DAV Practical Assignment 3

September 18, 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

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

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

[ ]:

	0	1	2
0	570	812	917
1	186	729	101
2	955	462	259
3	130	44	844
4	747	974	414
5	605	143	507
6	193	369	180
7	46	370	422
8	133	914	464
9	365	752	831
10	565	958	74
11	144	857	676
12	373	232	20
13	895	156	432
14	412	371	754
15	323	163	607

```
16  595  821  452
17  157  420  180
18  532  668  270
19  848  400   79
20  598  823  293
21    0  468  789
22  277  270  637
23  360  686  784
24   68  676  126
25   30  239  981
26  892  597  429
27  975  424  436
28  855  916  682
29  690  309  447
30  932  842  169
31  679  661  518
32  543  891  363
33  853  712  952
34  676  389  767
35  763  864   21
36  704  870  324
37  655  253  990
38  830  755  500
39  616  401  838
40  650   71  742
41  420  586  545
42   78  195  631
43  677  610  904
44  185  656  721
45  532  935  975
46  761  984  539
47  328  280  770
48  554  888  513
49  875  359  765
```

[ ]:
```
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
```

[ ]:

	0	1	2
0	570.0	812.0	917.0
1	186.0	729.0	101.0
2	955.0	462.0	259.0
3	NaN	44.0	844.0
4	747.0	974.0	414.0
5	605.0	143.0	507.0
6	NaN	NaN	180.0
7	46.0	370.0	422.0
8	133.0	914.0	464.0
9	365.0	752.0	831.0
10	565.0	958.0	74.0
11	144.0	857.0	676.0
12	373.0	232.0	20.0
13	895.0	156.0	432.0
14	412.0	371.0	754.0
15	323.0	163.0	607.0
16	595.0	821.0	452.0
17	157.0	420.0	180.0
18	NaN	668.0	270.0
19	848.0	400.0	79.0
20	598.0	823.0	293.0
21	0.0	468.0	789.0
22	NaN	270.0	637.0
23	NaN	686.0	784.0
24	68.0	676.0	126.0
25	30.0	NaN	981.0
26	892.0	597.0	429.0
27	975.0	424.0	436.0
28	855.0	916.0	NaN
29	690.0	309.0	447.0
30	932.0	842.0	169.0
31	679.0	661.0	518.0
32	543.0	NaN	363.0

```
33    NaN  712.0  952.0
34  676.0    NaN  767.0
35  763.0  864.0   21.0
36  704.0  870.0  324.0
37  655.0  253.0  990.0
38  830.0  755.0  500.0
39  616.0  401.0  838.0
40  650.0   71.0  742.0
41  420.0  586.0    NaN
42    NaN  195.0  631.0
43  677.0  610.0  904.0
44  185.0  656.0  721.0
45    NaN    NaN  975.0
46  761.0  984.0  539.0
47  328.0  280.0  770.0
48  554.0  888.0  513.0
49  875.0  359.0  765.0
```

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

[ ]:
```
0    8
1    5
2    2
dtype: int64
```

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

[ ]:
```
         1      2
0   812.0  917.0
1   729.0  101.0
2   462.0  259.0
3    44.0  844.0
4   974.0  414.0
5   143.0  507.0
6     NaN  180.0
7   370.0  422.0
8   914.0  464.0
9   752.0  831.0
10  958.0   74.0
11  857.0  676.0
12  232.0   20.0
13  156.0  432.0
14  371.0  754.0
15  163.0  607.0
16  821.0  452.0
17  420.0  180.0
```

```
18  668.0  270.0
19  400.0   79.0
20  823.0  293.0
21  468.0  789.0
22  270.0  637.0
23  686.0  784.0
24  676.0  126.0
25    NaN  981.0
26  597.0  429.0
27  424.0  436.0
28  916.0    NaN
29  309.0  447.0
30  842.0  169.0
31  661.0  518.0
32    NaN  363.0
33  712.0  952.0
34    NaN  767.0
35  864.0   21.0
36  870.0  324.0
37  253.0  990.0
38  755.0  500.0
39  401.0  838.0
40   71.0  742.0
41  586.0    NaN
42  195.0  631.0
43  610.0  904.0
44  656.0  721.0
45    NaN  975.0
46  984.0  539.0
47  280.0  770.0
48  888.0  513.0
49  359.0  765.0
```

[ ]:
```
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
```

```
0
```

```
2299.0
```

[ ]:
```
        0      1      2
1   186.0  729.0  101.0
2   955.0  462.0  259.0
3     NaN   44.0  844.0
4   747.0  974.0  414.0
5   605.0  143.0  507.0
```

```
6     NaN    NaN  180.0
7    46.0  370.0  422.0
8   133.0  914.0  464.0
9   365.0  752.0  831.0
10  565.0  958.0   74.0
11  144.0  857.0  676.0
12  373.0  232.0   20.0
13  895.0  156.0  432.0
14  412.0  371.0  754.0
15  323.0  163.0  607.0
16  595.0  821.0  452.0
17  157.0  420.0  180.0
18    NaN  668.0  270.0
19  848.0  400.0   79.0
20  598.0  823.0  293.0
21    0.0  468.0  789.0
22    NaN  270.0  637.0
23    NaN  686.0  784.0
24   68.0  676.0  126.0
25   30.0    NaN  981.0
26  892.0  597.0  429.0
27  975.0  424.0  436.0
28  855.0  916.0    NaN
29  690.0  309.0  447.0
30  932.0  842.0  169.0
31  679.0  661.0  518.0
32  543.0    NaN  363.0
33    NaN  712.0  952.0
34  676.0    NaN  767.0
35  763.0  864.0   21.0
36  704.0  870.0  324.0
37  655.0  253.0  990.0
38  830.0  755.0  500.0
39  616.0  401.0  838.0
40  650.0   71.0  742.0
41  420.0  586.0    NaN
42    NaN  195.0  631.0
43  677.0  610.0  904.0
44  185.0  656.0  721.0
45    NaN    NaN  975.0
46  761.0  984.0  539.0
47  328.0  280.0  770.0
48  554.0  888.0  513.0
49  875.0  359.0  765.0
```

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

[ ]:

	0	1	2
21	0.0	468.0	789.0
25	30.0	NaN	981.0
7	46.0	370.0	422.0
24	68.0	676.0	126.0
8	133.0	914.0	464.0
11	144.0	857.0	676.0
17	157.0	420.0	180.0
44	185.0	656.0	721.0
1	186.0	729.0	101.0
15	323.0	163.0	607.0
47	328.0	280.0	770.0
9	365.0	752.0	831.0
12	373.0	232.0	20.0
14	412.0	371.0	754.0
41	420.0	586.0	NaN
32	543.0	NaN	363.0
48	554.0	888.0	513.0
10	565.0	958.0	74.0
0	570.0	812.0	917.0
16	595.0	821.0	452.0
20	598.0	823.0	293.0
5	605.0	143.0	507.0
39	616.0	401.0	838.0
40	650.0	71.0	742.0
37	655.0	253.0	990.0
34	676.0	NaN	767.0
43	677.0	610.0	904.0
31	679.0	661.0	518.0
29	690.0	309.0	447.0
36	704.0	870.0	324.0
4	747.0	974.0	414.0
46	761.0	984.0	539.0
35	763.0	864.0	21.0
38	830.0	755.0	500.0
19	848.0	400.0	79.0
28	855.0	916.0	NaN
49	875.0	359.0	765.0
26	892.0	597.0	429.0
13	895.0	156.0	432.0
30	932.0	842.0	169.0
2	955.0	462.0	259.0
27	975.0	424.0	436.0
3	NaN	44.0	844.0
6	NaN	NaN	180.0
18	NaN	668.0	270.0
22	NaN	270.0	637.0

```
23    NaN  686.0  784.0
33    NaN  712.0  952.0
42    NaN  195.0  631.0
45    NaN    NaN  975.0
```

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

[ ]:
```
         0      1      2
0   570.0  812.0  917.0
1   186.0  729.0  101.0
2   955.0  462.0  259.0
3     NaN   44.0  844.0
4   747.0  974.0  414.0
5   605.0  143.0  507.0
7    46.0  370.0  422.0
8   133.0  914.0  464.0
9   365.0  752.0  831.0
10  565.0  958.0   74.0
11  144.0  857.0  676.0
12  373.0  232.0   20.0
13  895.0  156.0  432.0
14  412.0  371.0  754.0
15  323.0  163.0  607.0
16  595.0  821.0  452.0
17  157.0  420.0  180.0
19  848.0  400.0   79.0
20  598.0  823.0  293.0
21    0.0  468.0  789.0
24   68.0  676.0  126.0
25   30.0    NaN  981.0
26  892.0  597.0  429.0
27  975.0  424.0  436.0
28  855.0  916.0    NaN
29  690.0  309.0  447.0
30  932.0  842.0  169.0
31  679.0  661.0  518.0
32  543.0    NaN  363.0
34  676.0    NaN  767.0
35  763.0  864.0   21.0
36  704.0  870.0  324.0
37  655.0  253.0  990.0
38  830.0  755.0  500.0
39  616.0  401.0  838.0
40  650.0   71.0  742.0
41  420.0  586.0    NaN
43  677.0  610.0  904.0
44  185.0  656.0  721.0
```

```
46  761.0  984.0  539.0
47  328.0  280.0  770.0
48  554.0  888.0  513.0
49  875.0  359.0  765.0
```

[ ]:
```
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: 0.03292118315272979
Covariance between 2nd and 3rd columns: -19129.779623477294
```

[ ]:
```
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
```

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

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

[ ]:

	0	1	2	bins
0	570.0	812.0	917.0	(796.0, 990.0]
1	186.0	729.0	101.0	(19.03, 214.0]
2	955.0	462.0	259.0	(214.0, 408.0]
3	NaN	44.0	844.0	(796.0, 990.0]
4	747.0	974.0	414.0	(408.0, 602.0]
5	605.0	143.0	507.0	(408.0, 602.0]
6	NaN	NaN	180.0	(19.03, 214.0]
7	46.0	370.0	422.0	(408.0, 602.0]
8	133.0	914.0	464.0	(408.0, 602.0]
9	365.0	752.0	831.0	(796.0, 990.0]
10	565.0	958.0	74.0	(19.03, 214.0]
11	144.0	857.0	676.0	(602.0, 796.0]
12	373.0	232.0	20.0	(19.03, 214.0]

```
13  895.0  156.0  432.0  (408.0, 602.0]
14  412.0  371.0  754.0  (602.0, 796.0]
15  323.0  163.0  607.0  (602.0, 796.0]
16  595.0  821.0  452.0  (408.0, 602.0]
17  157.0  420.0  180.0  (19.03, 214.0]
18    NaN  668.0  270.0  (214.0, 408.0]
19  848.0  400.0   79.0  (19.03, 214.0]
20  598.0  823.0  293.0  (214.0, 408.0]
21    0.0  468.0  789.0  (602.0, 796.0]
22    NaN  270.0  637.0  (602.0, 796.0]
23    NaN  686.0  784.0  (602.0, 796.0]
24   68.0  676.0  126.0  (19.03, 214.0]
25   30.0    NaN  981.0  (796.0, 990.0]
26  892.0  597.0  429.0  (408.0, 602.0]
27  975.0  424.0  436.0  (408.0, 602.0]
28  855.0  916.0    NaN             NaN
29  690.0  309.0  447.0  (408.0, 602.0]
30  932.0  842.0  169.0  (19.03, 214.0]
31  679.0  661.0  518.0  (408.0, 602.0]
32  543.0    NaN  363.0  (214.0, 408.0]
33    NaN  712.0  952.0  (796.0, 990.0]
34  676.0    NaN  767.0  (602.0, 796.0]
35  763.0  864.0   21.0  (19.03, 214.0]
36  704.0  870.0  324.0  (214.0, 408.0]
37  655.0  253.0  990.0  (796.0, 990.0]
38  830.0  755.0  500.0  (408.0, 602.0]
39  616.0  401.0  838.0  (796.0, 990.0]
40  650.0   71.0  742.0  (602.0, 796.0]
41  420.0  586.0    NaN             NaN
42    NaN  195.0  631.0  (602.0, 796.0]
43  677.0  610.0  904.0  (796.0, 990.0]
44  185.0  656.0  721.0  (602.0, 796.0]
45    NaN    NaN  975.0  (796.0, 990.0]
46  761.0  984.0  539.0  (408.0, 602.0]
47  328.0  280.0  770.0  (602.0, 796.0]
48  554.0  888.0  513.0  (408.0, 602.0]
49  875.0  359.0  765.0  (602.0, 796.0]
```

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

```
[ ]: 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
```

```
[ ]:       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
```

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

```
[ ]:       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
```

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

```
[ ]:             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
```

```
[ ]: 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
```

```
[ ]: df
```

[ ]:

	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

[ ]:
```
#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)
```

[ ]:
```
df
```

[ ]:

	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

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

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

[ ]:
```
#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
```

[ ]:
```
new_df_grades
```

[ ]:

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

```
[ ]: 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
```

```
[ ]:              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.

```
[ ]: 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

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
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)

[ ]:
```
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

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
```