Practical Assignment 3

October 15, 2022

Name: Amartya Sinha Roll No: AC-1207

1. 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:
    1. Identify and count missing values in a dataframe.
    2. Drop the column having more than 5 null values.
    3. Identify the row label having maximum of the sum of all values in a row and drop that row.
    4. Sort the dataframe on the basis of the first column.
    5. Remove all duplicates from the first column.
    6. Find the correlation between first and second column and covariance between second and third column.
    7. Detect the outliers and remove the rows having outliers.
    8. Discretize second column and create 5 bins

[103]:

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

[104]:

```python
df1 = pd.DataFrame(np.random.randint(0, 1000, (50, 3)))
df1
```

[104]:

```
      0    1    2
0   721  103  191
1   376  191  427
2   533   36  196
3   272  188  136
4   751  551  982
5   590   49  526
6   850  439  966
7   237  112  333
8   426  730  632
9   372  623  668
10  296  896  250
11  404  588    8
12  562  409  356
13  767  120  671
14  295  336  761
15   77   81  197
```

```
16  679  884  240
17  628  138   44
18  838  608  876
19  520  186  976
20  102  874  129
21  736  582  830
22  277  922  270
23  263  521  835
24  507   90  757
25  479   78  222
26  276  337  483
27  687  523  939
28  627  600  576
29   77  410  415
30  257  158  693
31  655  806  227
32  439  637  583
33  323  495  484
34  106  553  222
35   29  286  816
36  299  458  767
37  209  871  182
38  520  780  558
39  983  912  935
40  899  260  383
41  782  446  445
42  980  705  217
43  994  128  573
44  974  377  597
45  262   20  135
46  178  614  950
47  974  631  989
48   97  803  995
49  140  168  946
```

[105]:

```
from itertools import product
from random import sample


total_nan = int(df1.size*0.1) #storing 10% of df size
possible_indices = list(product(range(df1.shape[0]), range(df1.shape[1]))) #creating list of all possible indices in df
random_indices = sample(possible_indices, total_nan) #selecting 10% random indices
```

```
x, y = zip(*random_indices)    #unzip random indices in x and y

np_arr = df1.to_numpy().astype(float)    #converting df1 to np_arr to insert nan
np_arr[x,y] = np.nan    #insert nan at (x,y) indices


final_df1 = pd.DataFrame(np_arr)

final_df1
```

```
[105]:     0      1      2
       0     NaN  103.0  191.0
       1   376.0  191.0  427.0
       2     NaN   36.0  196.0
       3   272.0  188.0  136.0
       4   751.0  551.0  982.0
       5   590.0   49.0  526.0
       6   850.0  439.0  966.0
       7     NaN  112.0  333.0
       8   426.0  730.0  632.0
       9   372.0  623.0  668.0
       10  296.0  896.0  250.0
       11  404.0  588.0    8.0
       12  562.0  409.0  356.0
       13  767.0  120.0  671.0
       14  295.0  336.0  761.0
       15    NaN   81.0  197.0
       16  679.0  884.0  240.0
       17  628.0  138.0    NaN
       18  838.0  608.0  876.0
       19  520.0  186.0  976.0
       20  102.0  874.0  129.0
       21  736.0  582.0  830.0
       22    NaN  922.0  270.0
       23  263.0    NaN  835.0
       24  507.0   90.0  757.0
       25  479.0    NaN  222.0
       26  276.0    NaN  483.0
       27  687.0  523.0  939.0
       28  627.0  600.0  576.0
       29   77.0  410.0  415.0
       30  257.0  158.0  693.0
       31  655.0  806.0  227.0
       32  439.0  637.0  583.0
```

```
33  323.0  495.0    NaN
34  106.0  553.0  222.0
35   29.0    NaN  816.0
36  299.0  458.0  767.0
37  209.0  871.0  182.0
38  520.0  780.0  558.0
39  983.0  912.0    NaN
40  899.0  260.0  383.0
41  782.0  446.0  445.0
42  980.0  705.0  217.0
43  994.0  128.0  573.0
44  974.0    NaN  597.0
45  262.0   20.0    NaN
46  178.0  614.0    NaN
47  974.0  631.0  989.0
48   97.0  803.0  995.0
49  140.0  168.0  946.0
```

[106]:

```
final_df1.isnull().sum()                #identify and count missing values in df
```

[106]:

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

[107]:

```
final_df1.dropna(axis=1, thresh=(len(final_df1)-5))         #thresh takes no of␣
 ↪min non nan values
```

[107]:

```
        0      1      2
0     NaN  103.0  191.0
1   376.0  191.0  427.0
2     NaN   36.0  196.0
3   272.0  188.0  136.0
4   751.0  551.0  982.0
5   590.0   49.0  526.0
6   850.0  439.0  966.0
7     NaN  112.0  333.0
8   426.0  730.0  632.0
9   372.0  623.0  668.0
10  296.0  896.0  250.0
11  404.0  588.0    8.0
12  562.0  409.0  356.0
13  767.0  120.0  671.0
14  295.0  336.0  761.0
15    NaN   81.0  197.0
16  679.0  884.0  240.0
17  628.0  138.0    NaN
```

```
18  838.0  608.0  876.0
19  520.0  186.0  976.0
20  102.0  874.0  129.0
21  736.0  582.0  830.0
22    NaN  922.0  270.0
23  263.0    NaN  835.0
24  507.0   90.0  757.0
25  479.0    NaN  222.0
26  276.0    NaN  483.0
27  687.0  523.0  939.0
28  627.0  600.0  576.0
29   77.0  410.0  415.0
30  257.0  158.0  693.0
31  655.0  806.0  227.0
32  439.0  637.0  583.0
33  323.0  495.0    NaN
34  106.0  553.0  222.0
35   29.0    NaN  816.0
36  299.0  458.0  767.0
37  209.0  871.0  182.0
38  520.0  780.0  558.0
39  983.0  912.0    NaN
40  899.0  260.0  383.0
41  782.0  446.0  445.0
42  980.0  705.0  217.0
43  994.0  128.0  573.0
44  974.0    NaN  597.0
45  262.0   20.0    NaN
46  178.0  614.0    NaN
47  974.0  631.0  989.0
48   97.0  803.0  995.0
49  140.0  168.0  946.0
```

[108]:

```
row_sum = final_df1.sum(axis=1)                 #store sum of all rows

display(row_sum.idxmax(), row_sum.max())        #display row index and max value
final_df1.drop(row_sum.idxmax())                #drop row using index
```

```
47
```

```
2594.0
```

[108]:

```
       0      1      2
0    NaN  103.0  191.0
1  376.0  191.0  427.0
2    NaN   36.0  196.0
3  272.0  188.0  136.0
4  751.0  551.0  982.0
```

```
5   590.0   49.0  526.0
6   850.0  439.0  966.0
7     NaN  112.0  333.0
8   426.0  730.0  632.0
9   372.0  623.0  668.0
10  296.0  896.0  250.0
11  404.0  588.0    8.0
12  562.0  409.0  356.0
13  767.0  120.0  671.0
14  295.0  336.0  761.0
15    NaN   81.0  197.0
16  679.0  884.0  240.0
17  628.0  138.0    NaN
18  838.0  608.0  876.0
19  520.0  186.0  976.0
20  102.0  874.0  129.0
21  736.0  582.0  830.0
22    NaN  922.0  270.0
23  263.0    NaN  835.0
24  507.0   90.0  757.0
25  479.0    NaN  222.0
26  276.0    NaN  483.0
27  687.0  523.0  939.0
28  627.0  600.0  576.0
29   77.0  410.0  415.0
30  257.0  158.0  693.0
31  655.0  806.0  227.0
32  439.0  637.0  583.0
33  323.0  495.0    NaN
34  106.0  553.0  222.0
35   29.0    NaN  816.0
36  299.0  458.0  767.0
37  209.0  871.0  182.0
38  520.0  780.0  558.0
39  983.0  912.0    NaN
40  899.0  260.0  383.0
41  782.0  446.0  445.0
42  980.0  705.0  217.0
43  994.0  128.0  573.0
44  974.0    NaN  597.0
45  262.0   20.0    NaN
46  178.0  614.0    NaN
48   97.0  803.0  995.0
49  140.0  168.0  946.0
```

```
[109]: final_df1.sort_values(0)          #sort df on the basis of first column
```

```
[109]:         0      1      2
       35   29.0    NaN  816.0
       29   77.0  410.0  415.0
       48   97.0  803.0  995.0
       20  102.0  874.0  129.0
       34  106.0  553.0  222.0
       49  140.0  168.0  946.0
       46  178.0  614.0    NaN
       37  209.0  871.0  182.0
       30  257.0  158.0  693.0
       45  262.0   20.0    NaN
       23  263.0    NaN  835.0
       3   272.0  188.0  136.0
       26  276.0    NaN  483.0
       14  295.0  336.0  761.0
       10  296.0  896.0  250.0
       36  299.0  458.0  767.0
       33  323.0  495.0    NaN
       9   372.0  623.0  668.0
       1   376.0  191.0  427.0
       11  404.0  588.0    8.0
       8   426.0  730.0  632.0
       32  439.0  637.0  583.0
       25  479.0    NaN  222.0
       24  507.0   90.0  757.0
       19  520.0  186.0  976.0
       38  520.0  780.0  558.0
       12  562.0  409.0  356.0
       5   590.0   49.0  526.0
       28  627.0  600.0  576.0
       17  628.0  138.0    NaN
       31  655.0  806.0  227.0
       16  679.0  884.0  240.0
       27  687.0  523.0  939.0
       21  736.0  582.0  830.0
       4   751.0  551.0  982.0
       13  767.0  120.0  671.0
       41  782.0  446.0  445.0
       18  838.0  608.0  876.0
       6   850.0  439.0  966.0
       40  899.0  260.0  383.0
       44  974.0    NaN  597.0
       47  974.0  631.0  989.0
       42  980.0  705.0  217.0
       39  983.0  912.0    NaN
       43  994.0  128.0  573.0
       0     NaN  103.0  191.0
```

```
2      NaN   36.0  196.0
7      NaN  112.0  333.0
15     NaN   81.0  197.0
22     NaN  922.0  270.0
```

[110]:

```
final_df1.drop_duplicates(subset=0)          #remove duplicate from 1st column
```

[110]:

```
        0      1      2
0     NaN  103.0  191.0
1   376.0  191.0  427.0
3   272.0  188.0  136.0
4   751.0  551.0  982.0
5   590.0   49.0  526.0
6   850.0  439.0  966.0
8   426.0  730.0  632.0
9   372.0  623.0  668.0
10  296.0  896.0  250.0
11  404.0  588.0    8.0
12  562.0  409.0  356.0
13  767.0  120.0  671.0
14  295.0  336.0  761.0
16  679.0  884.0  240.0
17  628.0  138.0    NaN
18  838.0  608.0  876.0
19  520.0  186.0  976.0
20  102.0  874.0  129.0
21  736.0  582.0  830.0
23  263.0    NaN  835.0
24  507.0   90.0  757.0
25  479.0    NaN  222.0
26  276.0    NaN  483.0
27  687.0  523.0  939.0
28  627.0  600.0  576.0
29   77.0  410.0  415.0
30  257.0  158.0  693.0
31  655.0  806.0  227.0
32  439.0  637.0  583.0
33  323.0  495.0    NaN
34  106.0  553.0  222.0
35   29.0    NaN  816.0
36  299.0  458.0  767.0
37  209.0  871.0  182.0
39  983.0  912.0    NaN
40  899.0  260.0  383.0
41  782.0  446.0  445.0
42  980.0  705.0  217.0
43  994.0  128.0  573.0
```

```
44  974.0    NaN  597.0
45  262.0   20.0    NaN
46  178.0  614.0    NaN
48   97.0  803.0  995.0
49  140.0  168.0  946.0
```

[111]:

```
print("Correlation betweeen 1st and 2nd columns:",final_df1[0].
 ↪corr(final_df1[1]))      #Correlation b/w 1st & 2nd cols
print("Covariance between 2nd and 3rd columns:", final_df1[1].
 ↪cov(final_df1[2]))        #Covariance b/w 2nd & 3rd cols
```

```
Correlation betweeen 1st and 2nd columns: -0.0019256533483172668
Covariance between 2nd and 3rd columns: -8244.820512820515
```

[112]:

```
df_mean, df_std = final_df1[1].mean(), final_df1[1].std()
# cut_off = 3*df_std
upper = df_mean + df_std*2
final_df1[final_df1[1]>(upper)]
# lower, upper = df_mean - cut_off, df_mean + cut_off

# outliers = [x for x in final_df1 if x<lower or x>upper]

# from scipy import stats
# final_df1[(np.abs(stats.zscore(final_df1))<3).all(axis=1)]      #detect␣
 ↪outliers and remove rows having outliers
```

[112]:

```
Empty DataFrame
Columns: [0, 1, 2]
Index: []
```

[113]:

```
final_df1['bins'] = pd.cut(final_df1[2], 5)                     #discretize 2nd␣
 ↪col & remove row with outliers
final_df1
```

[113]:

```
        0      1      2            bins
0     NaN  103.0  191.0  (7.013, 205.4]
1   376.0  191.0  427.0  (402.8, 600.2]
2     NaN   36.0  196.0  (7.013, 205.4]
3   272.0  188.0  136.0  (7.013, 205.4]
4   751.0  551.0  982.0  (797.6, 995.0]
5   590.0   49.0  526.0  (402.8, 600.2]
6   850.0  439.0  966.0  (797.6, 995.0]
7     NaN  112.0  333.0  (205.4, 402.8]
8   426.0  730.0  632.0  (600.2, 797.6]
9   372.0  623.0  668.0  (600.2, 797.6]
10  296.0  896.0  250.0  (205.4, 402.8]
11  404.0  588.0    8.0  (7.013, 205.4]
12  562.0  409.0  356.0  (205.4, 402.8]
13  767.0  120.0  671.0  (600.2, 797.6]
14  295.0  336.0  761.0  (600.2, 797.6]
15    NaN   81.0  197.0  (7.013, 205.4]
16  679.0  884.0  240.0  (205.4, 402.8]
17  628.0  138.0    NaN             NaN
18  838.0  608.0  876.0  (797.6, 995.0]
19  520.0  186.0  976.0  (797.6, 995.0]
20  102.0  874.0  129.0  (7.013, 205.4]
21  736.0  582.0  830.0  (797.6, 995.0]
22    NaN  922.0  270.0  (205.4, 402.8]
23  263.0    NaN  835.0  (797.6, 995.0]
24  507.0   90.0  757.0  (600.2, 797.6]
25  479.0    NaN  222.0  (205.4, 402.8]
26  276.0    NaN  483.0  (402.8, 600.2]
27  687.0  523.0  939.0  (797.6, 995.0]
28  627.0  600.0  576.0  (402.8, 600.2]
29   77.0  410.0  415.0  (402.8, 600.2]
30  257.0  158.0  693.0  (600.2, 797.6]
31  655.0  806.0  227.0  (205.4, 402.8]
32  439.0  637.0  583.0  (402.8, 600.2]
33  323.0  495.0    NaN             NaN
34  106.0  553.0  222.0  (205.4, 402.8]
35   29.0    NaN  816.0  (797.6, 995.0]
36  299.0  458.0  767.0  (600.2, 797.6]
37  209.0  871.0  182.0  (7.013, 205.4]
38  520.0  780.0  558.0  (402.8, 600.2]
39  983.0  912.0    NaN             NaN
40  899.0  260.0  383.0  (205.4, 402.8]
41  782.0  446.0  445.0  (402.8, 600.2]
42  980.0  705.0  217.0  (205.4, 402.8]
43  994.0  128.0  573.0  (402.8, 600.2]
44  974.0    NaN  597.0  (402.8, 600.2]
45  262.0   20.0    NaN             NaN
46  178.0  614.0    NaN             NaN
47  974.0  631.0  989.0  (797.6, 995.0]
48   97.0  803.0  995.0  (797.6, 995.0]
49  140.0  168.0  946.0  (797.6, 995.0]
```

2. Create a data frame to store marks of M students for n subjects and do the following:

    1. Find average marks for each student and add as a column
    2. Display average marks of each subject and add as a new row
    3. Compute descriptive statistics subject-wise
    4. Compute grade obtained by each student as per the examination policy of ur course (use lambda function)
    5. Find frequency of each grade for your class
    6. Find frequency of each grade obtained by each student and create a new DF as the following and set Rollno as the row index of the DF

[114]:
```
marks = {'Name': ['Amartya', 'Shahnwaz', 'Nilesh', 'Aditya'], 'DAV':[90, 80,␣
 ↪85, 70], 'IT':[95,100,96, 85], 'MP':[80,85,70, 65], 'ToC':[85,99,90, 62]}
df = pd.DataFrame(marks)
df
```

[114]:
```
       Name  DAV   IT  MP  ToC
0   Amartya   90   95  80   85
1  Shahnwaz   80  100  85   99
2    Nilesh   85   96  70   90
3    Aditya   70   85  65   62
```

[115]:
```
df['Average'] = df[['DAV','IT','MP','ToC']].mean(axis=1)           #add␣
 ↪average marks of each student in column
df
```

[115]:
```
       Name  DAV   IT  MP  ToC  Average
0   Amartya   90   95  80   85    87.50
1  Shahnwaz   80  100  85   99    91.00
2    Nilesh   85   96  70   90    85.25
3    Aditya   70   85  65   62    70.50
```

[116]:
```
df.loc['Sub Avg'] = df[['DAV','IT','MP','ToC']].mean().round(decimals=1)          ␣
 ↪    #add avg marks of each sub in row
df
```

[116]:
```
              Name   DAV     IT    MP   ToC  Average
0          Amartya  90.0   95.0  80.0  85.0    87.50
1         Shahnwaz  80.0  100.0  85.0  99.0    91.00
2           Nilesh  85.0   96.0  70.0  90.0    85.25
3           Aditya  70.0   85.0  65.0  62.0    70.50
Sub Avg        NaN  81.2   94.0  75.0  84.0      NaN
```

[117]:
```
display(df[['DAV','IT','MP','ToC']][0:-1].describe())          #didn't include␣
 ↪sub avg for descriptive statistics
```

```
             DAV          IT         MP        ToC
count   4.000000    4.000000   4.000000   4.000000
mean   81.250000   94.000000  75.000000  84.000000
std     8.539126    6.377042   9.128709  15.769168
min    70.000000   85.000000  65.000000  62.000000
25%    77.500000   92.500000  68.750000  79.250000
50%    82.500000   95.500000  75.000000  87.500000
75%    86.250000   97.000000  81.250000  92.250000
max    90.000000  100.000000  85.000000  99.000000
```

[118]:
```
df
```

[118]:

```
             Name   DAV     IT    MP   ToC  Average
0        Amartya  90.0   95.0  80.0  85.0    87.50
1       Shahnwaz  80.0  100.0  85.0  99.0    91.00
2         Nilesh  85.0   96.0  70.0  90.0    85.25
3        Aditya  70.0   85.0  65.0  62.0    70.50
Sub Avg       NaN  81.2   94.0  75.0  84.0      NaN
```

[119]:

```
#calculating grades for each student on the basis of average marks
df['Grades'] = df.apply(lambda x: 'A' if x['Average']>90 else ('B' if␣
 ↪x['Average']>80 else ('C' if x['Average']>70 else ('F' if x['Average']<33␣
 ↪else ('D')))), axis=1).head(-1)
```

[120]:

```
df
```

[120]:

```
             Name   DAV     IT    MP   ToC  Average Grades
0        Amartya  90.0   95.0  80.0  85.0    87.50      B
1       Shahnwaz  80.0  100.0  85.0  99.0    91.00      A
2         Nilesh  85.0   96.0  70.0  90.0    85.25      B
3        Aditya  70.0   85.0  65.0  62.0    70.50      C
Sub Avg       NaN  81.2   94.0  75.0  84.0      NaN    NaN
```

[121]:

```
df['Grades'].value_counts()                          #count frequency of each grade
```

[121]:

```
B    2
A    1
C    1
Name: Grades, dtype: int64
```

[122]:

```
#add grades of each student subjeect wise
new_df_grades = pd.DataFrame()
new_df_grades['Name'] = df['Name'].head(-1)
new_df_grades.index.names=['Roll No']
for sub in ['DAV', 'IT', 'MP', 'ToC']:
    new_df_grades[sub] = df.apply(lambda x: 'A' if x[sub]>90 else ('B' if␣
 ↪x[sub]>80 else ('C' if x[sub]>70 else ('F' if x[sub]<33 else ('D')))),␣
 ↪axis=1).head(-1)

new_df_grades.index+=1
```

[123]:

```
new_df_grades
```

[123]:

```
             Name DAV IT MP ToC
Roll No
1        Amartya   B  A  C   B
2       Shahnwaz   C  A  B   A
3         Nilesh   B  A  D   B
4        Aditya   D  B  D   D
```

[124]:
```
new_df_grades['Max Grade Obtained'] = new_df_grades.mode(axis=1)[0]
# new_df_grades['Freq'] = new_df_grades.value_counts().mode()
# new_df_grades.mode(axis=1)
new_df_grades['Frequency']=[new_df_grades[['DAV','IT','MP','ToC']].iloc[i].
 ↪value_counts().max() for i in range(len(new_df_grades))]

new_df_grades
```

[124]:
```
              Name DAV IT MP ToC Max Grade Obtained  Frequency
Roll No
1          Amartya   B  A  C   B                  B          2
2         Shahnwaz   C  A  B   A                  A          2
3           Nilesh   B  A  D   B                  B          2
4          Aditya    D  B  D   D                  D          3
```

3. Input two lists of hobbies where hobbies may be same in two lists as well as a list may have duplicate hobbies. Create a data series for the hobbies s.t. hobby type is the index label and count of that hobby is its value.

[125]:
```
hobby_lst1 = ['chess', 'cricket', 'traveling', 'dance', 'coding', 'chess',␣
 ↪'traveling', 'traveling']
hobby_lst2 = ['cricket', 'traveling', 'coding', 'dance', 'chess', 'traveling',␣
 ↪'dance', 'traveling']

hobby_series = pd.Series(dtype=int)

for i in hobby_lst1:
    hobby_series[i] = hobby_lst1.count(i)
    if i in hobby_series.index:
        hobby_series[i] += hobby_lst2.count(i)
display(hobby_series)
```

```
chess        3
cricket      2
traveling    6
dance        3
coding       2
dtype: int64
```

4. Consider two csv files of students of years 2019 and 2020 having following details (student name, hobby, course) where courses are (Cshons, bcomhons, PSCS) and hobbies are (writing, painting, music, dancing). Answer the following:

    1. Find all hobbies types for each course in both years
    2. Find hobbies which are there in 2019 but not in 2020 for each course
    3. Find common hobbies in both year
    4. Find course name in which students are exploring all hobbies
    5. Find count of students exploring each hobby in both year

[126]:
```
SD2019 = pd.DataFrame(pd.read_csv('StudentData2019.csv'))
SD2020 = pd.DataFrame(pd.read_csv('StudentData2020.csv'))
complete_data = pd.concat([SD2019,SD2020])
display(SD2019)
display(SD2020)
```

```
  Student Name     Hobby Course
0       Nilesh   Dancing     CS
1     Shahnwaz  Painting   PMCS
2      Prakash     Music   BCom
3       Divyam     Music     CS
4      Amartya   Writing     CS
5         Ayan   Dancing   PMCS
6      Avinash     Music   BCom
```

```
  Student Name     Hobby Course
0         Asad   Dancing     CS
1    Deepanshu  Painting   PMCS
2       Sahiba   Writing     CS
3       Shreya     Music     CS
4      Tanisha   Dancing   PMCS
5       Khushi   Writing   BCom
6         Yash     Music   BCom
7      Avinash  Painting     CS
8      Rishabh   Writing     CS
```

[127]:
```
SD2019.loc[[2,4]]
```

[127]:
```
  Student Name    Hobby Course
2      Prakash    Music   BCom
4      Amartya  Writing     CS
```

[128]:
```
A_D=complete_data.groupby(by=['Course','Hobby']).first().drop(columns='Student␣
 ↪Name')
A_D                    #hobby types for each course in both years
```

[128]:
```
Empty DataFrame
Columns: []
Index: [(BCom, Music), (BCom, Writing), (CS, Dancing), (CS, Music), (CS,
Painting), (CS, Writing), (PMCS, Dancing), (PMCS, Painting)]
```

[129]:
```
SD19=SD2019.groupby('Course').apply(lambda x:x['Hobby'].unique())      ␣
 ↪#hobbies which are there in 2019 but not in 2020 for each course
display(SD19)
```

```
Course
BCom                       [Music]
CS        [Dancing, Music, Writing]
PMCS            [Painting, Dancing]
```

```
dtype: object
```

```
[130]: SD20=SD2020.groupby('Course').apply(lambda x:x['Hobby'].unique())
```

```
[131]: SD20
```

```
[131]: Course
       BCom                            [Writing, Music]
       CS       [Dancing, Writing, Music, Painting]
       PMCS                         [Painting, Dancing]
       dtype: object
```

```
[132]: SD19.map(set)-SD20.map(set)
```

```
[132]: Course
       BCom     {}
       CS       {}
       PMCS     {}
       dtype: object
```

```
[133]: set(SD2019['Hobby']) & set(SD2020['Hobby'])         #common hobbies in both␣
         ↪years
```

```
[133]: {'Dancing', 'Music', 'Painting', 'Writing'}
```

```
[134]: SD20            #Course name in which students are exploring all hobbies
```

```
[134]: Course
       BCom                            [Writing, Music]
       CS       [Dancing, Writing, Music, Painting]
       PMCS                         [Painting, Dancing]
       dtype: object
```

```
[145]: # c = [SD20['Course'].iloc[i] for i in range(len(SD20)) if(len(SD20[0].
         ↪iloc[i])==4)]
       # c
       for i, j in enumerate(SD20):
           if(len(j) == 4):
               print(SD20.keys()[i])
```

```
CS
```

```
[136]: complete_data.groupby('Hobby')['Student Name'].count()      #student count␣
         ↪exploring each hobby in both years
```

```
[136]: Hobby
       Dancing     4
       Music       5
       Painting    3
```

```
Writing     4
Name: Student Name, dtype: int64
```

5. Use csv file handling to do the following:

    1. Read two csv files, remove all rows with any null value. If two files are compatible in terms of record structure combine them and store as in a new file
    2. Create a new data frame ‘New’ storing specified requirement ahead for each column as a row. If column is numeric then maintain mean, median, standard deviation else maintain number of distinct values, value which is appearing maximum time and its count. Assign user-specified row labels
    3. Store New DF as an excel file

[137]:

```
#a) Read two csv file and remove all the null values.
# If two files are compatible in terms of record structure then combine them␣
 ↪and save them as new file.
import pandas as pd
md1=pd.read_csv('marksdata1.csv')
md2=pd.read_csv('marksdata2.csv')
display(md1)
display(md2)
md1.dropna(inplace=True)
md2.dropna(inplace=True)
display(md1)
display(md2)
if(len(md1.columns)==len(md2.columns)):
    all_md=pd.concat([md1,md2],ignore_index=True)

all_md.to_csv('AllMarksData.csv')
#b) Create a new dataframe 'NEW' storing speciefied requirement ahead of each␣
 ↪column as a row.
# If column is numeric then maintain mean, median, standard deviation else␣
 ↪maintain number of distinct values,
# value which is appearing maximum time amd its count. Assign user-specifiedrow␣
 ↪labels.
New=all_md.copy()
New
New.loc['Mean']=New.mean(numeric_only=True)
New.loc['Median']=New.median(numeric_only=True)
New.loc['Standard Deviation']=New.std(numeric_only=True)
New

New.loc['Distinct Values']=New.nunique()
New.loc['MaximumCount']=[New[i].value_counts().max() for i in New.columns]
New.loc['Max_value']=[New[i].value_counts().idxmax() for i in New.columns]
New
New.to_excel('NewMarksDataQ5.xlsx')
```

```
       Name  Python  Java    JS
0  Abhishek    78.0   NaN  80.0
1     Aditi    88.0  79.0  79.0
2    Aditya    97.0  98.0   NaN
3      Aman     NaN  79.0  75.0
4   Amartya    96.0  94.0  89.0
```

```
      Name  Python  Java  JS
0     Amit      76    86  89
1  Amitesh      93    70  78
2    Ankit      97    91  76
3   Ananya      99    95  93
4     Anam      92    99  96
5   Divyam      90    99  87
```

```
      Name  Python  Java    JS
1    Aditi    88.0  79.0  79.0
4  Amartya    96.0  94.0  89.0
```

```
      Name  Python  Java  JS
0     Amit      76    86  89
1  Amitesh      93    70  78
2    Ankit      97    91  76
3   Ananya      99    95  93
4     Anam      92    99  96
5   Divyam      90    99  87
```

6. Create a database with two tables, where first table is having grades obtained by students in a class (computed from Qs 2) and another table is having range of marks for that grade (say grade A range is min 95 max100). Use both tables to display average marks for each student (use database manipulation and retrieval)

[138]:

```
import sqlite3

conn = sqlite3.connect('q6_database')
c = conn.cursor()
c.execute('DROP TABLE IF EXISTS Marks_Range')
c.execute('DROP TABLE IF EXISTS Grade_Table')
conn.commit()
c.execute('CREATE TABLE IF NOT EXISTS Grade_Table (Name text, Grades text)')

c.execute('CREATE TABLE IF NOT EXISTS Marks_Range (Grade text, Min number, Max␣
 ↪number)')

# df['Grades'] = df.apply(lambda x: 'A' if x['Average']>90 else ('B' if␣
 ↪x['Average']>80 else ('C' if x['Average']>70 else ('F' if x['Average']<33␣
 ↪else ('D')))), axis=1).head(-1)

c.execute('''INSERT INTO Marks_Range VALUES('A', 91, 100)''')
```

```
c.execute('''INSERT INTO Marks_Range VALUES('B', 81, 90)''')
c.execute('''INSERT INTO Marks_Range VALUES('C', 71, 80)''')
c.execute('''INSERT INTO Marks_Range VALUES('D', 33, 70)''')
c.execute('''INSERT INTO Marks_Range VALUES('F', 0, 32)''')

df[['Name', 'Grades']].head(-1).to_sql('Grade_Table', conn,␣
 ↪if_exists='replace', index=False)

c.execute('SELECT * FROM Grade_Table')
print('Grade_Table Table:')
for row in c.fetchall():
    print (row)

c.execute('SELECT * FROM Marks_Range')
print('\nMarks_Range Table:')
for row in c.fetchall():
    print (row)

c.execute('SELECT Grade_Table.Name, (Marks_Range.Min+Marks_Range.Max)/2 FROM␣
 ↪Grade_Table join Marks_Range on Grade_Table.Grades = Marks_Range.Grade')
print('\nAvg Marks:')
for row in c.fetchall():
    print(row)
```

```
Grade_Table Table:
('Amartya', 'B')
('Shahnwaz', 'A')
('Nilesh', 'B')
('Aditya', 'C')

Marks_Range Table:
('A', 91, 100)
('B', 81, 90)
('C', 71, 80)
('D', 33, 70)
('F', 0, 32)

Avg Marks:
('Amartya', 85)
('Shahnwaz', 95)
('Nilesh', 85)
('Aditya', 75)
```

7. Given is a folder ‘XXX’ containing 10 years sec options files of format csv/xIxs where record structure of each file is same and is as given below:

   Name of file sec-2015

   (Rollno, name, course, semester, year, choicel,choice2,choice3) (you may use same file multiple times by doing minor changes)

Do the following:

1. Find total number of students who have opted first choice as ‘Python Programming’ in all files. Choice to be searched and folder path/name needs to be passed as system arguments to the program
2. Compute total number of students filling choices per year. Donot consider duplicate records and records with no choice filled

[2]:

```
import sys
import glob
import pandas as pd

path=sys.argv[1]
choice=sys.argv[2]

secFiles=glob.glob(path+'/*.csv')
allSecFiles=(pd.read_csv(file) for file in secFiles)

y=2021
secDF=pd.concat(allSecFiles)
secDF.reset_index(inplace=True)

#print(secDF)
print('The total number of students who have opted for python are:')
print(sum(secDF['choice1']=='python'))

allSecFiles=[pd.read_csv(file) for file in secFiles]

for i in allSecFiles:
    y=y-1
    print('Total number of students filled choices in year',y)
    i.dropna(subset=['choice1','choice2','choice3'],inplace=True,how='all')
    #print(i)

    print(len(i)-(i.duplicated(subset=['choice1','choice2','choice3']).sum()))
```

```
[amartya@Firebolt DAV]$ python Prac_Ass3_Q7.py 'XXX' 'python'
The total number of students who have opted for python are:
9
Total number of students filled choices in year 2020
3
Total number of students filled choices in year 2019
3
Total number of students filled choices in year 2018
3
Total number of students filled choices in year 2017
3
```

```
Total number of students filled choices in year 2016
3
Total number of students filled choices in year 2015
3
Total number of students filled choices in year 2014
2
Total number of students filled choices in year 2013
2
Total number of students filled choices in year 2012
3
Total number of students filled choices in year 2011
3
```

8. Use Web API to download data from a URL and display some useful information

```
[ ]: import requests

     url = requests.get('https://api.github.com/users/amartyasinha918')
     json_file = url.json()

     my_df = pd.DataFrame(json_file, index=[0])

     display(my_df)
```

```
             login        id               node_id  \
0  amartyasinha918  81137946  MDQ6VXNlcjgxMTM3OTQ2

                                           avatar_url gravatar_id  \
0  https://avatars.githubusercontent.com/u/811379…

                                           url  \
0  https://api.github.com/users/amartyasinha918

                             html_url  \
0  https://github.com/amartyasinha918

                                       followers_url  \
0  https://api.github.com/users/amartyasinha918/f…

                                       following_url  \
0  https://api.github.com/users/amartyasinha918/f…

                                           gists_url  … email hireable  \
0  https://api.github.com/users/amartyasinha918/g…  …  None     None

    bio twitter_username public_repos public_gists followers  following  \
0  None  amartyasinha918           25            0         8          2

             created_at             updated_at
```

```
0  2021-03-22T04:47:04Z  2022-08-24T12:35:39Z

[1 rows x 32 columns]
```

[ ]:

```
my_df[['login', 'name', 'public_repos', 'followers', 'following', 'created_at']]
```

[ ]:

```
             login          name  public_repos  followers  following  \
0  amartyasinha918  Amartya Sinha            25          8          2

             created_at
0  2021-03-22T04:47:04Z
```