Part_4_GroupBy_Operations

October 31, 2022

[2]:
```python
import pandas as pd
import numpy as np
```

[4]:
```python
df = pd.read_csv('/content/sample_data/mpg.csv')
```

[ ]:
```python
df.head()
```

[ ]:
```
    mpg  cylinders  displacement horsepower  weight  acceleration  model_year  \
0  18.0          8         307.0        130    3504          12.0          70
1  15.0          8         350.0        165    3693          11.5          70
2  18.0          8         318.0        150    3436          11.0          70
3  16.0          8         304.0        150    3433          12.0          70
4  17.0          8         302.0        140    3449          10.5          70

   origin                       name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino
```

0.1 Groupby Operations:

It allows us to examine/view data on 'per category' basis.

It means we will view the data from the point of view that perticular category/column of the dataframe.

For example, we want to view the whole data from point of view of category/column 'model_year'. We will write it as:

```
df.groupby('model_year')
```

Calling groupy creates a lazy object which waits to be evaluated by an aggregate method call e.g. sum(), mean(), max(), min() etc.

0.2 Why to use groupby operations?

There are a number of reasons to use groupby operations in pandas.

1. One reason is to calculate statistics such as mean, median, mode, etc. on groups of data.
2. Another reason to use groupby operations is to split data into groups for further analysis. For example, you could group data by gender and then analyze the data within each group separately.
3. It helps to answer questions like how many rows you have per category?

[5]:
```
df['model_year'].value_counts()
```

[5]:
```
73    40
78    36
76    34
82    31
75    30
70    29
79    29
80    29
81    29
71    28
72    28
77    28
74    27
Name: model_year, dtype: int64
```

0.3 Question: Show the average number of cylidners usage, average displacement etc. all for each miles per gallons category?

[6]:
```
df.groupby('mpg').mean()
```

[6]:
```
      cylinders  displacement   weight  acceleration  model_year  origin
mpg
9.0         8.0        304.00  4732.00     18.500000        70.0     1.0
10.0        8.0        333.50  4495.50     14.500000        70.0     1.0
11.0        8.0        374.25  4419.00     12.375000        72.0     1.0
12.0        8.0        394.50  4786.50     12.083333        72.5     1.0
13.0        8.0        353.00  4254.45     12.935000        73.3     1.0
…          …             …        …             …           …       …
43.4        4.0         90.00  2335.00     23.700000        80.0     2.0
44.0        4.0         97.00  2130.00     24.600000        82.0     2.0
44.3        4.0         90.00  2085.00     21.700000        80.0     2.0
44.6        4.0         91.00  1850.00     13.800000        80.0     3.0
46.6        4.0         86.00  2110.00     17.900000        80.0     3.0

[129 rows x 6 columns]
```

0.4 Question: Show the maximum number of cylidners usage, displacement etc. all for each miles per gallons category?

```
[ ]: df.groupby('mpg').max()
```

```
[ ]:       cylinders  displacement horsepower  weight  acceleration  model_year  \
mpg
9.0            8         304.0        193    4732          18.5          70
10.0           8         360.0        215    4615          15.0          70
11.0           8         429.0        210    4997          14.0          73
12.0           8         455.0        225    4955          13.5          73
13.0           8         440.0        215    5140          16.0          76
…          …           …        …     …           …         …
43.4           4          90.0         48    2335          23.7          80
44.0           4          97.0         52    2130          24.6          82
44.3           4          90.0         48    2085          21.7          80
44.6           4          91.0         67    1850          13.8          80
46.6           4          86.0         65    2110          17.9          80

      origin                        name
mpg
9.0        1                   hi 1200d
10.0       1                  ford f250
11.0       1           oldsmobile omega
12.0       1  oldsmobile vista cruiser
13.0       1       pontiac safari (sw)
…       …                        …
43.4       2        vw dasher (diesel)
44.0       2                 vw pickup
44.3       2       vw rabbit c (diesel)
44.6       3       honda civic 1500 gl
46.6       3                 mazda glc

[129 rows x 8 columns]
```

0.5 Question: Show the maximum horsepower usage for each miles per gallons (mpg) category?

```
[8]: df.groupby('mpg').max()['horsepower']
```

```
[8]: mpg
9.0     193
10.0    215
11.0    210
12.0    225
```

```
13.0    215
       …
43.4     48
44.0     52
44.3     48
44.6     67
46.6     65
Name: horsepower, Length: 129, dtype: object
```

1 Multi-level hierarchy:

1.1 Question: Show the average displacement for each model year (outer index) and miles per gallons (mpg) (inner index) category?

```
[13]: df.groupby(['model_year', 'mpg']).mean()['displacement']
```

```
[13]: model_year  mpg
      70          9.0     304.00
                  10.0    333.50
                  11.0    318.00
                  14.0    428.80
                  15.0    390.40
                           …
      82          34.0    110.00
                  36.0    113.00
                  37.0     91.00
                  38.0    137.25
                  44.0     97.00
      Name: displacement, Length: 264, dtype: float64
```

Question: Which columns in the dataset are the columns and which are index?

Answer: The columns in the groupby call are the named indexes now and all remaining are columns to use.

```
[ ]: df.groupby(['model_year', 'mpg']).mean().columns
```

```
[ ]: Index(['cylinders', 'displacement', 'weight', 'acceleration', 'origin'],
     dtype='object')
```

```
[ ]: df.groupby(['model_year', 'mpg']).mean().index
```

```
[ ]: MultiIndex([(70,  9.0),
                 (70, 10.0),
                 (70, 11.0),
                 (70, 14.0),
                 (70, 15.0),
                 (70, 16.0),
                 (70, 17.0),
                 (70, 18.0),
                 (70, 21.0),
                 (70, 22.0),
                 …
                 (82, 27.0),
                 (82, 28.0),
                 (82, 29.0),
                 (82, 31.0),
                 (82, 32.0),
                 (82, 34.0),
                 (82, 36.0),
                 (82, 37.0),
                 (82, 38.0),
                 (82, 44.0)],
                names=['model_year', 'mpg'], length=264)
```

[ ]:
```
df.groupby(['model_year', 'mpg']).mean().index.names
```

[ ]:
```
FrozenList(['model_year', 'mpg'])
```

[ ]:
```
# To show the levels of indexes: we have used two columns in groupby so it has␣
↪two levels here.

df.groupby(['model_year', 'mpg']).mean().index.levels
```

[ ]:
```
FrozenList([[70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82], [9.0, 10.0,
11.0, 12.0, 13.0, 14.0, 14.5, 15.0, 15.5, 16.0, 16.2, 16.5, 16.9, 17.0, 17.5,
17.6, 17.7, 18.0, 18.1, 18.2, 18.5, 18.6, 19.0, 19.1, 19.2, 19.4, 19.8, 19.9,
20.0, 20.2, 20.3, 20.5, 20.6, 20.8, 21.0, 21.1, 21.5, 21.6, 22.0, 22.3, 22.4,
22.5, 23.0, 23.2, 23.5, 23.6, 23.7, 23.8, 23.9, 24.0, 24.2, 24.3, 24.5, 25.0,
25.1, 25.4, 25.5, 25.8, 26.0, 26.4, 26.5, 26.6, 26.8, 27.0, 27.2, 27.4, 27.5,
27.9, 28.0, 28.1, 28.4, 28.8, 29.0, 29.5, 29.8, 29.9, 30.0, 30.5, 30.7, 30.9,
31.0, 31.3, 31.5, 31.6, 31.8, 31.9, 32.0, 32.1, 32.2, 32.3, 32.4, 32.7, 32.8,
32.9, 33.0, 33.5, 33.7, 33.8, 34.0, 34.1, …]])
```

1.2 Question: Show the average acceleration for the model year '70' (outer index) and all miles per gallons (mpg) (inner index) category?

[14]:
```
df.groupby(['model_year', 'mpg']).mean().loc[70]
```

[14]:
```
      cylinders  displacement  weight  acceleration  origin
mpg
```

```
9.0     8.000000    304.000000  4732.0     18.500000     1.0
10.0    8.000000    333.500000  4495.5     14.500000     1.0
11.0    8.000000    318.000000  4382.0     13.500000     1.0
14.0    8.000000    428.800000  3957.2      9.100000     1.0
15.0    8.000000    390.400000  3841.6      9.900000     1.0
16.0    8.000000    304.000000  3433.0     12.000000     1.0
17.0    8.000000    302.000000  3449.0     10.500000     1.0
18.0    7.333333    274.666667  3238.0     12.833333     1.0
21.0    6.000000    199.500000  2617.5     15.500000     1.0
22.0    6.000000    198.000000  2833.0     15.500000     1.0
24.0    4.000000    110.000000  2401.0     14.750000     2.5
25.0    4.000000    107.000000  2523.5     17.500000     2.0
26.0    4.000000    109.000000  2034.5     16.500000     2.0
27.0    4.000000     97.000000  2130.0     14.500000     3.0
```

There's no model_year (70) written here because it is known that the whole data is shown for the particular year 70.

1.3 Question: Show the average acceleration for the model year '70' and '82'(outer index) and all miles per gallons (mpg) (inner index) category?

[15]:
```
df.groupby(['model_year', 'mpg']).mean().loc[[70, 82]]
```

[15]:
```
                  cylinders  displacement       weight  acceleration    origin
model_year mpg
70         9.0     8.000000    304.000000  4732.000000     18.500000  1.000000
           10.0    8.000000    333.500000  4495.500000     14.500000  1.000000
           11.0    8.000000    318.000000  4382.000000     13.500000  1.000000
           14.0    8.000000    428.800000  3957.200000      9.100000  1.000000
           15.0    8.000000    390.400000  3841.600000      9.900000  1.000000
           16.0    8.000000    304.000000  3433.000000     12.000000  1.000000
           17.0    8.000000    302.000000  3449.000000     10.500000  1.000000
           18.0    7.333333    274.666667  3238.000000     12.833333  1.000000
           21.0    6.000000    199.500000  2617.500000     15.500000  1.000000
           22.0    6.000000    198.000000  2833.000000     15.500000  1.000000
           24.0    4.000000    110.000000  2401.000000     14.750000  2.500000
           25.0    4.000000    107.000000  2523.500000     17.500000  2.000000
           26.0    4.000000    109.000000  2034.500000     16.500000  2.000000
           27.0    4.000000     97.000000  2130.000000     14.500000  3.000000
82         22.0    6.000000    232.000000  2835.000000     14.700000  1.000000
           23.0    4.000000    151.000000  3035.000000     20.500000  1.000000
           24.0    4.000000    140.000000  2865.000000     16.400000  1.000000
           25.0    6.000000    181.000000  2945.000000     16.400000  1.000000
           26.0    4.000000    156.000000  2585.000000     14.500000  1.000000
           27.0    4.000000    138.500000  2778.750000     17.375000  1.000000
           28.0    4.000000    116.000000  2615.000000     19.100000  1.000000
```

```
29.0    4.000000    135.000000  2525.000000     16.000000  1.000000
31.0    4.000000    107.333333  2421.666667     17.733333  1.666667
32.0    4.000000    123.333333  2308.333333     13.733333  2.333333
34.0    4.000000    110.000000  2320.000000     17.450000  2.000000
36.0    4.000000    113.000000  2168.000000     14.920000  2.000000
37.0    4.000000     91.000000  2025.000000     18.200000  3.000000
38.0    4.500000    137.250000  2275.000000     15.725000  2.000000
44.0    4.000000     97.000000  2130.000000     24.600000  2.000000
```

1.4 Question: Show the average of all columns for the model year '70' (outer index) and miles per gallons (mpg) 24.0 (inner index) category?

[ ]:
```python
df.groupby(['model_year', 'mpg']).mean().loc[(70, 24.0)]
```

[ ]:
```
cylinders          4.00
displacement     110.00
weight          2401.00
acceleration      14.75
origin             2.50
Name: (70, 24.0), dtype: float64
```

Nota bene:

- To show 2 values of same category, use [**val1, val2**]
- To show values of 2 differnet category, use [**(cat1_val, cat2_val)**]: Note tuple ( ) here inside [ ]

2 Cross Section:

Suppose we have outer index as **model_year** and inner index as **mpg**. We have seen the ways of checking all mpg's for one model_year or more than one model year. But we were'nt able to see what if we want to see all model's year for one mpg? Means we just want to give inner index value.

Sound Strange?

Its true that outer index is stronger than the inner index but we should have some method that can help us to deal with inner index too! For the weaker :)

So, the way to do it is by using cross section.

Syntax:

```
df.xs(key=val_to_eval, level='name_of_cat')
```

Disadvantage: It can't handle multi-index.

[29]:
```python
new_df = df.groupby(['model_year', 'mpg']).mean()
```

```
[32]: new_df.xs(key=22.0, level = 'mpg')
```

```
[32]:             cylinders  displacement  weight  acceleration  origin
      model_year
      70                6.0         198.0  2833.0         15.50     1.0
      71                4.0         140.0  2408.0         19.00     1.0
      72                4.0         121.5  2453.0         17.00     1.5
      73                4.0         108.0  2379.0         16.50     3.0
      75                4.0         121.0  2945.0         14.50     2.0
      76                6.0         237.5  3293.0         14.95     1.0
      77                6.0         146.0  2815.0         14.50     3.0
      82                6.0         232.0  2835.0         14.70     1.0
```

```
[34]: # To swap levels, make model_year as inner level and mpg as outer level

      new_df.swaplevel()
```

```
[34]:                  cylinders  displacement  weight  acceleration  origin
      mpg  model_year
      9.0  70                8.0        304.00  4732.0        18.500     1.0
      10.0 70                8.0        333.50  4495.5        14.500     1.0
      11.0 70                8.0        318.00  4382.0        13.500     1.0
      14.0 70                8.0        428.80  3957.2         9.100     1.0
      15.0 70                8.0        390.40  3841.6         9.900     1.0
      …                    …           …      …            …       …
      34.0 82                4.0        110.00  2320.0        17.450     2.0
      36.0 82                4.0        113.00  2168.0        14.920     2.0
      37.0 82                4.0         91.00  2025.0        18.200     3.0
      38.0 82                4.5        137.25  2275.0        15.725     2.0
      44.0 82                4.0         97.00  2130.0        24.600     2.0

      [264 rows x 5 columns]
```

3 Sorting in multi-level indexes:

```
[36]: new_df.sort_index(level='mpg', ascending = False)
```

```
[36]:                  cylinders  displacement  weight  acceleration  origin
      model_year mpg
      80         46.6        4.0          86.0  2110.0          17.9     3.0
                 44.6        4.0          91.0  1850.0          13.8     3.0
                 44.3        4.0          90.0  2085.0          21.7     2.0
      82         44.0        4.0          97.0  2130.0          24.6     2.0
      80         43.4        4.0          90.0  2335.0          23.7     2.0
      …                    …           …      …            …       …
```

```
73          11.0        8.0         375.0  4330.5         12.5      1.0
72          11.0        8.0         429.0  4633.0         11.0      1.0
70          11.0        8.0         318.0  4382.0         13.5      1.0
            10.0        8.0         333.5  4495.5         14.5      1.0
             9.0        8.0         304.0  4732.0         18.5      1.0

[264 rows x 5 columns]
```

4 Aggregate Functions:

Sometimes it doesn't make sense to apply one same function on all columns e.g. we can't imagine mean age of someone. So, good way is to apply separate functions for separate columns using **agg** ([ ]) call.

[38]:
```
df.agg(['std', 'mean'])
```

[38]:
```
            mpg  cylinders  displacement       weight  acceleration  \
std    7.815984   1.701004    104.269838   846.841774      2.757689
mean  23.514573   5.454774    193.425879  2970.424623     15.568090

      model_year    origin
std     3.697627  0.802055
mean   76.010050  1.572864
```

[39]:
```
df.agg(['std', 'mean'])['mpg']
```

[39]:
```
std      7.815984
mean    23.514573
Name: mpg, dtype: float64
```

[40]:
```
df.agg({'mpg': ['mean', 'max'], 'weight': ['std']})
```

[40]:
```
            mpg      weight
mean  23.514573         NaN
max   46.600000         NaN
std         NaN  846.841774
```