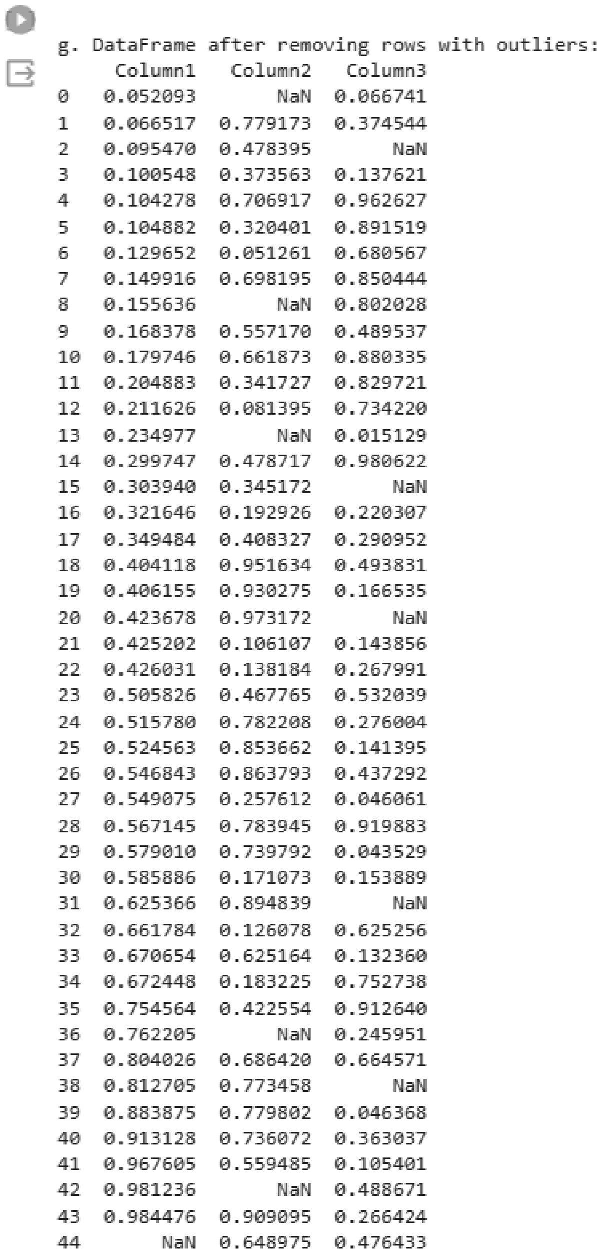
```
print("\nf. Column names before calculations:", df.columns)
if len(df.columns) >= 3:
  correlation = df[df.columns[0]].corr(df[df.columns[1]])
  covariance = df[df.columns[1]].cov(df[df.columns[2]])
  print(" Correlation between", df.columns[0], "and", df.columns[1], ":", correlation)
  print(" Covariance between", df.columns[1], "and", df.columns[2], ":", covariance)
else:
  print(" Insufficient columns to calculate correlation and covariance.")
```



```
f. Column names before calculations: Index(['Column1', 'Column2', 'Column3'], dtype='object')
 Correlation between Column1 and Column2 : 0.26354581891039663
 Covariance between Column2 and Column3 : -0.008760735586406365
```

[ ]

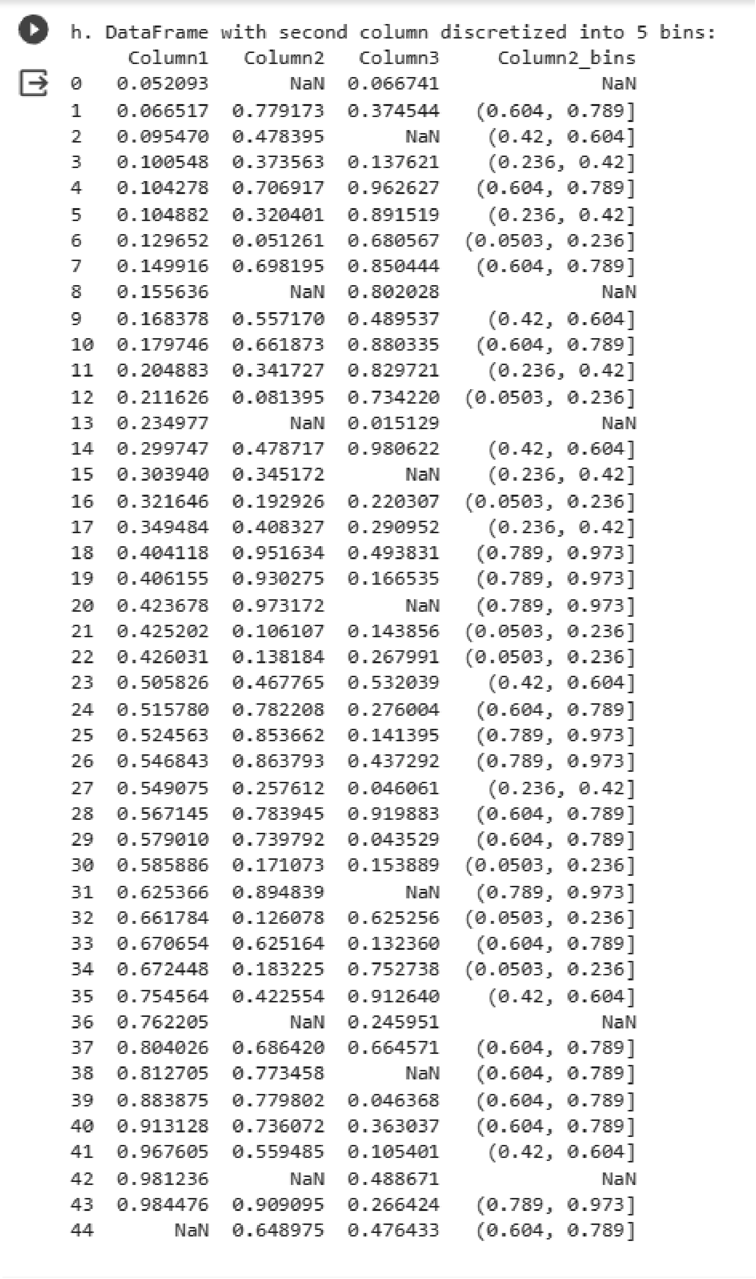
```
#g
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR)))
df = df[~outliers.any(axis=1)]
print("\ng. DataFrame after removing rows with outliers:")
print(df)
```



```
g. DataFrame after removing rows with outliers:
     Column1   Column2   Column3
0   0.052093       NaN  0.066741
1   0.066517  0.779173  0.374544
2   0.095470  0.478395       NaN
3   0.100548  0.373563  0.137621
4   0.104278  0.706917  0.962627
5   0.104882  0.320401  0.891519
6   0.129652  0.051261  0.680567
7   0.149916  0.698195  0.850444
8   0.155636       NaN  0.802028
9   0.168378  0.557170  0.489537
10  0.179746  0.661873  0.880335
11  0.204883  0.341727  0.829721
12  0.211626  0.081395  0.734220
13  0.234977       NaN  0.015129
14  0.299747  0.478717  0.980622
15  0.303940  0.345172       NaN
16  0.321646  0.192926  0.220307
17  0.349484  0.408327  0.290952
18  0.404118  0.951634  0.493831
19  0.406155  0.930275  0.166535
20  0.423678  0.973172       NaN
21  0.425202  0.106107  0.143856
22  0.426031  0.138184  0.267991
23  0.505826  0.467765  0.532039
24  0.515780  0.782208  0.276004
25  0.524563  0.853662  0.141395
26  0.546843  0.863793  0.437292
27  0.549075  0.257612  0.046061
28  0.567145  0.783945  0.919883
29  0.579010  0.739792  0.043529
30  0.585886  0.171073  0.153889
31  0.625366  0.894839       NaN
32  0.661784  0.126078  0.625256
33  0.670654  0.625164  0.132360
34  0.672448  0.183225  0.752738
35  0.754564  0.422554  0.912640
36  0.762205       NaN  0.245951
37  0.804026  0.686420  0.664571
38  0.812705  0.773458       NaN
39  0.883875  0.779802  0.046368
40  0.913128  0.736072  0.363037
41  0.967605  0.559485  0.105401
42  0.981236       NaN  0.488671
43  0.984476  0.909095  0.266424
44       NaN  0.648975  0.476433
```

[ ]

```
# h
df['Column2_bins'] = pd.cut(df['Column2'], bins=5)
print("\nh. DataFrame with second column discretized into 5 bins:")
print(df)
```



```
h. DataFrame with second column discretized into 5 bins:
     Column1   Column2   Column3      Column2_bins
0   0.052093       NaN  0.066741               NaN
1   0.066517  0.779173  0.374544    (0.604, 0.789]
2   0.095470  0.478395       NaN     (0.42, 0.604]
3   0.100548  0.373563  0.137621     (0.236, 0.42]
4   0.104278  0.706917  0.962627    (0.604, 0.789]
5   0.104882  0.320401  0.891519     (0.236, 0.42]
6   0.129652  0.051261  0.680567   (0.0503, 0.236]
7   0.149916  0.698195  0.850444    (0.604, 0.789]
8   0.155636       NaN  0.802028               NaN
9   0.168378  0.557170  0.489537     (0.42, 0.604]
10  0.179746  0.661873  0.880335    (0.604, 0.789]
11  0.204883  0.341727  0.829721     (0.236, 0.42]
12  0.211626  0.081395  0.734220   (0.0503, 0.236]
13  0.234977       NaN  0.015129               NaN
14  0.299747  0.478717  0.980622     (0.42, 0.604]
15  0.303940  0.345172       NaN     (0.236, 0.42]
16  0.321646  0.192926  0.220307   (0.0503, 0.236]
17  0.349484  0.408327  0.290952     (0.236, 0.42]
18  0.404118  0.951634  0.493831    (0.789, 0.973]
19  0.406155  0.930275  0.166535    (0.789, 0.973]
20  0.423678  0.973172       NaN    (0.789, 0.973]
21  0.425202  0.106107  0.143856   (0.0503, 0.236]
22  0.426031  0.138184  0.267991   (0.0503, 0.236]
23  0.505826  0.467765  0.532039     (0.42, 0.604]
24  0.515780  0.782208  0.276004    (0.604, 0.789]
25  0.524563  0.853662  0.141395    (0.789, 0.973]
26  0.546843  0.863793  0.437292    (0.789, 0.973]
27  0.549075  0.257612  0.046061     (0.236, 0.42]
28  0.567145  0.783945  0.919883    (0.604, 0.789]
29  0.579010  0.739792  0.043529    (0.604, 0.789]
30  0.585886  0.171073  0.153889   (0.0503, 0.236]
31  0.625366  0.894839       NaN    (0.789, 0.973]
32  0.661784  0.126078  0.625256   (0.0503, 0.236]
33  0.670654  0.625164  0.132360    (0.604, 0.789]
34  0.672448  0.183225  0.752738   (0.0503, 0.236]
35  0.754564  0.422554  0.912640     (0.42, 0.604]
36  0.762205       NaN  0.245951               NaN
37  0.804026  0.686420  0.664571    (0.604, 0.789]
38  0.812705  0.773458       NaN    (0.604, 0.789]
39  0.883875  0.779802  0.046368    (0.604, 0.789]
40  0.913128  0.736072  0.363037    (0.604, 0.789]
41  0.967605  0.559485  0.105401     (0.42, 0.604]
42  0.981236       NaN  0.488671               NaN
43  0.984476  0.909095  0.266424    (0.789, 0.973]
44       NaN  0.648975  0.476433    (0.604, 0.789]
```

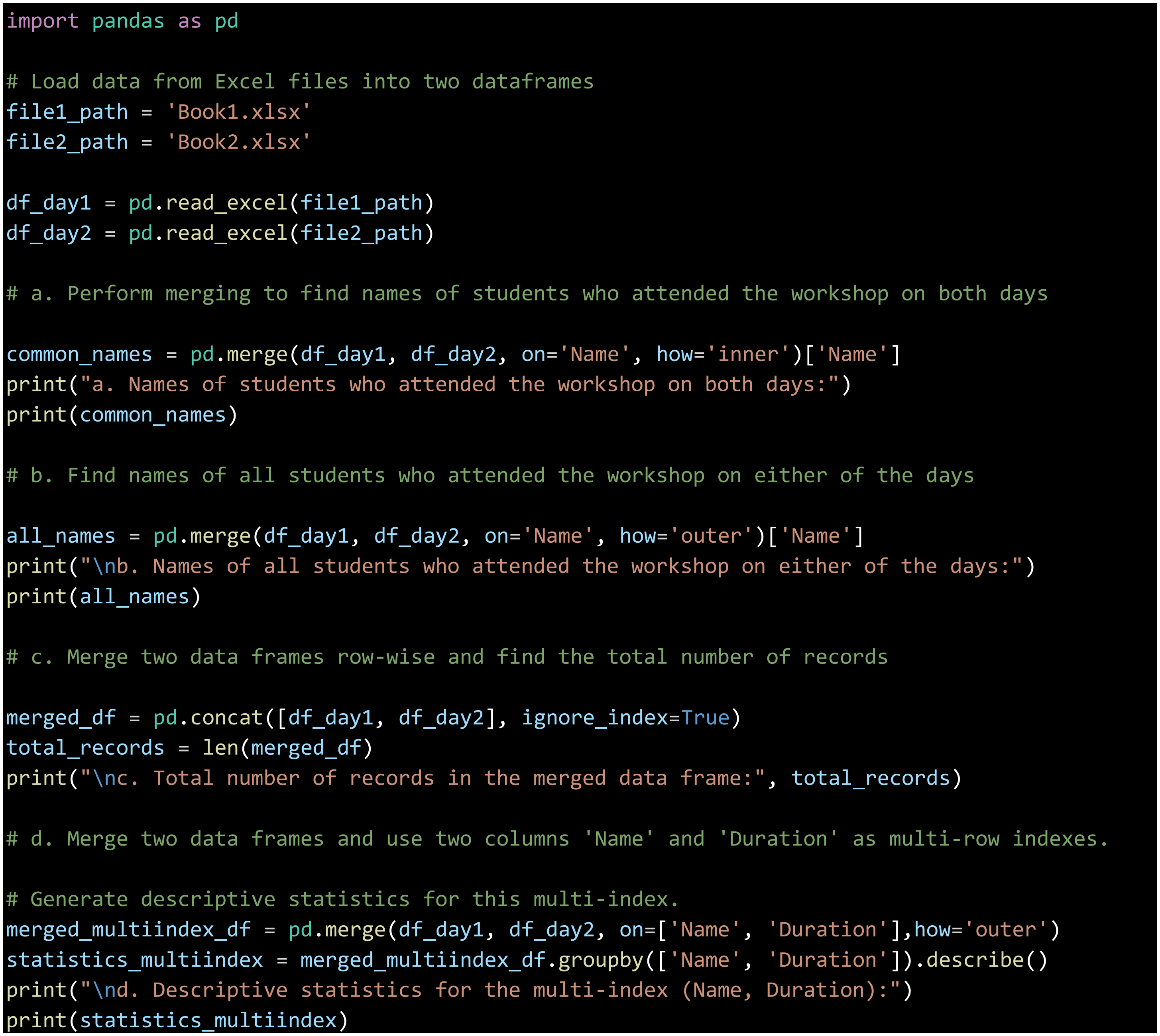
4.Consider two excel files having attendance of a workshop’s participants for two days. Each file has three fields ‘Name’, ‘Time of joining’, duration (in minutes) where names are unique within a file. Note that duration may take one of three values (30, 40, 50) only. Import the data into two dataframes 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.

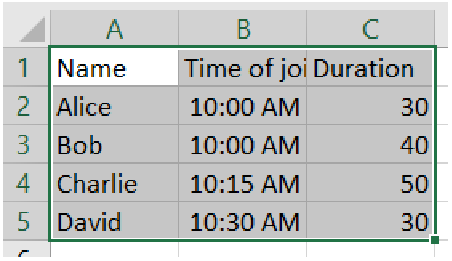
c. Merge two data frames row-wise and find the total number of records in the data frame.

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



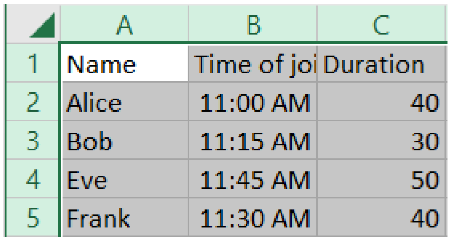
```
import pandas as pd

# Load data from Excel files into two dataframes
file1_path = 'Book1.xlsx'
file2_path = 'Book2.xlsx'

df_day1 = pd.read_excel(file1_path)
df_day2 = pd.read_excel(file2_path)

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

common_names = pd.merge(df_day1, df_day2, on='Name', how='inner')['Name']
print("a. Names of students who attended the workshop on both days:")
print(common_names)

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

all_names = pd.merge(df_day1, df_day2, on='Name', how='outer')['Name']
print("\nb. Names of all students who attended the workshop on either of the days:")
print(all_names)

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

merged_df = pd.concat([df_day1, df_day2], ignore_index=True)
total_records = len(merged_df)
print("\nc. Total number of records in the merged data frame:", total_records)

# d. Merge two data frames and use two columns 'Name' and 'Duration' as multi-row indexes.

# Generate descriptive statistics for this multi-index.
merged_multiindex_df = pd.merge(df_day1, df_day2, on=['Name', 'Duration'],how='outer')
statistics_multiindex = merged_multiindex_df.groupby(['Name', 'Duration']).describe()
print("\nd. Descriptive statistics for the multi-index (Name, Duration):")
print(statistics_multiindex)
```

BOOK1

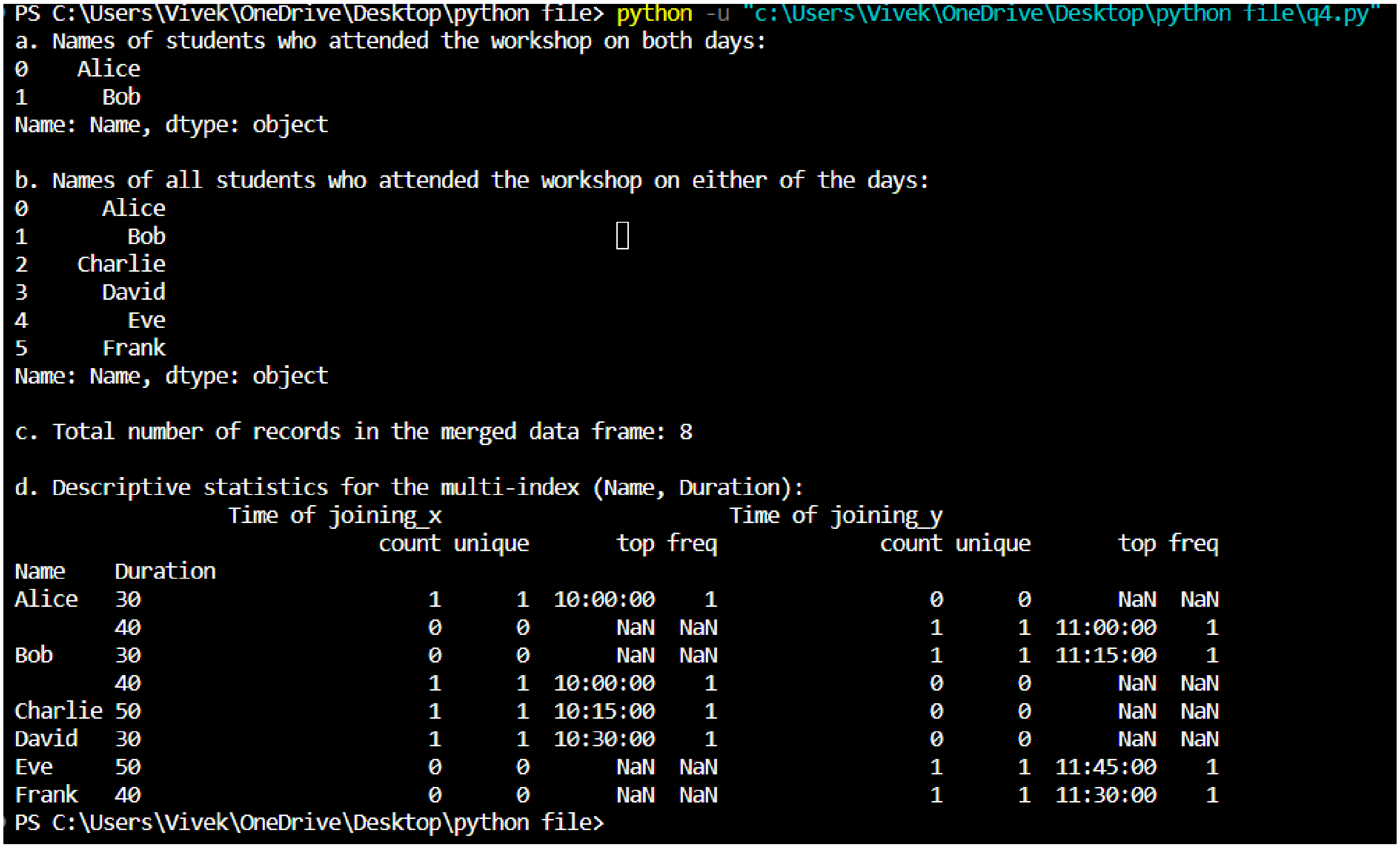


	A	B	C
1	Name	Time of joi	Duration
2	Alice	10:00 AM	30
3	Bob	10:00 AM	40
4	Charlie	10:15 AM	50
5	David	10:30 AM	30

BOOK2



	A	B	C
1	Name	Time of joi	Duration
2	Alice	11:00 AM	40
3	Bob	11:15 AM	30
4	Eve	11:45 AM	50
5	Frank	11:30 AM	40



```
PS C:\Users\Vivek\OneDrive\Desktop\python file> python -u "c:\Users\Vivek\OneDrive\Desktop\python file\q4.py"
a. Names of students who attended the workshop on both days:
0    Alice
1      Bob
Name: Name, dtype: object

b. Names of all students who attended the workshop on either of the days:
0      Alice
1        Bob
2    Charlie
3      David
4        Eve
5      Frank
Name: Name, dtype: object

c. Total number of records in the merged data frame: 8

d. Descriptive statistics for the multi-index (Name, Duration):
                 Time of joining_x                       Time of joining_y
                             count unique       top freq             count unique       top freq
Name    Duration
Alice   30                       1      1  10:00:00    1                 0      0       NaN  NaN
        40                       0      0       NaN  NaN                 1      1  11:00:00    1
Bob     30                       0      0       NaN  NaN                 1      1  11:15:00    1
        40                       1      1  10:00:00    1                 0      0       NaN  NaN
Charlie 50                       1      1  10:15:00    1                 0      0       NaN  NaN
David   30                       1      1  10:30:00    1                 0      0       NaN  NaN
Eve     50                       0      0       NaN  NaN                 1      1  11:45:00    1
Frank   40                       0      0       NaN  NaN                 1      1  11:30:00    1
PS C:\Users\Vivek\OneDrive\Desktop\python file>
```

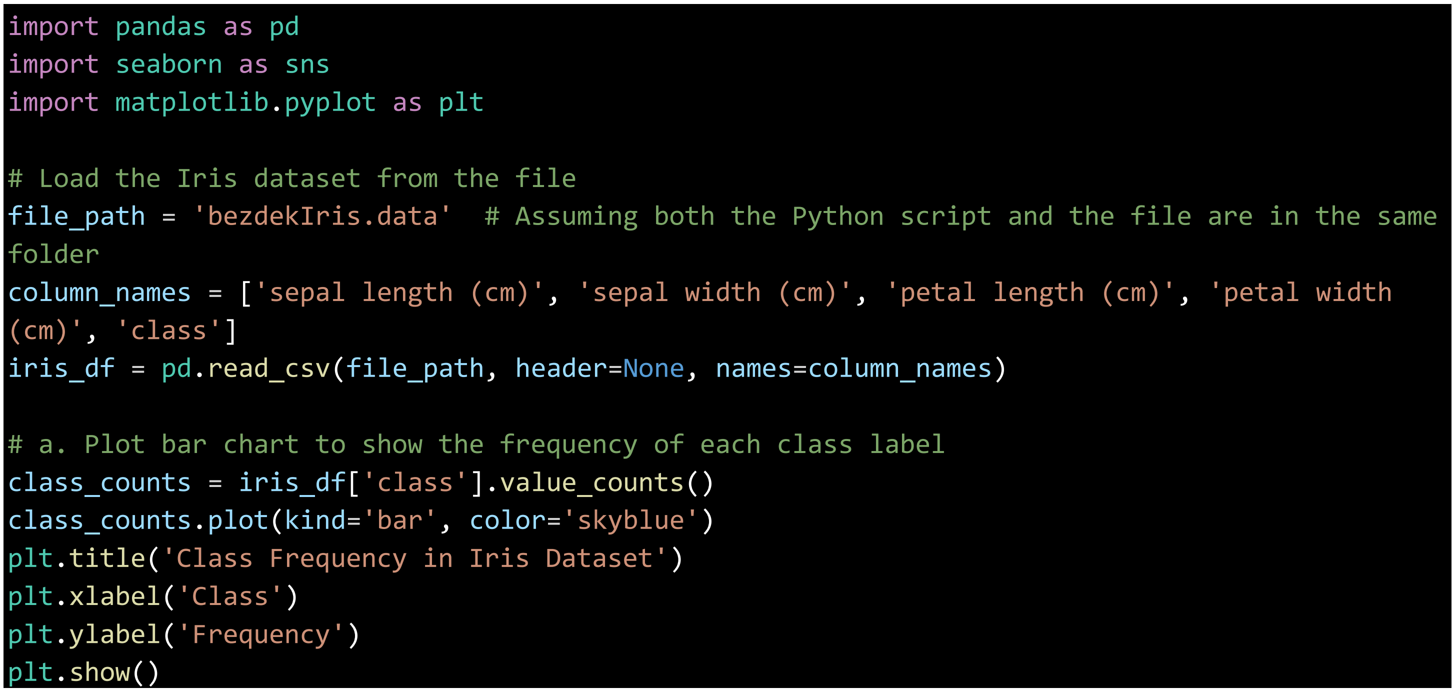
5. Taking Iris data, plot the following with proper legend and axis labels: (Download IRIS data from: https://archive.ics.uci.edu/ml/datasets/iris or import it from sklearn.datasets)

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

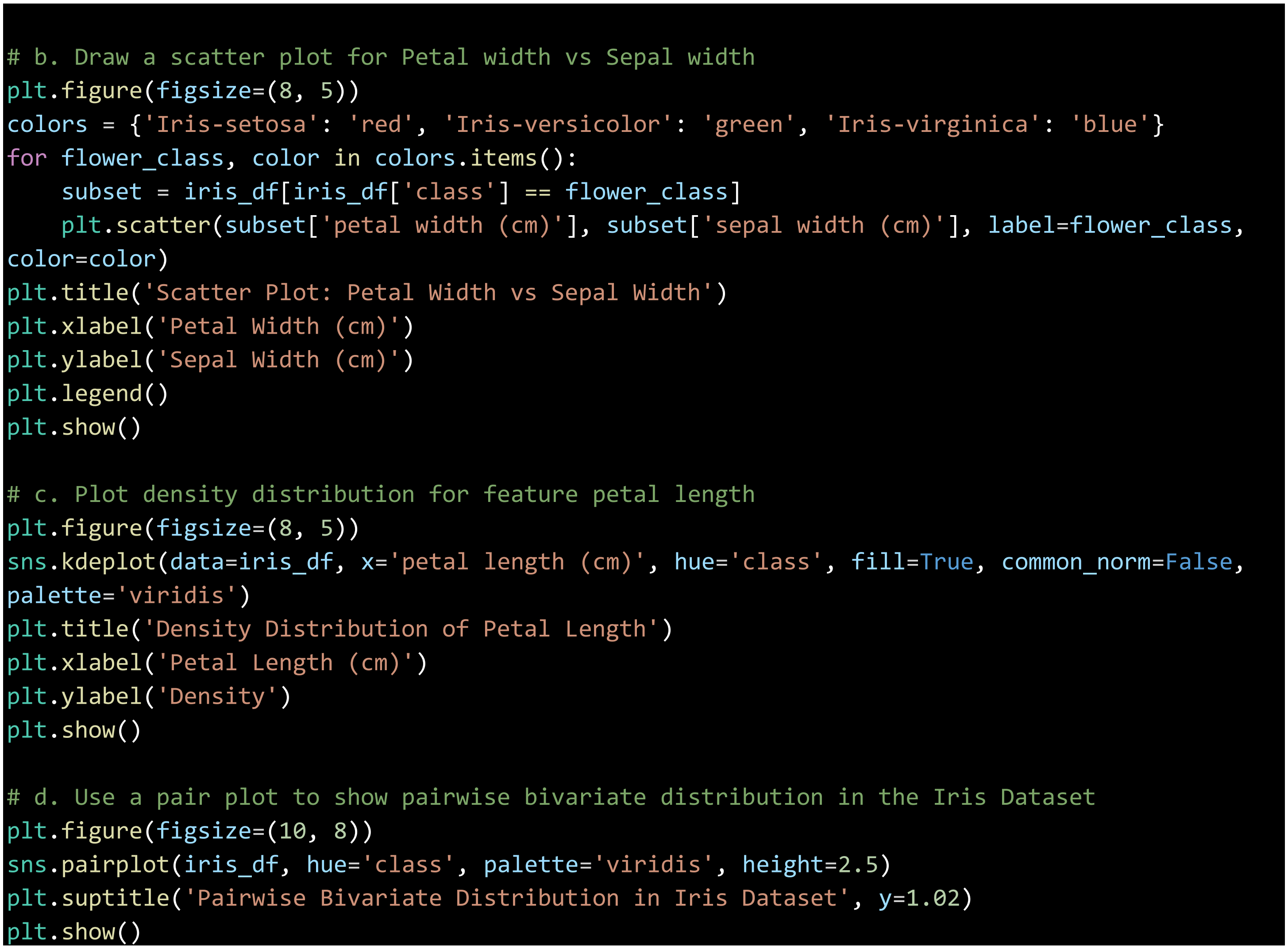
b. Draw a scatter plot for Petal width vs sepal width.

c. Plot density distribution for feature petal length.

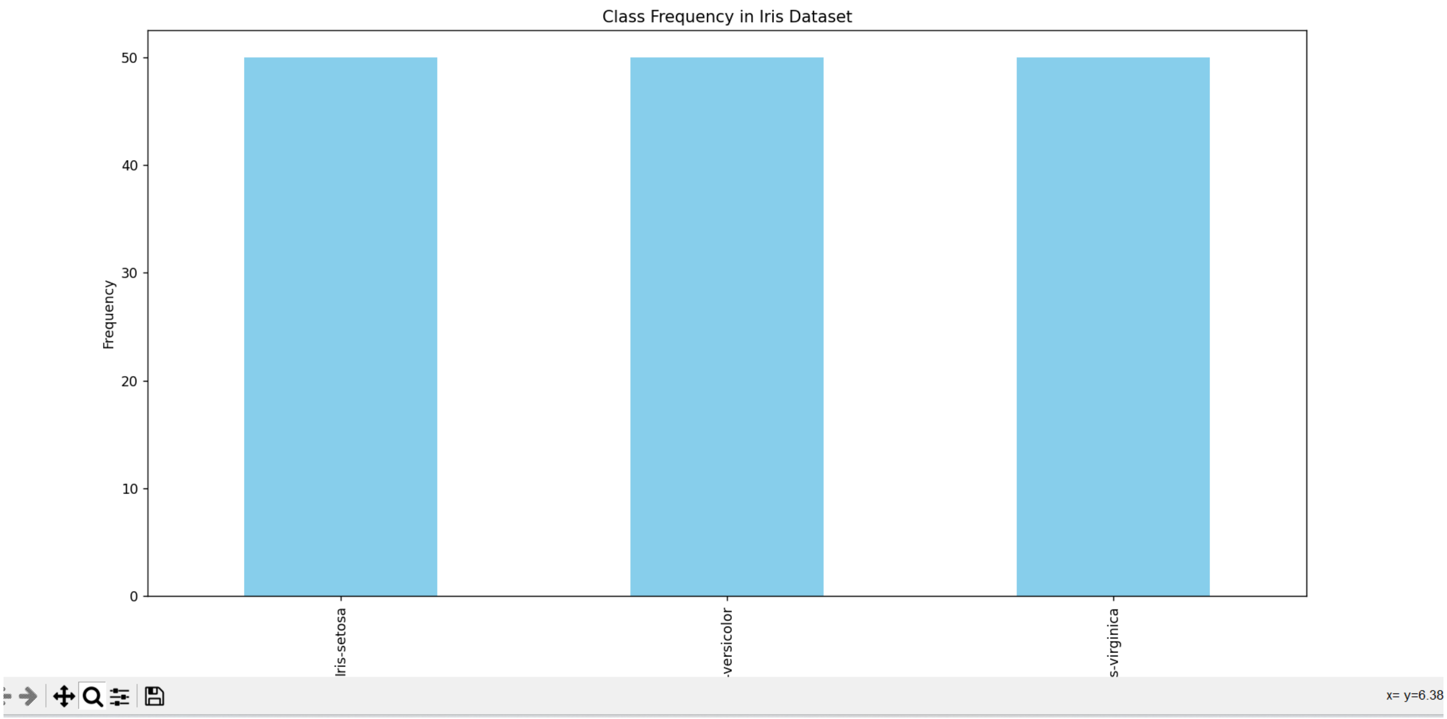
d. Use a pair plot to show pairwise bivariate distribution in the Iris Dataset

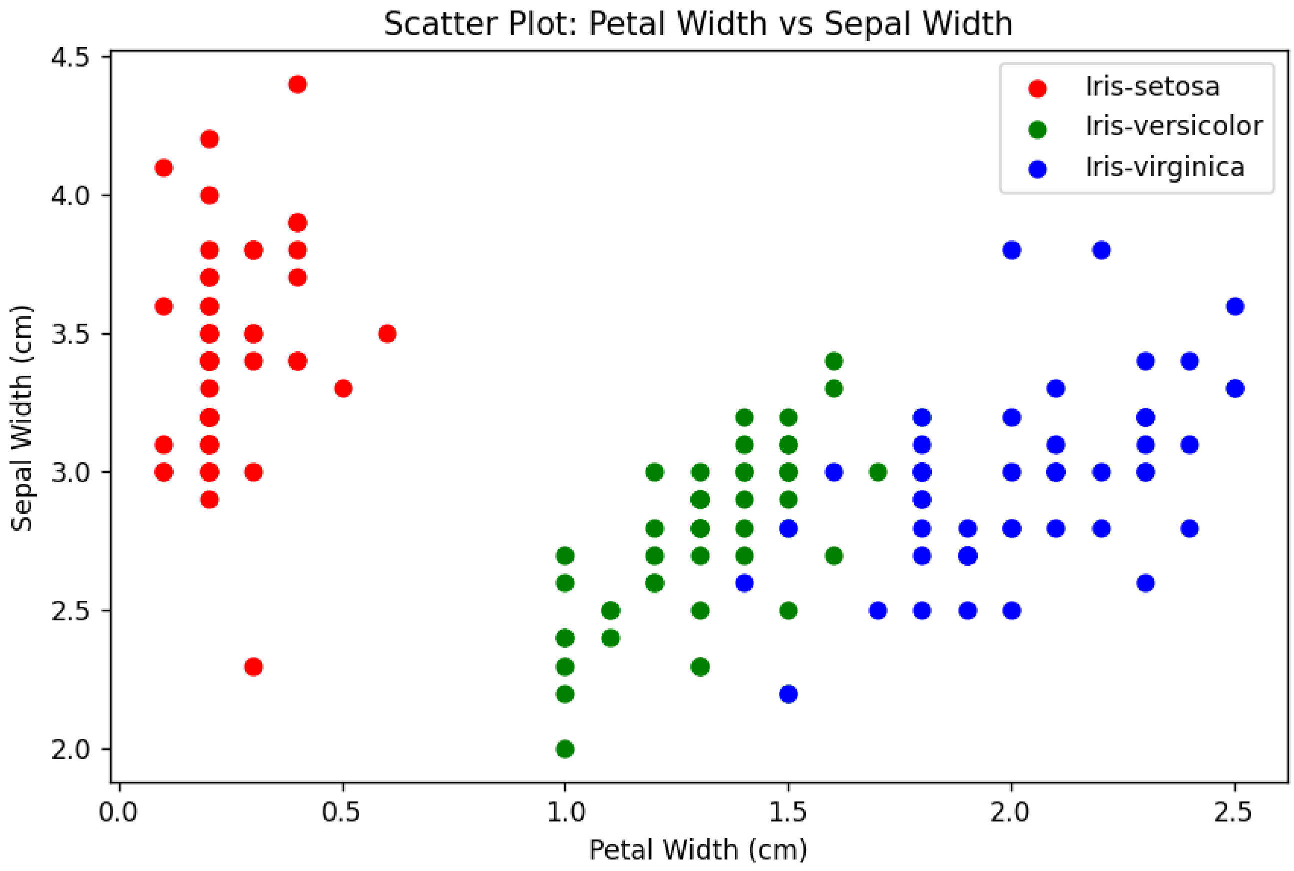


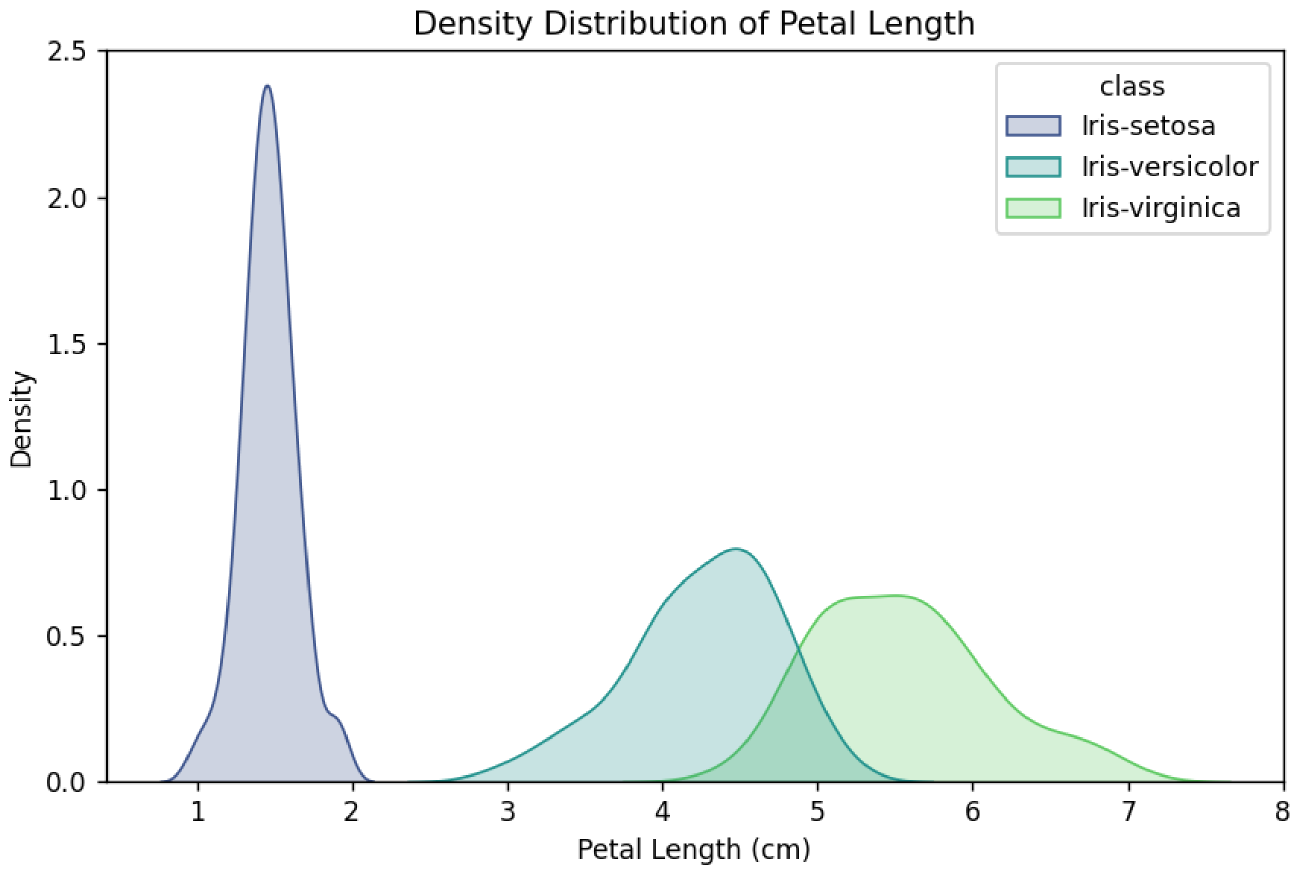
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the Iris dataset from the file
file_path = 'bezdekIris.data'  # Assuming both the Python script and the file are in the same
folder
column_names = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)', 'class']
iris_df = pd.read_csv(file_path, header=None, names=column_names)

# a. Plot bar chart to show the frequency of each class label
class_counts = iris_df['class'].value_counts()
class_counts.plot(kind='bar', color='skyblue')
plt.title('Class Frequency in Iris Dataset')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```

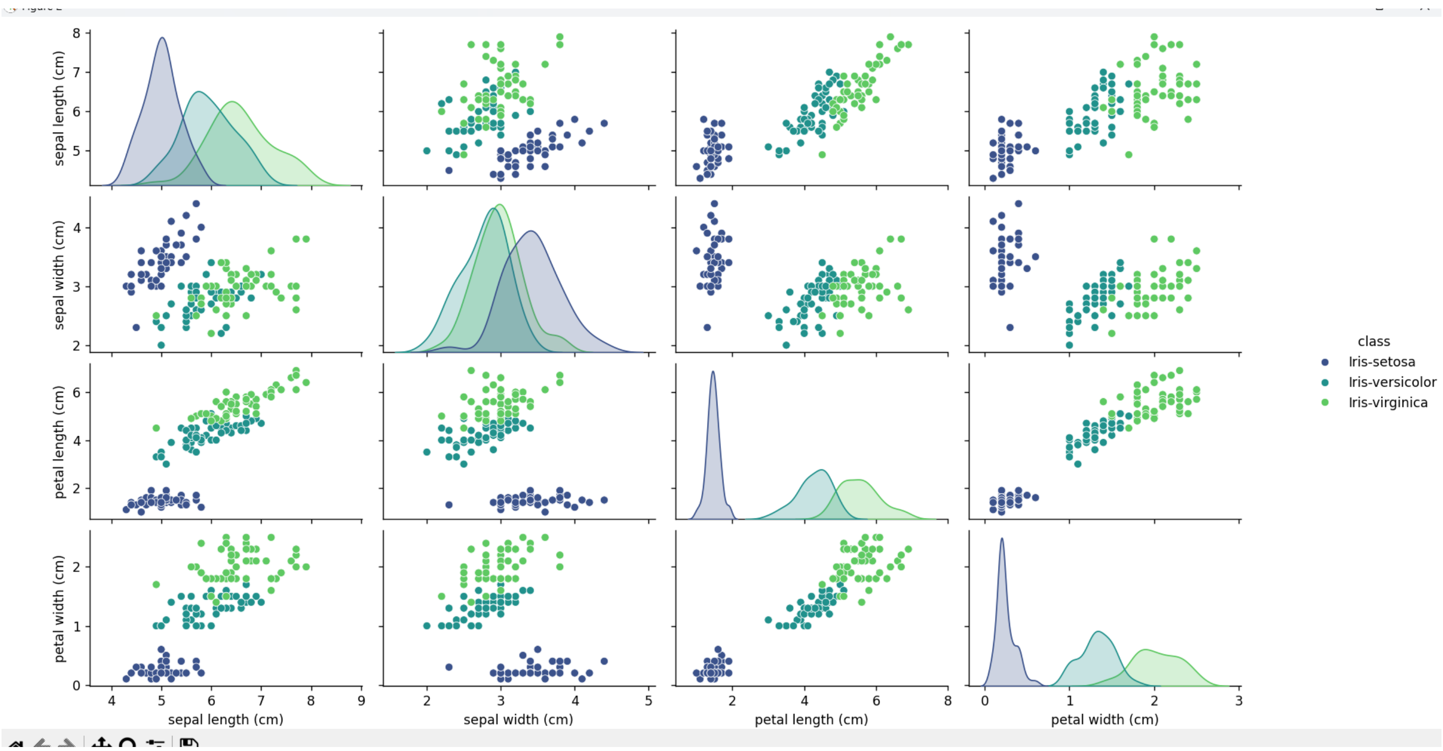


```
# b. Draw a scatter plot for Petal width vs Sepal width
plt.figure(figsize=(8, 5))
colors = {'Iris-setosa': 'red', 'Iris-versicolor': 'green', 'Iris-virginica': 'blue'}
for flower_class, color in colors.items():
    subset = iris_df[iris_df['class'] == flower_class]
    plt.scatter(subset['petal width (cm)'], subset['sepal width (cm)'], label=flower_class,
color=color)
plt.title('Scatter Plot: Petal Width vs Sepal Width')
plt.xlabel('Petal Width (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend()
plt.show()

# c. Plot density distribution for feature petal length
plt.figure(figsize=(8, 5))
sns.kdeplot(data=iris_df, x='petal length (cm)', hue='class', fill=True, common_norm=False,
palette='viridis')
plt.title('Density Distribution of Petal Length')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Density')
plt.show()

# d. Use a pair plot to show pairwise bivariate distribution in the Iris Dataset
plt.figure(figsize=(10, 8))
sns.pairplot(iris_df, hue='class', palette='viridis', height=2.5)
plt.suptitle('Pairwise Bivariate Distribution in Iris Dataset', y=1.02)
plt.show()
```




Scatter Plot: Petal Width vs Sepal Width
Sepal Width (cm)
Petal Width (cm)
4.5
4.0
3.5
3.0
2.5
2.0
0.0
0.5
1.0
1.5
2.0
2.5
Iris-setosa
Iris-versicolor
Iris-virginica


Density Distribution of Petal Length
Density
Petal Length (cm)
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7
8
class
Iris-setosa
Iris-versicolor
Iris-virginica



6.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.

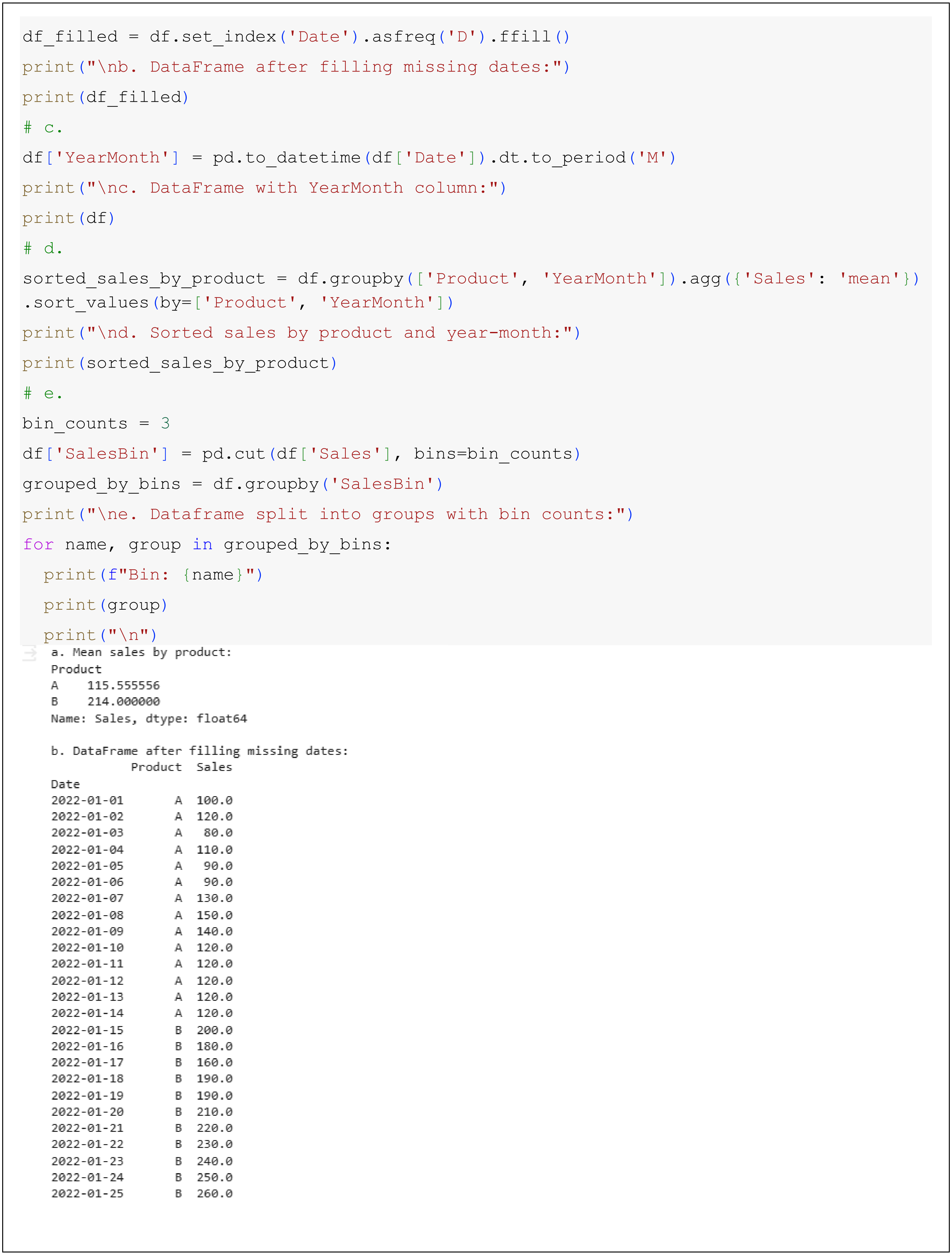
e. Split a given dataframe into groups with bin counts.

---

[1]

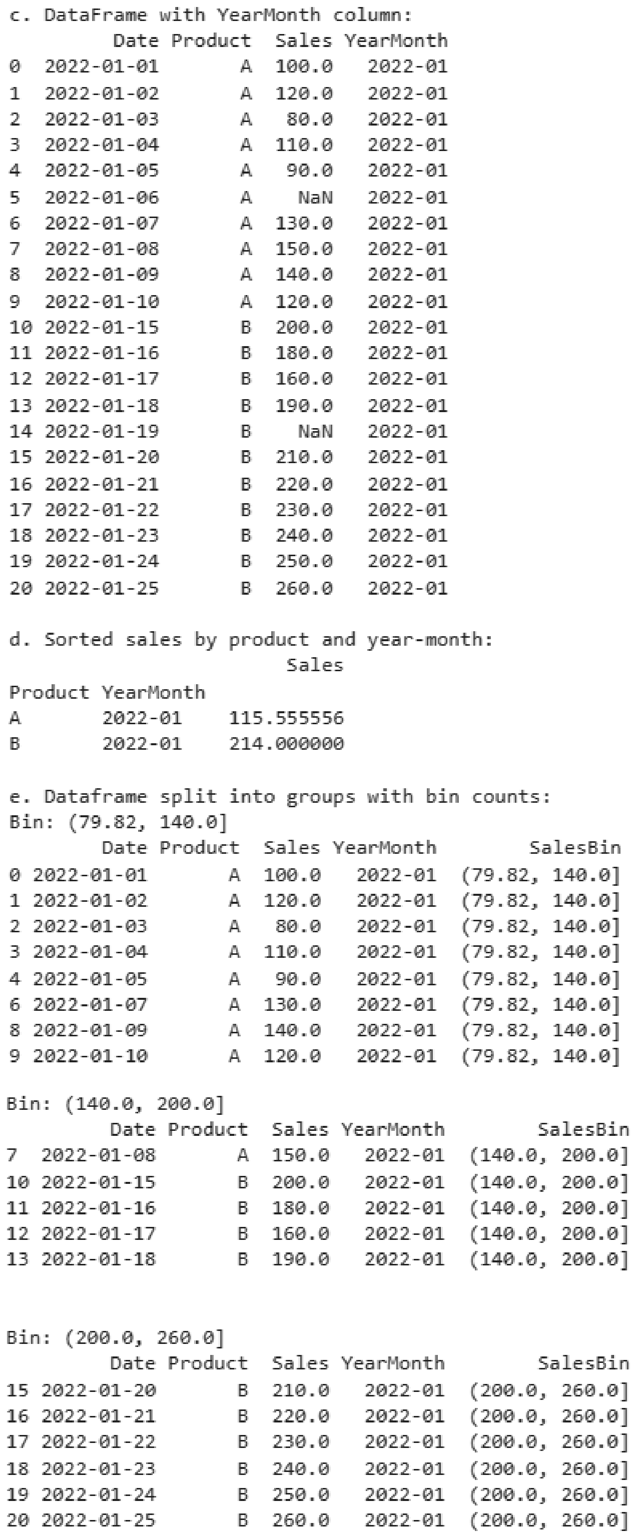
0s

```
import pandas as pd
import numpy as np
data = {
'Date': pd.date_range(start='2022-01-01', end='2022-01-10', freq='D').tolist() +
pd.date_range(start='2022-01-15', end='2022-01-25', freq='D').tolist(),
'Product': ['A'] * 10 + ['B'] * 11,
'Sales': [100, 120, 80, 110, 90, np.nan, 130, 150, 140, 120, 200, 180, 160, 190, np.nan, 210, 220, 230, 240, 250, 260]
}
df = pd.DataFrame(data)
# a.
mean_sales_by_product = df.groupby('Product')['Sales'].mean()
print("a. Mean sales by product:")
print(mean_sales_by_product)
# b.
```



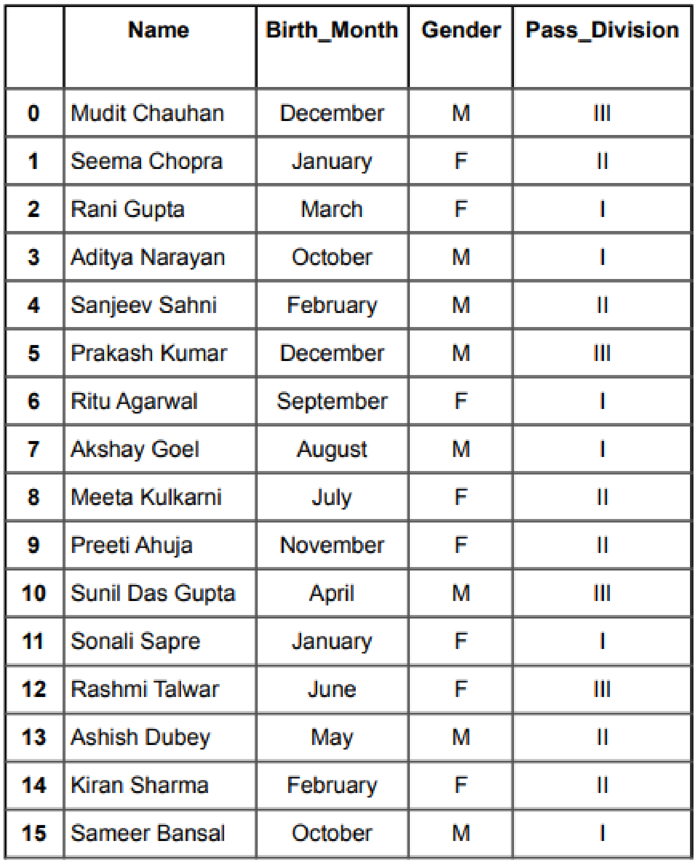
```python
df_filled = df.set_index('Date').asfreq('D').ffill()
print("\nb. DataFrame after filling missing dates:")
print(df_filled)
# c.
df['YearMonth'] = pd.to_datetime(df['Date']).dt.to_period('M')
print("\nc. DataFrame with YearMonth column:")
print(df)
# d.
sorted_sales_by_product = df.groupby(['Product', 'YearMonth']).agg({'Sales': 'mean'})
.sort_values(by=['Product', 'YearMonth'])
print("\nd. Sorted sales by product and year-month:")
print(sorted_sales_by_product)
# e.
bin_counts = 3
df['SalesBin'] = pd.cut(df['Sales'], bins=bin_counts)
grouped_by_bins = df.groupby('SalesBin')
print("\ne. Dataframe split into groups with bin counts:")
for name, group in grouped_by_bins:
  print(f"Bin: {name}")
  print(group)
  print("\n")
```

```
a. Mean sales by product:
Product
A    115.555556
B    214.000000
Name: Sales, dtype: float64

b. DataFrame after filling missing dates:
           Product  Sales
Date
2022-01-01       A  100.0
2022-01-02       A  120.0
2022-01-03       A   80.0
2022-01-04       A  110.0
2022-01-05       A   90.0
2022-01-06       A   90.0
2022-01-07       A  130.0
2022-01-08       A  150.0
2022-01-09       A  140.0
2022-01-10       A  120.0
2022-01-11       A  120.0
2022-01-12       A  120.0
2022-01-13       A  120.0
2022-01-14       A  120.0
2022-01-15       B  200.0
2022-01-16       B  180.0
2022-01-17       B  160.0
2022-01-18       B  190.0
2022-01-19       B  190.0
2022-01-20       B  210.0
2022-01-21       B  220.0
2022-01-22       B  230.0
2022-01-23       B  240.0
2022-01-24       B  250.0
2022-01-25       B  260.0
```



```
c. DataFrame with YearMonth column:
         Date Product  Sales YearMonth
0  2022-01-01       A  100.0   2022-01
1  2022-01-02       A  120.0   2022-01
2  2022-01-03       A   80.0   2022-01
3  2022-01-04       A  110.0   2022-01
4  2022-01-05       A   90.0   2022-01
5  2022-01-06       A    NaN   2022-01
6  2022-01-07       A  130.0   2022-01
7  2022-01-08       A  150.0   2022-01
8  2022-01-09       A  140.0   2022-01
9  2022-01-10       A  120.0   2022-01
10 2022-01-15       B  200.0   2022-01
11 2022-01-16       B  180.0   2022-01
12 2022-01-17       B  160.0   2022-01
13 2022-01-18       B  190.0   2022-01
14 2022-01-19       B    NaN   2022-01
15 2022-01-20       B  210.0   2022-01
16 2022-01-21       B  220.0   2022-01
17 2022-01-22       B  230.0   2022-01
18 2022-01-23       B  240.0   2022-01
19 2022-01-24       B  250.0   2022-01
20 2022-01-25       B  260.0   2022-01

d. Sorted sales by product and year-month:
                        Sales
Product YearMonth
A       2022-01    115.555556
B       2022-01    214.000000

e. Dataframe split into groups with bin counts:
Bin: (79.82, 140.0]
        Date Product  Sales YearMonth        SalesBin
0 2022-01-01       A  100.0   2022-01  (79.82, 140.0]
1 2022-01-02       A  120.0   2022-01  (79.82, 140.0]
2 2022-01-03       A   80.0   2022-01  (79.82, 140.0]
3 2022-01-04       A  110.0   2022-01  (79.82, 140.0]
4 2022-01-05       A   90.0   2022-01  (79.82, 140.0]
6 2022-01-07       A  130.0   2022-01  (79.82, 140.0]
8 2022-01-09       A  140.0   2022-01  (79.82, 140.0]
9 2022-01-10       A  120.0   2022-01  (79.82, 140.0]

Bin: (140.0, 200.0]
         Date Product  Sales YearMonth        SalesBin
7  2022-01-08       A  150.0   2022-01  (140.0, 200.0]
10 2022-01-15       B  200.0   2022-01  (140.0, 200.0]
11 2022-01-16       B  180.0   2022-01  (140.0, 200.0]
12 2022-01-17       B  160.0   2022-01  (140.0, 200.0]
13 2022-01-18       B  190.0   2022-01  (140.0, 200.0]


Bin: (200.0, 260.0]
         Date Product  Sales YearMonth        SalesBin
15 2022-01-20       B  210.0   2022-01  (200.0, 260.0]
16 2022-01-21       B  220.0   2022-01  (200.0, 260.0]
17 2022-01-22       B  230.0   2022-01  (200.0, 260.0]
18 2022-01-23       B  240.0   2022-01  (200.0, 260.0]
19 2022-01-24       B  250.0   2022-01  (200.0, 260.0]
20 2022-01-25       B  260.0   2022-01  (200.0, 260.0]
```

7. 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.

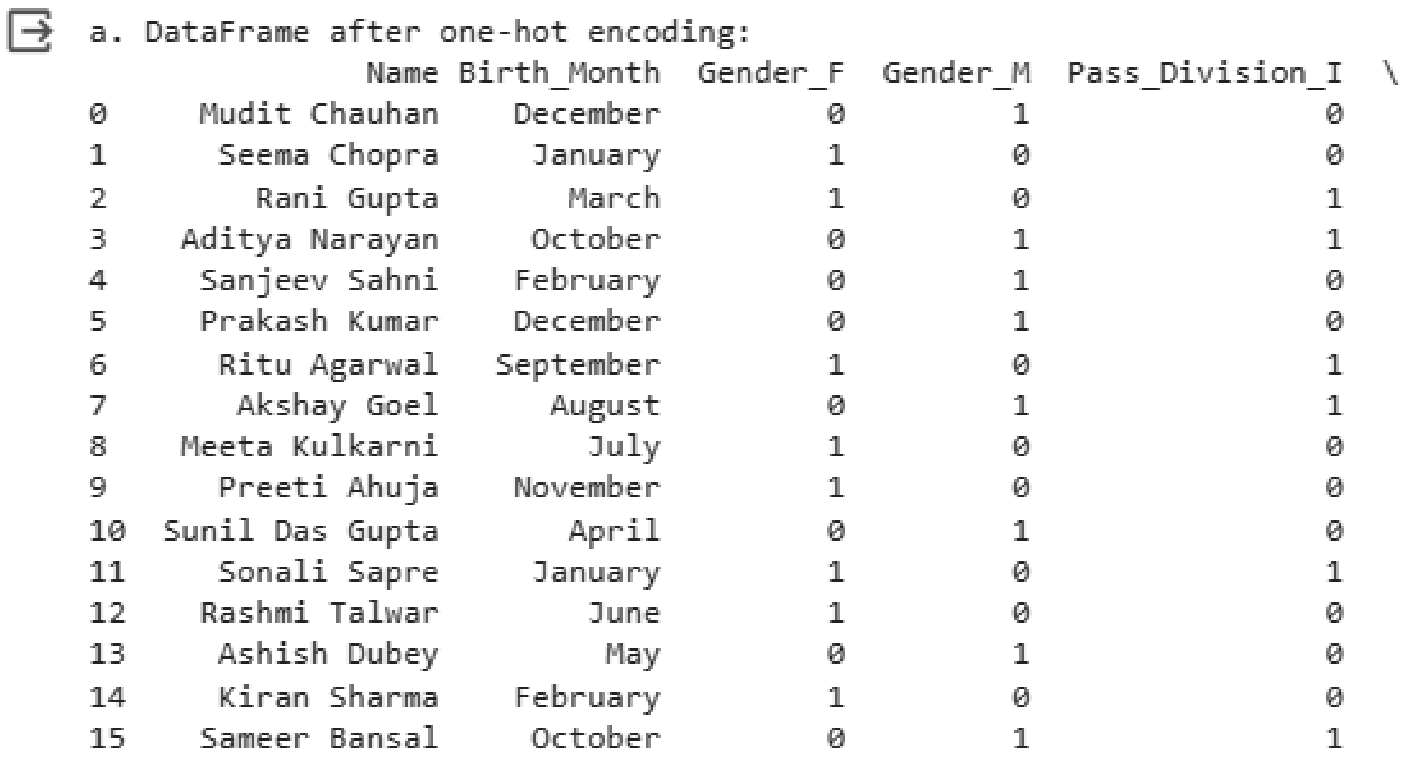
b. Sort this data frame on the “Birth Month” column (i.e. January to December). Hint: Convert Month to Categorical.

[3]

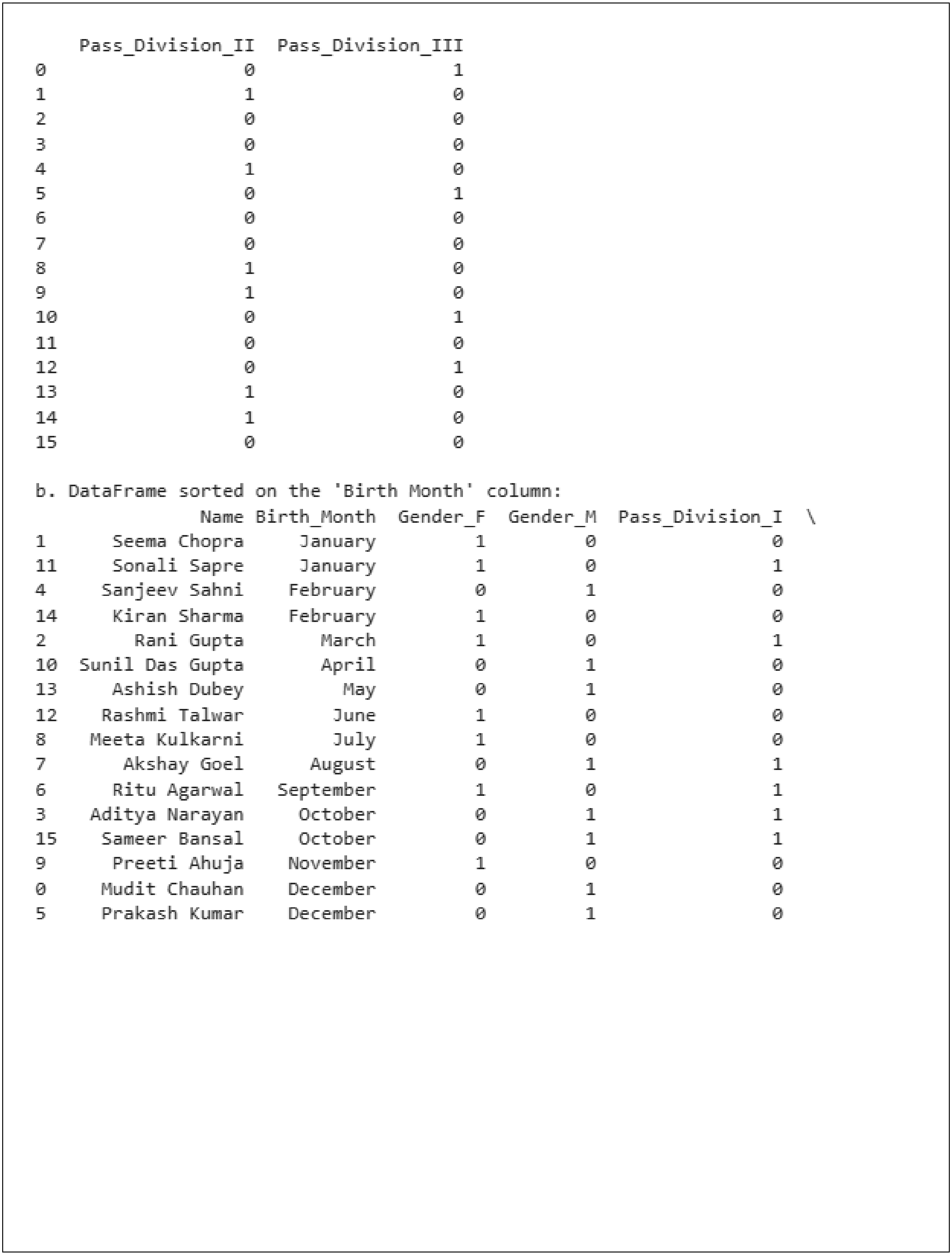
0s

```
import pandas as pd
# Your provided data
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']
}
```

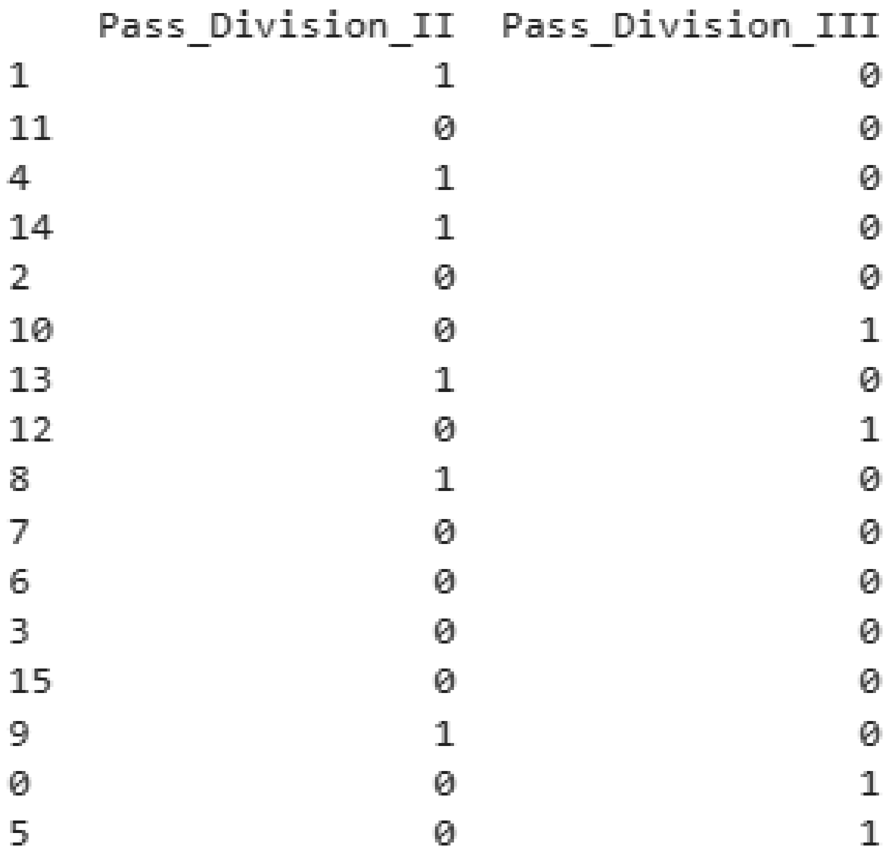
```
df = pd.DataFrame(data)
# a. Perform one hot encoding of the last two columns of categorical data using the get_dummies() function.
df_encoded = pd.get_dummies(df, columns=['Gender', 'Pass_Division'])
# b. Sort this data frame on the “Birth Month” column.
# Convert 'Birth_Month' to Categorical with custom order
month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August','September', 'October', 'November', 'December']
df_encoded['Birth_Month'] = pd.Categorical(df_encoded['Birth_Month'],categories=month_order, ordered=True)
# Sort the DataFrame based on 'Birth_Month'
df_sorted = df_encoded.sort_values(by='Birth_Month')
# Display the results
print("a. DataFrame after one-hot encoding:")
print(df_encoded)
print("\nb. DataFrame sorted on the 'Birth Month' column:")
print(df_sorted)
```



```
a. DataFrame after one-hot encoding:
               Name Birth_Month  Gender_F  Gender_M  Pass_Division_I  \
0     Mudit Chauhan    December         0         1                0
1      Seema Chopra     January         1         0                0
2        Rani Gupta       March         1         0                1
3    Aditya Narayan     October         0         1                1
4     Sanjeev Sahni    February         0         1                0
5     Prakash Kumar    December         0         1                0
6      Ritu Agarwal   September         1         0                1
7       Akshay Goel      August         0         1                1
8    Meeta Kulkarni        July         1         0                0
9      Preeti Ahuja    November         1         0                0
10  Sunil Das Gupta       April         0         1                0
11     Sonali Sapre     January         1         0                1
12    Rashmi Talwar        June         1         0                0
13     Ashish Dubey         May         0         1                0
14     Kiran Sharma    February         1         0                0
15    Sameer Bansal     October         0         1                1
```



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

b. DataFrame sorted on the 'Birth Month' column:
               Name Birth_Month  Gender_F  Gender_M  Pass_Division_I  \
1      Seema Chopra     January         1         0                0
11     Sonali Sapre     January         1         0                1
4     Sanjeev Sahni    February         0         1                0
14     Kiran Sharma    February         1         0                0
2        Rani Gupta       March         1         0                1
10  Sunil Das Gupta       April         0         1                0
13     Ashish Dubey         May         0         1                0
12    Rashmi Talwar        June         1         0                0
8    Meeta Kulkarni        July         1         0                0
7       Akshay Goel      August         0         1                1
6      Ritu Agarwal   September         1         0                1
3    Aditya Narayan     October         0         1                1
15    Sameer Bansal     October         0         1                1
9      Preeti Ahuja    November         1         0                0
0     Mudit Chauhan    December         0         1                0
5     Prakash Kumar    December         0         1                0
```



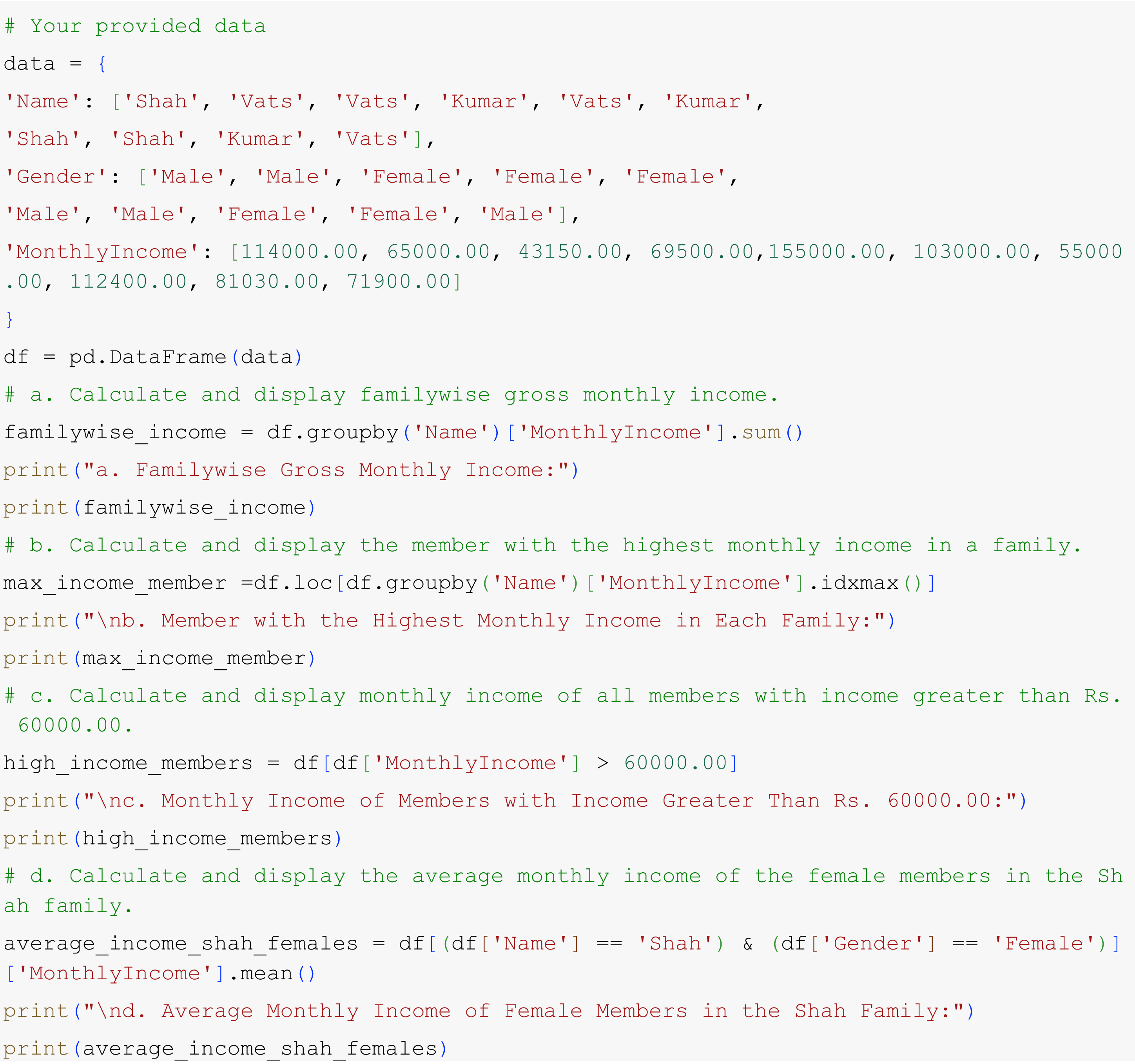
```
    Pass_Division_II  Pass_Division_III
1                  1                  0
11                 0                  0
4                  1                  0
14                 1                  0
2                  0                  0
10                 0                  1
13              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

```
8. 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.
```

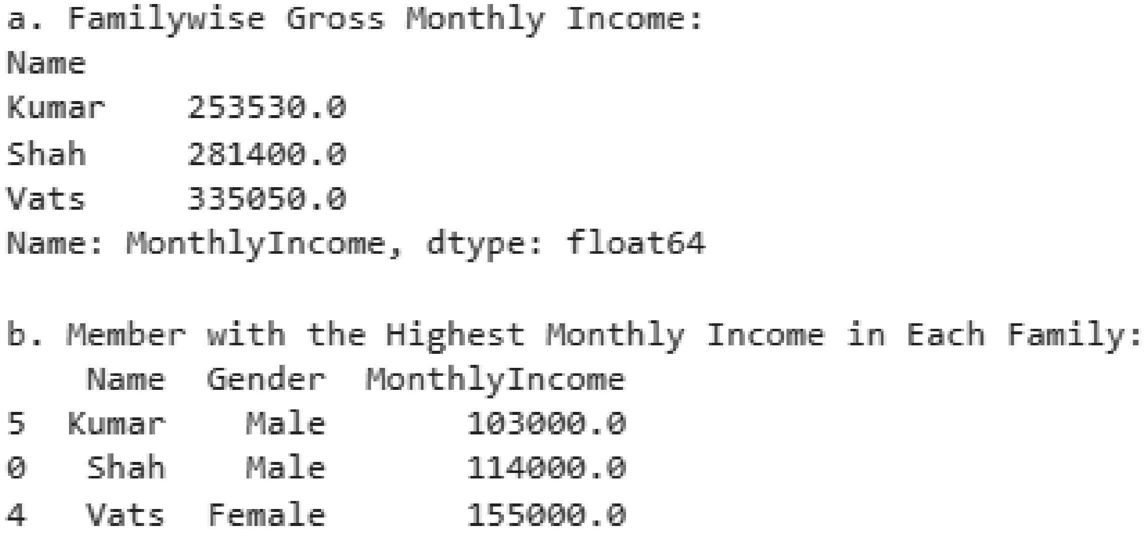
[5]

0s

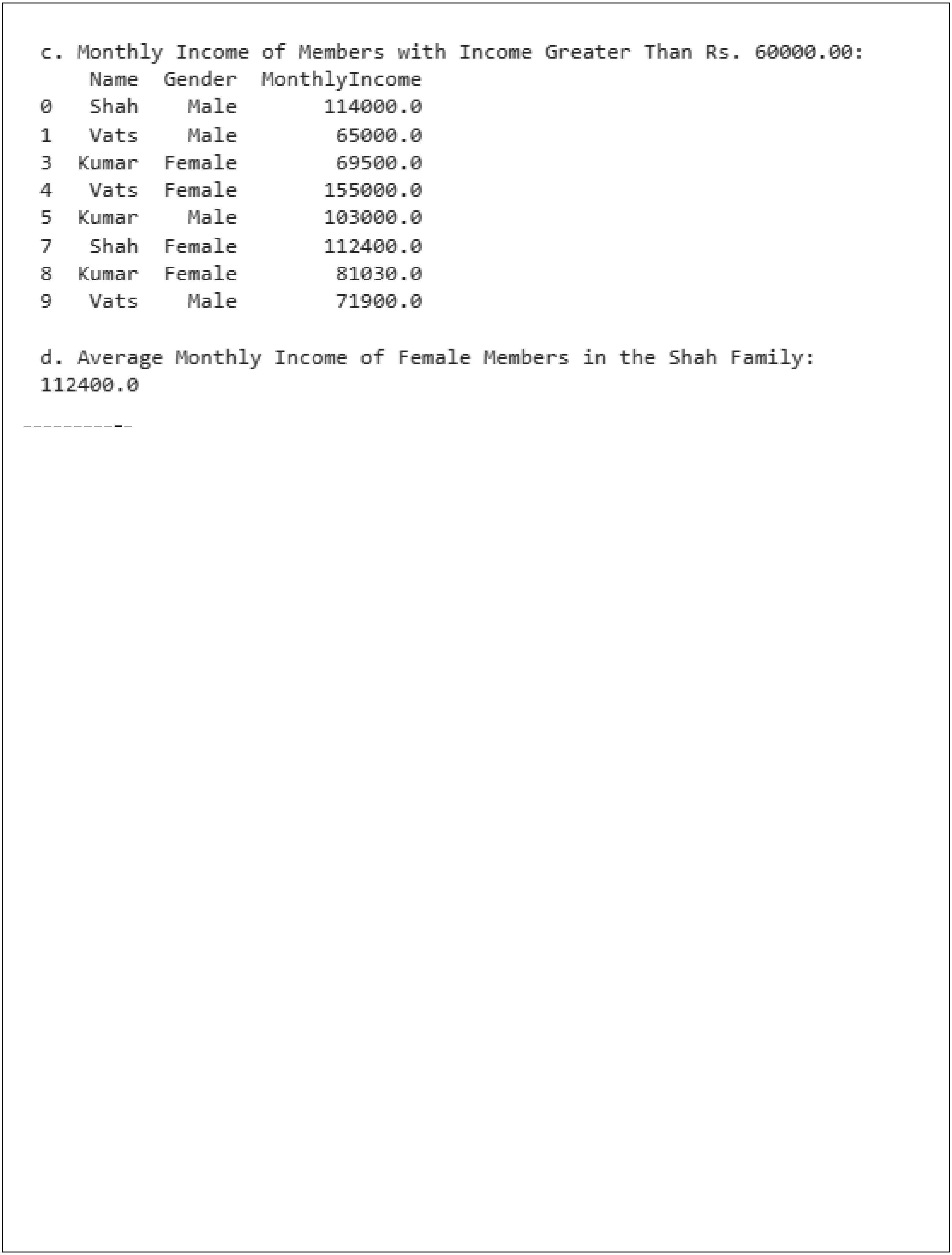
```python
import pandas as pd
```



```
# Your provided data
data = {
'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(data)
# a. Calculate and display familywise gross monthly income.
familywise_income = df.groupby('Name')['MonthlyIncome'].sum()
print("a. Familywise Gross Monthly Income:")
print(familywise_income)
# b. Calculate and display the member with the highest monthly income in a family.
max_income_member =df.loc[df.groupby('Name')['MonthlyIncome'].idxmax()]
print("\nb. Member with the Highest Monthly Income in Each Family:")
print(max_income_member)
# c. Calculate and display monthly income of all members with income greater than Rs.
 60000.00.
high_income_members = df[df['MonthlyIncome'] > 60000.00]
print("\nc. Monthly Income of Members with Income Greater Than Rs. 60000.00:")
print(high_income_members)
# d. Calculate and display the average monthly income of the female members in the Sh
ah family.
average_income_shah_females = df[(df['Name'] == 'Shah') & (df['Gender'] == 'Female')]
['MonthlyIncome'].mean()
print("\nd. Average Monthly Income of Female Members in the Shah Family:")
print(average_income_shah_females)
```



```
a. Familywise Gross Monthly Income:
Name
Kumar    253530.0
Shah     281400.0
Vats     335050.0
Name: MonthlyIncome, dtype: float64

b. Member with the Highest Monthly Income in Each Family:
    Name  Gender  MonthlyIncome
5  Kumar    Male       103000.0
0   Shah    Male       114000.0
4   Vats  Female       155000.0
```



```
c. Monthly Income of Members with Income Greater Than Rs. 60000.00:
    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. Average Monthly Income of Female Members in the Shah Family:
112400.0
```

```
-----------
```