"Only Knowledge can provide salvation"

KESHAV MAHAVIDYALAYA

UNIVERSITY OF DELHI

DATA ANALYSIS AND VISUALIZATION

Practical File

Submitted To:

Ms. Nidhi Passi

Submitted By:

Bhoomika Singh (2110834)
B.Sc. (H) Computer Science
Section - A
Exam Roll no. - 21035570019
Semester - V

INDEX

PRACTICAL – 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:

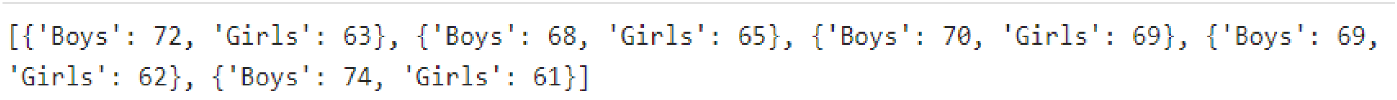
Original dictionary of lists:
{'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65, 69, 62, 61]}

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].

Code:

```
students = {"Boys":[72,68,70,69,74],"Girls":[63,65,69,62,61]}
data=[]

for i in range(len(students["Boys"])):
    x={}
    x["Boys"]=students["Boys"][i]
    x["Girls"]=students["Girls"][i]
    data.append(x)

print(data)
```

Output:



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

PRACTICAL – 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 n x 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.

Code:

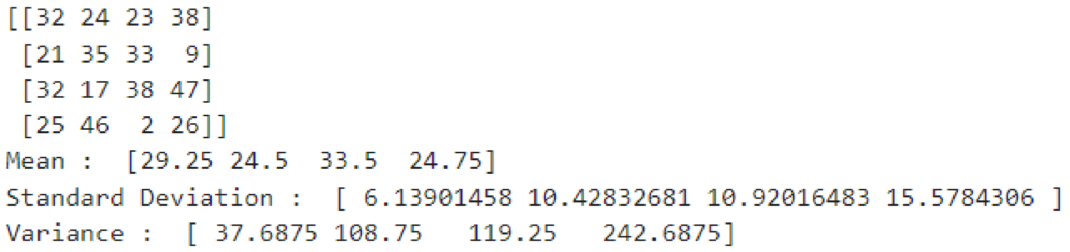
```
import numpy as np
from numpy import random
arr1 = np.random.randint(50,size=(4,4))
print(arr1)

# part a.
print("Mean : ",arr1.mean(axis=1))
print("Standard Deviation : ",arr1.std(axis=1))
print("Variance : ",arr1.var(axis=1))

# part b.
B = np.array([56,48,22,41,78,91,24,46,8,33])
print(B)
B = np.sort(B)
print(B)
for i in range(len(B)):
    print("Index of", B[i], "is: ", i)

# part c.
m=int(input("Enter Number of Row  : "))
n=int(input("Enter Number of Column  : "))

mn = np.random.randint(50,size=(m,n))
print(mn)
print("Shape of mn is: ",mn.shape)
print("Data Type of mn is: ",mn.dtype)
print("Type of mn is: ",type(mn))

nm = mn.reshape(n,m)
print("\nReshaped array nm is: ",nm)
print("Shape of nm is: ",nm.shape)
print("Data Type of nm is: ",nm.dtype)
print("Type of mn is: ",type(nm))
```

# part d.

```
data_arr= np.array([4,7,'xyz',0,3,'abc',0])
Zero=[]
non_zero=[]
NaN = []

for i in range(len(data_arr)):
    if(data_arr[i].isdigit()):
        if(data_arr[i]=='0'):
            Zero.append(i)
        else:
            non_zero.append(i)
    else:
        NaN.append(i)
```

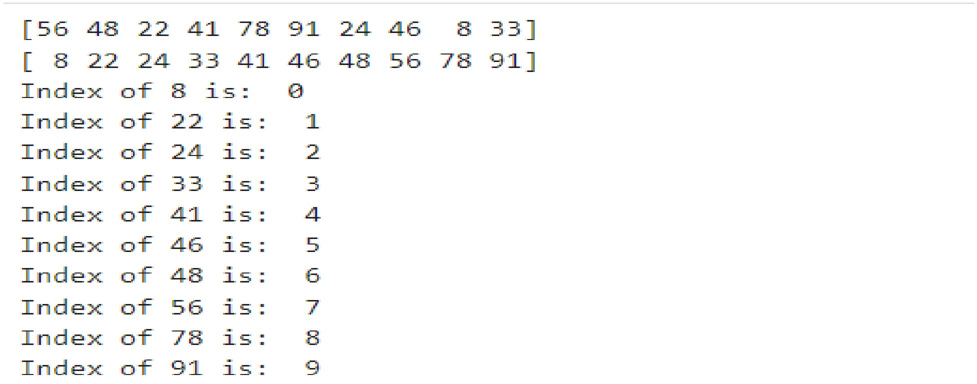
Output:

# part a.



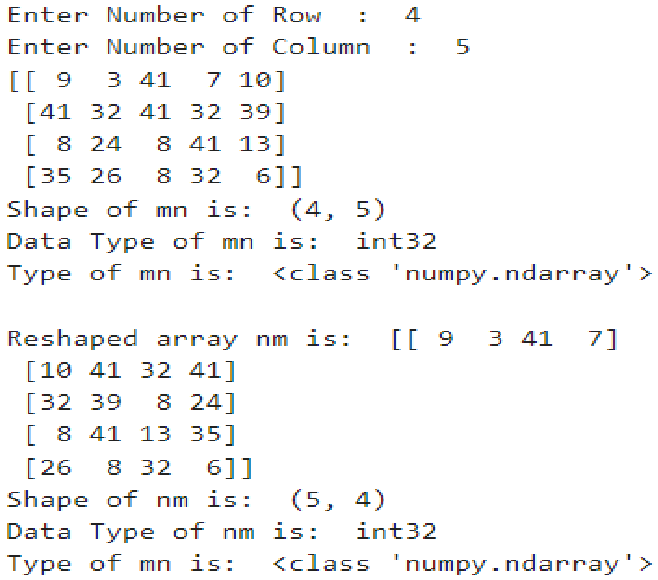
```
[[32 24 23 38]
 [21 35 33  9]
 [32 17 38 47]
 [25 46  2 26]]
Mean :  [29.25 24.5  33.5  24.75]
Standard Deviation :  [ 6.13901458 10.42832681 10.92016483 15.5784306 ]
Variance :  [ 37.6875 108.75   119.25   242.6875]
```

# part b.



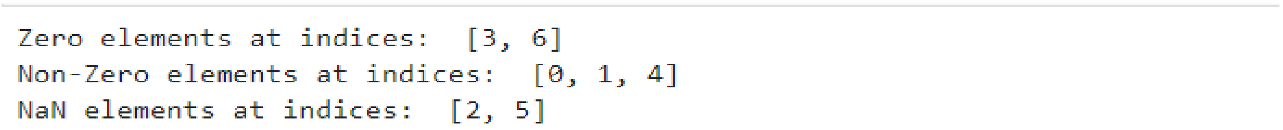
```
[56 48 22 41 78 91 24 46  8 33]
[ 8 22 24 33 41 46 48 56 78 91]
Index of 8 is:  0
Index of 22 is:  1
Index of 24 is:  2
Index of 33 is:  3
Index of 41 is:  4
Index of 46 is:  5
Index of 48 is:  6
Index of 56 is:  7
Index of 78 is:  8
Index of 91 is:  9
```

# part c.



```
Enter Number of Row  :  4
Enter Number of Column  :  5
[[ 9  3 41  7 10]
 [41 32 41 32 39]
 [ 8 24  8 41 13]
 [35 26  8 32  6]]
Shape of mn is:  (4, 5)
Data Type of mn is:  int32
Type of mn is:  <class 'numpy.ndarray'>

Reshaped array nm is:  [[ 9  3 41  7]
 [10 41 32 41]
 [32 39  8 24]
 [ 8 41 13 35]
 [26  8 32  6]]
Shape of nm is:  (5, 4)
Data Type of nm is:  int32
Type of mn is:  <class 'numpy.ndarray'>
```

# part d.



```
Zero elements at indices:  [3, 6]
Non-Zero elements at indices:  [0, 1, 4]
NaN elements at indices:  [2, 5]
```

PRACTICAL – 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

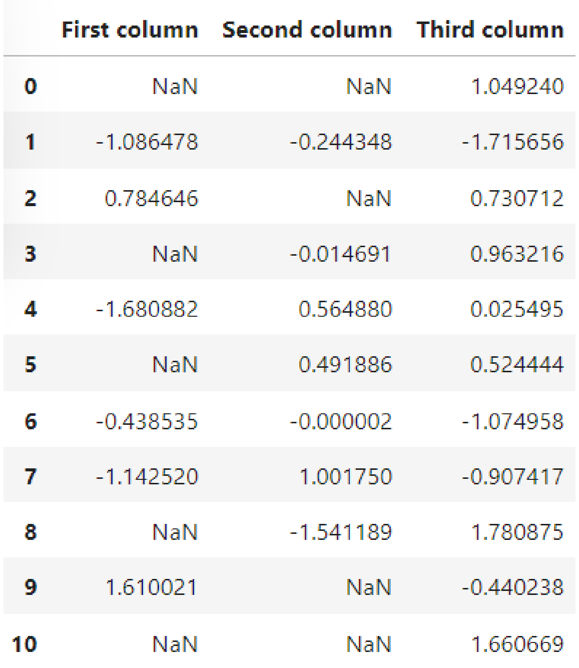
Code:

```
# Dataframe.
import pandas as pd
import numpy as np

data=np.random.randn(50,3)
null_index= np.random.choice([True,False],size=(50,3),p=[0.10,0.90])
data[null_index]=np.nan
df=pd.DataFrame(data,columns=["First column","Second column","Third column"])
df

# part a.
df1=df.isnull().sum()
df1

# part b.
thresh_val=df.shape[0] - 5
df.dropna(axis=1, thresh=thresh_val)

# part c.
a=df.sum(axis=1).idxmax()
df.drop(index=a)

# part d.
df.sort_values('First column', ignore_index=True)
```

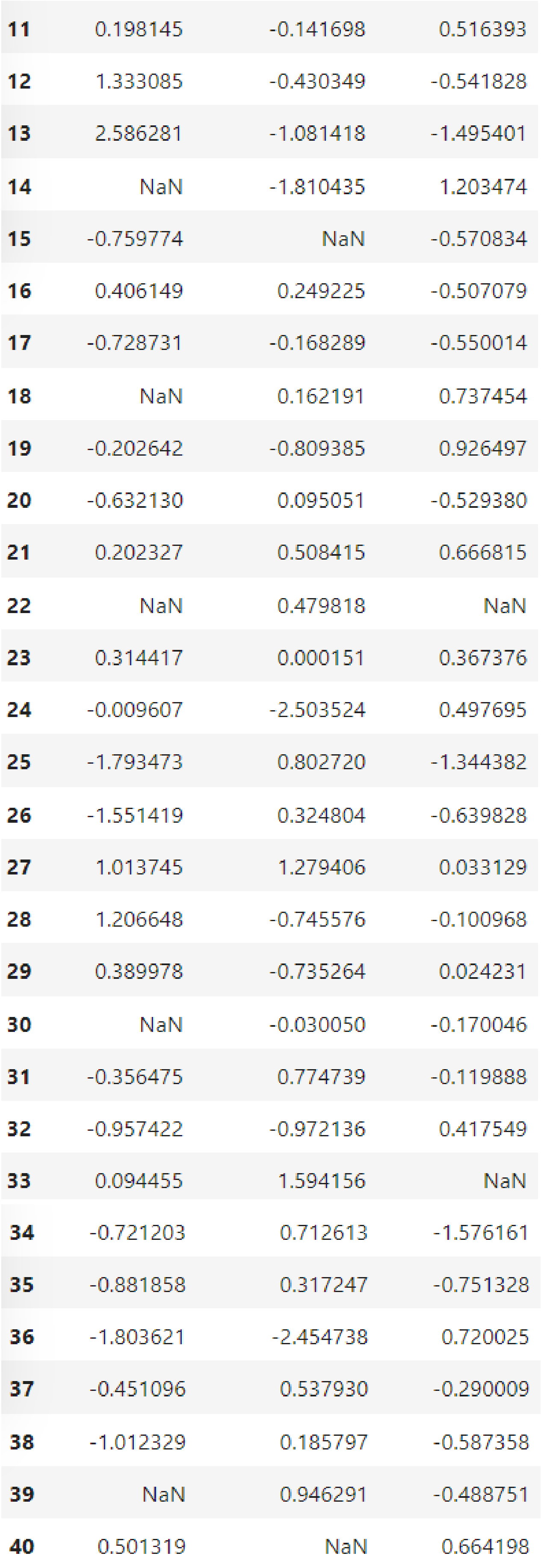
```
# part e.
df.drop_duplicates('First column')

# part f.
df['First column'].corr(df['Second column'])
df['Second column'].cov(df['Third column'])

# part g.
outlier=pd.Series(data=False,index=df.index)
for col in df.columns:

      Q1= df[col].quantile(0.25)
      Q3= df[col].quantile(0.75)
      IQR=Q3-Q1

      lower_bound = Q1-(1.5 * IQR)
      upper_bound = Q3+(1.5 * IQR)
      outlier |= (df[col] < lower_bound) | (df[col] > upper_bound)
df=df[~outlier]
df

# part h.
df['Binning']=pd.cut(np.array(df['Second column']),bins=5)
df
```

Output:

# Dataframe.



	First column	Second column	Third column
0	NaN	NaN	1.049240
1	-1.086478	-0.244348	-1.715656
2	0.784646	NaN	0.730712
3	NaN	-0.014691	0.963216
4	-1.680882	0.564880	0.025495
5	NaN	0.491886	0.524444
6	-0.438535	-0.000002	-1.074958
7	-1.142520	1.001750	-0.907417
8	NaN	-1.541189	1.780875
9	1.610021	NaN	-0.440238
10	NaN	NaN	1.660669



11	0.198145	-0.141698	0.516393
12	1.333085	-0.430349	-0.541828
13	2.586281	-1.081418	-1.495401
14	NaN	-1.810435	1.203474
15	-0.759774	NaN	-0.570834
16	0.406149	0.249225	-0.507079
17	-0.728731	-0.168289	-0.550014
18	NaN	0.162191	0.737454
19	-0.202642	-0.809385	0.926497
20	-0.632130	0.095051	-0.529380
21	0.202327	0.508415	0.666815
22	NaN	0.479818	NaN
23	0.314417	0.000151	0.367376
24	-0.009607	-2.503524	0.497695
25	-1.793473	0.802720	-1.344382
26	-1.551419	0.324804	-0.639828
27	1.013745	1.279406	0.033129
28	1.206648	-0.745576	-0.100968
29	0.389978	-0.735264	0.024231
30	NaN	-0.030050	-0.170046
31	-0.356475	0.774739	-0.119888
32	-0.957422	-0.972136	0.417549
33	0.094455	1.594156	NaN
34	-0.721203	0.712613	-1.576161
35	-0.881858	0.317247	-0.751328
36	-1.803621	-2.454738	0.720025
37	-0.451096	0.537930	-0.290009
38	-1.012329	0.185797	-0.587358
39	NaN	0.946291	-0.488751
40	0.501319	NaN	0.664198

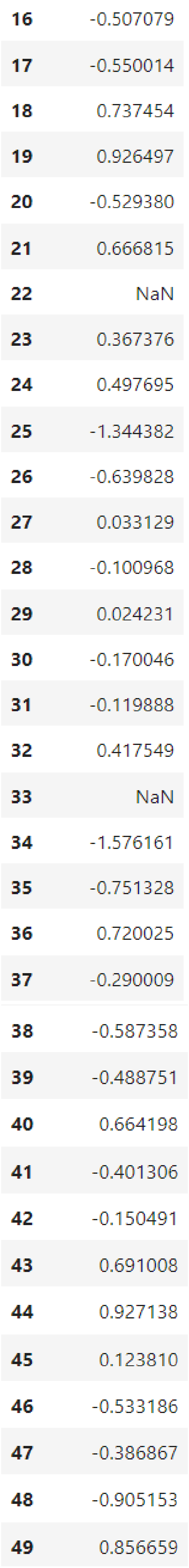
41	-1.169954	NaN	-0.401306
42	-0.510108	NaN	-0.150491
43	0.372117	NaN	0.691008
44	0.394212	-1.436297	0.927138
45	-0.613697	-1.746581	0.123810
46	-2.556021	0.349167	-0.533186
47	0.243551	2.146408	-0.386867
48	1.342335	NaN	-0.905153
49	-0.079612	-0.737700	0.856659

# part a.

```
First column     10
Second column    10
Third column      2
dtype: int64
```

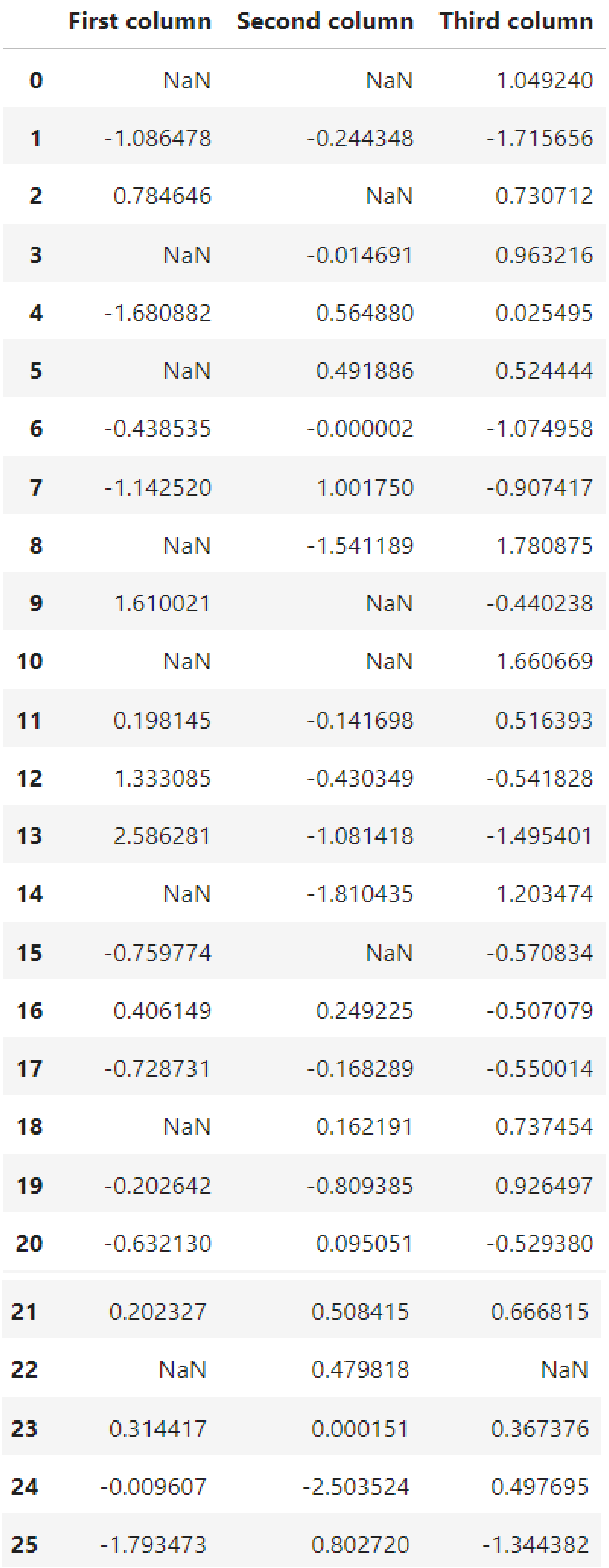
# part b.

	Third column
0	1.049240
1	-1.715656
2	0.730712
3	0.963216
4	0.025495
5	0.524444
6	-1.074958
7	-0.907417
8	1.780875
9	-0.440238
10	1.660669
11	0.516393
12	-0.541828
13	-1.495401
14	1.203474
15	-0.570834

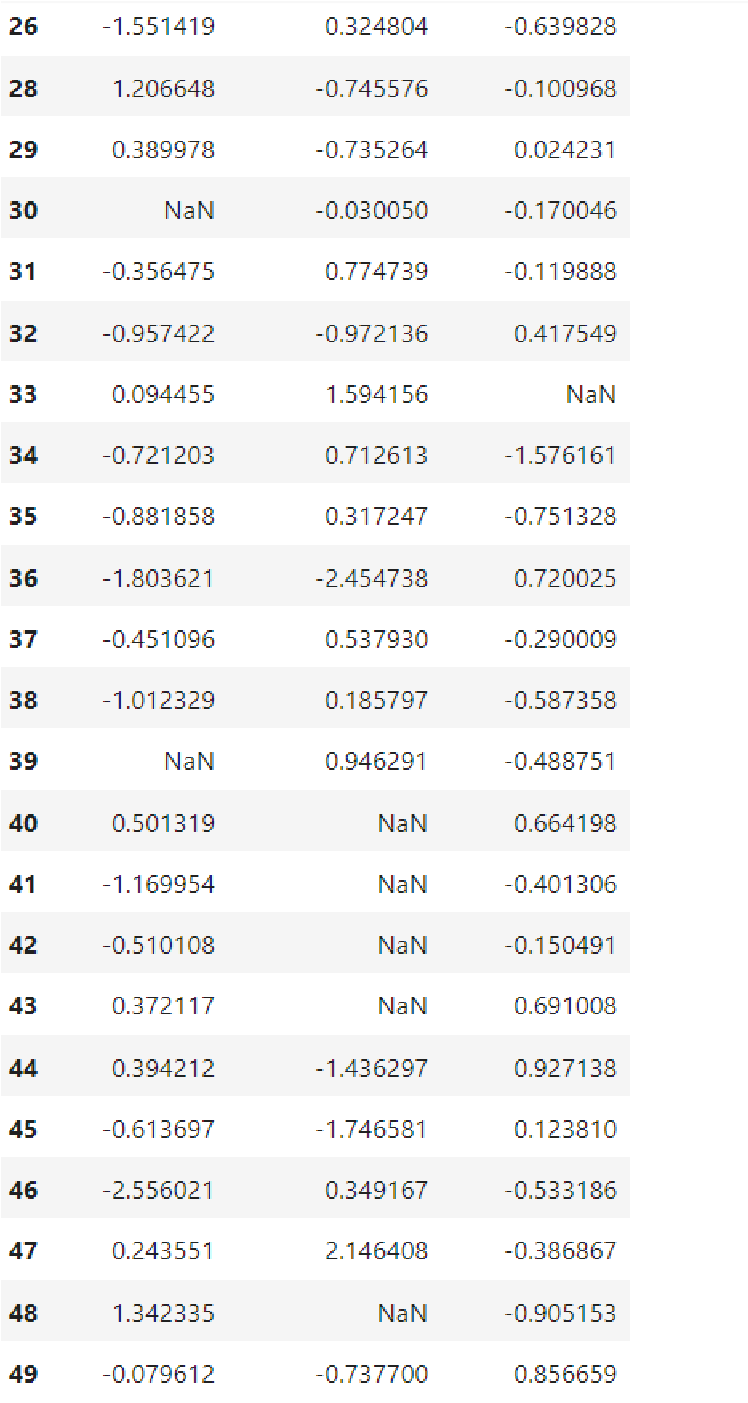


16	-0.507079
17	-0.550014
18	0.737454
19	0.926497
20	-0.529380
21	0.666815
22	NaN
23	0.367376
24	0.497695
25	-1.344382
26	-0.639828
27	0.033129
28	-0.100968
29	0.024231
30	-0.170046
31	-0.119888
32	0.417549
33	NaN
34	-1.576161
35	-0.751328
36	0.720025
37	-0.290009
38	-0.587358
39	-0.488751
40	0.664198
41	-0.401306
42	-0.150491
43	0.691008
44	0.927138
45	0.123810
46	-0.533186
47	-0.386867
48	-0.905153
49	0.856659

# part c.

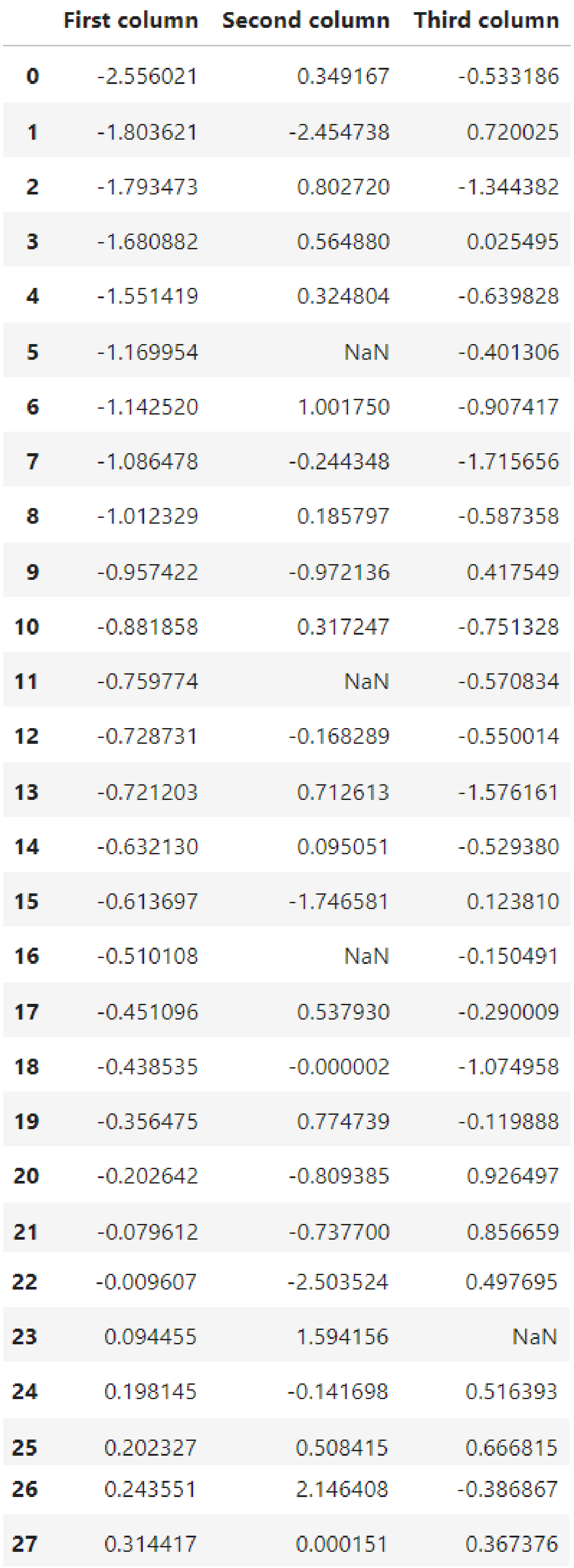


	First column	Second column	Third column
0	NaN	NaN	1.049240
1	-1.086478	-0.244348	-1.715656
2	0.784646	NaN	0.730712
3	NaN	-0.014691	0.963216
4	-1.680882	0.564880	0.025495
5	NaN	0.491886	0.524444
6	-0.438535	-0.000002	-1.074958
7	-1.142520	1.001750	-0.907417
8	NaN	-1.541189	1.780875
9	1.610021	NaN	-0.440238
10	NaN	NaN	1.660669
11	0.198145	-0.141698	0.516393
12	1.333085	-0.430349	-0.541828
13	2.586281	-1.081418	-1.495401
14	NaN	-1.810435	1.203474
15	-0.759774	NaN	-0.570834
16	0.406149	0.249225	-0.507079
17	-0.728731	-0.168289	-0.550014
18	NaN	0.162191	0.737454
19	-0.202642	-0.809385	0.926497
20	-0.632130	0.095051	-0.529380
21	0.202327	0.508415	0.666815
22	NaN	0.479818	NaN
23	0.314417	0.000151	0.367376
24	-0.009607	-2.503524	0.497695
25	-1.793473	0.802720	-1.344382

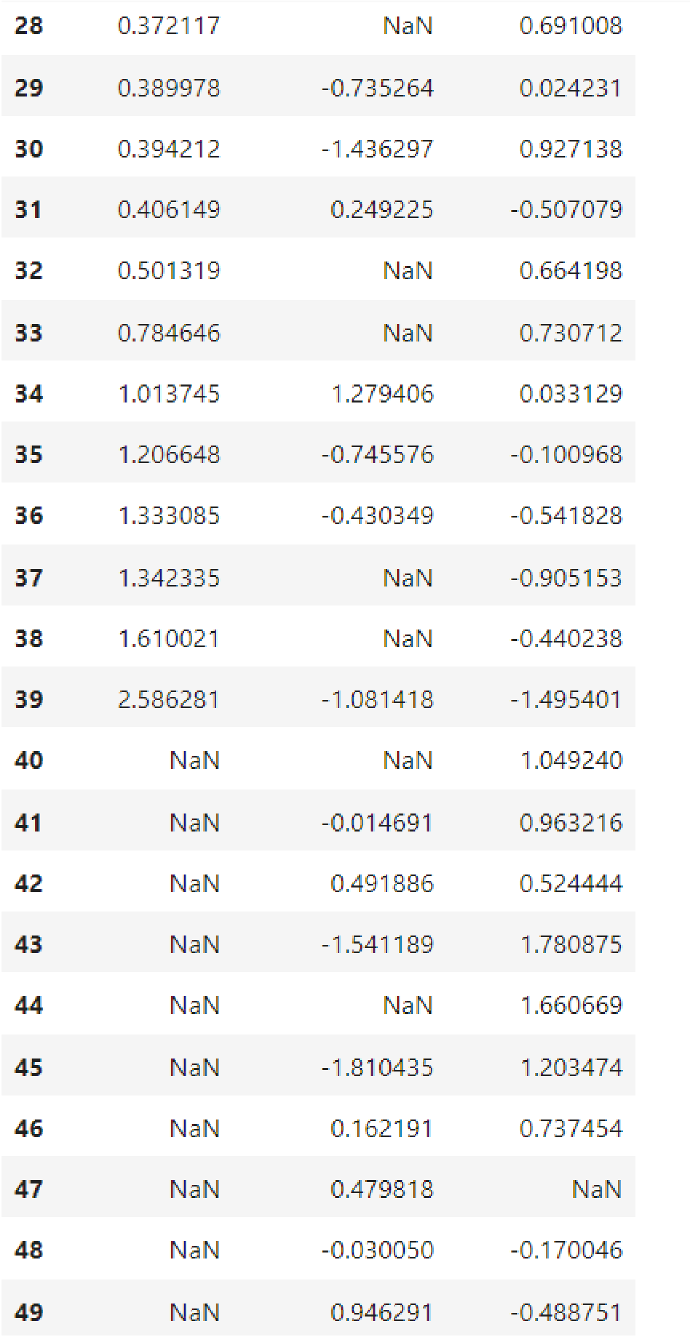


26	-1.551419	0.324804	-0.639828
28	1.206648	-0.745576	-0.100968
29	0.389978	-0.735264	0.024231
30	NaN	-0.030050	-0.170046
31	-0.356475	0.774739	-0.119888
32	-0.957422	-0.972136	0.417549
33	0.094455	1.594156	NaN
34	-0.721203	0.712613	-1.576161
35	-0.881858	0.317247	-0.751328
36	-1.803621	-2.454738	0.720025
37	-0.451096	0.537930	-0.290009
38	-1.012329	0.185797	-0.587358
39	NaN	0.946291	-0.488751
40	0.501319	NaN	0.664198
41	-1.169954	NaN	-0.401306
42	-0.510108	NaN	-0.150491
43	0.372117	NaN	0.691008
44	0.394212	-1.436297	0.927138
45	-0.613697	-1.746581	0.123810
46	-2.556021	0.349167	-0.533186
47	0.243551	2.146408	-0.386867
48	1.342335	NaN	-0.905153
49	-0.079612	-0.737700	0.856659

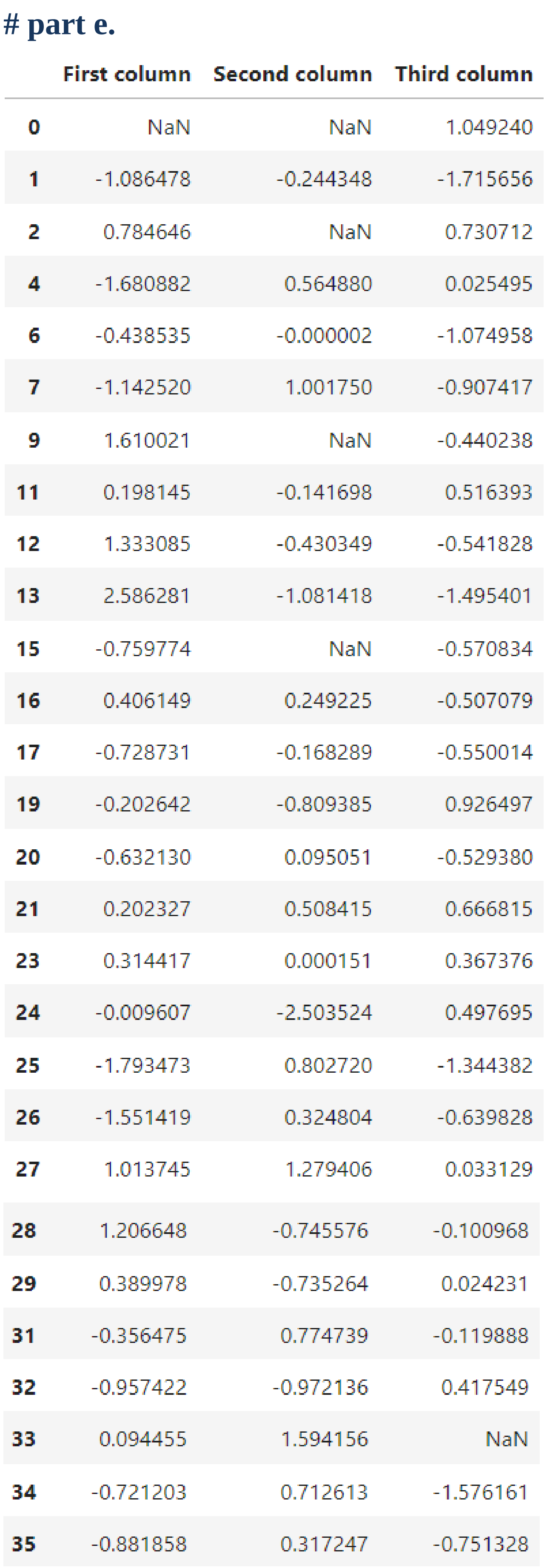
# part d.



	First column	Second column	Third column
0	-2.556021	0.349167	-0.533186
1	-1.803621	-2.454738	0.720025
2	-1.793473	0.802720	-1.344382
3	-1.680882	0.564880	0.025495
4	-1.551419	0.324804	-0.639828
5	-1.169954	NaN	-0.401306
6	-1.142520	1.001750	-0.907417
7	-1.086478	-0.244348	-1.715656
8	-1.012329	0.185797	-0.587358
9	-0.957422	-0.972136	0.417549
10	-0.881858	0.317247	-0.751328
11	-0.759774	NaN	-0.570834
12	-0.728731	-0.168289	-0.550014
13	-0.721203	0.712613	-1.576161
14	-0.632130	0.095051	-0.529380
15	-0.613697	-1.746581	0.123810
16	-0.510108	NaN	-0.150491
17	-0.451096	0.537930	-0.290009
18	-0.438535	-0.000002	-1.074958
19	-0.356475	0.774739	-0.119888
20	-0.202642	-0.809385	0.926497
21	-0.079612	-0.737700	0.856659
22	-0.009607	-2.503524	0.497695
23	0.094455	1.594156	NaN
24	0.198145	-0.141698	0.516393
25	0.202327	0.508415	0.666815
26	0.243551	2.146408	-0.386867
27	0.314417	0.000151	0.367376



28	0.372117	NaN	0.691008
29	0.389978	-0.735264	0.024231
30	0.394212	-1.436297	0.927138
31	0.406149	0.249225	-0.507079
32	0.501319	NaN	0.664198
33	0.784646	NaN	0.730712
34	1.013745	1.279406	0.033129
35	1.206648	-0.745576	-0.100968
36	1.333085	-0.430349	-0.541828
37	1.342335	NaN	-0.905153
38	1.610021	NaN	-0.440238
39	2.586281	-1.081418	-1.495401
40	NaN	NaN	1.049240
41	NaN	-0.014691	0.963216
42	NaN	0.491886	0.524444
43	NaN	-1.541189	1.780875
44	NaN	NaN	1.660669
45	NaN	-1.810435	1.203474
46	NaN	0.162191	0.737454
47	NaN	0.479818	NaN
48	NaN	-0.030050	-0.170046
49	NaN	0.946291	-0.488751



# part e.

	First column	Second column	Third column
0	NaN	NaN	1.049240
1	-1.086478	-0.244348	-1.715656
2	0.784646	NaN	0.730712
4	-1.680882	0.564880	0.025495
6	-0.438535	-0.000002	-1.074958
7	-1.142520	1.001750	-0.907417
9	1.610021	NaN	-0.440238
11	0.198145	-0.141698	0.516393
12	1.333085	-0.430349	-0.541828
13	2.586281	-1.081418	-1.495401
15	-0.759774	NaN	-0.570834
16	0.406149	0.249225	-0.507079
17	-0.728731	-0.168289	-0.550014
19	-0.202642	-0.809385	0.926497
20	-0.632130	0.095051	-0.529380
21	0.202327	0.508415	0.666815
23	0.314417	0.000151	0.367376
24	-0.009607	-2.503524	0.497695
25	-1.793473	0.802720	-1.344382
26	-1.551419	0.324804	-0.639828
27	1.013745	1.279406	0.033129
28	1.206648	-0.745576	-0.100968
29	0.389978	-0.735264	0.024231
31	-0.356475	0.774739	-0.119888
32	-0.957422	-0.972136	0.417549
33	0.094455	1.594156	NaN
34	-0.721203	0.712613	-1.576161
35	-0.881858	0.317247	-0.751328

36	-1.803621	-2.454738	0.720025
37	-0.451096	0.537930	-0.290009
38	-1.012329	0.185797	-0.587358
40	0.501319	NaN	0.664198
41	-1.169954	NaN	-0.401306
42	-0.510108	NaN	-0.150491
43	0.372117	NaN	0.691008
44	0.394212	-1.436297	0.927138
45	-0.613697	-1.746581	0.123810
46	-2.556021	0.349167	-0.533186
47	0.243551	2.146408	-0.386867
48	1.342335	NaN	-0.905153
49	-0.079612	-0.737700	0.856659

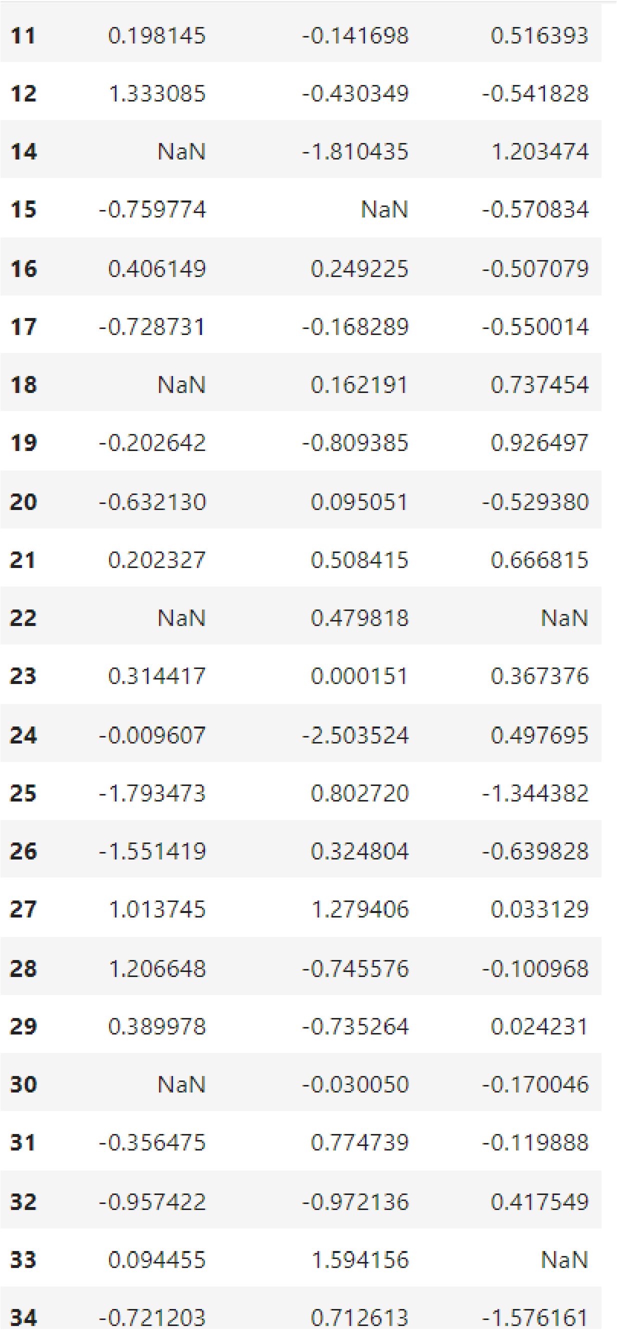
# part f.

```
-0.09560670752718779
```

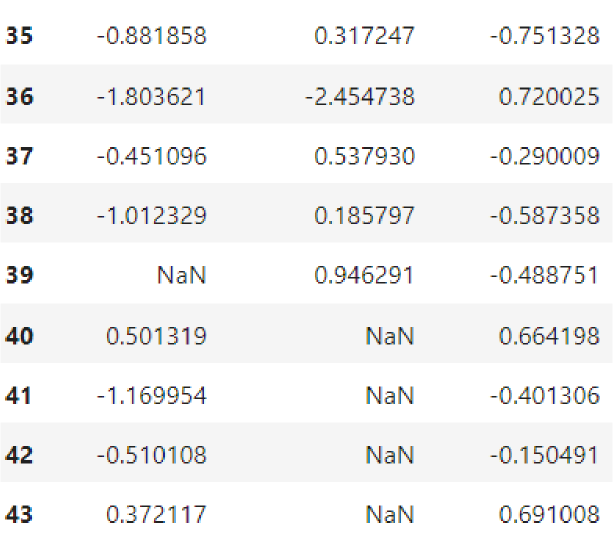
```
-0.3842857140042494
```

# part g.

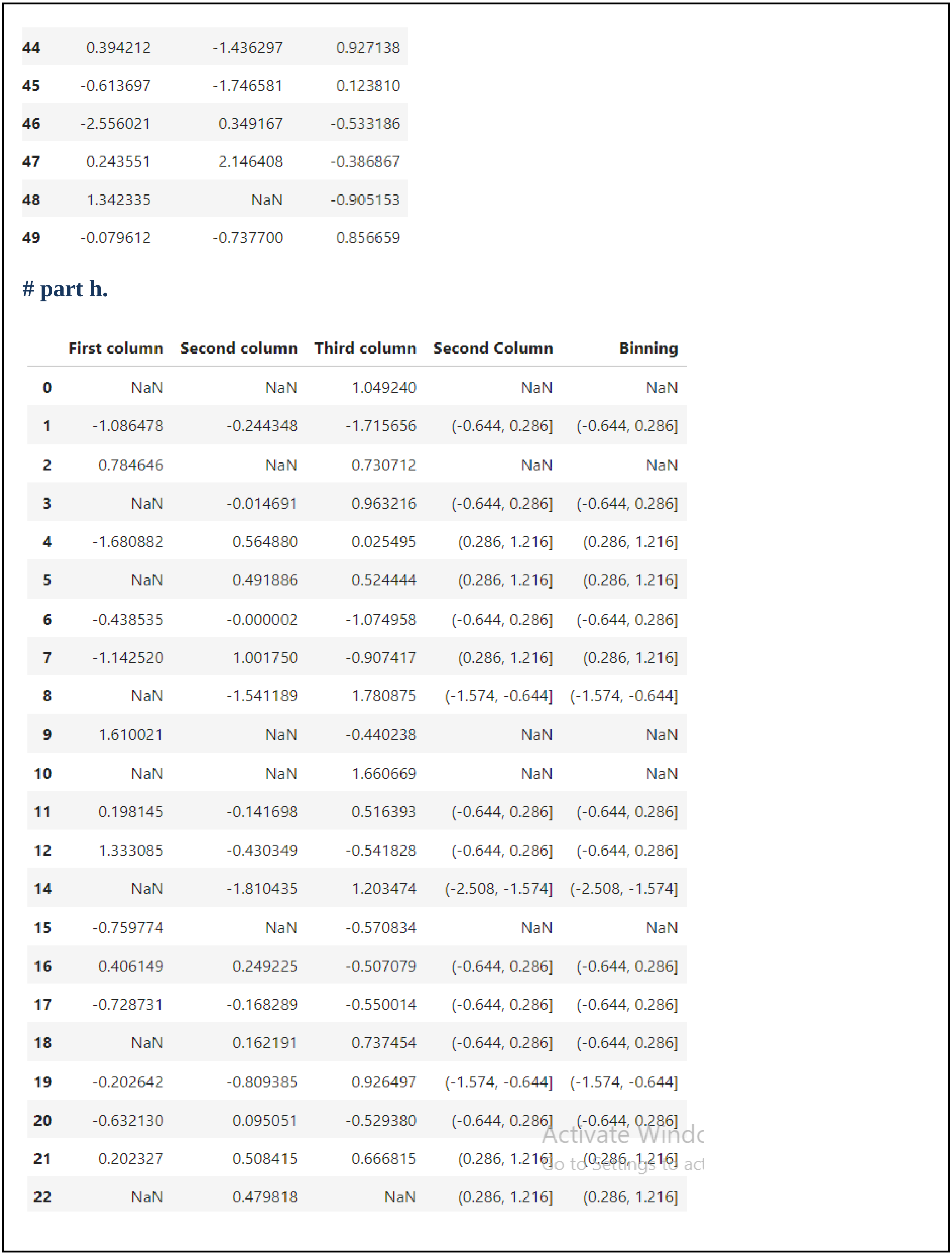
	First column	Second column	Third column
0	NaN	NaN	1.049240
1	-1.086478	-0.244348	-1.715656
2	0.784646	NaN	0.730712
3	NaN	-0.014691	0.963216
4	-1.680882	0.564880	0.025495
5	NaN	0.491886	0.524444
6	-0.438535	-0.000002	-1.074958
7	-1.142520	1.001750	-0.907417
8	NaN	-1.541189	1.780875
9	1.610021	NaN	-0.440238
10	NaN	NaN	1.660669



11	0.198145	-0.141698	0.516393
12	1.333085	-0.430349	-0.541828
14	NaN	-1.810435	1.203474
15	-0.759774	NaN	-0.570834
16	0.406149	0.249225	-0.507079
17	-0.728731	-0.168289	-0.550014
18	NaN	0.162191	0.737454
19	-0.202642	-0.809385	0.926497
20	-0.632130	0.095051	-0.529380
21	0.202327	0.508415	0.666815
22	NaN	0.479818	NaN
23	0.314417	0.000151	0.367376
24	-0.009607	-2.503524	0.497695
25	-1.793473	0.802720	-1.344382
26	-1.551419	0.324804	-0.639828
27	1.013745	1.279406	0.033129
28	1.206648	-0.745576	-0.100968
29	0.389978	-0.735264	0.024231
30	NaN	-0.030050	-0.170046
31	-0.356475	0.774739	-0.119888
32	-0.957422	-0.972136	0.417549
33	0.094455	1.594156	NaN
34	-0.721203	0.712613	-1.576161



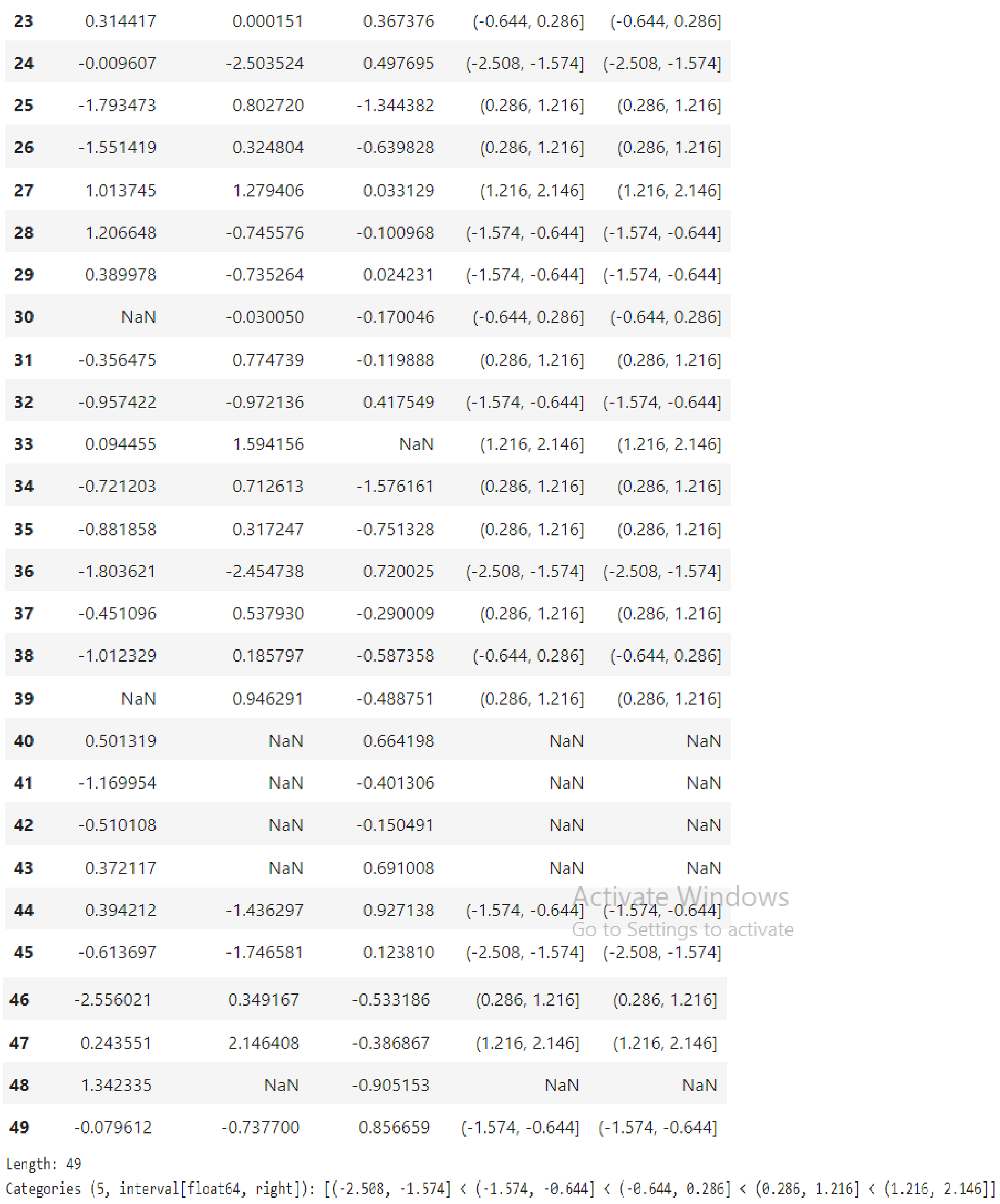
35	-0.881858	0.317247	-0.751328
36	-1.803621	-2.454738	0.720025
37	-0.451096	0.537930	-0.290009
38	-1.012329	0.185797	-0.587358
39	NaN	0.946291	-0.488751
40	0.501319	NaN	0.664198
41	-1.169954	NaN	-0.401306
42	-0.510108	NaN	-0.150491
43	0.372117	NaN	0.691008



44	0.394212	-1.436297	0.927138
45	-0.613697	-1.746581	0.123810
46	-2.556021	0.349167	-0.533186
47	0.243551	2.146408	-0.386867
48	1.342335	NaN	-0.905153
49	-0.079612	-0.737700	0.856659

# part h.

	First column	Second column	Third column	Second Column	Binning
0	NaN	NaN	1.049240	NaN	NaN
1	-1.086478	-0.244348	-1.715656	(-0.644, 0.286]	(-0.644, 0.286]
2	0.784646	NaN	0.730712	NaN	NaN
3	NaN	-0.014691	0.963216	(-0.644, 0.286]	(-0.644, 0.286]
4	-1.680882	0.564880	0.025495	(0.286, 1.216]	(0.286, 1.216]
5	NaN	0.491886	0.524444	(0.286, 1.216]	(0.286, 1.216]
6	-0.438535	-0.000002	-1.074958	(-0.644, 0.286]	(-0.644, 0.286]
7	-1.142520	1.001750	-0.907417	(0.286, 1.216]	(0.286, 1.216]
8	NaN	-1.541189	1.780875	(-1.574, -0.644]	(-1.574, -0.644]
9	1.610021	NaN	-0.440238	NaN	NaN
10	NaN	NaN	1.660669	NaN	NaN
11	0.198145	-0.141698	0.516393	(-0.644, 0.286]	(-0.644, 0.286]
12	1.333085	-0.430349	-0.541828	(-0.644, 0.286]	(-0.644, 0.286]
14	NaN	-1.810435	1.203474	(-2.508, -1.574]	(-2.508, -1.574]
15	-0.759774	NaN	-0.570834	NaN	NaN
16	0.406149	0.249225	-0.507079	(-0.644, 0.286]	(-0.644, 0.286]
17	-0.728731	-0.168289	-0.550014	(-0.644, 0.286]	(-0.644, 0.286]
18	NaN	0.162191	0.737454	(-0.644, 0.286]	(-0.644, 0.286]
19	-0.202642	-0.809385	0.926497	(-1.574, -0.644]	(-1.574, -0.644]
20	-0.632130	0.095051	-0.529380	(-0.644, 0.286]	(-0.644, 0.286]
21	0.202327	0.508415	0.666815	(0.286, 1.216]	(0.286, 1.216]
22	NaN	0.479818	NaN	(0.286, 1.216]	(0.286, 1.216]



23	0.314417	0.000151	0.367376	(-0.644, 0.286]	(-0.644, 0.286]
24	-0.009607	-2.503524	0.497695	(-2.508, -1.574]	(-2.508, -1.574]
25	-1.793473	0.802720	-1.344382	(0.286, 1.216]	(0.286, 1.216]
26	-1.551419	0.324804	-0.639828	(0.286, 1.216]	(0.286, 1.216]
27	1.013745	1.279406	0.033129	(1.216, 2.146]	(1.216, 2.146]
28	1.206648	-0.745576	-0.100968	(-1.574, -0.644]	(-1.574, -0.644]
29	0.389978	-0.735264	0.024231	(-1.574, -0.644]	(-1.574, -0.644]
30	NaN	-0.030050	-0.170046	(-0.644, 0.286]	(-0.644, 0.286]
31	-0.356475	0.774739	-0.119888	(0.286, 1.216]	(0.286, 1.216]
32	-0.957422	-0.972136	0.417549	(-1.574, -0.644]	(-1.574, -0.644]
33	0.094455	1.594156	NaN	(1.216, 2.146]	(1.216, 2.146]
34	-0.721203	0.712613	-1.576161	(0.286, 1.216]	(0.286, 1.216]
35	-0.881858	0.317247	-0.751328	(0.286, 1.216]	(0.286, 1.216]
36	-1.803621	-2.454738	0.720025	(-2.508, -1.574]	(-2.508, -1.574]
37	-0.451096	0.537930	-0.290009	(0.286, 1.216]	(0.286, 1.216]
38	-1.012329	0.185797	-0.587358	(-0.644, 0.286]	(-0.644, 0.286]
39	NaN	0.946291	-0.488751	(0.286, 1.216]	(0.286, 1.216]
40	0.501319	NaN	0.664198	NaN	NaN
41	-1.169954	NaN	-0.401306	NaN	NaN
42	-0.510108	NaN	-0.150491	NaN	NaN
43	0.372117	NaN	0.691008	NaN	NaN
44	0.394212	-1.436297	0.927138	(-1.574, -0.644]	(-1.574, -0.644]
45	-0.613697	-1.746581	0.123810	(-2.508, -1.574]	(-2.508, -1.574]
46	-2.556021	0.349167	-0.533186	(0.286, 1.216]	(0.286, 1.216]
47	0.243551	2.146408	-0.386867	(1.216, 2.146]	(1.216, 2.146]
48	1.342335	NaN	-0.905153	NaN	NaN
49	-0.079612	-0.737700	0.856659	(-1.574, -0.644]	(-1.574, -0.644]

```
Length: 49
Categories (5, interval[float64, right]): [(-2.508, -1.574] < (-1.574, -0.644] < (-0.644, 0.286] < (0.286, 1.216] < (1.216, 2.146]]
```

PRACTICAL – 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.

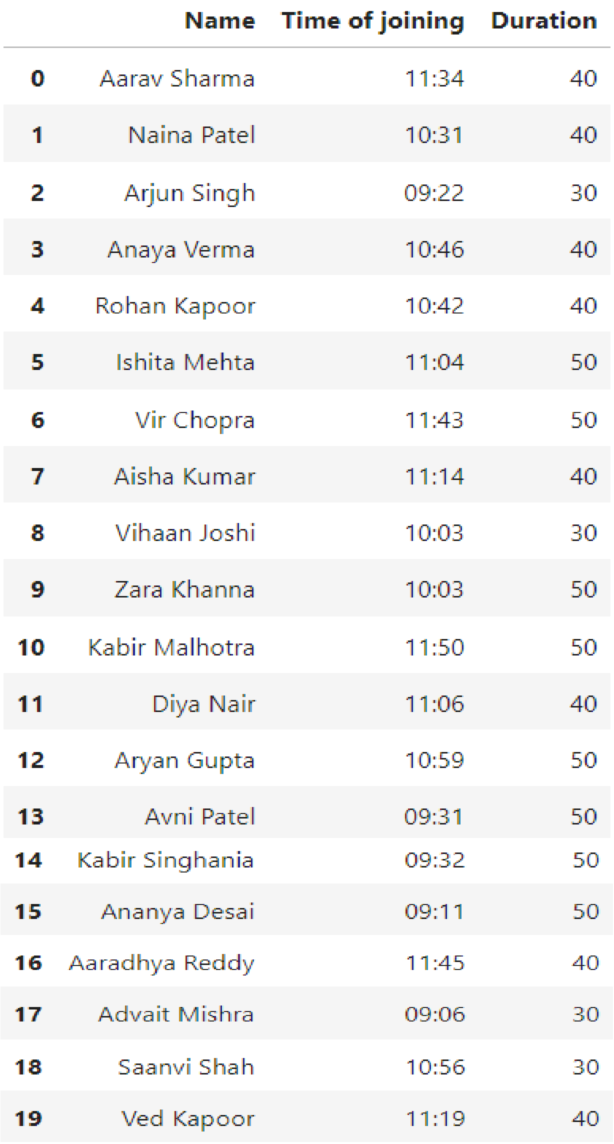
Code:

```
# Dataframes.
File1=r"day1_attendance.xlsx"
File2 = r"day2_attendance.xlsx"

import pandas as pd
df=pd.read_excel(File1)
df1=pd.read_excel(File2)
df
df1

# part a.
df.merge(df1,how="inner",on="Name")

# part b.
df.merge(df1, how="outer")

# part c.
a=pd.concat([df,df1],ignore_index=True, axis=0)
a
# total records in the dataframe are:
len(a)

# part d.
# merging dataframes
b=df.merge(df1,how="outer")
b
# setting columns as multi-indexes
c=b.set_index(keys=["Name","Duration"])
```

```
c
# descriptive statistics for this multi-index
c.describe()
```

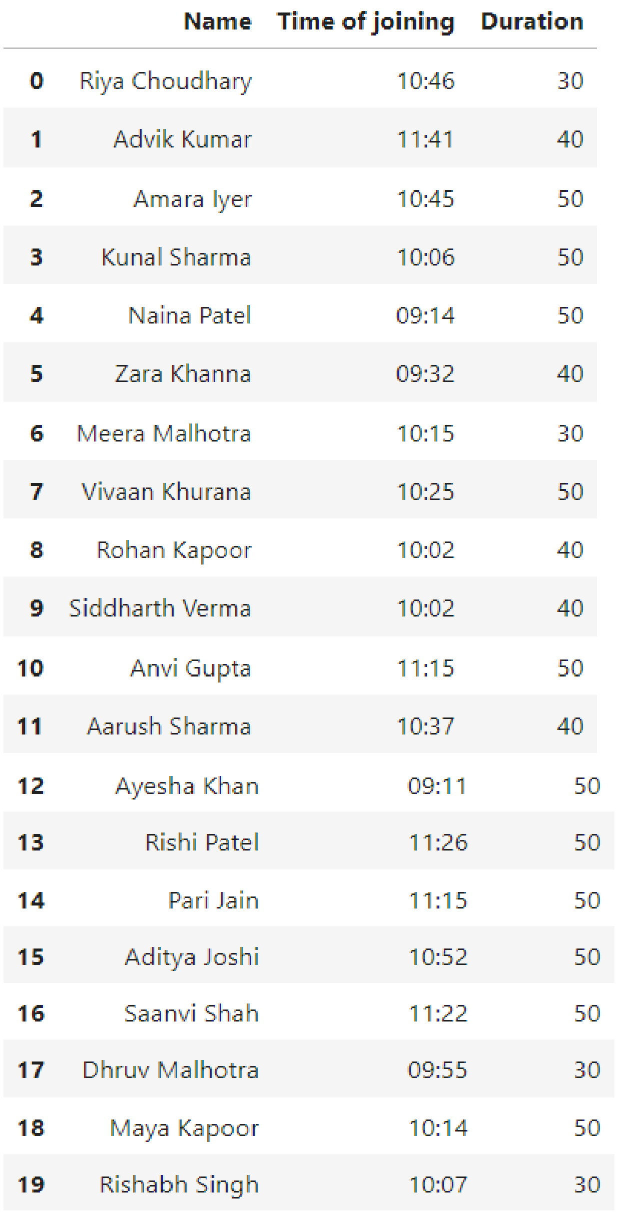
Output:

# Dataframe_1.



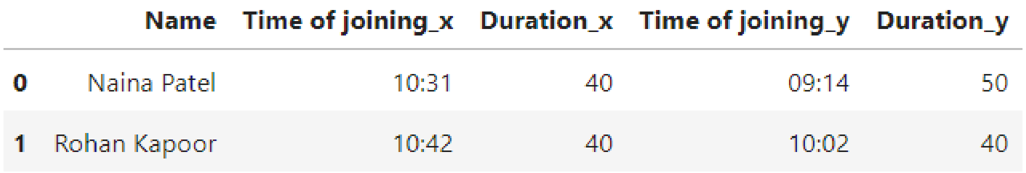
	Name	Time of joining	Duration
0	Aarav Sharma	11:34	40
1	Naina Patel	10:31	40
2	Arjun Singh	09:22	30
3	Anaya Verma	10:46	40
4	Rohan Kapoor	10:42	40
5	Ishita Mehta	11:04	50
6	Vir Chopra	11:43	50
7	Aisha Kumar	11:14	40
8	Vihaan Joshi	10:03	30
9	Zara Khanna	10:03	50
10	Kabir Malhotra	11:50	50
11	Diya Nair	11:06	40
12	Aryan Gupta	10:59	50
13	Avni Patel	09:31	50
14	Kabir Singhania	09:32	50
15	Ananya Desai	09:11	50
16	Aaradhya Reddy	11:45	40
17	Advait Mishra	09:06	30
18	Saanvi Shah	10:56	30
19	Ved Kapoor	11:19	40

# Dataframe_2.



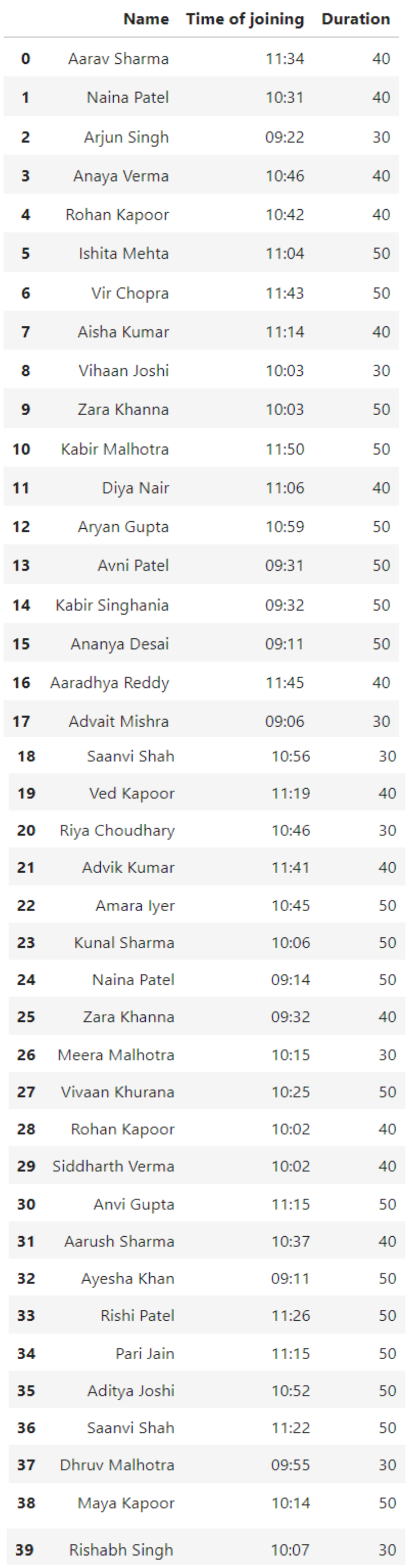
	Name	Time of joining	Duration
0	Riya Choudhary	10:46	30
1	Advik Kumar	11:41	40
2	Amara Iyer	10:45	50
3	Kunal Sharma	10:06	50
4	Naina Patel	09:14	50
5	Zara Khanna	09:32	40
6	Meera Malhotra	10:15	30
7	Vivaan Khurana	10:25	50
8	Rohan Kapoor	10:02	40
9	Siddharth Verma	10:02	40
10	Anvi Gupta	11:15	50
11	Aarush Sharma	10:37	40
12	Ayesha Khan	09:11	50
13	Rishi Patel	11:26	50
14	Pari Jain	11:15	50
15	Aditya Joshi	10:52	50
16	Saanvi Shah	11:22	50
17	Dhruv Malhotra	09:55	30
18	Maya Kapoor	10:14	50
19	Rishabh Singh	10:07	30

# part a.



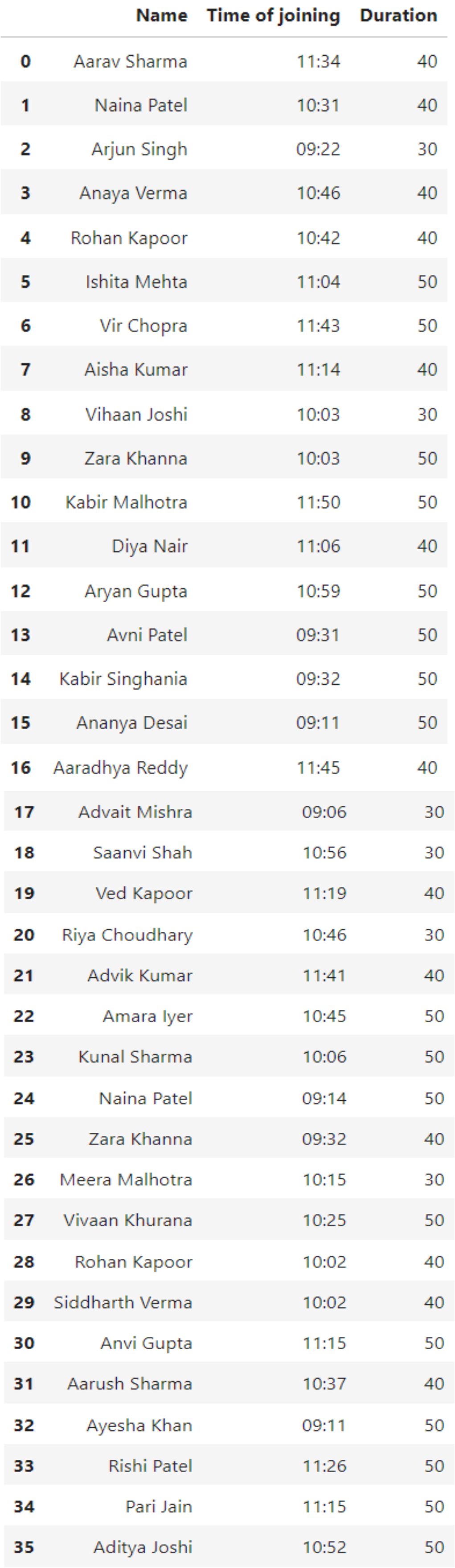
	Name	Time of joining_x	Duration_x	Time of joining_y	Duration_y
0	Naina Patel	10:31	40	09:14	50
1	Rohan Kapoor	10:42	40	10:02	40

# part b.



	Name	Time of joining	Duration
0	Aarav Sharma	11:34	40
1	Naina Patel	10:31	40
2	Arjun Singh	09:22	30
3	Anaya Verma	10:46	40
4	Rohan Kapoor	10:42	40
5	Ishita Mehta	11:04	50
6	Vir Chopra	11:43	50
7	Aisha Kumar	11:14	40
8	Vihaan Joshi	10:03	30
9	Zara Khanna	10:03	50
10	Kabir Malhotra	11:50	50
11	Diya Nair	11:06	40
12	Aryan Gupta	10:59	50
13	Avni Patel	09:31	50
14	Kabir Singhania	09:32	50
15	Ananya Desai	09:11	50
16	Aaradhya Reddy	11:45	40
17	Advait Mishra	09:06	30
18	Saanvi Shah	10:56	30
19	Ved Kapoor	11:19	40
20	Riya Choudhary	10:46	30
21	Advik Kumar	11:41	40
22	Amara Iyer	10:45	50
23	Kunal Sharma	10:06	50
24	Naina Patel	09:14	50
25	Zara Khanna	09:32	40
26	Meera Malhotra	10:15	30
27	Vivaan Khurana	10:25	50
28	Rohan Kapoor	10:02	40
29	Siddharth Verma	10:02	40
30	Anvi Gupta	11:15	50
31	Aarush Sharma	10:37	40
32	Ayesha Khan	09:11	50
33	Rishi Patel	11:26	50
34	Pari Jain	11:15	50
35	Aditya Joshi	10:52	50
36	Saanvi Shah	11:22	50
37	Dhruv Malhotra	09:55	30
38	Maya Kapoor	10:14	50
39	Rishabh Singh	10:07	30

# part c.



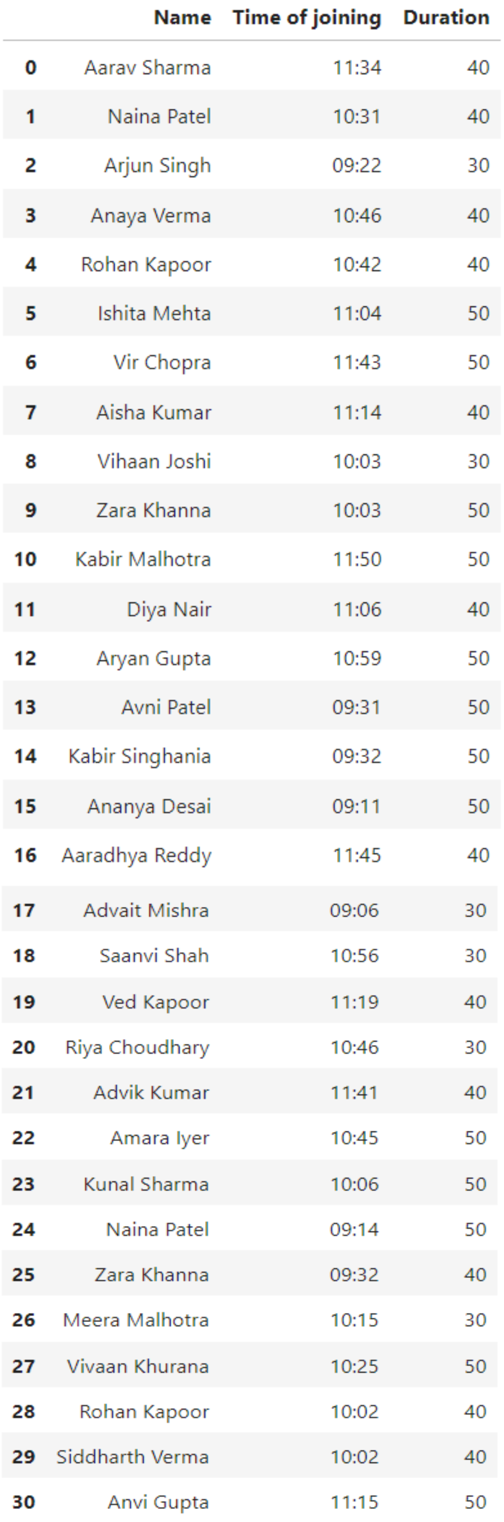
	Name	Time of joining	Duration
0	Aarav Sharma	11:34	40
1	Naina Patel	10:31	40
2	Arjun Singh	09:22	30
3	Anaya Verma	10:46	40
4	Rohan Kapoor	10:42	40
5	Ishita Mehta	11:04	50
6	Vir Chopra	11:43	50
7	Aisha Kumar	11:14	40
8	Vihaan Joshi	10:03	30
9	Zara Khanna	10:03	50
10	Kabir Malhotra	11:50	50
11	Diya Nair	11:06	40
12	Aryan Gupta	10:59	50
13	Avni Patel	09:31	50
14	Kabir Singhania	09:32	50
15	Ananya Desai	09:11	50
16	Aaradhya Reddy	11:45	40
17	Advait Mishra	09:06	30
18	Saanvi Shah	10:56	30
19	Ved Kapoor	11:19	40
20	Riya Choudhary	10:46	30
21	Advik Kumar	11:41	40
22	Amara Iyer	10:45	50
23	Kunal Sharma	10:06	50
24	Naina Patel	09:14	50
25	Zara Khanna	09:32	40
26	Meera Malhotra	10:15	30
27	Vivaan Khurana	10:25	50
28	Rohan Kapoor	10:02	40
29	Siddharth Verma	10:02	40
30	Anvi Gupta	11:15	50
31	Aarush Sharma	10:37	40
32	Ayesha Khan	09:11	50
33	Rishi Patel	11:26	50
34	Pari Jain	11:15	50
35	Aditya Joshi	10:52	50

# total records in the dataframe are:

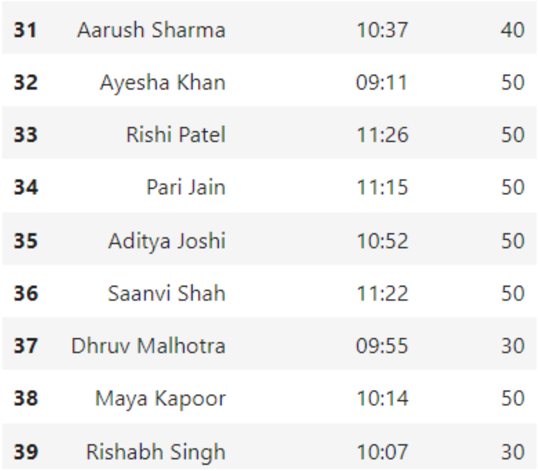
```
40
```

# part d.

# merging dataframes

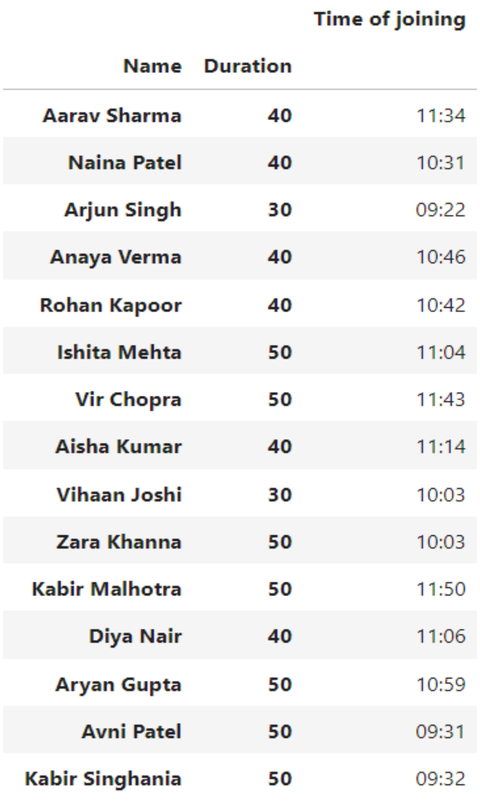
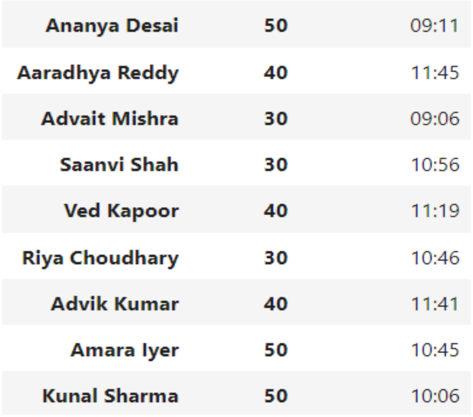


	Name	Time of joining	Duration
0	Aarav Sharma	11:34	40
1	Naina Patel	10:31	40
2	Arjun Singh	09:22	30
3	Anaya Verma	10:46	40
4	Rohan Kapoor	10:42	40
5	Ishita Mehta	11:04	50
6	Vir Chopra	11:43	50
7	Aisha Kumar	11:14	40
8	Vihaan Joshi	10:03	30
9	Zara Khanna	10:03	50
10	Kabir Malhotra	11:50	50
11	Diya Nair	11:06	40
12	Aryan Gupta	10:59	50
13	Avni Patel	09:31	50
14	Kabir Singhania	09:32	50
15	Ananya Desai	09:11	50
16	Aaradhya Reddy	11:45	40
17	Advait Mishra	09:06	30
18	Saanvi Shah	10:56	30
19	Ved Kapoor	11:19	40
20	Riya Choudhary	10:46	30
21	Advik Kumar	11:41	40
22	Amara Iyer	10:45	50
23	Kunal Sharma	10:06	50
24	Naina Patel	09:14	50
25	Zara Khanna	09:32	40
26	Meera Malhotra	10:15	30
27	Vivaan Khurana	10:25	50
28	Rohan Kapoor	10:02	40
29	Siddharth Verma	10:02	40
30	Anvi Gupta	11:15	50

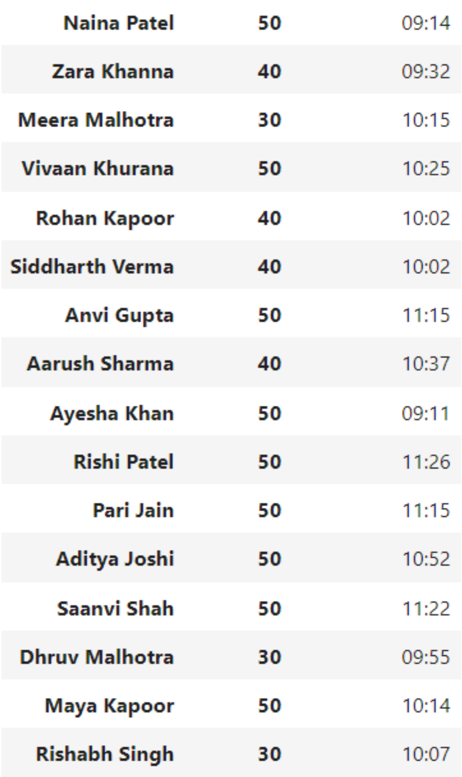


31	Aarush Sharma	10:37	40
32	Ayesha Khan	09:11	50
33	Rishi Patel	11:26	50
34	Pari Jain	11:15	50
35	Aditya Joshi	10:52	50
36	Saanvi Shah	11:22	50
37	Dhruv Malhotra	09:55	30
38	Maya Kapoor	10:14	50
39	Rishabh Singh	10:07	30

# setting columns as multi-indexes

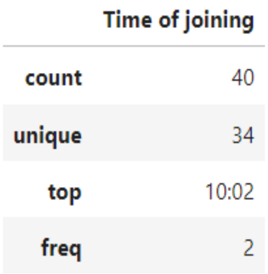



		Time of joining
Name	**Duration**	
Aarav Sharma	**40**	11:34
Naina Patel	**40**	10:31
Arjun Singh	**30**	09:22
Anaya Verma	**40**	10:46
Rohan Kapoor	**40**	10:42
Ishita Mehta	**50**	11:04
Vir Chopra	**50**	11:43
Aisha Kumar	**40**	11:14
Vihaan Joshi	**30**	10:03
Zara Khanna	**50**	10:03
Kabir Malhotra	**50**	11:50
Diya Nair	**40**	11:06
Aryan Gupta	**50**	10:59
Avni Patel	**50**	09:31
Kabir Singhania	**50**	09:32
Ananya Desai	**50**	09:11
Aaradhya Reddy	**40**	11:45
Advait Mishra	**30**	09:06
Saanvi Shah	**30**	10:56
Ved Kapoor	**40**	11:19
Riya Choudhary	**30**	10:46
Advik Kumar	**40**	11:41
Amara Iyer	**50**	10:45
Kunal Sharma	**50**	10:06



Naina Patel	50	09:14
Zara Khanna	40	09:32
Meera Malhotra	30	10:15
Vivaan Khurana	50	10:25
Rohan Kapoor	40	10:02
Siddharth Verma	40	10:02
Anvi Gupta	50	11:15
Aarush Sharma	40	10:37
Ayesha Khan	50	09:11
Rishi Patel	50	11:26
Pari Jain	50	11:15
Aditya Joshi	50	10:52
Saanvi Shah	50	11:22
Dhruv Malhotra	30	09:55
Maya Kapoor	50	10:14
Rishabh Singh	30	10:07

# descriptive statistics for this multi-index



	Time of joining
count	40
unique	34
top	10:02
freq	2

PRACTICAL – 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.

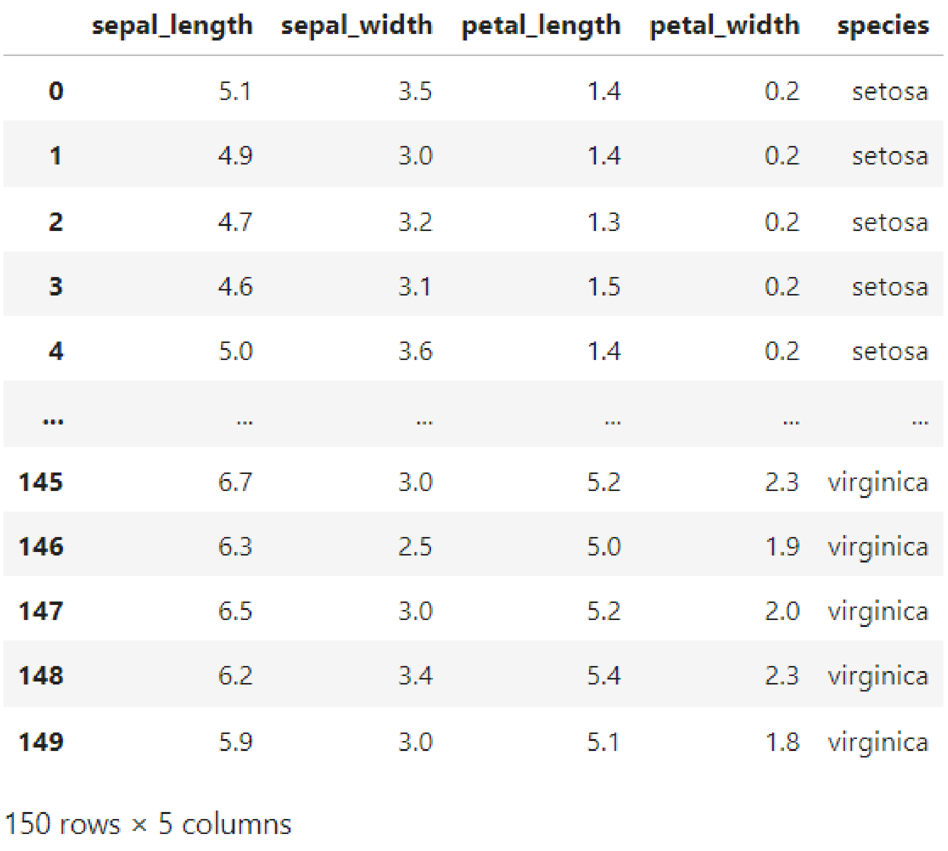
Code:

```
from matplotlib import pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

# Dataframe.
iris = load_iris()
iris_df = sns.load_dataset('iris')
iris_df

# part a.
plt.figure(figsize=(4, 4))
sns.countplot(x='species', hue ='species', data=iris_df, palette='husl')
plt.title('Frequency of Each Class Label in Iris Dataset')
plt.xlabel('Class Label')
plt.ylabel('Frequency')
plt.legend(loc = 'upper right', labels=['setosa','versicolor','virginica'])
plt.show()

# part b.
plt.figure(figsize=(6, 4))
sns.scatterplot(x='sepal_width', y='petal_width', data=iris_df, palette='husl', hue='petal_length')
plt.title('Scatter Plot: Petal Width vs Sepal Width')
plt.xlabel('Sepal Width (cm)')
plt.ylabel('Petal Width (cm)')
plt.legend(loc='upper right')
plt.show()
```

```
# part c.
plt.figure(figsize=(8, 4))
sns.kdeplot(iris_df['petal_length'], fill=True, label='Petal Length')
plt.title('Density Distribution of Petal Length in Iris Dataset')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Density')
plt.legend(loc='upper right')
plt.show()

# part d.
sns.set(style="ticks")
sns.pairplot(iris_df, markers=["o", "s", "D"])
plt.suptitle('Pairwise Bivariate Distribution in Iris Dataset', y=1.02)
plt.show()
```

Output:

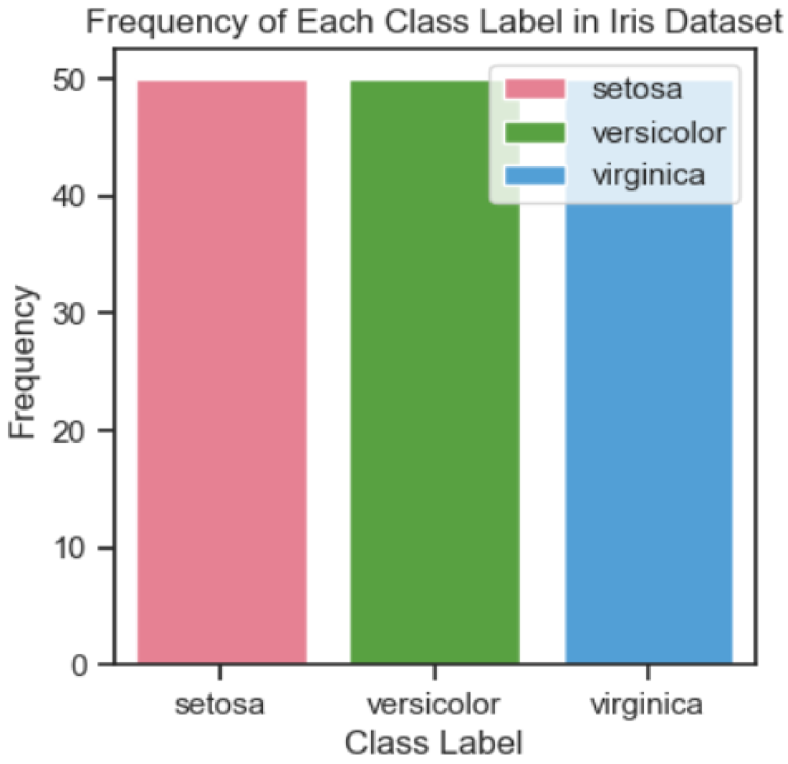
# Dataframe.



	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

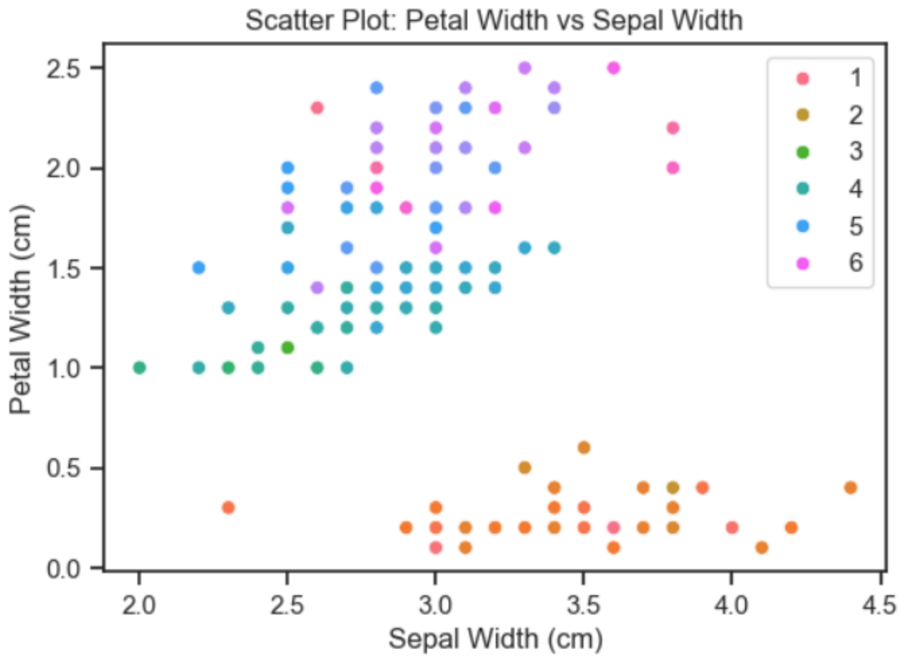
150 rows × 5 columns

# part a.



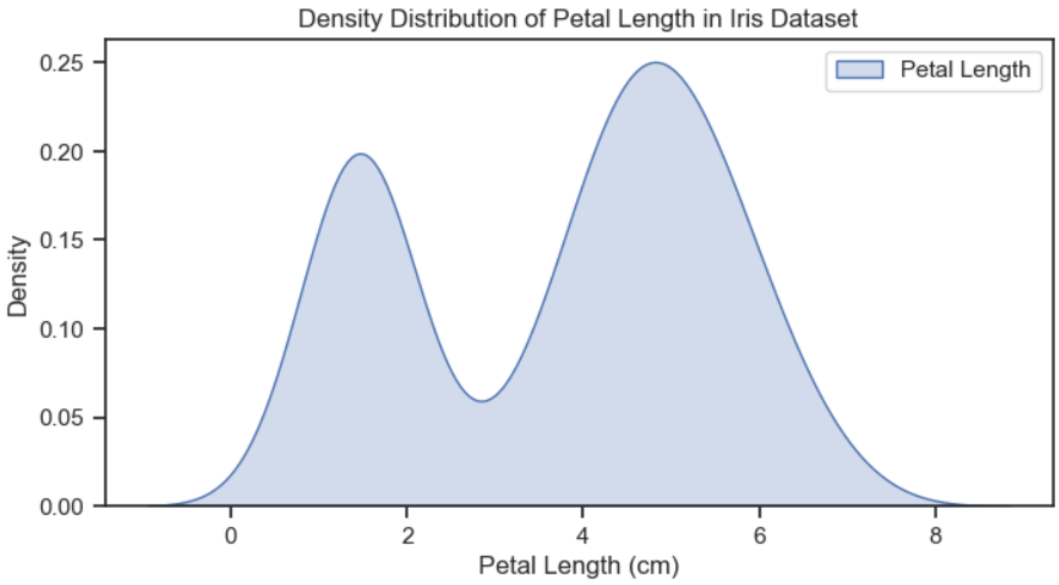
Frequency of Each Class Label in Iris Dataset
setosa
versicolor
virginica
Frequency
0
10
20
30
40
50
setosa
versicolor
virginica
Class Label

# part b.

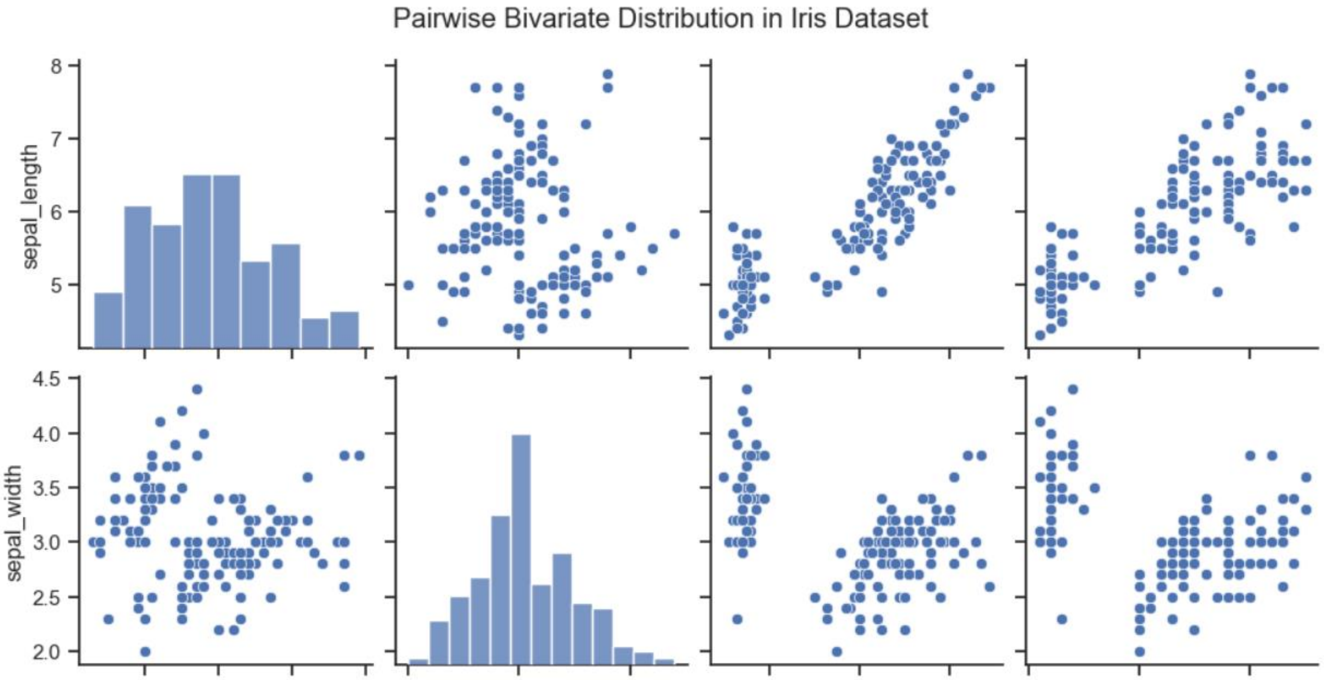


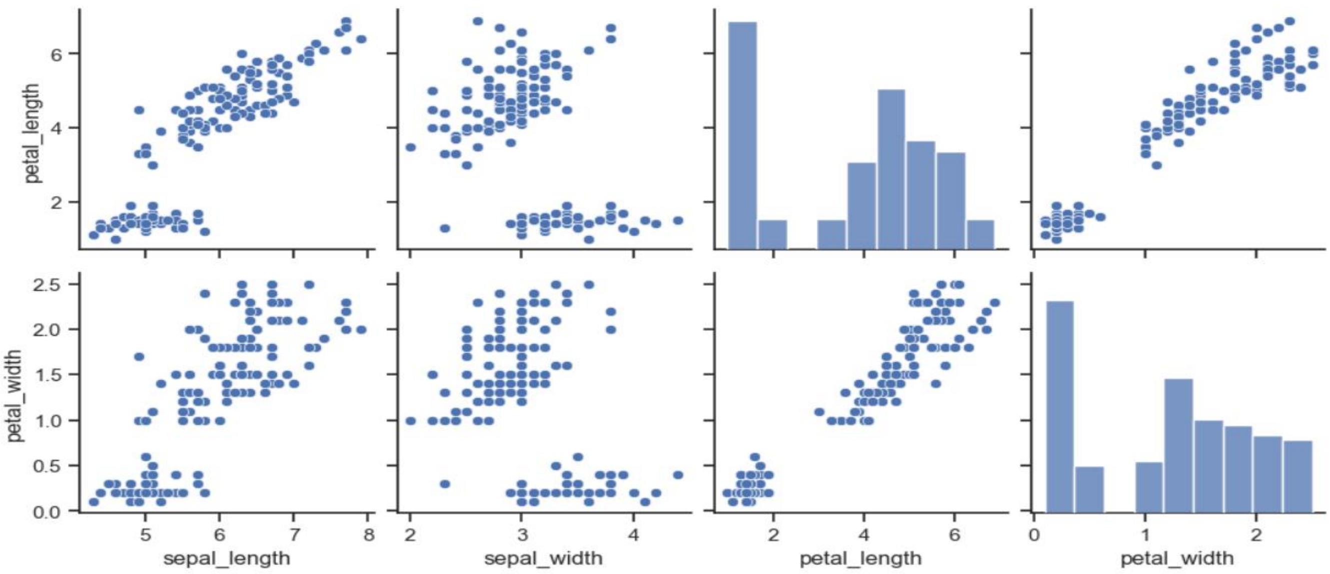
Scatter Plot: Petal Width vs Sepal Width
1
2
3
4
5
6
Petal Width (cm)
0.0
0.5
1.0
1.5
2.0
2.5
2.0
2.5
3.0
3.5
4.0
4.5
Sepal Width (cm)

# part c.


Density Distribution of Petal Length in Iris Dataset
Petal Length
Density
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
Petal Length (cm)

# part d.


Pairwise Bivariate Distribution in Iris Dataset
sepal_length
8
7
6
5
sepal_width
4.5
4.0
3.5
3.0
2.5
2.0


petal_length
6
4
2
petal_width
2.5
2.0
1.5
1.0
0.5
0.0
5
6
7
8
sepal_length
2
3
4
sepal_width
2
4
6
petal_length
0
1
2
petal_width

PRACTICAL – 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.

Code:

```
# Dataset.
import pandas as pd
data=pd.read_csv('GameSales.csv',encoding='latin-1')
data
```

```
# part a.
import numpy as np
data.groupby('Genre')['Global'].mean()
```

```
# part b.
data['Year'].fillna(inplace=True,method='ffill')
```

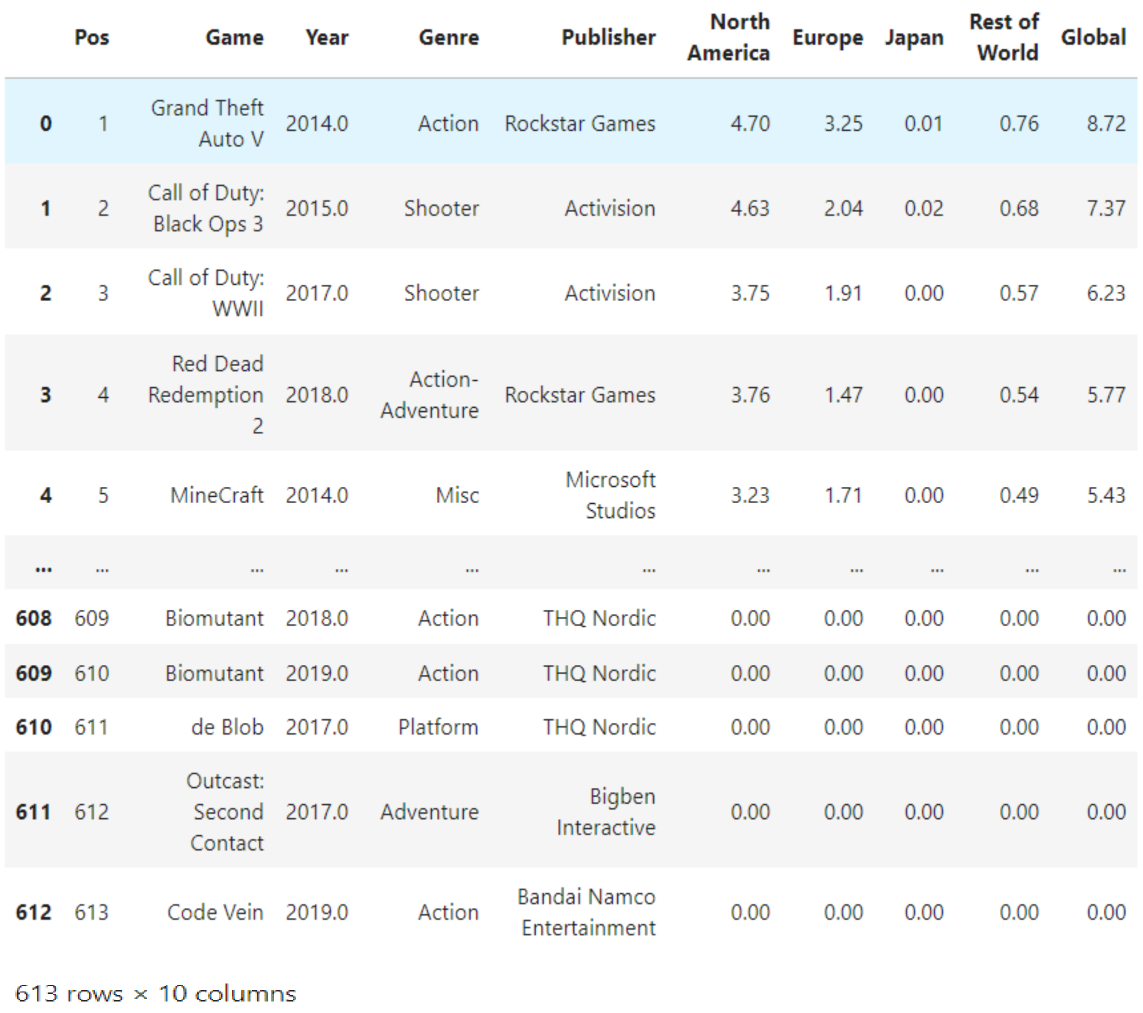
```
# part c.
from datetime import datetime
date=list()
for i in data['Year']:
    date.append(datetime.strptime(str(i),'%Y.%M'))
data['Year']=date
data
```

```
# part d.
pd.pivot_table(data,values='Global',index=['Genre','Year'],aggfunc='mean').sort_values(by='Global')
```

```
# part e.
grouped=data.groupby(['Genre',pd.cut(data['Global'],5)])
grouped.size().unstack()
```

Output:

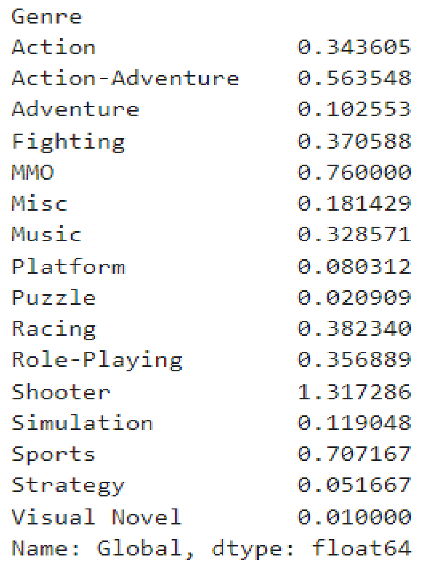
# Dataset.



	Pos	Game	Year	Genre	Publisher	North America	Europe	Japan	Rest of World	Global
0	1	Grand Theft Auto V	2014.0	Action	Rockstar Games	4.70	3.25	0.01	0.76	8.72
1	2	Call of Duty: Black Ops 3	2015.0	Shooter	Activision	4.63	2.04	0.02	0.68	7.37
2	3	Call of Duty: WWII	2017.0	Shooter	Activision	3.75	1.91	0.00	0.57	6.23
3	4	Red Dead Redemption 2	2018.0	Action-Adventure	Rockstar Games	3.76	1.47	0.00	0.54	5.77
4	5	MineCraft	2014.0	Misc	Microsoft Studios	3.23	1.71	0.00	0.49	5.43
...	...	...	...	...	...	...	...	...	...	...
608	609	Biomutant	2018.0	Action	THQ Nordic	0.00	0.00	0.00	0.00	0.00
609	610	Biomutant	2019.0	Action	THQ Nordic	0.00	0.00	0.00	0.00	0.00
610	611	de Blob	2017.0	Platform	THQ Nordic	0.00	0.00	0.00	0.00	0.00
611	612	Outcast: Second Contact	2017.0	Adventure	Bigben Interactive	0.00	0.00	0.00	0.00	0.00
612	613	Code Vein	2019.0	Action	Bandai Namco Entertainment	0.00	0.00	0.00	0.00	0.00

613 rows × 10 columns

# part a.

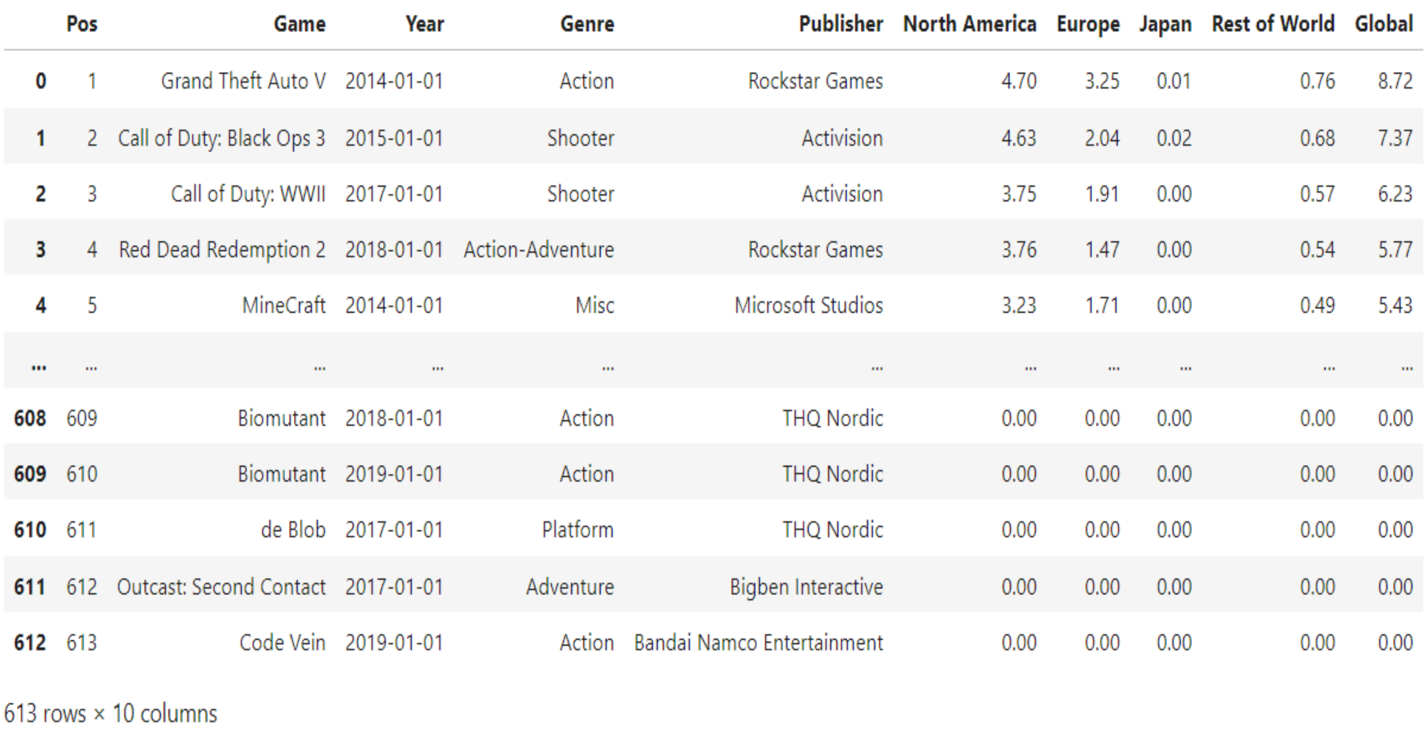


```
Genre
Action                 0.343605
Action-Adventure       0.563548
Adventure              0.102553
Fighting               0.370588
MMO                    0.760000
Misc                   0.181429
Music                  0.328571
Platform               0.080312
Puzzle                 0.020909
Racing                 0.382340
Role-Playing           0.356889
Shooter                1.317286
Simulation             0.119048
Sports                 0.707167
Strategy               0.051667
Visual Novel           0.010000
Name: Global, dtype: float64
```

# part b.

Replaced all the missing dates with values of previous non-missing dates

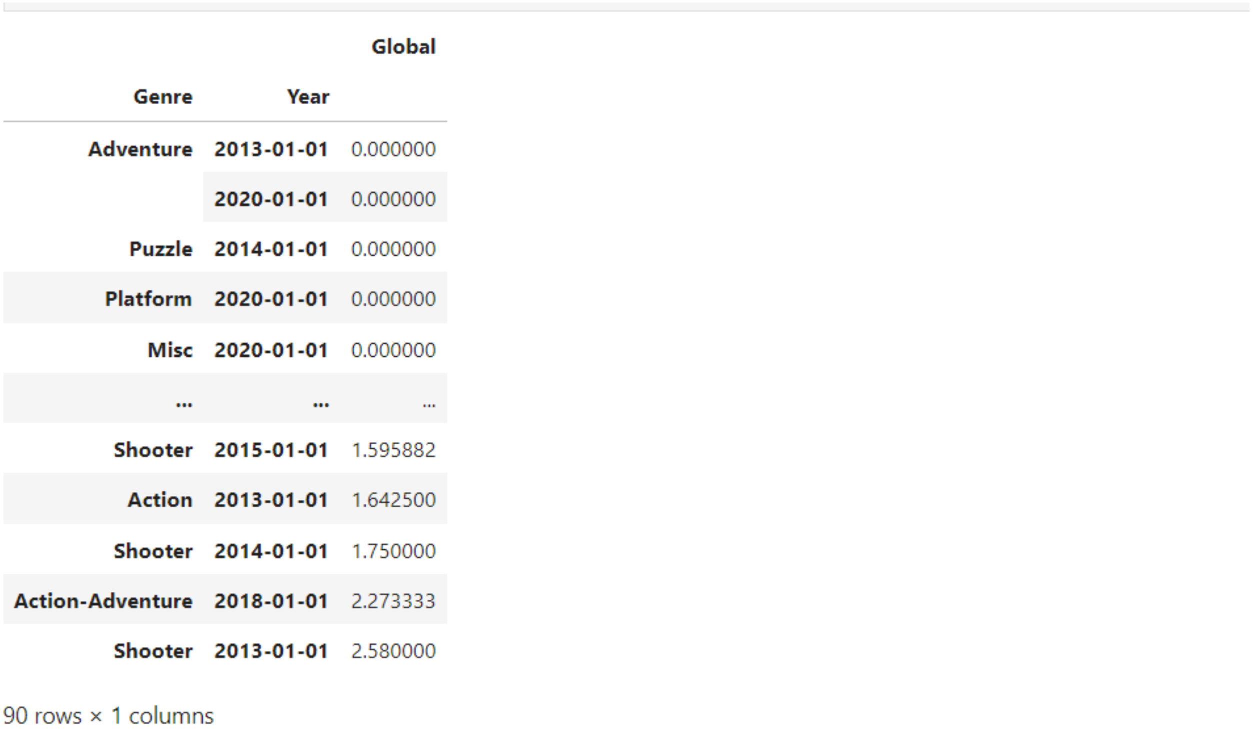
# part c.



	Pos	Game	Year	Genre	Publisher	North America	Europe	Japan	Rest of World	Global
0	1	Grand Theft Auto V	2014-01-01	Action	Rockstar Games	4.70	3.25	0.01	0.76	8.72
1	2	Call of Duty: Black Ops 3	2015-01-01	Shooter	Activision	4.63	2.04	0.02	0.68	7.37
2	3	Call of Duty: WWII	2017-01-01	Shooter	Activision	3.75	1.91	0.00	0.57	6.23
3	4	Red Dead Redemption 2	2018-01-01	Action-Adventure	Rockstar Games	3.76	1.47	0.00	0.54	5.77
4	5	MineCraft	2014-01-01	Misc	Microsoft Studios	3.23	1.71	0.00	0.49	5.43
...	...	...	...	...	...	...	...	...	...	...
608	609	Biomutant	2018-01-01	Action	THQ Nordic	0.00	0.00	0.00	0.00	0.00
609	610	Biomutant	2019-01-01	Action	THQ Nordic	0.00	0.00	0.00	0.00	0.00
610	611	de Blob	2017-01-01	Platform	THQ Nordic	0.00	0.00	0.00	0.00	0.00
611	612	Outcast: Second Contact	2017-01-01	Adventure	Bigben Interactive	0.00	0.00	0.00	0.00	0.00
612	613	Code Vein	2019-01-01	Action	Bandai Namco Entertainment	0.00	0.00	0.00	0.00	0.00

613 rows × 10 columns

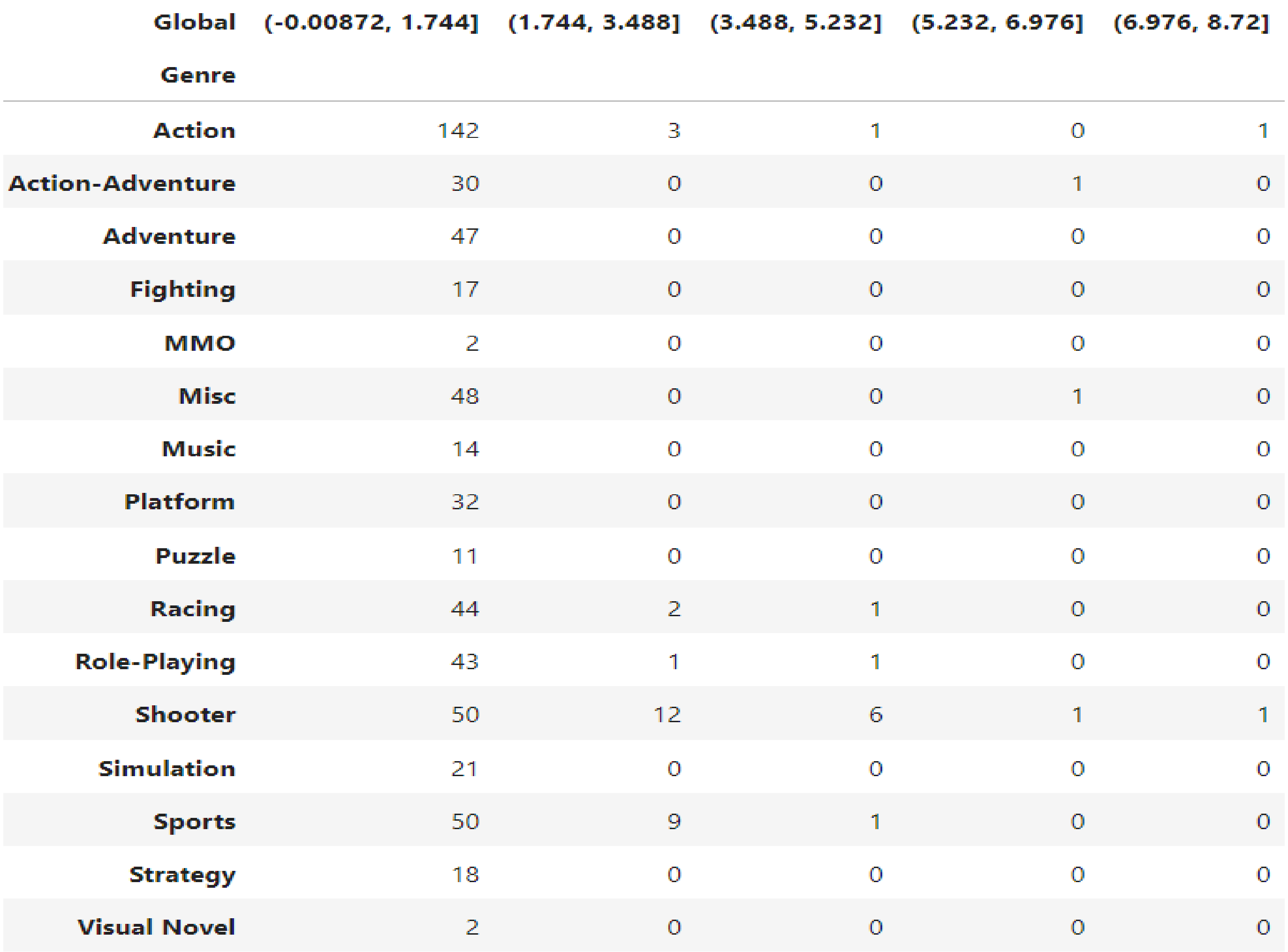
# part d.



		Global
Genre	**Year**	
Adventure	**2013-01-01**	0.000000
	2020-01-01	0.000000
Puzzle	**2014-01-01**	0.000000
Platform	**2020-01-01**	0.000000
Misc	**2020-01-01**	0.000000
...	**...**	...
Shooter	**2015-01-01**	1.595882
Action	**2013-01-01**	1.642500
Shooter	**2014-01-01**	1.750000
Action-Adventure	**2018-01-01**	2.273333
Shooter	**2013-01-01**	2.580000

90 rows × 1 columns

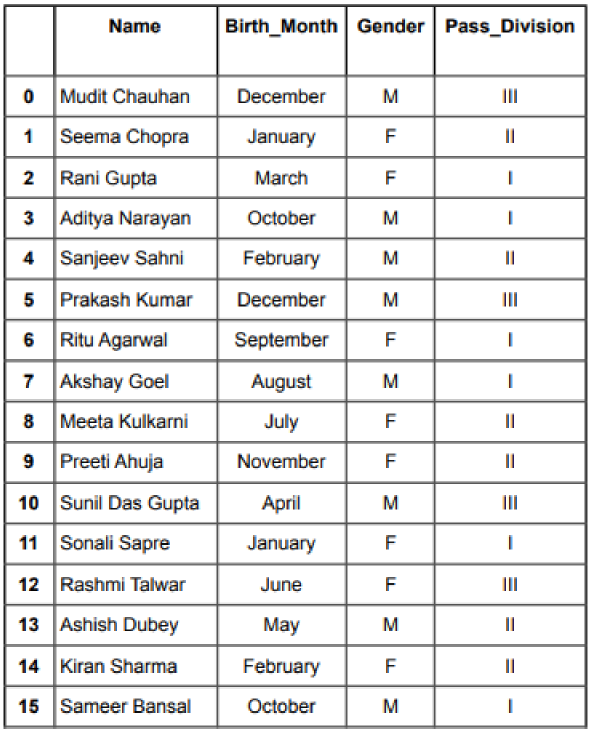
# part e.



Global	(-0.00872, 1.744]	(1.744, 3.488]	(3.488, 5.232]	(5.232, 6.976]	(6.976, 8.72]
Genre					
Action	142	3	1	0	1
Action-Adventure	30	0	0	1	0
Adventure	47	0	0	0	0
Fighting	17	0	0	0	0
MMO	2	0	0	0	0
Misc	48	0	0	1	0
Music	14	0	0	0	0
Platform	32	0	0	0	0
Puzzle	11	0	0	0	0
Racing	44	2	1	0	0
Role-Playing	43	1	1	0	0
Shooter	50	12	6	1	1
Simulation	21	0	0	0	0
Sports	50	9	1	0	0
Strategy	18	0	0	0	0
Visual Novel	2	0	0	0	0

PRACTICAL – 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.

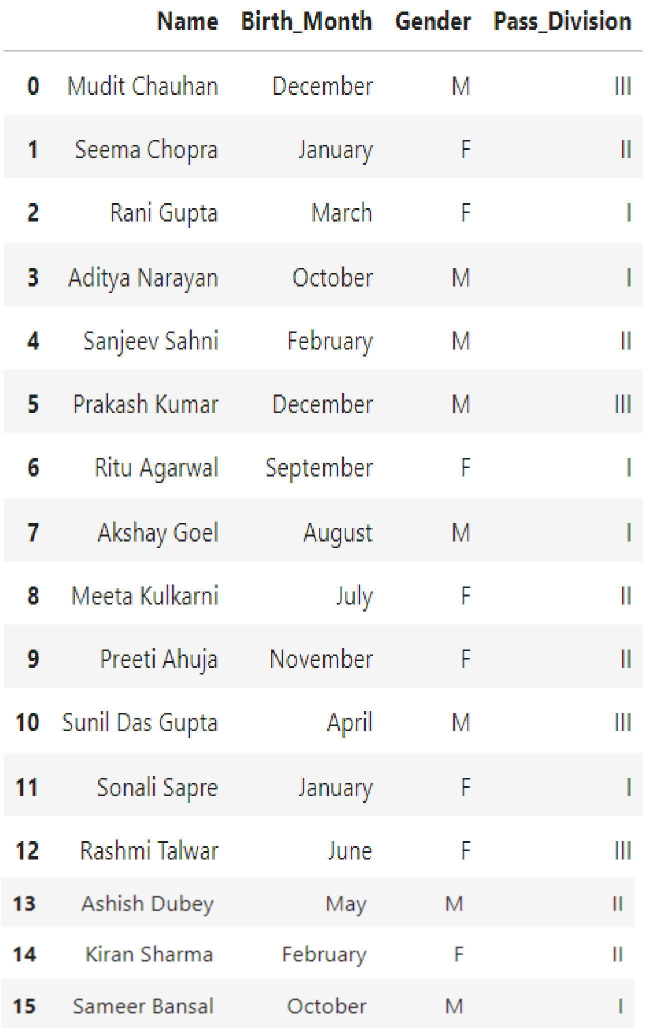
Code:

```
import numpy as np
import pandas as pd

# Dataframe.
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);
```

```
df

# part a.
hot_encoding_df=pd.get_dummies(df,columns=['Gender','Pass_Division'])
hot_encoding_df

# part b.
Month=['January','February','March','April','May','June','July','August','September','October','November','December']
df['Birth_Month']=pd.Categorical(df['Birth_Month'] , categories=Month , ordered=True)
df
# to dislay categories of categorical data
df['Birth_Month'].cat.categories
# codes of categorical data
df['Birth_Month'].cat.codes
```

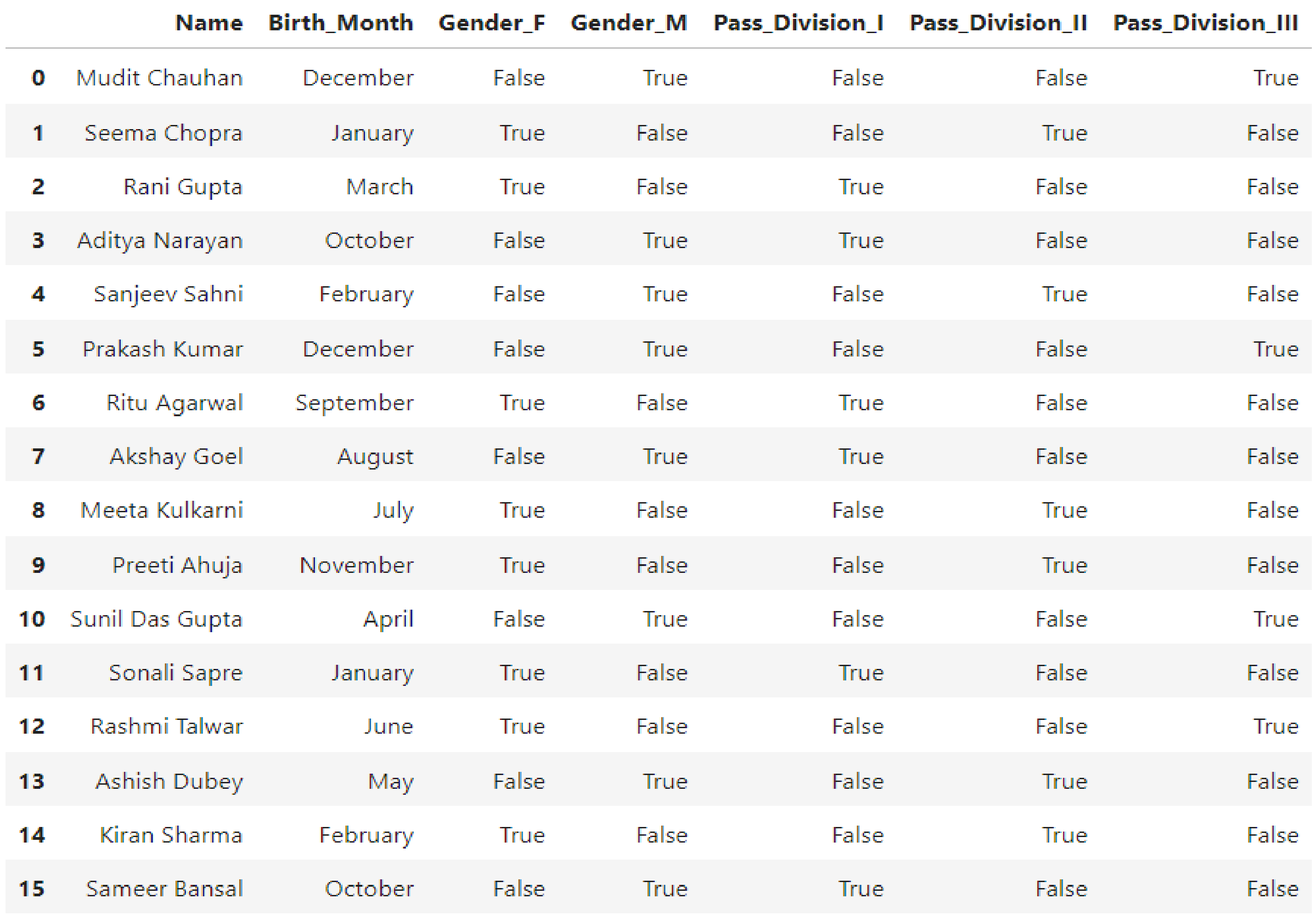
Output:

# Dataframe.



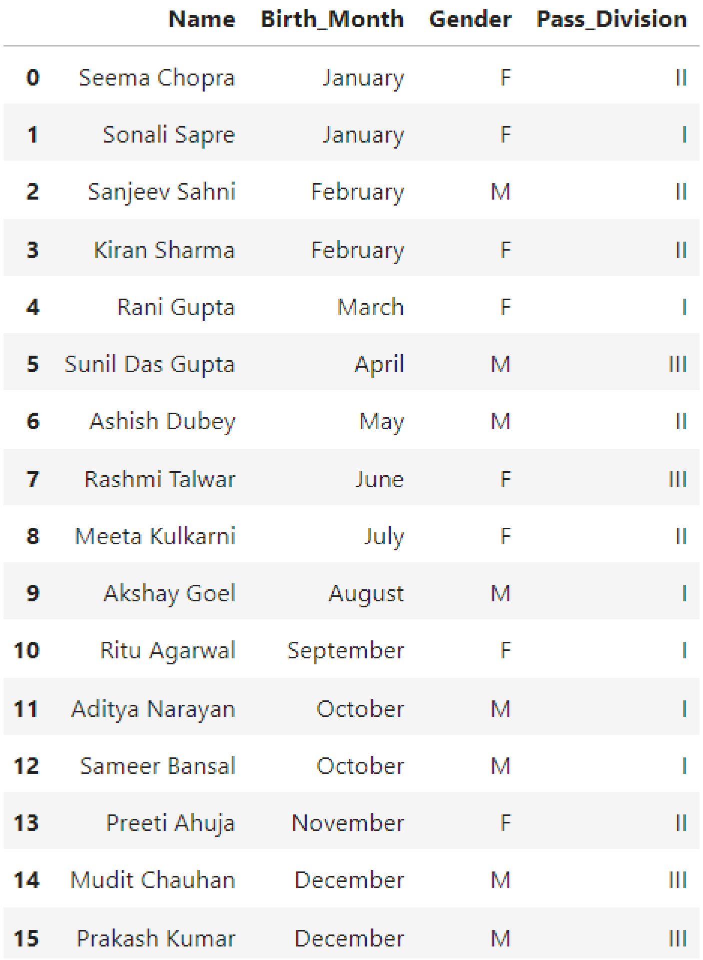
	Name	Birth_Month	Gender	Pass_Division
0	Mudit Chauhan	December	M	III
1	Seema Chopra	January	F	II
2	Rani Gupta	March	F	I
3	Aditya Narayan	October	M	I
4	Sanjeev Sahni	February	M	II
5	Prakash Kumar	December	M	III
6	Ritu Agarwal	September	F	I
7	Akshay Goel	August	M	I
8	Meeta Kulkarni	July	F	II
9	Preeti Ahuja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

# part a.



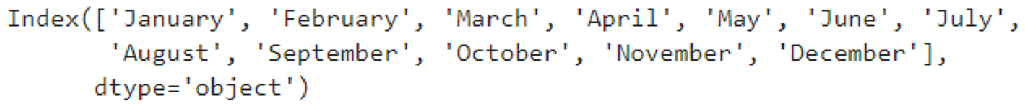
	Name	Birth_Month	Gender_F	Gender_M	Pass_Division_I	Pass_Division_II	Pass_Division_III
0	Mudit Chauhan	December	False	True	False	False	True
1	Seema Chopra	January	True	False	False	True	False
2	Rani Gupta	March	True	False	True	False	False
3	Aditya Narayan	October	False	True	True	False	False
4	Sanjeev Sahni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

# part b.



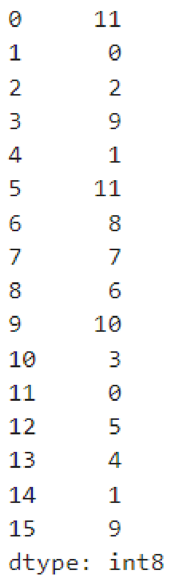
	Name	Birth_Month	Gender	Pass_Division
0	Seema Chopra	January	F	II
1	Sonali Sapre	January	F	I
2	Sanjeev Sahni	February	M	II
3	Kiran Sharma	February	F	II
4	Rani Gupta	March	F	I
5	Sunil Das Gupta	April	M	III
6	Ashish Dubey	May	M	II
7	Rashmi Talwar	June	F	III
8	Meeta Kulkarni	July	F	II
9	Akshay Goel	August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

Categories



```
Index(['January', 'February', 'March', 'April', 'May', 'June', 'July',
       'August', 'September', 'October', 'November', 'December'],
      dtype='object')
```

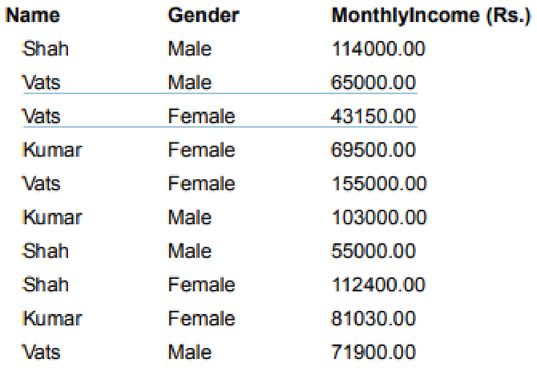
Code



```
0     11
1      0
2      2
3      9
4      1
5     11
6      8
7      7
8      6
9     10
10     3
11     0
12     5
13     4
14     1
15     9
dtype: int8
```

PRACTICAL – 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.

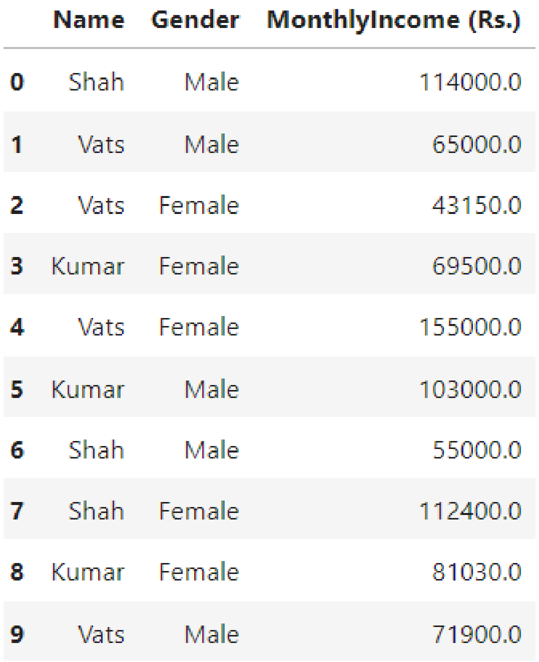
Code:

```
import numpy as np
import pandas as pd

# Dataframe.
data={'Name':['Shah','Vats','Vats','Kumar','Vats','Kumar','Shah','Shah','Kumar','Vats'],
     'Gender':['Male','Male','Female','Female','Female','Male','Male','Female','Female','Male'],
     'MonthlyIncome
(Rs.)':[114000.00,65000.00,43150.00,69500.00,155000.00,103000.00,55000.00,112400.00,81030.00,71900.00]
     }
df=pd.DataFrame(data);
df

# part a.
familywise_income = df.groupby('Name')['MonthlyIncome (Rs.)'].sum()
familywise_income

# part b.
idx = df.groupby('Name')['MonthlyIncome (Rs.)'].idxmax()
highest_income_members = df.loc[idx, ['Name', 'Gender', 'MonthlyIncome (Rs.)']]
```

```
highest_income_members

# part c.
high_income_members = df[df['MonthlyIncome (Rs.)'] > 60000.00]
high_income_members[['Name', 'Gender', 'MonthlyIncome (Rs.)']]

# part d.
shah_female_avg_income = df[(df['Name'] == 'Shah') & (df['Gender'] == 'Female')]['MonthlyIncome (Rs.)'].mean()
shah_female_avg_income
```

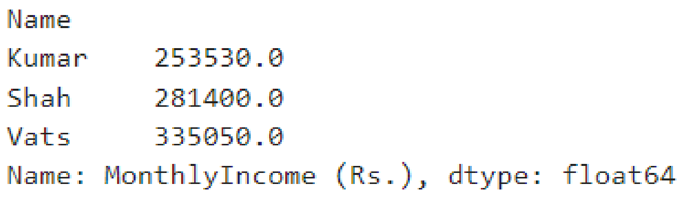
Output:

# Dataframe.



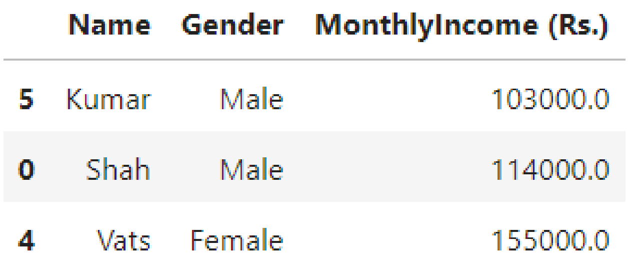
	Name	Gender	MonthlyIncome (Rs.)
0	Shah	Male	114000.0
1	Vats	Male	65000.0
2	Vats	Female	43150.0
3	Kumar	Female	69500.0
4	Vats	Female	155000.0
5	Kumar	Male	103000.0
6	Shah	Male	55000.0
7	Shah	Female	112400.0
8	Kumar	Female	81030.0
9	Vats	Male	71900.0

# part a.



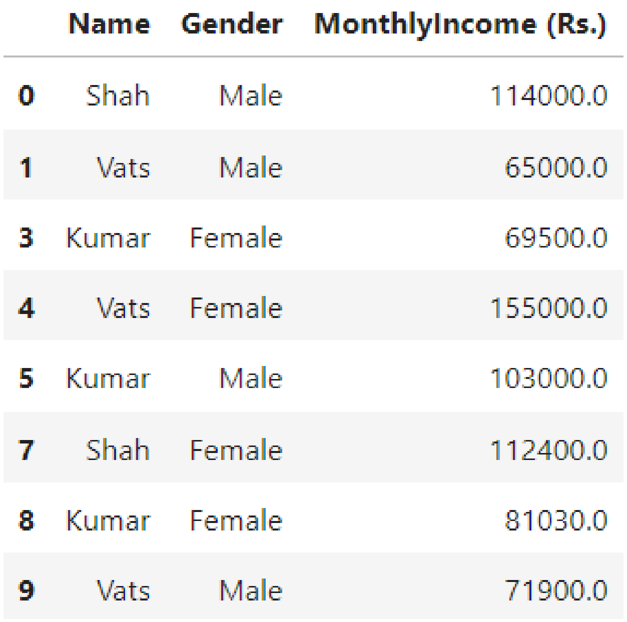
```
Name
Kumar    253530.0
Shah     281400.0
Vats     335050.0
Name: MonthlyIncome (Rs.), dtype: float64
```

# part b.



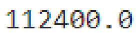
	Name	Gender	MonthlyIncome (Rs.)
5	Kumar	Male	103000.0
0	Shah	Male	114000.0
4	Vats	Female	155000.0

# part c.



	Name	Gender	MonthlyIncome (Rs.)
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

# part d.



```
112400.0
```