DAV : Assignment 2 (Pandas)

Name : DHIRENDRA KUMAR PATEL

ROLL : 16027

```
import pandas as pd
import numpy as np
src="/content/drive/MyDrive/Classroom/DAV sem 3/16027_DHIRENDRA_KUMAR_PATEL/assignment_2_pandas/csvdata"
```

Q1. Use a dataset of your choice from Open Data Portal (https:// data.gov.in/, UCI repository) . Load a Pandas dataframe with a selected dataset. Identify and count the missing values in a dataframe. Clean the data after removing noise as follows

a) Detect the outliers and remove the rows having outliers

```
df1=pd.read_csv(f"{src}/q1.csv")
df1
```

	City Name	Zone Number	Zone Name	Type of hospital_Private or Public	Total no. of beds	Number of COVID beds	Number of ICU beds	Number of ventilators or ABD
0	Chennai	Zone 1	Thiruvottiyur	Thiruvottiyur UCHC	65	0	0	0
1	Chennai	Zone 1	Thiruvottiyur	Thiruvottiyur GH	50	0	0	0
2	Chennai	Zone 1	Thiruvottiyur	Akash Hospital	120	50	50	4
3	Chennai	Zone 1	Thiruvottiyur	Sugam Hospital	102	51	2	2
4	Chennai	Zone 1	Thiruvottiyur	Suman Hospital	35	10	10	2
...	...	...	...	...	...	...	...	...
310	Chennai	Zone 15	Sholinganallur	Swaram Hospital, No: 13, Duraiswamy 2nd street...	30	0	1	1
311	Chennai	Zone 15	Sholinganallur	Sri Sugam Hospital, No: 189, OMR,Sholinganallu...	6	0	0	0
312	Chennai	Zone 15	Sholinganallur	Trust Life Hospital, Panaiyur, Chennai-119	5	0	0	0
313	Chennai	Zone 15	Sholinganallur	Ragas General Hospital, E.C.R Uthandi, Ch-119	100	0	0	0

b) Drop duplicate rows.

```
df1.drop_duplicates(inplace=True)
df1
```

	City Name	Zone Number	Zone Name	Type of hospital_Private or Public	Total no. of beds	Number of COVID beds	Number of ICU beds	Number ventilat or
0	Chennai	Zone 1	Thiruvottiyur	Thiruvottiyur UCHC	65	0	0	
1	Chennai	Zone 1	Thiruvottiyur	Thiruvottiyur GH	50	0	0	
2	Chennai	Zone 1	Thiruvottiyur	Akash Hospital	120	50	50	
3	Chennai	Zone 1	Thiruvottiyur	Sugam Hospital	102	51	2	
4	Chennai	Zone 1	Thiruvottiyur	Suman Hospital	35	10	10	
...	...	...	...	...	...	...	...	
310	Chennai	Zone 15	Sholinganallur	Swaram Hospital, No: 13, Duraiswamy 2nd street...	30	0	1	
311	Chennai	Zone 15	Sholinganallur	Sri Sugam Hospital, No: 189, OMR,Sholinganallu...	6	0	0	

c) Identify the most positively correlated attributes and negatively correlated attributes

```
cmatrix=df1.corr(numeric_only=True)
print("Correlation Matrix :\n")
cmatrix[np.eye(len(cmatrix),dtype=bool)]=np.nan  #replace correlation = 1 with NaN
cmatrix
```

```
Correlation Matrix :
```

	Total no. of beds	Number of COVID beds	Number of ICU beds	Number of ventilators or ABD
Total no. of beds	NaN	0.779262	0.699359	0.648733
Number of COVID beds	0.779262	NaN	0.828389	0.557429
Number of ICU beds	0.699359	0.828389	NaN	0.674267

```
print("Maximum Correlated : \n",cmatrix.stack().idxmax()," : ", cmatrix.max().max())
print("\nMinimum Correlated : \n",cmatrix.stack().idxmin()," : ", cmatrix.min().min())
```

```
Maximum Correlated :
 ('Number of COVID beds', 'Number of ICU beds')  :  0.8283885567139659

Minimum Correlated :
 ('Number of COVID beds', 'Number of ventilators or ABD')  :  0.5574292113754715
```

Q2. 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}]

What are multiple ways to do it? Give at least 3 methods to achieve it? Explain each method as the comment of your code.

```
d1={'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65, 69, 62, 61]}
df1=pd.DataFrame(d1)
def f(arow):      #apply function to DataFrame that takes each row and returns dictionary containing Boy-Girl pairs of that row, then st
  return {"Boys":arow["Boys"],"Girls":arow["Girls"]}
print(df1,"\n")
l1=df1.apply(f,axis=1).tolist()
print(l1)
```

```
   Boys  Girls
0    72     63
1    68     65
2    70     69
3    69     62
4    74     61

[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls': 65}, {'Boys': 70, 'Girls': 69}, {'Boys': 69, 'Girls': 62}, {'Boys': 74, 'Girls': 6
```

```
l2=[]
for i in range(len(d1["Boys"])):   #iterate both lists in the dictionary, then make dictionary of the Boy-Girl index-wise pairs and appe
  l2.append({"Boys":d1["Boys"][i],"Girls":d1["Girls"][i]})
print(l2)
```

```
[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls': 65}, {'Boys': 70, 'Girls': 69}, {'Boys': 69, 'Girls': 62}, {'Boys': 74, 'Girls': 6
```

```
l3=[]
for i,j in zip(d1["Boys"],d1["Girls"]):   #use zip to map the dictionary lists and generate zipped Boy-Girl pairs based on index of list
  l3.append({"Boys":i,"Girls":j})
print(l3)
```

```
[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls': 65}, {'Boys': 70, 'Girls': 69}, {'Boys': 69, 'Girls': 62}, {'Boys': 74, 'Girls': 6
```

```
l4=df1.to_dict(orient="records")  #use to_dict function of DataFrame with orientation=records which gives list of dictionary like [{colu
print(l4)
```

```
[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls': 65}, {'Boys': 70, 'Girls': 69}, {'Boys': 69, 'Girls': 62}, {'Boys': 74, 'Girls': 6
```

Q3.Create a dataframe having at least 5 columns and 100 rows to store numeric data generated using a random function. Replace 25% of the values by null values whose index positions are generated using random function. Do the following:

```
df3=pd.DataFrame(np.random.randint(100,size=(200,10)))
df3
```

	0	1	2	3	4	5	6	7	8	9
0	90	85	51	46	57	63	6	15	63	35
1	24	85	36	74	37	13	31	52	16	36
2	99	34	48	38	99	78	30	96	85	59
3	18	51	85	92	86	65	28	70	57	26
4	57	69	43	49	67	38	63	13	70	85
...	...	...	...	...	...	...	...	...	...	...
195	52	51	61	8	51	88	9	39	62	84
196	47	45	27	0	84	18	77	30	7	52
197	66	74	92	28	1	74	56	30	12	66
198	55	58	15	78	92	69	39	6	41	35
199	56	97	13	21	85	16	72	75	26	88

200 rows × 10 columns

```
nancount=0
nanneeded=df3.size*0.25
while nancount<nanneeded:
  x=np.random.randint(0,df3.shape[0])
  y=np.random.randint(0,df3.shape[1])
  if not pd.isnull(df3.iloc[x,y]):
    df3.iloc[x,y]=np.nan
    nancount+=1
df3
```

	0	1	2	3	4	5	6	7	8	9
0	NaN	85.0	51.0	46.0	57.0	63.0	NaN	15.0	63.0	35.0
1	24.0	85.0	36.0	74.0	37.0	13.0	31.0	NaN	16.0	36.0
2	99.0	34.0	48.0	38.0	99.0	78.0	NaN	96.0	NaN	NaN
3	NaN	51.0	85.0	NaN	86.0	NaN	NaN	70.0	NaN	NaN
4	NaN	69.0	43.0	49.0	67.0	38.0	63.0	13.0	NaN	85.0
...	...	...	...	...	...	...	...	...	...	...
195	52.0	51.0	61.0	8.0	51.0	NaN	9.0	NaN	NaN	84.0
196	47.0	45.0	27.0	0.0	84.0	18.0	77.0	NaN	7.0	52.0
197	66.0	74.0	NaN	NaN	1.0	74.0	56.0	30.0	12.0	66.0
198	55.0	NaN	15.0	78.0	92.0	NaN	39.0	6.0	NaN	35.0
199	56.0	97.0	13.0	21.0	85.0	16.0	NaN	75.0	26.0	NaN

200 rows × 10 columns

a. Identify and count missing values in a dataframe.

```
count = df3.isnull().sum().sum()
print("Total Null Values : ", count)
```

```
Total Null Values :  500
```

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

```
not_more_than_5_nulls = df3.isnull().sum()<=5
df3.loc[:,not_more_than_5_nulls]
#this deletes all columns as the question has asked for minimum 100 rows, 5 columns with 25% of all values to be NaN. Meaning, all the c
```

0
1
2
3
4
...
195
196
197
198
199

200 rows × 0 columns

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

```
max_rowsum=df3.sum(axis=1).max()
print("Maximum Sum in a Row is : ",max_rowsum)
no_drop_condition=(df3.sum(axis=1) != max_rowsum)
print("Dataframe after dropping max sum row :")
df3.loc[no_drop_condition,:]
```

```
Maximum Sum in a Row is :  630.0
Dataframe after dropping max sum row :
```

	0	1	2	3	4	5	6	7	8	9
0	NaN	85.0	51.0	46.0	57.0	63.0	NaN	15.0	63.0	35.0
1	24.0	85.0	36.0	74.0	37.0	13.0	31.0	NaN	16.0	36.0
2	99.0	34.0	48.0	38.0	99.0	78.0	NaN	96.0	NaN	NaN
3	NaN	51.0	85.0	NaN	86.0	NaN	NaN	70.0	NaN	NaN
4	NaN	69.0	43.0	49.0	67.0	38.0	63.0	13.0	NaN	85.0
...	...	...	...	...	...	...	...	...	...	...
195	52.0	51.0	61.0	8.0	51.0	NaN	9.0	NaN	NaN	84.0
196	47.0	45.0	27.0	0.0	84.0	18.0	77.0	NaN	7.0	52.0
197	66.0	74.0	NaN	NaN	1.0	74.0	56.0	30.0	12.0	66.0
198	55.0	NaN	15.0	78.0	92.0	NaN	39.0	6.0	NaN	35.0
199	56.0	97.0	13.0	21.0	85.0	16.0	NaN	75.0	26.0	NaN

199 rows × 10 columns

d. Sort the data frame on the basis of the first column.

```
df3.sort_values(by=[0],axis=0)
```

	0	1	2	3	4	5	6	7	8	9
174	0.0	38.0	15.0	72.0	34.0	46.0	30.0	23.0	95.0	6.0
78	0.0	NaN	59.0	NaN	52.0	92.0	NaN	99.0	NaN	60.0
105	2.0	NaN	82.0	NaN	NaN	4.0	26.0	45.0	25.0	93.0
126	3.0	77.0	62.0	95.0	53.0	56.0	4.0	26.0	58.0	81.0
15	3.0	NaN	NaN	NaN	42.0	17.0	95.0	61.0	33.0	NaN

e. Remove all duplicates from the first column.

```
df3.drop_duplicates(subset=[0])
```

	0	1	2	3	4	5	6	7	8	9
0	NaN	85.0	51.0	46.0	57.0	63.0	NaN	15.0	63.0	35.0
1	24.0	85.0	36.0	74.0	37.0	13.0	31.0	NaN	16.0	36.0
2	99.0	34.0	48.0	38.0	99.0	78.0	NaN	96.0	NaN	NaN
5	32.0	72.0	99.0	87.0	41.0	89.0	95.0	21.0	86.0	NaN
6	4.0	8.0	42.0	34.0	33.0	88.0	NaN	25.0	40.0	80.0
...	...	...	...	...	...	...	...	...	...	...
182	78.0	69.0	57.0	NaN	7.0	64.0	NaN	25.0	70.0	0.0
183	9.0	97.0	97.0	76.0	16.0	99.0	52.0	12.0	80.0	92.0
188	52.0	NaN	72.0	NaN	56.0	88.0	54.0	NaN	39.0	NaN
193	81.0	86.0	55.0	NaN	27.0	NaN	NaN	2.0	NaN	10.0
197	66.0	74.0	NaN	NaN	1.0	74.0	56.0	30.0	12.0	66.0

80 rows × 10 columns

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.

```
print("Correlation between first and second column : ",df3[0].corr(df3[1]))
print("Covariance between second and third column : ",df3[1].cov(df3[2]))
```

```
Correlation between first and second column :  -0.05360938407044749
Covariance between second and third column :  156.76629183037528
```

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

```
z_score_threshold = 5
z_scores = (df3 - df3.mean()) / df3.std()
print("DataFrame after removing rows with outliers :\n")
df3[(z_scores.abs() < z_score_threshold).all(axis=1)]
```

```
DataFrame after removing rows with outliers :
```

h. Discretize second column and create 5 bins

```
41  30.0  96.0  31.0  77.0  71.0  52.0  77.0  58.0  57.0  76.0
```

```
pd.cut(df3[1], bins=5)
```

```
0      (78.4, 98.0]
1      (78.4, 98.0]
2      (19.6, 39.2]
3      (39.2, 58.8]
4      (58.8, 78.4]
           ...
195    (39.2, 58.8]
196    (39.2, 58.8]
197    (58.8, 78.4]
198             NaN
199    (78.4, 98.0]
Name: 1, Length: 200, dtype: category
Categories (5, interval[float64, right]): [(-0.098, 19.6] < (19.6, 39.2] < (39.2, 58.8] <
                                           (58.8, 78.4] < (78.4, 98.0]]
```

```
174   0.0  28.0  15.0  73.0  24.0  46.0  20.0  23.0  95.0   6.0
```

Q4.Consider two excel files having attendance of a workshop’s participants for two days. Each file has three fields ‘Name’, ‘Time of joining’, duration (in minutes) where names are unique within a file. Note that duration may take one of three values (30, 40, 50) only. Import the data into two dataframes and do the following:

```
ws1=pd.read_csv(f"{src}/workshop1.csv")
ws2=pd.read_csv(f"{src}/workshop2.csv")
print("Workshop 1 :\n",ws1)
print("\nWorkshop 2 :\n",ws2)
```

```
Workshop 1 :
           Name Time of joining  Duration
0        Rahul            8:30        30
1        Rohit            5:29        40
2      Chandan            2:30        50
3        Deepu            5:27        40
4       Anvesh            4:50        30
5       Arfiya            8:20        40
6       Bijaya           19:40        30
7   Bratadipta            4:28        30
8         Ansh            3:50        40
9         Neha           12:30        50
10         Dev            5:29        40
11        Shiv           12:39        40
12    Shivansh           18:40        50
13        Kali           12:40        50
14  Pari verma           15:30        40
15     Radhika            3:40        50
16      Kartik            3:56        30
17        Anvi            4:02        40
18        Devi            2:50        50

Workshop 2 :
           Name Time of joining  Duration
0       Akshat            4:20        40
1         Durg            7:20        30
2        Deepu            5:27        40
3       Pankaj           18:40        40
4       Anvesh            4:50        30
5       Arfiya            8:20        40
6        Rahul           12:34        50
7       Bijaya           19:40        30
8        Rohit            3:50        50
9   Bratadipta            4:28        30
10        Ansh            3:50        40
11        Neha           12:30        50
12     Devansh            5:29        40
13       Shiva           12:39        40
14    Shivansh           18:40        50
15      Kapali           12:40        50
16        Pari           15:30        40
17     Radhika            3:40        50
18      Kartik            3:56        30
19      Anvika            4:02        40
20      Devika            2:50        30
21        Devi            2:50        50
```

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

```
ws3=pd.merge(ws1,ws2,on="Name",how="inner")
print(ws3)
```

```
          Name Time of joining_x  Duration_x Time of joining_y  Duration_y
0        Rahul              8:30          30             12:34          50
1        Rohit              5:29          40              3:50          50
2        Deepu              5:27          40              5:27          40
3       Anvesh              4:50          30              4:50          30
4       Arfiya              8:20          40              8:20          40
5       Bijaya             19:40          30             19:40          30
6   Bratadipta              4:28          30              4:28          30
7         Ansh              3:50          40              3:50          40
8         Neha             12:30          50             12:30          50
9     Shivansh             18:40          50             18:40          50
10     Radhika              3:40          50              3:40          50
11      Kartik              3:56          30              3:56          30
12        Devi              2:50          50              2:50          50
```

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

```
ws4=pd.merge(ws1,ws2,on="Name",how="outer")
print(ws4)
```

```
          Name Time of joining_x  Duration_x Time of joining_y  Duration_y
0        Rahul              8:30        30.0             12:34        50.0
1        Rohit              5:29        40.0              3:50        50.0
2      Chandan              2:30        50.0               NaN         NaN
3        Deepu              5:27        40.0              5:27        40.0
4       Anvesh              4:50        30.0              4:50        30.0
5       Arfiya              8:20        40.0              8:20        40.0
6       Bijaya             19:40        30.0             19:40        30.0
7   Bratadipta              4:28        30.0              4:28        30.0
8         Ansh              3:50        40.0              3:50        40.0
9         Neha             12:30        50.0             12:30        50.0
10         Dev              5:29        40.0               NaN         NaN
11        Shiv             12:39        40.0               NaN         NaN
12    Shivansh             18:40        50.0             18:40        50.0
13        Kali             12:40        50.0               NaN         NaN
14  Pari verma             15:30        40.0               NaN         NaN
15     Radhika              3:40        50.0              3:40        50.0
16      Kartik              3:56        30.0              3:56        30.0
17        Anvi              4:02        40.0               NaN         NaN
18        Devi              2:50        50.0              2:50        50.0
19      Akshat               NaN         NaN              4:20        40.0
20        Durg               NaN         NaN              7:20        30.0
21     Pankaj                NaN         NaN             18:40        40.0
22     Devansh               NaN         NaN              5:29        40.0
23       Shiva               NaN         NaN             12:39        40.0
24      Kapali               NaN         NaN             12:40        50.0
25        Pari               NaN         NaN             15:30        40.0
26      Anvika               NaN         NaN              4:02        40.0
27      Devika               NaN         NaN              2:50        30.0
```

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

```
ws5=pd.merge(ws1,ws2,on="Name",how="outer")
ws5.count(axis=0)
```

```
Name                 28
Time of joining_x    19
Duration_x           19
Time of joining_y    22
Duration_y           22
dtype: int64
```

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

```
ws6 = pd.merge(ws1, ws2, on=['Name', 'Duration'], how='outer')
ws6.set_index(['Name', 'Duration'], inplace=True)
statistics = ws6.describe()
print("Merged DataFrame with Multi-Index :\n",ws6)
print("\nDescriptive Statistics for Multi-Index :\n",statistics)
```

```
Merged DataFrame with Multi-Index :
                     Time of joining_x Time of joining_y
Name       Duration
Rahul      30                     8:30               NaN
Rohit      40                     5:29               NaN
```

```
Chandan    50             2:30          NaN
Deepu      40             5:27         5:27
Anvesh     30             4:50         4:50
Arfiya     40             8:20         8:20
Bijaya     30            19:40        19:40
Bratadipta 30             4:28         4:28
Ansh       40             3:50         3:50
Neha       50            12:30        12:30
Dev        40             5:29          NaN
Shiv       40            12:39          NaN
Shivansh   50            18:40        18:40
Kali       50            12:40          NaN
Pari verma 40            15:30          NaN
Radhika    50             3:40         3:40
Kartik     30             3:56         3:56
Anvi       40             4:02          NaN
Devi       50             2:50         2:50
Akshat     40              NaN         4:20
Durg       30              NaN         7:20
Pankaj     40              NaN        18:40
Rahul      50              NaN        12:34
Rohit      50              NaN         3:50
Devansh    40              NaN         5:29
Shiva      40              NaN        12:39
Kapali     50              NaN        12:40
Pari       40              NaN        15:30
Anvika     40              NaN         4:02
Devika     30              NaN         2:50

Descriptive Statistics for Multi-Index :
        Time of joining_x Time of joining_y
count                  19                22
unique                 18                19
top                  5:29              3:50
freq                    2                 2
```

Q5.Consider a data frame containing data about students i.e. name, gender and passing division:

```
stdf1=pd.read_csv(f"{src}/students.csv")
stdf1
```

	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.

```
pd.get_dummies(stdf1, columns=["Gender","Pass_Division"])
```

	Name	Birth_Month	Gender_F	Gender_M	Pass_Division_I	Pass_Division_II	Pass
0	Mudit Chauhan	December	0	1	0	0	
1	Seema Chopra	January	1	0	0	1	
2	Rani Gupta	March	1	0	1	0	
3	Aditya Narayan	October	0	1	1	0	
4	Sanjeev Sahni	February	0	1	0	1	
5	Prakash Kumar	December	0	1	0	0	
6	Ritu Agarwal	September	1	0	1	0	
7	Akshay Goel	August	0	1	1	0	

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

```
stdf1['Birth_Month'] = pd.Categorical(stdf1['Birth_Month'], categories=['January', 'February', 'March', 'April', 'May', 'June', 'July',
stdf1.sort_values(by='Birth_Month')
```

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

Q6.Consider the following data frame containing a family name, gender of the family member and her/his monthly income in each record. Write a program in Python using Pandas to perform the following:

```
fam1=pd.read_csv(f"{src}/family.csv")
fam1
```

	FamilyName	Gender	MonthlyIncome(Rs.)
0	Shah	Male	44000.0
1	Vats	Male	65000.0
2	Vats	Female	43150.0
4	Vats	Female	255000.0

A. Calculate and display familywise gross monthly income.

```
print("Familywise gross monthly income : ")
fam1.groupby('FamilyName')['MonthlyIncome(Rs.)'].sum()
```

```
Familywise gross monthly income :
FamilyName
Kumar    250530.0
Shah     211400.0
Vats     435050.0
Name: MonthlyIncome(Rs.), dtype: float64
```

B. Display the highest and lowest monthly income for each family name.

```
print("Highest income for each family :")
fam1.groupby('FamilyName')['MonthlyIncome(Rs.)'].max()
```

```
Highest income for each family :
FamilyName
Kumar    103000.0
Shah     112400.0
Vats     255000.0
Name: MonthlyIncome(Rs.), dtype: float64
```

```
print("Lowest income for each family :")
fam1.groupby('FamilyName')['MonthlyIncome(Rs.)'].min()
```

```
Lowest income for each family :
FamilyName
Kumar    66500.0
Shah     44000.0
Vats     43150.0
Name: MonthlyIncome(Rs.), dtype: float64
```

C. Calculate and display monthly income of all members earning income less than Rs. 80000.00.

```
print("Monthly income of all members earning less than Rs. 80000.00 :")
fam1[fam1['MonthlyIncome(Rs.)']<80000]
```

```
Monthly income of all members earning less than Rs. 80000.00 :
```

	FamilyName	Gender	MonthlyIncome(Rs.)
0	Shah	Male	44000.0
1	Vats	Male	65000.0
2	Vats	Female	43150.0
3	Kumar	Female	66500.0
6	Shah	Male	55000.0
9	Vats	Male	71900.0

D. Calculate and display the average monthly income of the female members in the Shah family.

```
print("Average monthly income of Shah family female members :", ( fam1[(fam1["FamilyName"]=="Shah") & (fam1["Gender"]=="Female")] ) ["Mon
```

```
Average monthly income of Shah family female members : 112400.0
```

E. Calculate and display monthly income of all members with income greater than Rs. 60000.00.

```
print("Monthly income of all members with income greater than Rs. 60000.00 :")
fam1[fam1['MonthlyIncome(Rs.)']>60000]
```

```
Monthly income of all members with income greater than Rs. 60000.00 :
```

	FamilyName	Gender	MonthlyIncome(Rs.)
1	Vats	Male	65000.0
3	Kumar	Female	66500.0
4	Vats	Female	255000.0
5	Kumar	Male	103000.0
7	Shah	Female	112400.0
8	Kumar	Female	81030.0

F. Display total number of females along with their average monthly income.

```
female_count = fam1[fam1['Gender'] == 'Female'].groupby('FamilyName')['Gender'].count()
monthly_female_income =fam1[fam1['Gender']=="Female"].groupby("FamilyName").mean(numeric_only=True)
pd.merge(female_count, monthly_female_income, on="FamilyName")
```

	Gender	MonthlyIncome(Rs.)
FamilyName		
Kumar	2	73765.0
Shah	1	112400.0
Vats	2	149075.0

G. Delete rows with Monthly income less than the average income of all members

```
avg_income = fam1["MonthlyIncome(Rs.)"].mean()
print("Average income of all members :", avg_income)
print("After deleting rows with Monthly income less than the average income :")
fam1[fam1.loc[:,"MonthlyIncome(Rs.)"]>avg_income]
```

```
Average income of all members : 89698.0
After deleting rows with Monthly income less than the average income :
```

	FamilyName	Gender	MonthlyIncome(Rs.)
4	Vats	Female	255000.0
5	Kumar	Male	103000.0
7	Shah	Female	112400.0

Q7.Using the parsed.csv file, complete the following exercises to practise your pandas skills:

```
par1=pd.read_csv(f"{src}/parsed.csv")
par1
```

	alert	cdi	code	detail	dmin	fe
0	NaN	NaN	37389218	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.008693	Na
1	NaN	NaN	37389202	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.020030	Na
2	NaN	4.4	37389194	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.021370	28
3	NaN	NaN	37389186	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.026180	Na
4	NaN	NaN	73096941	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.077990	Na
...	...	...	...	...	...	
9327	NaN	NaN	73086771	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.018060	Na

a. Find the 95th percentile of earthquake magnitude in Japan using the magType of 'mb'.

9328 NaN NaN 38063967 https://earthquake.usgs.gov/fdsnws/event/1/que... 0.030410 Na

```
japan_eq=par1[(par1['parsed_place'] == 'Japan') & (par1['magType'] == 'mb')]
print("95th percentile of earthquake magnitude in Japan using the magType of 'mb' : ")
japan_eq['mag'].quantile(0.95)
```

```
95th percentile of earthquake magnitude in Japan using the magType of 'mb' :
4.9
```

b. Find the percentage of earthquakes in Indonesia that were coupled with tsunamis.

9330 NaN NaN 38063959 https://earthquake.usgs.gov/fdsnws/event/1/que... 0.018650 Na

```
indonesia_eq = par1[par1['parsed_place'] == 'Indonesia']
indonesia_eq_count = len(indonesia_eq)
tsunami_eq = len(indonesia_eq[indonesia_eq['tsunami'] == 1])
print("Percentage of earthquakes in Indonesia that were coupled with tsunamis :")
(tsunami_eq/indonesia_eq_count)*100
```

```
Percentage of earthquakes in Indonesia that were coupled with tsunamis :
23.12925170068027
```

c. Get summary statistics for earthquakes in Nevada.

```
nevada_eq = par1[par1['parsed_place'] == 'Nevada']
nevada_eq.describe()
```

	cdi	dmin	felt	gap	mag	mmi	nst	
count	15.000000	681.000000	15.000000	681.000000	681.000000	1.00	681.000000	681.0(
mean	2.440000	0.166199	2.400000	153.668120	0.500073	2.84	12.618209	0.1!
std	0.501142	0.166228	4.626013	68.735302	0.696710	NaN	9.866963	0.0!
min	2.000000	0.001000	1.000000	29.140000	-0.500000	2.84	3.000000	0.0(
25%	2.000000	0.053000	1.000000	97.380000	-0.100000	2.84	6.000000	0.1(
50%	2.200000	0.112000	1.000000	149.140000	0.400000	2.84	10.000000	0.1
75%	2.900000	0.233000	1.000000	199.720000	0.900000	2.84	16.000000	0.1!
max	3.300000	1.414000	19.000000	355.910000	2.900000	2.84	61.000000	0.8(

d. Add a column to the dataframe indicating whether or not the earthquake happened in a country or US state that is on the Ring of Fire. Use Bolivia, Chile, Ecuador, Peru, Costa Rica, Guatemala, Mexico (be careful not to select New Mexico), Japan, Philippines, Indonesia, New Zealand, Antarctica (look for Antarctic), Canada, Fiji, Alaska, Washington, California, Russia, Taiwan, Tonga, and Kermadec Islands.

```
rings_of_fire = ['Bolivia', 'Chile', 'Ecuador', 'Peru', 'Costa Rica', 'Guatemala', 'Mexico', 'Japan', 'Philippines', 'Indonesia', 'New Z
par1['Exists in Ring of fire?'] = par1['parsed_place'].isin(rings_of_fire)
par1
```

code	detail	dmin	felt	gap	
37389218	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.008693	NaN	85.0	,ci373892
37389202	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.020030	NaN	79.0	,ci373892
37389194	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.021370	28.0	21.0	,ci373891
37389186	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.026180	NaN	39.0	,ci373891
73096941	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.077990	NaN	192.0	,nc730969
...	...	...	...	...	
73086771	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.018060	NaN	185.0	,nc730867
38063967	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.030410	NaN	50.0	,ci380639
2018261000	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.452600	NaN	276.0	,pr20182610
38063959	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.018650	NaN	61.0	,ci380639
38063935	https://earthquake.usgs.gov/fdsnws/event/1/que...	0.016980	NaN	39.0	,ci380639

s

e. Calculate the number of earthquakes in the Ring of Fire locations and the number outside them.

```
print("Number of earthquakes in the Ring of Fire locations :",len(par1[par1['Exists in Ring of fire?'] == True]))
print("Number of earthquakes outside the Ring of Fire locations :",len(par1[par1['Exists in Ring of fire?'] == False]))
```

```
Number of earthquakes in the Ring of Fire locations : 7184
Number of earthquakes outside the Ring of Fire locations : 2148
```

f. Find the tsunami count along the Ring of Fire.

```
print("Tsunami count along the Ring of Fire locations:", len(par1[(par1['Exists in Ring of fire?']==True) & (par1['tsunami']==1)]))
```

```
Tsunami count along the Ring of Fire locations: 45
```

Q8.Using the CSV files in the earthquakes.csv folder, Write a program in Python using Pandas to perform the following:

```
eq1=pd.read_csv(f"{src}/earthquakes.csv")
eq1
```

	mag	magType	time	place	tsunami	parsed_place
0	1.35	ml	1539475168010	9km NE of Aguanga, CA	0	California
1	1.29	ml	1539475129610	9km NE of Aguanga, CA	0	California
2	3.42	ml	1539475062610	8km NE of Aguanga, CA	0	California
3	0.44	ml	1539474978070	9km NE of Aguanga, CA	0	California
4	2.16	md	1539474716050	10km NW of Avenal, CA	0	California
...	...	...	...	...	...	...
9327	0.62	md	1537230228060	9km ENE of Mammoth Lakes, CA	0	California

a. With the earthquakes.csv file, select all the earthquakes in Japan with a magType of mb and a magnitude of 4.9 or greater.

```
eq1[(eq1['parsed_place']=='Japan') & (eq1['magType']=='mb') & (eq1['mag']>=4.9)]
```

	mag	magType	time	place	tsunami	parsed_place
1563	4.9	mb	1538977532250	293km ESE of Iwo Jima, Japan	0	Japan
2576	5.4	mb	1538697528010	37km E of Tomakomai, Japan	0	Japan
3072	4.9	mb	1538579732490	15km ENE of Hasaki, Japan	0	Japan

b. Create bins for each full number of magnitude (for example, the first bin is 0-1, the second is 1-2, and so on) with a magType of ml and count how many are in each bin.

```
ml_eq = eq1[eq1['magType'] == 'ml']
bins = list(range(int(ml_eq['mag'].min()), int(ml_eq['mag'].max()) + 1))
binned_eq = pd.cut(ml_eq['mag'], bins, right=False)
binned_counts = binned_eq.value_counts().sort_index()
print(binned_counts)
```

```
[-1, 0)     446
[0, 1)     2072
[1, 2)     3126
[2, 3)      985
[3, 4)      153
[4, 5)        6
Name: mag, dtype: int64
```

c. Build a crosstab with the earthquake data between the tsunami column and the magType column. Rather than showing the frequency count, show the maximum magnitude that was observed for each combination. Put the magType along the columns.

```
pd.crosstab(eq1['tsunami'], eq1['magType'], values=eq1['mag'], aggfunc='max')
```

magType	mb	mb_lg	md	mh	ml	ms_20	mw	mwb	mwr	mww
tsunami										
0	5.6	3.5	4.11	1.1	4.2	NaN	3.83	5.8	4.8	6.0
1	6.1	NaN	NaN	NaN	5.1	5.7	4.41	NaN	NaN	7.5

Q9.Using the faang.csv file, group by the ticker and resample to monthly frequency. Make the following aggregations:

```
fng1=pd.read_csv(f"{src}/faang.csv", parse_dates=['date'],index_col="date")
fng1_resampled=fng1.groupby('ticker').resample('M')
```

a. Mean of the opening price

```
fng1_resampled['open'].mean()
```

```
ticker  date      
AAPL    2018-01-31     170.714690
        2018-02-28     164.562753
        2018-03-31     172.421381
        2018-04-30     167.332895
        2018-05-31     182.635582
        2018-06-30     186.605843
        2018-07-31     188.065786
        2018-08-31     210.460287
        2018-09-30     220.611742
        2018-10-31     219.489426
        2018-11-30     190.828681
        2018-12-31     164.537405
AMZN    2018-01-31    1301.377143
        2018-02-28    1447.112632
        2018-03-31    1542.160476
        2018-04-30    1475.841905
        2018-05-31    1590.474545
        2018-06-30    1699.088571
        2018-07-31    1786.305714
        2018-08-31    1891.957826
        2018-09-30    1969.239474
        2018-10-31    1799.630870
        2018-11-30    1622.323810
        2018-12-31    1572.922105
FB      2018-01-31     184.364762
        2018-02-28     180.721579
        2018-03-31     173.449524
        2018-04-30     164.163557
        2018-05-31     181.910509
        2018-06-30     194.974067
        2018-07-31     199.332143
        2018-08-31     177.598443
        2018-09-30     164.232895
        2018-10-31     154.873261
        2018-11-30     141.762857
        2018-12-31     137.529474
GOOG    2018-01-31    1127.200952
        2018-02-28    1088.629474
        2018-03-31    1096.108095
        2018-04-30    1038.415238
        2018-05-31    1064.021364
        2018-06-30    1136.396190
        2018-07-31    1183.464286
        2018-08-31    1226.156957
        2018-09-30    1176.878421
        2018-10-31    1116.082174
        2018-11-30    1054.971429
        2018-12-31    1042.620000
NFLX    2018-01-31     231.269286
        2018-02-28     270.873158
        2018-03-31     312.712857
        2018-04-30     309.129529
        2018-05-31     329.779759
        2018-06-30     384.557595
        2018-07-31     380.969090
        2018-08-31     345.409591
        2018-09-30     363.326842
```

b. Maximum of the high price

```
fng1_resampled['high'].max()
```

```
ticker  date      
AAPL    2018-01-31     176.6782
        2018-02-28     177.9059
        2018-03-31     180.7477
        2018-04-30     176.2526
        2018-05-31     187.9311
        2018-06-30     192.0247
        2018-07-31     193.7650
        2018-08-31     227.1001
        2018-09-30     227.8939
        2018-10-31     231.6645
        2018-11-30     220.6405
        2018-12-31     184.1501
AMZN    2018-01-31    1472.5800
        2018-02-28    1528.7000
        2018-03-31    1617.5400
        2018-04-30    1638.1000
        2018-05-31    1635.0000
```

```
        2018-06-30    1763.1000
        2018-07-31    1880.0500
        2018-08-31    2025.5700
        2018-09-30    2050.5000
        2018-10-31    2033.1900
        2018-11-30    1784.0000
        2018-12-31    1778.3400
FB      2018-01-31     190.6600
        2018-02-28     195.3200
        2018-03-31     186.1000
        2018-04-30     177.1000
        2018-05-31     192.7200
        2018-06-30     203.5500
        2018-07-31     218.6200
        2018-08-31     188.3000
        2018-09-30     173.8900
        2018-10-31     165.8800
        2018-11-30     154.1300
        2018-12-31     147.1900
GOOG    2018-01-31    1186.8900
        2018-02-28    1174.0000
        2018-03-31    1177.0500
        2018-04-30    1094.1600
        2018-05-31    1110.7500
        2018-06-30    1186.2900
        2018-07-31    1273.8900
        2018-08-31    1256.5000
        2018-09-30    1212.9900
        2018-10-31    1209.9600
        2018-11-30    1095.5700
        2018-12-31    1124.6500
NFLX    2018-01-31     286.8100
        2018-02-28     297.3600
        2018-03-31     333.9800
        2018-04-30     338.8200
        2018-05-31     356.1000
        2018-06-30     423.2056
        2018-07-31     419.7700
        2018-08-31     376.8085
        2018-09-30     383.2000
```

c. Minimum of the low price

```
fng1_resampled['low'].min()
```

```
        2018-04-30     158.2207
        2018-05-31     162.7911
        2018-06-30     178.7056
        2018-07-31     181.3655
        2018-08-31     195.0999
        2018-09-30     213.6351
        2018-10-31     204.4963
        2018-11-30     169.5328
        2018-12-31     145.9639
AMZN    2018-01-31    1170.5100
        2018-02-28    1265.9300
        2018-03-31    1365.2000
        2018-04-30    1352.8800
        2018-05-31    1546.0200
        2018-06-30    1635.0900
        2018-07-31    1678.0600
        2018-08-31    1776.0200
        2018-09-30    1865.0000
        2018-10-31    1476.3600
        2018-11-30    1420.0000
        2018-12-31    1307.0000
FB      2018-01-31     175.8000
        2018-02-28     167.1800
        2018-03-31     149.0200
        2018-04-30     150.5100
        2018-05-31     170.2300
        2018-06-30     186.4300
        2018-07-31     166.5600
        2018-08-31     170.2700
        2018-09-30     158.8656
        2018-10-31     139.0300
        2018-11-30     126.8500
        2018-12-31     123.0200
GOOG    2018-01-31    1045.2300
        2018-02-28     992.5600
        2018-03-31     980.6400
        2018-04-30     990.3700
        2018-05-31    1006.2900
        2018-06-30    1096.0100
```

```
        2018-11-30     996.0200
        2018-12-31     970.1100
NFLX    2018-01-31     195.4200
        2018-02-28     236.1100
        2018-03-31     275.9000
        2018-04-30     271.2239
        2018-05-31     305.7300
        2018-06-30     352.8200
        2018-07-31     328.0000
        2018-08-31     310.9280
        2018-09-30     335.8300
        2018-10-31     271.2093
        2018-11-30     250.0000
        2018-12-31     231.2300
Name: low, dtype: float64
```

d. Mean of the closing price

```
fng1_resampled['close'].mean()
```

```
        2018-04-30     167.286924
        2018-05-31     183.207418
        2018-06-30     186.508652
        2018-07-31     188.179724
        2018-08-31     211.477743
        2018-09-30     220.356353
        2018-10-31     219.137822
        2018-11-30     190.246652
        2018-12-31     163.564732
AMZN    2018-01-31    1309.010952
        2018-02-28    1442.363158
        2018-03-31    1540.367619
        2018-04-30    1468.220476
        2018-05-31    1594.903636
        2018-06-30    1698.823810
        2018-07-31    1784.649048
        2018-08-31    1897.851304
        2018-09-30    1966.077895
        2018-10-31    1782.058261
        2018-11-30    1625.483810
        2018-12-31    1559.443158
FB      2018-01-31     184.962857
        2018-02-28     180.269474
        2018-03-31     173.489524
        2018-04-30     163.810476
        2018-05-31     182.930000
        2018-06-30     195.267619
        2018-07-31     199.967143
        2018-08-31     177.491957
        2018-09-30     164.377368
        2018-10-31     154.187826
        2018-11-30     141.635714
        2018-12-31     137.161053
GOOG    2018-01-31    1130.770476
        2018-02-28    1088.206842
        2018-03-31    1091.490476
        2018-04-30    1035.696190
        2018-05-31    1069.275909
        2018-06-30    1137.626667
        2018-07-31    1187.590476
        2018-08-31    1225.671739
        2018-09-30    1175.808947
        2018-10-31    1110.940435
        2018-11-30    1056.162381
        2018-12-31    1037.420526
NFLX    2018-01-31     232.908095
        2018-02-28     271.443684
        2018-03-31     312.228095
        2018-04-30     307.466190
        2018-05-31     331.536818
        2018-06-30     384.133333
        2018-07-31     381.515238
        2018-08-31     346.257826
        2018-09-30     362.641579
        2018-10-31     335.445652
        2018-11-30     290.344762
        2018-12-31     265.302368
Name: close, dtype: float64
```

e. Sum of the volume traded

```
fng1_resampled['volume'].sum()
```

```
ticker  date
AAPL    2018-01-31    659679440
```

```
            2018-02-28    927894473
            2018-03-31    713727447
            2018-04-30    666360147
            2018-05-31    620976206
            2018-06-30    527624365
            2018-07-31    393843881
            2018-08-31    700318837
            2018-09-30    678972040
            2018-10-31    789748068
            2018-11-30    961321947
            2018-12-31    898917007
AMZN        2018-01-31     96371290
            2018-02-28    137784020
            2018-03-31    130400151
            2018-04-30    129945743
            2018-05-31     71615299
            2018-06-30     85941510
            2018-07-31     97629820
            2018-08-31     96575676
            2018-09-30     94445693
            2018-10-31    183228552
            2018-11-30    139290208
            2018-12-31    154812304
FB          2018-01-31    495655736
            2018-02-28    516621991
            2018-03-31    996232472
            2018-04-30    751130388
            2018-05-31    401144183
            2018-06-30    387265765
            2018-07-31    652763259
            2018-08-31    549016789
            2018-09-30    500468912
            2018-10-31    622446235
            2018-11-30    518150415
            2018-12-31    558786249
GOOG        2018-01-31     28738485
            2018-02-28     42384105
            2018-03-31     45430049
            2018-04-30     41773275
            2018-05-31     31849196
            2018-06-30     32103642
            2018-07-31     31953386
            2018-08-31     28820379
            2018-09-30     28863199
            2018-10-31     48496167
            2018-11-30     36735570
            2018-12-31     40256461
NFLX        2018-01-31    238377533
            2018-02-28    184585819
            2018-03-31    263449491
            2018-04-30    262064417
            2018-05-31    142051114
            2018-06-30    244032001
            2018-07-31    305487432
            2018-08-31    213144082
```