In [17]:
```
import pandas as pd
import numpy as np
```

In [18]:
```
diamonds=pd.read_csv('diamonds.csv')
print(diamonds)
```

```
       carat        cut color clarity  depth  table  price
0       0.23      Ideal     E     SI2   61.5   55.0    326
1       0.21    Premium     E     SI1   59.8   61.0    326
2       0.23       Good     E     VS1   56.9   65.0    327
3       0.29    Premium     I     VS2   62.4   58.0    334
4       0.31       Good     J     SI2   63.3   58.0    335
...      ...        ...   ...     ...    ...    ...    ...
53935   0.72      Ideal     D     SI1   60.8   57.0   2757
53936   0.72       Good     D     SI1   63.1   55.0   2757
53937   0.70  Very Good     D     SI1   62.8   60.0   2757
53938   0.86    Premium     H     SI2   61.0   58.0   2757
53939   0.75      Ideal     D     SI2   62.2   55.0   2757

[53940 rows x 7 columns]
```

In [19]:
```
# displaying the unique values of  a column
cut_list=diamonds['cut'].unique()
print(cut_list)
print()

# Number of unique values
n=diamonds['cut'].nunique()
print(f'the number of unique values is {n}')
print()
```

```
['Ideal' 'Premium' 'Good' 'Very Good' 'Fair']

the number of unique values is 5
```

Aggregation functions

Aggregation	Description
count()	Total number of items
first(), last()	First and last item
mean(), median()	Mean and median
min(), max()	Minimum and maximum
std(), var()	Standard deviation and variance

mad() **Mean absolute deviation**

prod() **Product of all items**

sum() **Sum of all items**

These are all methods of DataFrame and Series objects.

In [25]:
```
# examples
print(diamonds.mean())
print()
print(diamonds.agg('mean'))
print()
print(diamonds[['carat','price']].agg(['mean','sum']))
```

```
carat       0.797940
depth      61.749405
table      57.457184
price    3932.799722
dtype: float64

carat       0.797940
depth      61.749405
table      57.457184
price    3932.799722
dtype: float64

             carat         price
mean      0.79794  3.932800e+03
sum   43040.87000  2.121352e+08
```

In [27]:
```
diamonds.aggregate({"carat":['max', 'min'],
                    "price":['max', 'sum','mean'],
                    "depth":['min', 'max']})
```

Out[27]:

	carat	price	depth
max	5.01	1.882300e+04	79.0
min	0.20	NaN	43.0
sum	NaN	2.121352e+08	NaN
mean	NaN	3.932800e+03	NaN

Groupby

The 3 Steps of a Groupby Process

Any groupby process involves some combination of the following 3 steps:

1 - Splitting the original object into groups based on the defined criteria.

2 - Applying a function to each group.

3 - Combining the results.

Splitting the Original DataFrame into Groups

In [39]:
```
grpcut=diamonds.groupby('cut')   # we are splitting the diamonds dataframe per
print(grpcut)
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000017942593BE0>
```

grpcut is a GroupBy object

In [40]:
```
#groups and indices attributes of the groupby object

print(grpcut.indices)
print()
print(grpcut.groups)    # numbers in the list are the row number.
print()
```

```
{'Fair': array([    8,    91,    97, ..., 53863, 53879, 53882], dtype=int64),
'Good': array([    2,     4,    10, ..., 53916, 53927, 53936], dtype=int64),
'Ideal': array([    0,    11,    13, ..., 53929, 53935, 53939], dtype=int64),
'Premium': array([    1,     3,    12, ..., 53931, 53934, 53938], dtype=int6
4), 'Very Good': array([    5,     6,     7, ..., 53932, 53933, 53937], dtype
=int64)}

{'Fair': [8, 91, 97, 123, 124, 128, 129, 204, 227, 241, 255, 296, 298, 314, 3
52, 356, 359, 369, 376, 384, 385, 423, 439, 440, 443, 472, 514, 526, 581, 59
5, 635, 663, 676, 703, 704, 706, 712, 713, 714, 719, 750, 770, 771, 777, 788,
801, 839, 879, 880, 883, 884, 895, 897, 898, 899, 919, 929, 933, 934, 938, 93
9, 967, 1019, 1096, 1097, 1098, 1140, 1183, 1199, 1227, 1235, 1238, 1267, 127
0, 1275, 1319, 1324, 1355, 1357, 1359, 1361, 1362, 1364, 1412, 1423, 1438, 14
69, 1498, 1506, 1515, 1523, 1524, 1527, 1552, 1554, 1597, 1598, 1599, 1608, 1
664, ...], 'Good': [2, 4, 10, 17, 18, 20, 35, 36, 37, 42, 43, 44, 47, 59, 74,
84, 95, 96, 145, 169, 175, 184, 189, 190, 203, 221, 238, 239, 243, 244, 251,
272, 279, 285, 305, 320, 321, 333, 360, 380, 381, 388, 400, 403, 428, 447, 46
1, 475, 476, 482, 499, 504, 511, 518, 528, 537, 544, 567, 590, 654, 658, 664,
667, 677, 694, 705, 710, 723, 724, 725, 726, 729, 748, 753, 769, 803, 859, 86
3, 868, 874, 882, 905, 906, 932, 948, 958, 965, 974, 997, 1002, 1009, 1037, 1
041, 1054, 1075, 1091, 1092, 1093, 1094, 1128, ...], 'Ideal': [0, 11, 13, 16,
39, 40, 41, 51, 52, 55, 60, 62, 63, 65, 66, 82, 83, 90, 92, 102, 104, 105, 10
7, 108, 109, 110, 111, 114, 115, 117, 118, 119, 120, 121, 130, 132, 138, 139,
144, 149, 151, 155, 156, 159, 163, 164, 167, 168, 170, 173, 174, 179, 180, 18
1, 182, 183, 185, 191, 198, 207, 209, 212, 213, 214, 216, 217, 220, 224, 229,
233, 234, 237, 240, 248, 249, 250, 256, 258, 262, 265, 269, 273, 274, 278, 29
1, 292, 293, 294, 295, 300, 302, 303, 308, 309, 312, 313, 315, 316, 318, 326,
...], 'Premium': [1, 3, 12, 14, 15, 26, 45, 53, 54, 56, 61, 64, 68, 69, 72, 7
3, 85, 86, 87, 88, 89, 99, 101, 103, 106, 112, 116, 125, 126, 135, 137, 140,
141, 150, 152, 153, 157, 158, 160, 171, 178, 187, 192, 193, 194, 195, 196, 19
7, 199, 201, 202, 205, 211, 215, 222, 225, 226, 228, 242, 245, 246, 252, 253,
254, 257, 259, 260, 263, 264, 266, 267, 268, 277, 280, 281, 282, 283, 284, 28
8, 289, 290, 297, 306, 311, 317, 319, 322, 323, 324, 327, 328, 334, 335, 336,
337, 338, 339, 353, 354, 357, ...], 'Very Good': [5, 6, 7, 9, 19, 21, 22, 23,
24, 25, 27, 28, 29, 30, 31, 32, 33, 34, 38, 46, 48, 49, 50, 57, 58, 67, 70, 7
1, 75, 76, 77, 78, 79, 80, 81, 93, 94, 98, 100, 113, 122, 127, 131, 133, 134,
136, 142, 143, 146, 147, 148, 154, 161, 162, 165, 166, 172, 176, 177, 186, 18
8, 200, 206, 208, 210, 218, 219, 223, 230, 231, 232, 235, 236, 247, 261, 270,
271, 275, 276, 286, 287, 299, 301, 304, 307, 310, 325, 329, 330, 340, 342, 34
3, 344, 345, 347, 361, 362, 364, 365, 367, ...]}
```

In [41]:
```
# groupby object methods

print(grpcut.size())       # method to  display group sizes (how many rows).
print()
print(grpcut.first())      # preview the result with the first or last entry f
print()
print(grpcut.last())
print()
```

```
cut
Fair          1610
Good          4906
Ideal        21551
Premium      13791
Very Good    12082
dtype: int64

           carat color clarity  depth  table  price
cut
Fair        0.22     E     VS2   65.1   61.0    337
Good        0.23     E     VS1   56.9   65.0    327
Ideal       0.23     E     SI2   61.5   55.0    326
Premium     0.21     E     SI1   59.8   61.0    326
Very Good   0.24     J    VVS2   62.8   57.0    336

           carat color clarity  depth  table  price
cut
Fair        0.71     D     VS1   65.4   59.0   2747
Good        0.72     D     SI1   63.1   55.0   2757
Ideal       0.75     D     SI2   62.2   55.0   2757
Premium     0.86     H     SI2   61.0   58.0   2757
Very Good   0.70     D     SI1   62.8   60.0   2757
```

In [42]:
```
print(grpcut.get_group('Good')  )# Get Specific  Group
print()
print(grpcut.get_group('Good')[['carat','price']] )     # Get Specific Columns
print()
```

```
       carat   cut color clarity  depth  table  price
2       0.23  Good     E     VS1   56.9   65.0    327
4       0.31  Good     J     SI2   63.3   58.0    335
10      0.30  Good     J     SI1   64.0   55.0    339
17      0.30  Good     J     SI1   63.4   54.0    351
18      0.30  Good     J     SI1   63.8   56.0    351
...      ...   ...   ...     ...    ...    ...    ...
53913   0.80  Good     G     VS2   64.2   58.0   2753
53914   0.84  Good     I     VS1   63.7   59.0   2753
53916   0.74  Good     D     SI1   63.1   59.0   2753
53927   0.79  Good     F     SI1   58.1   59.0   2756
53936   0.72  Good     D     SI1   63.1   55.0   2757

[4906 rows x 7 columns]

       carat  price
2       0.23    327
4       0.31    335
10      0.30    339
17      0.30    351
18      0.30    351
...      ...    ...
53913   0.80   2753
53914   0.84   2753
53916   0.74   2753
53927   0.79   2756
53936   0.72   2757

[4906 rows x 2 columns]
```

Iterating over groups

In [43]:
```
for name, group in grpcut:
    print(name)
    print(group)
    print()
```

```
Fair
       carat   cut color clarity  depth  table  price
8       0.22  Fair     E     VS2   65.1   61.0    337
91      0.86  Fair     E     SI2   55.1   69.0   2757
97      0.96  Fair     F     SI2   66.3   62.0   2759
123     0.70  Fair     F     VS2   64.5   57.0   2762
124     0.70  Fair     F     VS2   65.3   55.0   2762
...      ...   ...   ...     ...    ...    ...    ...
53757   0.72  Fair     F     VS2   55.4   64.0   2724
53800   0.90  Fair     I     VS1   68.7   62.0   2732
53863   1.00  Fair     I     SI2   66.8   56.0   2743
53879   1.04  Fair     G     SI2   65.2   57.0   2745
53882   0.71  Fair     D     VS1   65.4   59.0   2747

[1610 rows x 7 columns]

Good
       carat   cut color clarity  depth  table  price
2       0.23  Good     E     VS1   56.9   65.0    327
4       0.31  Good     J     SI2   63.3   58.0    335
10      0.30  Good     J     SI1   64.0   55.0    339
17      0.30  Good     J     SI1   63.4   54.0    351
18      0.30  Good     J     SI1   63.8   56.0    351
...      ...   ...   ...     ...    ...    ...    ...
53913   0.80  Good     G     VS2   64.2   58.0   2753
53914   0.84  Good     I     VS1   63.7   59.0   2753
53916   0.74  Good     D     SI1   63.1   59.0   2753
53927   0.79  Good     F     SI1   58.1   59.0   2756
53936   0.72  Good     D     SI1   63.1   55.0   2757

[4906 rows x 7 columns]

Ideal
       carat    cut color clarity  depth  table  price
0       0.23  Ideal     E     SI2   61.5   55.0    326
11      0.23  Ideal     J     VS1   62.8   56.0    340
13      0.31  Ideal     J     SI2   62.2   54.0    344
16      0.30  Ideal     I     SI2   62.0   54.0    348
39      0.33  Ideal     I     SI2   61.8   55.0    403
...      ...    ...   ...     ...    ...    ...    ...
53925   0.79  Ideal     I     SI1   61.6   56.0   2756
53926   0.71  Ideal     E     SI1   61.9   56.0   2756
53929   0.71  Ideal     G     VS1   61.4   56.0   2756
53935   0.72  Ideal     D     SI1   60.8   57.0   2757
53939   0.75  Ideal     D     SI2   62.2   55.0   2757

[21551 rows x 7 columns]

Premium
       carat      cut color clarity  depth  table  price
1       0.21  Premium     E     SI1   59.8   61.0    326
3       0.29  Premium     I     VS2   62.4   58.0    334
12      0.22  Premium     F     SI1   60.4   61.0    342
14      0.20  Premium     E     SI2   60.2   62.0    345
15      0.32  Premium     E      I1   60.9   58.0    345
...      ...      ...   ...     ...    ...    ...    ...
53928   0.79  Premium     E     SI2   61.4   58.0   2756
```

```
53930   0.71  Premium     E     SI1   60.5   55.0   2756
53931   0.71  Premium     F     SI1   59.8   62.0   2756
53934   0.72  Premium     D     SI1   62.7   59.0   2757
53938   0.86  Premium     H     SI2   61.0   58.0   2757

[13791 rows x 7 columns]

Very Good
       carat        cut color clarity  depth  table  price
5       0.24  Very Good     J    VVS2   62.8   57.0    336
6       0.24  Very Good     I    VVS1   62.3   57.0    336
7       0.26  Very Good     H     SI1   61.9   55.0    337
9       0.23  Very Good     H     VS1   59.4   61.0    338
19      0.30  Very Good     J     SI1   62.7   59.0    351
...      ...        ...   ...     ...    ...    ...    ...
53921   0.70  Very Good     E     VS2   62.8   60.0   2755
53922   0.70  Very Good     D     VS1   63.1   59.0   2755
53932   0.70  Very Good     E     VS2   60.5   59.0   2757
53933   0.70  Very Good     E     VS2   61.2   59.0   2757
53937   0.70  Very Good     D     SI1   62.8   60.0   2757

[12082 rows x 7 columns]
```

Pandas Groupby for Data Aggregation

Aggregate functions in the Pandas package:

count() – Number of non-null observations

sum() – Sum of values

mean() – Mean of values

median() – Arithmetic median of values

min() – Minimum

max() – Maximum

mode() – Mode

std() – Standard deviation

var() – Variance

In [44]:
```
grpcut_mean=grpcut.mean()  #  mean values for each numeric column by group.
print(grpcut_mean)
```

```
              carat      depth      table        price
cut
Fair       1.046137  64.041677  59.053789  4358.757764
Good       0.849185  62.365879  58.694639  3928.864452
Ideal      0.702837  61.709401  55.951668  3457.541970
Premium    0.891955  61.264673  58.746095  4584.257704
Very Good  0.806381  61.818275  57.956150  3981.759891
```

In [45]:
```
grpcut_mean_price=grpcut['price'].mean()    # calculate mean for the 'price' c
print(grpcut_mean_price)
print(type(grpcut_mean_price))  # cutgrp_mean_price is a Serie
```

```
cut
Fair         4358.757764
Good         3928.864452
Ideal        3457.541970
Premium      4584.257704
Very Good    3981.759891
Name: price, dtype: float64
<class 'pandas.core.series.Series'>
```

Groupby multiple columns - count diamonds per cut and color

In [46]:
```
grp_cut_color=diamonds.groupby(['cut','color'])
print(grp_cut_color.count())
```

		carat	clarity	depth	table	price
cut	color					
Fair	D	163	163	163	163	163
	E	224	224	224	224	224
	F	312	312	312	312	312
	G	314	314	314	314	314
	H	303	303	303	303	303
	I	175	175	175	175	175
	J	119	119	119	119	119
Good	D	662	662	662	662	662
	E	933	933	933	933	933
	F	909	909	909	909	909
	G	871	871	871	871	871
	H	702	702	702	702	702
	I	522	522	522	522	522
	J	307	307	307	307	307
Ideal	D	2834	2834	2834	2834	2834
	E	3903	3903	3903	3903	3903
	F	3826	3826	3826	3826	3826
	G	4884	4884	4884	4884	4884
	H	3115	3115	3115	3115	3115
	I	2093	2093	2093	2093	2093
	J	896	896	896	896	896
Premium	D	1603	1603	1603	1603	1603
	E	2337	2337	2337	2337	2337
	F	2331	2331	2331	2331	2331
	G	2924	2924	2924	2924	2924
	H	2360	2360	2360	2360	2360
	I	1428	1428	1428	1428	1428
	J	808	808	808	808	808
Very Good	D	1513	1513	1513	1513	1513
	E	2400	2400	2400	2400	2400
	F	2164	2164	2164	2164	2164
	G	2299	2299	2299	2299	2299
	H	1824	1824	1824	1824	1824
	I	1204	1204	1204	1204	1204
	J	678	678	678	678	678

In [47]:
```
print(grp_cut_color['price'].sum())
```

```
cut        color
Fair       D          699443
           E          824838
           F         1194025
           G         1331126
           H         1556112
           I          819953
           J          592103
Good       D         2254363
           E         3194260
           F         3177637
           G         3591553
           H         3001931
           I         2650994
           J         1404271
Ideal      D         7450854
           E        10138238
           F        12912518
           G        18171930
           H        12115278
           I         9317974
           J         4406695
Premium    D         5820962
           E         8270443
           F        10081319
           G        13160170
           H        12311428
           I         8491146
           J         5086030
Very Good  D         5250817
           E         7715165
           F         8177367
           G         8903461
           H         8272552
           I         6328079
           J         3460182
Name: price, dtype: int64
```

Applying Multiple Aggregation Functions at Once using agg()

In [48]:
```
grp_cut_color[['price']].agg([np.sum, np.mean, np.std])
```

Out[48]:

		price		
		sum	mean	std
cut	color			
Fair	D	699443	4291.061350	3286.114238
	E	824838	3682.312500	2976.651645
	F	1194025	3827.003205	3223.302685
	G	1331126	4239.254777	3609.644379
	H	1556112	5135.683168	3886.481847
	I	819953	4685.445714	3730.271132
	J	592103	4975.655462	4050.458933
Good	D	2254363	3405.382175	3175.148710
	E	3194260	3423.644159	3330.702061
	F	3177637	3495.750275	3202.411187
	G	3591553	4123.482204	3702.504718
	H	3001931	4276.254986	4020.660488
	I	2650994	5078.532567	4631.702141
	J	1404271	4574.172638	3707.790845
Ideal	D	7450854	2629.094566	3001.069919
	E	10138238	2597.550090	2956.007149
	F	12912518	3374.939362	3766.635328
	G	18171930	3720.706388	4006.262468
	H	12115278	3889.334831	4013.375228
	I	9317974	4451.970377	4505.150405
	J	4406695	4918.186384	4476.206836
Premium	D	5820962	3631.292576	3711.634010
	E	8270443	3538.914420	3794.987184
	F	10081319	4324.890176	4012.022756
	G	13160170	4500.742134	4356.571034
	H	12311428	5216.706780	4466.189717
	I	8491146	5946.180672	5053.746146
	J	5086030	6294.591584	4788.936691

		price		
		sum	mean	std
cut	color			
Very Good	D	5250817	3470.467284	3523.753268
	E	7715165	3214.652083	3408.023634
	F	8177367	3778.820240	3786.124033
	G	8903461	3872.753806	3861.375464
	H	8272552	4535.390351	4185.798202
	I	6328079	5255.879568	4687.104775
	J	3460182	5103.513274	4135.652742

Applying custom function

In [49]:
```
# The function return the number of diamonds with color 'D' for each group
def get_D_color(df):              # df is the DataFrame of each group
    return len(df[df["color"] == 'D'])

# Set the custom function as the parameter of apply()
print(grpcut.apply(get_D_color))
```

```
cut
Fair          163
Good          662
Ideal        2834
Premium      1603
Very Good    1513
dtype: int64
```

Filtering: filter() , query()

In [50]:
```
grpcut_mean=grpcut.mean()  #  mean values for each numeric column by group.
print(grpcut.size())
print(grpcut_mean)
print(grpcut['price'].filter(lambda x: x.mean() > 4400))
```

```
cut
Fair          1610
Good          4906
Ideal        21551
Premium      13791
Very Good    12082
dtype: int64
              carat      depth      table        price
cut
Fair       1.046137  64.041677  59.053789  4358.757764
Good       0.849185  62.365879  58.694639  3928.864452
Ideal      0.702837  61.709401  55.951668  3457.541970
Premium    0.891955  61.264673  58.746095  4584.257704
Very Good  0.806381  61.818275  57.956150  3981.759891
1           326
3           334
12          342
14          345
15          345
```

In [51]:
```
print(diamonds.query("color=='D'").groupby('cut').size())

print()
```

```
cut
Fair          163
Good          662
Ideal        2834
Premium      1603
Very Good    1513
dtype: int64
```

Groupby and NaN

In [52]:
```
df = pd.DataFrame(
    [
        (1, 'B', 121, 10.1, True),
        (2, 'C', 145, 5.5, False),
        (3, 'A', 345, 4.5, False),
        (4, 'A', 112, np.nan, True),
        (5, 'C', 105, 2.1, False),
        (6, np.nan, 435, 7.8, True),
        (7, np.nan, 521, np.nan, True),
        (8, 'B', 322, 8.7, True),
        (9, 'C', 213, 5.8, True),
        (10, 'B', 718, 9.1, False),
    ],
    columns=['colA', 'colB', 'colC', 'colD', 'colE'])
print(df)
```

```
   colA colB  colC  colD   colE
0     1    B   121  10.1   True
1     2    C   145   5.5  False
2     3    A   345   4.5  False
3     4    A   112   NaN   True
4     5    C   105   2.1  False
5     6  NaN   435   7.8   True
6     7  NaN   521   NaN   True
7     8    B   322   8.7   True
8     9    C   213   5.8   True
9    10    B   718   9.1  False
```

In [53]:
```
print(df.groupby('colB')['colD'].sum())
print()
print(df.groupby('colB', dropna=False)['colD'].sum())
print()
#df.fillna(-1).groupby('colB').sum()
```

```
colB
A     4.5
B    27.9
C    13.4
Name: colD, dtype: float64

colB
A       4.5
B      27.9
C      13.4
NaN     7.8
Name: colD, dtype: float64
```

In [54]:
```
print(df.groupby('colB').count())
print()
print(df.groupby('colB', dropna=False).count())
```

```
      colA  colC  colD  colE
colB
A        2     2     1     2
B        3     3     3     3
C        3     3     3     3

      colA  colC  colD  colE
colB
A        2     2     1     2
B        3     3     3     3
C        3     3     3     3
NaN      2     2     1     2
```