(http://www.pieriandata.com)

Groupby Operations and Multi-level Index

In [1]:
```
import numpy as np
import pandas as pd
```

Data

In [2]:
```
df = pd.read_csv('mpg.csv')
```

In [3]:
```
df
```

Out[3]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	
0	18.0	8	307.0	130	3504	12.0	70	1	ch cl
1	15.0	8	350.0	165	3693	11.5	70	1	;
2	18.0	8	318.0	150	3436	11.0	70	1	ply s
3	16.0	8	304.0	150	3433	12.0	70	1	re
4	17.0	8	302.0	140	3449	10.5	70	1	
...	...	...	...	...	...	...	...	...	
393	27.0	4	140.0	86	2790	15.6	82	1	m
394	44.0	4	97.0	52	2130	24.6	82	2	
395	32.0	4	135.0	84	2295	11.6	82	1	ra
396	28.0	4	120.0	79	2625	18.6	82	1	
397	31.0	4	119.0	82	2720	19.4	82	1	ch

398 rows × 9 columns

groupby() method

In [4]:
```
# Creates a groupby object waiting for an aggregate method
df.groupby('model_year')
```

Out[4]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x00000246790FEC88
>
```

Adding an aggregate method call. To use a grouped object, you need to tell pandas how you want to aggregate the data.

Common Options:

```
mean(): Compute mean of groups
sum(): Compute sum of group values
size(): Compute group sizes
count(): Compute count of group
std(): Standard deviation of groups
var(): Compute variance of groups
sem(): Standard error of the mean of groups
describe(): Generates descriptive statistics
first(): Compute first of group values
```

In [5]:

```python
# model_year becomes the index! It is NOT a column name,it is now the name
df.groupby('model_year').mean()
```

Out[5]:

model_year	mpg	cylinders	displacement	weight	acceleration	origin
70	17.689655	6.758621	281.413793	3372.793103	12.948276	1.310345
71	21.250000	5.571429	209.750000	2995.428571	15.142857	1.428571
72	18.714286	5.821429	218.375000	3237.714286	15.125000	1.535714
73	17.100000	6.375000	256.875000	3419.025000	14.312500	1.375000
74	22.703704	5.259259	171.740741	2877.925926	16.203704	1.666667
75	20.266667	5.600000	205.533333	3176.800000	16.050000	1.466667
76	21.573529	5.647059	197.794118	3078.735294	15.941176	1.470588
77	23.375000	5.464286	191.392857	2997.357143	15.435714	1.571429
78	24.061111	5.361111	177.805556	2861.805556	15.805556	1.611111
79	25.093103	5.827586	206.689655	3055.344828	15.813793	1.275862
80	33.696552	4.137931	115.827586	2436.655172	16.934483	2.206897
81	30.334483	4.620690	135.310345	2522.931034	16.306897	1.965517
82	31.709677	4.193548	128.870968	2453.548387	16.638710	1.645161

In [6]:

```python
avg_year = df.groupby('model_year').mean()
```

In [7]:

```python
avg_year.index
```

Out[7]:

```
Int64Index([70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82], dtype='in
t64', name='model_year')
```

In [8]:

```python
avg_year.columns
```

Out[8]:

```
Index(['mpg', 'cylinders', 'displacement', 'weight', 'acceleration', 'orig
in'], dtype='object')
```

In [9]:
```
avg_year['mpg']
```

Out[9]:
```
model_year
70    17.689655
71    21.250000
72    18.714286
73    17.100000
74    22.703704
75    20.266667
76    21.573529
77    23.375000
78    24.061111
79    25.093103
80    33.696552
81    30.334483
82    31.709677
Name: mpg, dtype: float64
```

In [10]:
```
df.groupby('model_year').mean()['mpg']
```

Out[10]:
```
model_year
70    17.689655
71    21.250000
72    18.714286
73    17.100000
74    22.703704
75    20.266667
76    21.573529
77    23.375000
78    24.061111
79    25.093103
80    33.696552
81    30.334483
82    31.709677
Name: mpg, dtype: float64
```

In [11]:
```
df.groupby('model_year').describe()
```

Out[11]:

	mpg								cylinders	
	count	mean	std	min	25%	50%	75%	max	count	mean
model_year										
70	29.0	17.689655	5.339231	9.0	14.000	16.00	22.000	27.0	29.0	6.758621
71	28.0	21.250000	6.591942	12.0	15.500	19.00	27.000	35.0	28.0	5.571429
72	28.0	18.714286	5.435529	11.0	13.750	18.50	23.000	28.0	28.0	5.821429
73	40.0	17.100000	4.700245	11.0	13.000	16.00	20.000	29.0	40.0	6.375000
74	27.0	22.703704	6.420010	13.0	16.000	24.00	27.000	32.0	27.0	5.259259
75	30.0	20.266667	4.940566	13.0	16.000	19.50	23.000	33.0	30.0	5.600000
76	34.0	21.573529	5.889297	13.0	16.750	21.00	26.375	33.0	34.0	5.647059
77	28.0	23.375000	6.675862	15.0	17.375	21.75	30.000	36.0	28.0	5.464286
78	36.0	24.061111	6.898044	16.2	19.350	20.70	28.000	43.1	36.0	5.361111
79	29.0	25.093103	6.794217	15.5	19.200	23.90	31.800	37.3	29.0	5.827586
80	29.0	33.696552	7.037983	19.1	29.800	32.70	38.100	46.6	29.0	4.137931
81	29.0	30.334483	5.591465	17.6	26.600	31.60	34.400	39.1	29.0	4.620690
82	31.0	31.709677	5.392548	22.0	27.000	32.00	36.000	44.0	31.0	4.193548

13 rows × 48 columns

In [12]:
```
df.groupby('model_year').describe().transpose()
```

Out[12]:

	model_year	70	71	72	73	74
mpg	count	29.000000	28.000000	28.000000	40.000000	27.000000
	mean	17.689655	21.250000	18.714286	17.100000	22.703704
	std	5.339231	6.591942	5.435529	4.700245	6.420010
	min	9.000000	12.000000	11.000000	11.000000	13.000000
	25%	14.000000	15.500000	13.750000	13.000000	16.000000
	50%	16.000000	19.000000	18.500000	16.000000	24.000000
	75%	22.000000	27.000000	23.000000	20.000000	27.000000
	max	27.000000	35.000000	28.000000	29.000000	32.000000
cylinders	count	29.000000	28.000000	28.000000	40.000000	27.000000
	mean	6.758621	5.571429	5.821429	6.375000	5.259259
	std	1.724926	1.665079	2.073708	1.807215	1.583390
	min	4.000000	4.000000	3.000000	3.000000	4.000000
	25%	6.000000	4.000000	4.000000	4.000000	4.000000
	50%	8.000000	6.000000	4.000000	7.000000	4.000000
	75%	8.000000	6.500000	8.000000	8.000000	6.000000
	max	8.000000	8.000000	8.000000	8.000000	8.000000
displacement	count	29.000000	28.000000	28.000000	40.000000	27.000000
	mean	281.413793	209.750000	218.375000	256.875000	171.740741
	std	124.421380	115.102410	123.781964	121.722085	92.601127
	min	97.000000	71.000000	70.000000	68.000000	71.000000
	25%	198.000000	97.750000	109.250000	121.750000	90.000000
	50%	307.000000	228.500000	131.000000	276.000000	122.000000
	75%	383.000000	273.000000	326.000000	350.250000	250.000000
	max	455.000000	400.000000	429.000000	455.000000	350.000000
weight	count	29.000000	28.000000	28.000000	40.000000	27.000000
	mean	3372.793103	2995.428571	3237.714286	3419.025000	2877.925926
	std	852.868663	1061.830859	974.520960	974.809133	949.308571
	min	1835.000000	1613.000000	2100.000000	1867.000000	1649.000000
	25%	2648.000000	2110.750000	2285.500000	2554.500000	2116.500000
	50%	3449.000000	2798.000000	2956.000000	3338.500000	2489.000000
	75%	4312.000000	3603.250000	4169.750000	4247.250000	3622.500000
	max	4732.000000	5140.000000	4633.000000	4997.000000	4699.000000

	model_year	70	71	72	73	74
acceleration	count	29.000000	28.000000	28.000000	40.000000	27.000000
	mean	12.948276	15.142857	15.125000	14.312500	16.203704
	std	3.330982	2.666171	2.850032	2.754222	1.688532
	min	8.000000	11.500000	11.000000	9.500000	13.500000
	25%	10.000000	13.375000	13.375000	12.500000	15.250000
	50%	12.500000	14.500000	14.500000	14.000000	16.000000
	75%	15.000000	16.125000	16.625000	16.000000	17.000000
	max	20.500000	20.500000	23.500000	21.000000	21.000000
origin	count	29.000000	28.000000	28.000000	40.000000	27.000000
	mean	1.310345	1.428571	1.535714	1.375000	1.666667
	std	0.603765	0.741798	0.792658	0.667467	0.832050
	min	1.000000	1.000000	1.000000	1.000000	1.000000
	25%	1.000000	1.000000	1.000000	1.000000	1.000000
	50%	1.000000	1.000000	1.000000	1.000000	1.000000
	75%	1.000000	2.000000	2.000000	2.000000	2.000000
	max	3.000000	3.000000	3.000000	3.000000	3.000000

Groupby Multiple Columns

Let's explore average mpg per year per cylinder count

In [13]:
```
df.groupby(['model_year','cylinders']).mean()
```

Out[13]:

model_year	cylinders	mpg	displacement	weight	acceleration	origin
70	4	25.285714	107.000000	2292.571429	16.000000	2.285714
	6	20.500000	199.000000	2710.500000	15.500000	1.000000
	8	14.111111	367.555556	3940.055556	11.194444	1.000000
71	4	27.461538	101.846154	2056.384615	16.961538	1.923077
	6	18.000000	243.375000	3171.875000	14.750000	1.000000
	8	13.428571	371.714286	4537.714286	12.214286	1.000000
72	3	19.000000	70.000000	2330.000000	13.500000	3.000000
	4	23.428571	111.535714	2382.642857	17.214286	1.928571
	8	13.615385	344.846154	4228.384615	13.000000	1.000000
73	3	18.000000	70.000000	2124.000000	13.500000	3.000000
	4	22.727273	109.272727	2338.090909	17.136364	2.000000
	6	19.000000	212.250000	2917.125000	15.687500	1.250000
	8	13.200000	365.250000	4279.050000	12.250000	1.000000
74	4	27.800000	96.533333	2151.466667	16.400000	2.200000
	6	17.857143	230.428571	3320.000000	16.857143	1.000000
	8	14.200000	315.200000	4438.400000	14.700000	1.000000
75	4	25.250000	114.833333	2489.250000	15.833333	2.166667
	6	17.583333	233.750000	3398.333333	17.708333	1.000000
	8	15.666667	330.500000	4108.833333	13.166667	1.000000
76	4	26.766667	106.333333	2306.600000	16.866667	1.866667
	6	20.000000	221.400000	3349.600000	17.000000	1.300000
	8	14.666667	324.000000	4064.666667	13.222222	1.000000
77	3	21.500000	80.000000	2720.000000	13.500000	3.000000
	4	29.107143	106.500000	2205.071429	16.064286	1.857143
	6	19.500000	220.400000	3383.000000	16.900000	1.400000
	8	16.000000	335.750000	4177.500000	13.662500	1.000000
78	4	29.576471	112.117647	2296.764706	16.282353	2.117647
	5	20.300000	131.000000	2830.000000	15.900000	2.000000
	6	19.066667	213.250000	3314.166667	16.391667	1.166667
	8	19.050000	300.833333	3563.333333	13.266667	1.000000
79	4	31.525000	113.583333	2357.583333	15.991667	1.583333
	5	25.400000	183.000000	3530.000000	20.100000	2.000000
	6	22.950000	205.666667	3025.833333	15.433333	1.000000
	8	18.630000	321.400000	3862.900000	15.400000	1.000000
80	3	23.700000	70.000000	2420.000000	12.500000	3.000000
	4	34.612000	111.000000	2360.080000	17.144000	2.200000
	5	36.400000	121.000000	2950.000000	19.900000	2.000000
	6	25.900000	196.500000	3145.500000	15.050000	2.000000

		mpg	displacement	weight	acceleration	origin
model_year	**cylinders**					
81	**4**	32.814286	108.857143	2275.476190	16.466667	2.095238
	6	23.428571	184.000000	3093.571429	15.442857	1.714286
	8	26.600000	350.000000	3725.000000	19.000000	1.000000
82	**4**	32.071429	118.571429	2402.321429	16.703571	1.714286
	6	28.333333	225.000000	2931.666667	16.033333	1.000000

In [14]:
```
df.groupby(['model_year','cylinders']).mean().index
```

Out[14]:
```
MultiIndex([(70, 4),
            (70, 6),
            (70, 8),
            (71, 4),
            (71, 6),
            (71, 8),
            (72, 3),
            (72, 4),
            (72, 8),
            (73, 3),
            (73, 4),
            (73, 6),
            (73, 8),
            (74, 4),
            (74, 6),
            (74, 8),
            (75, 4),
            (75, 6),
            (75, 8),
            (76, 4),
            (76, 6),
            (76, 8),
            (77, 3),
            (77, 4),
            (77, 6),
            (77, 8),
            (78, 4),
            (78, 5),
            (78, 6),
            (78, 8),
            (79, 4),
            (79, 5),
            (79, 6),
            (79, 8),
            (80, 3),
            (80, 4),
            (80, 5),
            (80, 6),
            (81, 4),
            (81, 6),
            (81, 8),
            (82, 4),
            (82, 6)],
           names=['model_year', 'cylinders'])
```

MultiIndex

The MultiIndex Object

In [15]:
```
year_cyl = df.groupby(['model_year','cylinders']).mean()
```

In [16]:
```
year_cyl
```

Out[16]:

model_year	cylinders	mpg	displacement	weight	acceleration	origin
70	4	25.285714	107.000000	2292.571429	16.000000	2.285714
	6	20.500000	199.000000	2710.500000	15.500000	1.000000
	8	14.111111	367.555556	3940.055556	11.194444	1.000000
71	4	27.461538	101.846154	2056.384615	16.961538	1.923077
	6	18.000000	243.375000	3171.875000	14.750000	1.000000
	8	13.428571	371.714286	4537.714286	12.214286	1.000000
72	3	19.000000	70.000000	2330.000000	13.500000	3.000000
	4	23.428571	111.535714	2382.642857	17.214286	1.928571
	8	13.615385	344.846154	4228.384615	13.000000	1.000000
73	3	18.000000	70.000000	2124.000000	13.500000	3.000000
	4	22.727273	109.272727	2338.090909	17.136364	2.000000
	6	19.000000	212.250000	2917.125000	15.687500	1.250000
	8	13.200000	365.250000	4279.050000	12.250000	1.000000
74	4	27.800000	96.533333	2151.466667	16.400000	2.200000
	6	17.857143	230.428571	3320.000000	16.857143	1.000000
	8	14.200000	315.200000	4438.400000	14.700000	1.000000
75	4	25.250000	114.833333	2489.250000	15.833333	2.166667
	6	17.583333	233.750000	3398.333333	17.708333	1.000000
	8	15.666667	330.500000	4108.833333	13.166667	1.000000
76	4	26.766667	106.333333	2306.600000	16.866667	1.866667
	6	20.000000	221.400000	3349.600000	17.000000	1.300000
	8	14.666667	324.000000	4064.666667	13.222222	1.000000
77	3	21.500000	80.000000	2720.000000	13.500000	3.000000
	4	29.107143	106.500000	2205.071429	16.064286	1.857143
	6	19.500000	220.400000	3383.000000	16.900000	1.400000
	8	16.000000	335.750000	4177.500000	13.662500	1.000000
78	4	29.576471	112.117647	2296.764706	16.282353	2.117647
	5	20.300000	131.000000	2830.000000	15.900000	2.000000
	6	19.066667	213.250000	3314.166667	16.391667	1.166667
	8	19.050000	300.833333	3563.333333	13.266667	1.000000
79	4	31.525000	113.583333	2357.583333	15.991667	1.583333
	5	25.400000	183.000000	3530.000000	20.100000	2.000000
	6	22.950000	205.666667	3025.833333	15.433333	1.000000
	8	18.630000	321.400000	3862.900000	15.400000	1.000000
80	3	23.700000	70.000000	2420.000000	12.500000	3.000000
	4	34.612000	111.000000	2360.080000	17.144000	2.200000
	5	36.400000	121.000000	2950.000000	19.900000	2.000000
	6	25.900000	196.500000	3145.500000	15.050000	2.000000

		mpg	displacement	weight	acceleration	origin
model_year	**cylinders**					
81	**4**	32.814286	108.857143	2275.476190	16.466667	2.095238
	6	23.428571	184.000000	3093.571429	15.442857	1.714286
	8	26.600000	350.000000	3725.000000	19.000000	1.000000
82	**4**	32.071429	118.571429	2402.321429	16.703571	1.714286
	6	28.333333	225.000000	2931.666667	16.033333	1.000000

In [17]:
```
year_cyl.index
```

Out[17]:
```
MultiIndex([(70, 4),
            (70, 6),
            (70, 8),
            (71, 4),
            (71, 6),
            (71, 8),
            (72, 3),
            (72, 4),
            (72, 8),
            (73, 3),
            (73, 4),
            (73, 6),
            (73, 8),
            (74, 4),
            (74, 6),
            (74, 8),
            (75, 4),
            (75, 6),
            (75, 8),
            (76, 4),
            (76, 6),
            (76, 8),
            (77, 3),
            (77, 4),
            (77, 6),
            (77, 8),
            (78, 4),
            (78, 5),
            (78, 6),
            (78, 8),
            (79, 4),
            (79, 5),
            (79, 6),
            (79, 8),
            (80, 3),
            (80, 4),
            (80, 5),
            (80, 6),
            (81, 4),
            (81, 6),
            (81, 8),
            (82, 4),
            (82, 6)],
           names=['model_year', 'cylinders'])
```

In [18]:
```
year_cyl.index.levels
```

Out[18]:
```
FrozenList([[70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82], [3, 4, 5, 6, 8]])
```

In [19]:
```
year_cyl.index.names
```

Out[19]:
```
FrozenList(['model_year', 'cylinders'])
```

Indexing with the Hierarchical Index

Full Documentation: [https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html (https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html)](https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html)

In [20]:
```
year_cyl.head()
```

Out[20]:

		mpg	displacement	weight	acceleration	origin
model_year	**cylinders**					
70	**4**	25.285714	107.000000	2292.571429	16.000000	2.285714
	6	20.500000	199.000000	2710.500000	15.500000	1.000000
	8	14.111111	367.555556	3940.055556	11.194444	1.000000
71	**4**	27.461538	101.846154	2056.384615	16.961538	1.923077
	6	18.000000	243.375000	3171.875000	14.750000	1.000000

Grab Based on Outside Index

In [21]:
```
year_cyl.loc[70]
```

Out[21]:

	mpg	displacement	weight	acceleration	origin
cylinders					
4	25.285714	107.000000	2292.571429	16.000000	2.285714
6	20.500000	199.000000	2710.500000	15.500000	1.000000
8	14.111111	367.555556	3940.055556	11.194444	1.000000

In [22]:
```
year_cyl.loc[[70,72]]
```

Out[22]:

model_year	cylinders	mpg	displacement	weight	acceleration	origin
70	4	25.285714	107.000000	2292.571429	16.000000	2.285714
	6	20.500000	199.000000	2710.500000	15.500000	1.000000
	8	14.111111	367.555556	3940.055556	11.194444	1.000000
72	3	19.000000	70.000000	2330.000000	13.500000	3.000000
	4	23.428571	111.535714	2382.642857	17.214286	1.928571
	8	13.615385	344.846154	4228.384615	13.000000	1.000000

Grab a Single Row

In [23]:
```
year_cyl.loc[(70,8)]
```

Out[23]:
```
mpg                14.111111
displacement      367.555556
weight           3940.055556
acceleration       11.194444
origin              1.000000
Name: (70, 8), dtype: float64
```

Grab Based on Cross-section with .xs()

This method takes a `key` argument to select data at a particular level of a MultiIndex.

Parameters

```
key : label or tuple of label
    Label contained in the index, or partially in a MultiIndex.
axis : {0 or 'index', 1 or 'columns'}, default 0
    Axis to retrieve cross-section on.
level : object, defaults to first n levels (n=1 or len(key))
    In case of a key partially contained in a MultiIndex, indicate
    which levels are used. Levels can be referred by label or posi
tion.
```

In [24]:
```
year_cyl.xs(key=70,axis=0,level='model_year')
```

Out[24]:

cylinders	mpg	displacement	weight	acceleration	origin
4	25.285714	107.000000	2292.571429	16.000000	2.285714
6	20.500000	199.000000	2710.500000	15.500000	1.000000
8	14.111111	367.555556	3940.055556	11.194444	1.000000

In [25]:
```
# Mean column values for 4 cylinders per year
year_cyl.xs(key=4,axis=0,level='cylinders')
```

Out[25]:

model_year	mpg	displacement	weight	acceleration	origin
70	25.285714	107.000000	2292.571429	16.000000	2.285714
71	27.461538	101.846154	2056.384615	16.961538	1.923077
72	23.428571	111.535714	2382.642857	17.214286	1.928571
73	22.727273	109.272727	2338.090909	17.136364	2.000000
74	27.800000	96.533333	2151.466667	16.400000	2.200000
75	25.250000	114.833333	2489.250000	15.833333	2.166667
76	26.766667	106.333333	2306.600000	16.866667	1.866667
77	29.107143	106.500000	2205.071429	16.064286	1.857143
78	29.576471	112.117647	2296.764706	16.282353	2.117647
79	31.525000	113.583333	2357.583333	15.991667	1.583333
80	34.612000	111.000000	2360.080000	17.144000	2.200000
81	32.814286	108.857143	2275.476190	16.466667	2.095238
82	32.071429	118.571429	2402.321429	16.703571	1.714286

Careful note!

Keep in mind, its usually much easier to filter out values **before** running a groupby() call, so you should attempt to filter out any values/categories you don't want to use. For example, its much easier to remove **4** cylinder cars before the groupby() call, very difficult to this sort of thing after a group by.

In [26]:
```
df[df['cylinders'].isin([6,8])].groupby(['model_year','cylinders']).mean()
```

Out[26]:

model_year	cylinders	mpg	displacement	weight	acceleration	origin
70	6	20.500000	199.000000	2710.500000	15.500000	1.000000
	8	14.111111	367.555556	3940.055556	11.194444	1.000000
71	6	18.000000	243.375000	3171.875000	14.750000	1.000000
	8	13.428571	371.714286	4537.714286	12.214286	1.000000
72	8	13.615385	344.846154	4228.384615	13.000000	1.000000
73	6	19.000000	212.250000	2917.125000	15.687500	1.250000
	8	13.200000	365.250000	4279.050000	12.250000	1.000000
74	6	17.857143	230.428571	3320.000000	16.857143	1.000000
	8	14.200000	315.200000	4438.400000	14.700000	1.000000
75	6	17.583333	233.750000	3398.333333	17.708333	1.000000
	8	15.666667	330.500000	4108.833333	13.166667	1.000000
76	6	20.000000	221.400000	3349.600000	17.000000	1.300000
	8	14.666667	324.000000	4064.666667	13.222222	1.000000
77	6	19.500000	220.400000	3383.000000	16.900000	1.400000
	8	16.000000	335.750000	4177.500000	13.662500	1.000000
78	6	19.066667	213.250000	3314.166667	16.391667	1.166667
	8	19.050000	300.833333	3563.333333	13.266667	1.000000
79	6	22.950000	205.666667	3025.833333	15.433333	1.000000
	8	18.630000	321.400000	3862.900000	15.400000	1.000000
80	6	25.900000	196.500000	3145.500000	15.050000	2.000000
81	6	23.428571	184.000000	3093.571429	15.442857	1.714286
	8	26.600000	350.000000	3725.000000	19.000000	1.000000
82	6	28.333333	225.000000	2931.666667	16.033333	1.000000

Swap Levels

- Swapping Levels: https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#swapping-levels-with-swaplevel (https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#swapping-levels-with-swaplevel)
- Generalized Method is reorder_levels: https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#reordering-levels-with-reorder-levels (https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#reordering-levels-with-reorder-levels)

In [27]:
```
year_cyl.swaplevel().head()
```

Out[27]:

		mpg	displacement	weight	acceleration	origin
cylinders	**model_year**					
4	**70**	25.285714	107.000000	2292.571429	16.000000	2.285714
6	**70**	20.500000	199.000000	2710.500000	15.500000	1.000000
8	**70**	14.111111	367.555556	3940.055556	11.194444	1.000000
4	**71**	27.461538	101.846154	2056.384615	16.961538	1.923077
6	**71**	18.000000	243.375000	3171.875000	14.750000	1.000000

Sorting MultiIndex

- https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#sorting-a-multiindex (https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#sorting-a-multiindex)

In [28]:
```
year_cyl.sort_index(level='model_year',ascending=False)
```

Out[28]:

model_year	cylinders	mpg	displacement	weight	acceleration	origin
82	6	28.333333	225.000000	2931.666667	16.033333	1.000000
	4	32.071429	118.571429	2402.321429	16.703571	1.714286
81	8	26.600000	350.000000	3725.000000	19.000000	1.000000
	6	23.428571	184.000000	3093.571429	15.442857	1.714286
	4	32.814286	108.857143	2275.476190	16.466667	2.095238
80	6	25.900000	196.500000	3145.500000	15.050000	2.000000
	5	36.400000	121.000000	2950.000000	19.900000	2.000000
	4	34.612000	111.000000	2360.080000	17.144000	2.200000
	3	23.700000	70.000000	2420.000000	12.500000	3.000000
79	8	18.630000	321.400000	3862.900000	15.400000	1.000000
	6	22.950000	205.666667	3025.833333	15.433333	1.000000
	5	25.400000	183.000000	3530.000000	20.100000	2.000000
	4	31.525000	113.583333	2357.583333	15.991667	1.583333
78	8	19.050000	300.833333	3563.333333	13.266667	1.000000
	6	19.066667	213.250000	3314.166667	16.391667	1.166667
	5	20.300000	131.000000	2830.000000	15.900000	2.000000
	4	29.576471	112.117647	2296.764706	16.282353	2.117647
77	8	16.000000	335.750000	4177.500000	13.662500	1.000000
	6	19.500000	220.400000	3383.000000	16.900000	1.400000
	4	29.107143	106.500000	2205.071429	16.064286	1.857143
	3	21.500000	80.000000	2720.000000	13.500000	3.000000
76	8	14.666667	324.000000	4064.666667	13.222222	1.000000
	6	20.000000	221.400000	3349.600000	17.000000	1.300000
	4	26.766667	106.333333	2306.600000	16.866667	1.866667
75	8	15.666667	330.500000	4108.833333	13.166667	1.000000
	6	17.583333	233.750000	3398.333333	17.708333	1.000000
	4	25.250000	114.833333	2489.250000	15.833333	2.166667
74	8	14.200000	315.200000	4438.400000	14.700000	1.000000
	6	17.857143	230.428571	3320.000000	16.857143	1.000000
	4	27.800000	96.533333	2151.466667	16.400000	2.200000
73	8	13.200000	365.250000	4279.050000	12.250000	1.000000
	6	19.000000	212.250000	2917.125000	15.687500	1.250000
	4	22.727273	109.272727	2338.090909	17.136364	2.000000
	3	18.000000	70.000000	2124.000000	13.500000	3.000000
72	8	13.615385	344.846154	4228.384615	13.000000	1.000000
	4	23.428571	111.535714	2382.642857	17.214286	1.928571
	3	19.000000	70.000000	2330.000000	13.500000	3.000000

		mpg	displacement	weight	acceleration	origin
model_year	**cylinders**					
	8	13.428571	371.714286	4537.714286	12.214286	1.000000
71	**6**	18.000000	243.375000	3171.875000	14.750000	1.000000
	4	27.461538	101.846154	2056.384615	16.961538	1.923077
	8	14.111111	367.555556	3940.055556	11.194444	1.000000
70	**6**	20.500000	199.000000	2710.500000	15.500000	1.000000
	4	25.285714	107.000000	2292.571429	16.000000	2.285714

In [29]:
```
year_cyl.sort_index(level='cylinders',ascending=False)
```

Out[29]:

		mpg	displacement	weight	acceleration	origin
model_year	**cylinders**					
81	**8**	26.600000	350.000000	3725.000000	19.000000	1.000000
79	**8**	18.630000	321.400000	3862.900000	15.400000	1.000000
78	**8**	19.050000	300.833333	3563.333333	13.266667	1.000000
77	**8**	16.000000	335.750000	4177.500000	13.662500	1.000000
76	**8**	14.666667	324.000000	4064.666667	13.222222	1.000000
75	**8**	15.666667	330.500000	4108.833333	13.166667	1.000000
74	**8**	14.200000	315.200000	4438.400000	14.700000	1.000000
73	**8**	13.200000	365.250000	4279.050000	12.250000	1.000000
72	**8**	13.615385	344.846154	4228.384615	13.000000	1.000000
71	**8**	13.428571	371.714286	4537.714286	12.214286	1.000000
70	**8**	14.111111	367.555556	3940.055556	11.194444	1.000000
82	**6**	28.333333	225.000000	2931.666667	16.033333	1.000000
81	**6**	23.428571	184.000000	3093.571429	15.442857	1.714286
80	**6**	25.900000	196.500000	3145.500000	15.050000	2.000000
79	**6**	22.950000	205.666667	3025.833333	15.433333	1.000000
78	**6**	19.066667	213.250000	3314.166667	16.391667	1.166667
77	**6**	19.500000	220.400000	3383.000000	16.900000	1.400000
76	**6**	20.000000	221.400000	3349.600000	17.000000	1.300000
75	**6**	17.583333	233.750000	3398.333333	17.708333	1.000000
74	**6**	17.857143	230.428571	3320.000000	16.857143	1.000000
73	**6**	19.000000	212.250000	2917.125000	15.687500	1.250000
71	**6**	18.000000	243.375000	3171.875000	14.750000	1.000000
70	**6**	20.500000	199.000000	2710.500000	15.500000	1.000000
80	**5**	36.400000	121.000000	2950.000000	19.900000	2.000000
79	**5**	25.400000	183.000000	3530.000000	20.100000	2.000000
78	**5**	20.300000	131.000000	2830.000000	15.900000	2.000000
82	**4**	32.071429	118.571429	2402.321429	16.703571	1.714286
81	**4**	32.814286	108.857143	2275.476190	16.466667	2.095238
80	**4**	34.612000	111.000000	2360.080000	17.144000	2.200000
79	**4**	31.525000	113.583333	2357.583333	15.991667	1.583333
78	**4**	29.576471	112.117647	2296.764706	16.282353	2.117647
77	**4**	29.107143	106.500000	2205.071429	16.064286	1.857143
76	**4**	26.766667	106.333333	2306.600000	16.866667	1.866667
75	**4**	25.250000	114.833333	2489.250000	15.833333	2.166667
74	**4**	27.800000	96.533333	2151.466667	16.400000	2.200000
73	**4**	22.727273	109.272727	2338.090909	17.136364	2.000000
72	**4**	23.428571	111.535714	2382.642857	17.214286	1.928571
71	**4**	27.461538	101.846154	2056.384615	16.961538	1.923077

model_year	cylinders	mpg	displacement	weight	acceleration	origin
70	**4**	25.285714	107.000000	2292.571429	16.000000	2.285714
80	**3**	23.700000	70.000000	2420.000000	12.500000	3.000000
77	**3**	21.500000	80.000000	2720.000000	13.500000	3.000000
73	**3**	18.000000	70.000000	2124.000000	13.500000	3.000000
72	**3**	19.000000	70.000000	2330.000000	13.500000	3.000000

Advanced: agg() method

The agg() method allows you to customize what aggregate functions you want per category

In [33]:
```
df
```

Out[33]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	
0	18.0	8	307.0	130	3504	12.0	70	1	ch cl
1	15.0	8	350.0	165	3693	11.5	70	1	
2	18.0	8	318.0	150	3436	11.0	70	1	ply s
3	16.0	8	304.0	150	3433	12.0	70	1	re
4	17.0	8	302.0	140	3449	10.5	70	1	
...	...	...	...	...	...	...	...	...	
393	27.0	4	140.0	86	2790	15.6	82	1	m
394	44.0	4	97.0	52	2130	24.6	82	2	
395	32.0	4	135.0	84	2295	11.6	82	1	ra
396	28.0	4	120.0	79	2625	18.6	82	1	
397	31.0	4	119.0	82	2720	19.4	82	1	cl

398 rows × 9 columns

agg() on a DataFrame

In [35]:
```
# These strings need to match up with built-in method names
df.agg(['median','mean'])
```

Out[35]:

	mpg	cylinders	displacement	weight	acceleration	model_year	origin
median	23.000000	4.000000	148.500000	2803.500000	15.50000	76.00000	1.000000
mean	23.514573	5.454774	193.425879	2970.424623	15.56809	76.01005	1.572864

In [41]:
```
df.agg(['sum','mean'])[['mpg','weight']]
```

Out[41]:

	mpg	weight
sum	9358.800000	1.182229e+06
mean	23.514573	2.970425e+03

Specify aggregate methods per column

agg() is very powerful,allowing you to pass in a dictionary where the keys are the columns and the values are a list of aggregate methods.

In [43]:
```
df.agg({'mpg':['median','mean'],'weight':['mean','std']})
```

Out[43]:

	mpg	weight
mean	23.514573	2970.424623
median	23.000000	NaN
std	NaN	846.841774

agg() with groupby()

In [44]:
```
df.groupby('model_year').agg({'mpg':['median','mean'],'weight':['mean','std
```

Out[44]:

	mpg		weight	
	median	mean	mean	std
model_year				
70	16.00	17.689655	3372.793103	852.868663
71	19.00	21.250000	2995.428571	1061.830859
72	18.50	18.714286	3237.714286	974.520960
73	16.00	17.100000	3419.025000	974.809133
74	24.00	22.703704	2877.925926	949.308571
75	19.50	20.266667	3176.800000	765.179781
76	21.00	21.573529	3078.735294	821.371481
77	21.75	23.375000	2997.357143	912.825902
78	20.70	24.061111	2861.805556	626.023907
79	23.90	25.093103	3055.344828	747.881497
80	32.70	33.696552	2436.655172	432.235491
81	31.60	30.334483	2522.931034	533.600501
82	32.00	31.709677	2453.548387	354.276713