10.6 Exercise: Aggregates

- Name: **Congxin (David) Xu**
- Computing ID: **cx2rx**

Part 1:

Write a summary about each aggregate function mentioned (np.mean, np.std, np.var, np.argmin, np.argmax, np.median, np.percentile).

- np.mean() is used to calculate the average value of a list of numbers or a nested list of numbers.
- np.std() is used to calculate the standard deviation of a list of numbers or a nested list of numbers.
- np.var() is used to calculate the variance of a list of numbers or a nested list of numbers.
- np.argmin() is used to get the index value of the minimum value of a list of numbers or a nested list of numbers.
- np.argmax() is used to get the index value of the maximum value of a list of numbers or a nested list of numbers.
- np.median() is used to calculate the median value of a list of numbers or a nested list of numbers.

Part 2:

Come up with a small sample of code (with appropriate setup/data) that illustrates the usage of at least four (4) of the aggregate functions (from Part 1). Include the results/output as well.

In [1]:

```python
import numpy as np

# Example 1: np.mean()
A = [[1,2,3], [4,5,12]]
# (1 + 2 + 3 + 4 + 5 + 12) / 6 = 27 / 6 = 4.5
np.mean(A)
```

Out[1]:

```
4.5
```

In [2]:

```python
# Example 2: np.median
B = [1,2,3,5,1,5,61,7,3] # Ordered List B: [1, 1, 2, 3, 3, 5, 5, 7, 61] --> median = 3
np.median(B)
```

Out[2]:

```
3.0
```

In [3]:

```
# Example 3: np.argmax and np.argnub
C = [1,2,3,100,0,5,100,7,9] # We know that the maximum value in this list is 100, but t
here are 100s in this list
                            # the first 100 is at index 3 and the second 100 is at inde
x 6. np.argmax will only
                            # return the index of the first maximum value, which is 3.
np.argmax(C)
```

Out[3]:

```
3
```

In [4]:

```
# Example 4: np.percentile
D = [1,2,4,5,0,6,8,7,3] # Ordered List B: [0, 1, 2, 3, 4, 5, 6, 7, 8] --> 25% percentil
e will be 2
np.percentile(D, 25)
```

Out[4]:

```
2.0
```

Part 3

Find a sample data set (or create a reasonable data set and place in an appropriate structure, e.g., DataFrame or other). Using GroupBy and splitting using more than one key, perform some queries against that data. Explain what question you're trying to answer and show the results/output. Come up with a total of three (3) different queries.

In [5]:

```
import pandas as pd
data = pd.read_csv("clean_diamond_data.csv")
data.head()
```

Out[5]:

	carat	clarity	color	cut	price
0	0.30	VS1	I	Good	229
1	0.30	SI2	F	Very Good	262
2	0.31	SI2	G	Very Good	262
3	0.29	SI2	E	Very Good	263
4	0.30	SI1	J	Very Good	269

In [6]:

```
# Query 1: Find out the number of diamonds by color and by cut
data.groupby(["color", 'cut']).size()
```

Out[6]:

```
color  cut
D      Astor Ideal      410
       Good            2276
       Ideal          20270
       Very Good      10998
E      Astor Ideal      506
       Good            2462
       Ideal          21390
       Very Good      11633
F      Astor Ideal      507
       Good            2531
       Ideal          22433
       Very Good      11500
G      Astor Ideal      620
       Good            2281
       Ideal          21021
       Very Good      10437
H      Astor Ideal      538
       Good            1848
       Ideal          17328
       Very Good       7717
I      Astor Ideal      522
       Good            1702
       Ideal          16633
       Very Good       7161
J      Astor Ideal      269
       Good            1306
       Ideal           9681
       Very Good       4658
dtype: int64
```

In [7]:

```
# Query 2: Find out the average price of a diamond for each clarity by cut
data[['clarity', 'cut', 'price']].groupby(['clarity', 'cut']).agg('mean')
```

Out[7]:

		price
clarity	**cut**	
FL	**Astor Ideal**	4141.000000
	Good	171428.833333
	Ideal	24634.819292
	Very Good	97460.664634
IF	**Astor Ideal**	3888.123288
	Good	15863.193182
	Ideal	6337.480622
	Very Good	11385.122351
SI1	**Astor Ideal**	3829.383430
	Good	4644.109361
	Ideal	3639.630512
	Very Good	4948.145100
SI2	**Astor Ideal**	2174.860465
	Good	3941.782960
	Ideal	2952.450864
	Very Good	3743.539358
VS1	**Astor Ideal**	4298.202546
	Good	6566.943024
	Ideal	5586.883285
	Very Good	8688.191471
VS2	**Astor Ideal**	3798.241212
	Good	6828.726505
	Ideal	4968.526129
	Very Good	7585.547694
VVS1	**Astor Ideal**	5098.471698
	Good	5933.037838
	Ideal	4683.246190
	Very Good	7038.141964
VVS2	**Astor Ideal**	3934.745614
	Good	6857.590006
	Ideal	5341.666184
	Very Good	7072.037922

In [8]:

```
# Query 3: Find out the max & min in carat and max & min price within each clarity and color
data[['clarity', 'color', 'carat', 'price']].groupby(['clarity', 'color']).agg([('Maximum', 'max'), ('Minimum', 'min')])
```

Out[8]:

		carat		price	
		Maximum	Minimum	Maximum	Minimum
clarity	color				
FL	D	15.37	0.23	2317596	778
	E	10.08	0.24	801378	629
	F	4.21	0.24	181614	532
	G	6.16	0.30	225593	627
	H	3.03	0.24	50405	522
	I	3.30	1.02	36872	4285
	J	1.31	0.90	5878	2623
IF	D	7.60	0.23	786736	543
	E	10.25	0.23	967776	370
	F	12.10	0.23	1191901	446
	G	10.62	0.23	450832	409
	H	5.60	0.23	175970	327
	I	5.41	0.23	103836	326
	J	8.05	0.23	183361	390
SI1	D	15.38	0.23	755279	319
	E	10.07	0.23	498323	300
	F	11.08	0.23	298065	277
	G	20.10	0.23	856860	294
	H	11.06	0.23	323213	279
	I	11.01	0.23	372241	278
	J	12.24	0.23	295367	269
SI2	D	11.58	0.23	407965	315
	E	5.29	0.23	90512	263
	F	7.33	0.23	128350	262
	G	7.06	0.23	131621	262
	H	6.04	0.24	104067	273
	I	10.23	0.23	251302	279
	J	10.64	0.25	188418	300
VS1	D	10.27	0.23	944545	318
	E	10.20	0.23	726829	309
	F	15.06	0.23	1047585	297
	G	11.03	0.23	575328	298
	H	12.12	0.23	558048	338
	I	15.16	0.23	636777	229
	J	20.45	0.23	781752	304

		carat		price	
		Maximum	**Minimum**	**Maximum**	**Minimum**
clarity	**color**				
VS2	**D**	8.52	0.23	429031	293
	E	10.02	0.23	456256	327
	F	10.64	0.23	589226	309
	G	10.03	0.23	407377	296
	H	20.05	0.23	1582746	305
	I	14.01	0.23	373391	292
	J	15.32	0.24	463169	287
VVS1	**D**	15.88	0.23	1666041	379
	E	9.09	0.23	701009	369
	F	10.05	0.23	510307	324
	G	10.86	0.23	724799	347
	H	10.71	0.23	386488	341
	I	10.02	0.23	303445	299
	J	8.04	0.23	207519	313
VVS2	**D**	10.51	0.23	1169933	346
	E	8.52	0.23	532730	355
	F	11.28	0.23	1201892	306
	G	16.06	0.23	1284065	318
	H	10.96	0.23	584827	313
	I	20.01	0.23	705603	274
	J	10.02	0.23	217543	322