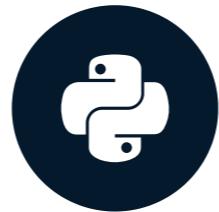


Summary statistics

DATA MANIPULATION WITH PANDAS



Maggie Matsui

Senior Content Developer at DataCamp

Summarizing numerical data

```
dogs["height_cm"].mean()
```

```
49.714285714285715
```

- `.median()` , `.mode()`
- `.min()` , `.max()`
- `.var()` , `.std()`
- `.sum()`
- `.quantile()`

Summarizing dates

Oldest dog:

```
dogs["date_of_birth"].min()
```

```
'2011-12-11'
```

Youngest dog:

```
dogs["date_of_birth"].max()
```

```
'2018-02-27'
```

The .agg() method

```
def pct30(column):  
    return column.quantile(0.3)  
  
dogs["weight_kg"].agg(pct30)
```

The .agg() method allows you to apply your own custom functions to a DataFrame, as well as apply functions to more than one column of a DataFrame at once, making your aggregations super-efficient.

22.59999999999998

34- Pandas DataFrames: Aggregation

<https://www.youtube.com/watch?v=2I2E1ZbF8pg>

Summaries on multiple columns

```
dogs[["weight_kg", "height_cm"]].agg(pct30)
```

```
weight_kg      22.6
height_cm     45.4
dtype: float64
```

Multiple summaries

```
def pct40(column):  
    return column.quantile(0.4)
```

```
dogs["weight_kg"].agg([pct30, pct40])
```

```
pct30    22.6  
pct40    24.0  
Name: weight_kg, dtype: float64
```

Cumulative sum



```
dogs["weight_kg"]
```

```
0    24  
1    24  
2    24  
3    17  
4    29  
5     2  
6   74
```

```
Name: weight_kg, dtype: int64
```

```
dogs["weight_kg"].cumsum()
```

```
0    24  
1    48  
2    72  
3    89  
4   118  
5   120  
6   194
```

```
Name: weight_kg, dtype: int64
```

Calling cumsum on a column returns not just one number, but a number for each row of the DataFrame. The first number returned, or the number in the zeroth index, is the first dog's weight. The next number is the sum of the first and second dogs' weights. The third number is the sum of the first, second, and third dogs' weights, and so on. The last number is the sum of all the dogs' weights.

Cumulative statistics

- `.cummax()`
- `.cummin()`
- `.cumprod()`

Walmart

```
sales.head()
```

| | store | type | dept | date | weekly_sales | is_holiday | temp_c | fuel_price | unemp |
|---|-------|------|------|------------|--------------|------------|--------|------------|-------|
| 0 | 1 | A | 1 | 2010-02-05 | 24924.50 | False | 5.73 | 0.679 | 8.106 |
| 1 | 1 | A | 2 | 2010-02-05 | 50605.27 | False | 5.73 | 0.679 | 8.106 |
| 2 | 1 | A | 3 | 2010-02-05 | 13740.12 | False | 5.73 | 0.679 | 8.106 |
| 3 | 1 | A | 4 | 2010-02-05 | 39954.04 | False | 5.73 | 0.679 | 8.106 |
| 4 | 1 | A | 5 | 2010-02-05 | 32229.38 | False | 5.73 | 0.679 | 8.106 |

Let's practice!

DATA MANIPULATION WITH PANDAS

Counting

DATA MANIPULATION WITH PANDAS



Maggie Matsui

Senior Content Developer at DataCamp

Avoiding double counting



Vet visits

```
print(vet_visits)
```

```
      date      name     breed  weight_kg
0  2018-09-02    Bella  Labrador     24.87
1  2019-06-07     Max  Labrador     28.35
2  2018-01-17   Stella Chihuahua     1.51
3  2019-10-19    Lucy  Chow Chow     24.07
..       ...
71 2018-01-20   Stella Chihuahua     2.83
72 2019-06-07     Max  Chow Chow     24.01
73 2018-08-20    Lucy  Chow Chow     24.40
74 2019-04-22     Max  Labrador     28.54
```

Dropping duplicate names

```
vet_visits.drop_duplicates(subset="name")
```

| | date | name | breed | weight_kg |
|----|------------|---------|-------------|-----------|
| 0 | 2018-09-02 | Bella | Labrador | 24.87 |
| 1 | 2019-06-07 | Max | Chow Chow | 24.01 |
| 2 | 2019-03-19 | Charlie | Poodle | 24.95 |
| 3 | 2018-01-17 | Stella | Chihuahua | 1.51 |
| 4 | 2019-10-19 | Lucy | Chow Chow | 24.07 |
| 7 | 2019-03-30 | Cooper | Schnauzer | 16.91 |
| 10 | 2019-01-04 | Bernie | St. Bernard | 74.98 |
| (6 | 2019-06-07 | Max | Labrador | 28.35) |

4. Dropping duplicate names

Let's try to fix this by removing rows that contain a dog name already listed earlier in the dataset, or in other words; we'll extract a dog with each name from the dataset once. We can do this using the `drop_duplicates` method. It takes an argument, `subset`, which is the column we want to find our duplicates based on - in this case, we want all the unique names. Now we have a list of dogs where each one appears once. We have Max the Chow Chow, but where did Max the Labrador go? Because we have two different dogs with the same name, we'll need to consider more than just name when dropping duplicates.

Dropping duplicate pairs

```
unique_dogs = vet_visits.drop_duplicates(subset=["name", "breed"])
print(unique_dogs)
```

| | date | name | breed | weight_kg |
|----|------------|---------|-------------|-----------|
| 0 | 2018-09-02 | Bella | Labrador | 24.87 |
| 1 | 2019-03-13 | Max | Chow Chow | 24.13 |
| 2 | 2019-03-19 | Charlie | Poodle | 24.95 |
| 3 | 2018-01-17 | Stella | Chihuahua | 1.51 |
| 4 | 2019-10-19 | Lucy | Chow Chow | 24.07 |
| 6 | 2019-06-07 | Max | Labrador | 28.35 |
| 7 | 2019-03-30 | Cooper | Schnauzer | 16.91 |
| 10 | 2019-01-04 | Bernie | St. Bernard | 74.98 |

5. Dropping duplicate pairs

Since Max and Max are different breeds, we can drop the rows with pairs of name and breed listed earlier in the dataset. To base our duplicate dropping on multiple columns, we can pass a list of column names to the subset argument, in this case, name and breed. Now both Maxes have been included, and we can start counting.

Easy as 1, 2, 3

```
unique_dogs["breed"].value_counts()
```

```
Labrador      2  
Schnauzer     1  
St. Bernard    1  
Chow Chow      2  
Poodle         1  
Chihuahua      1  
Name: breed, dtype: int64
```

To count the dog of each breed.

```
unique_dogs["breed"].value_counts(sort=True)
```

```
Labrador      2  
Chow Chow      2  
Schnauzer     1  
St. Bernard    1  
Poodle         1  
Chihuahua      1  
Name: breed, dtype: int64
```

The normalize argument can be used to turn the counts into **proportions** of the total. 25% of the dogs that go to this vet are Labradors.

```
unique_dogs["breed"].value_counts(normalize=True)
```

```
Labrador      0.250
Chow Chow     0.250
Schnauzer     0.125
St. Bernard    0.125
Poodle        0.125
Chihuahua     0.125
Name: breed, dtype: float64
```

Let's practice!

DATA MANIPULATION WITH PANDAS

Grouped summary statistics

DATA MANIPULATION WITH PANDAS



Maggie Matsui

Senior Content Developer at DataCamp

Summaries by group

```
dogs[dogs["color"] == "Black"]["weight_kg"].mean()  
dogs[dogs["color"] == "Brown"]["weight_kg"].mean()  
dogs[dogs["color"] == "White"]["weight_kg"].mean()  
dogs[dogs["color"] == "Gray"]["weight_kg"].mean()  
dogs[dogs["color"] == "Tan"]["weight_kg"].mean()
```

```
26.0  
24.0  
74.0  
17.0  
2.0
```

Grouped summaries

```
dogs.groupby("color")["weight_kg"].mean()
```

```
color
Black      26.5
Brown      24.0
Gray       17.0
Tan        2.0
White      74.0
Name: weight_kg, dtype: float64
```

3. Grouped summaries

That's where the `groupby` method comes in. We can group by the color variable, select the weight column, and take the mean. This will give us the mean weight for each dog color. This was just one line of code compared to the five we had to write before to get the same results.

Multiple grouped summaries

```
dogs.groupby("color")["weight_kg"].agg([min, max, sum])
```

| color | min | max | sum |
|-------|-----|-----|-----|
| Black | 24 | 29 | 53 |
| Brown | 24 | 24 | 48 |
| Gray | 17 | 17 | 17 |
| Tan | 2 | 2 | 2 |
| White | 74 | 74 | 74 |

4. Multiple grouped summaries

Just like with ungrouped summary statistics, we can use the `agg` method to get multiple statistics. Here, we pass a list of functions into `agg` after grouping by color. This gives us the minimum, maximum, and sum of the different colored dogs' weights.

Grouping by multiple variables

```
dogs.groupby(["color", "breed"])["weight_kg"].mean()
```

```
color   breed
Black   Chow  Chow      25
          Labrador      29
          Poodle        24
Brown   Chow  Chow      24
          Labrador      24
Gray    Schnauzer      17
Tan     Chihuahua      2
White   St. Bernard    74
Name: weight_kg, dtype: int64
```

5. Grouping by multiple variables

You can also group by multiple columns and calculate summary statistics. Here, we group by color and breed, select the weight column and take the mean. This gives us the mean weight of each breed of each color.

Many groups, many summaries

```
dogs.groupby(["color", "breed"])[["weight_kg", "height_cm"]].mean()
```

| | | weight_kg | height_cm |
|-------|-------------|-----------|-----------|
| color | breed | | |
| Black | Labrador | 29 | 59 |
| | Poodle | 24 | 43 |
| Brown | Chow Chow | 24 | 46 |
| | Labrador | 24 | 56 |
| Gray | Schnauzer | 17 | 49 |
| Tan | Chihuahua | 2 | 18 |
| White | St. Bernard | 74 | 77 |

6. Many groups, many summaries
You can also group by multiple columns and aggregate by multiple columns.

Let's practice!

DATA MANIPULATION WITH PANDAS

Pivot tables

DATA MANIPULATION WITH PANDAS



Maggie Matsui

Senior Content Developer at DataCamp

Group by to pivot table

The "values" argument is the column that you want to summarize, and the index column is the column that you want to group by. By default, pivot table takes the mean value for each group.

```
dogs.groupby("color")["weight_kg"].mean()
```

```
color
Black    26
Brown    24
Gray     17
Tan      2
White    74
Name: weight_kg, dtype: int64
```

```
dogs.pivot_table(values="weight_kg",
                  index="color")
```

```
           weight_kg
color
Black        26.5
Brown        24.0
Gray         17.0
Tan          2.0
White        74.0
```

Different statistics

3. Different statistics

If we want a different summary statistic, we can use the `aggfunc` argument and pass it a function. Here, we take the median for each dog color using NumPy's `median` function.

```
import numpy as np  
dogs.pivot_table(values="weight_kg", index="color", aggfunc=np.median)
```

```
weight_kg  
color  
Black      26.5  
Brown      24.0  
Gray       17.0  
Tan        2.0  
White     74.0
```

Multiple statistics

```
dogs.pivot_table(values="weight_kg", index="color", aggfunc=[np.mean, np.median])
```

| color | mean | median |
|-------|-----------|-----------|
| | weight_kg | weight_kg |
| Black | 26.5 | 26.5 |
| Brown | 24.0 | 24.0 |
| Gray | 17.0 | 17.0 |
| Tan | 2.0 | 2.0 |
| White | 74.0 | 74.0 |

Pivot on two variables

```
dogs.groupby(["color", "breed"])["weight_kg"].mean()
```

```
dogs.pivot_table(values="weight_kg", index="color", columns="breed")
```

| breed | Chihuahua | Chow Chow | Labrador | Poodle | Schnauzer | St. Bernard |
|-------|-----------|-----------|----------|--------|-----------|-------------|
| color | | | | | | |
| Black | NaN | NaN | 29.0 | 24.0 | NaN | NaN |
| Brown | NaN | 24.0 | 24.0 | NaN | NaN | NaN |
| Gray | NaN | NaN | NaN | NaN | 17.0 | NaN |
| Tan | 2.0 | NaN | NaN | NaN | NaN | NaN |
| White | NaN | NaN | NaN | NaN | NaN | 74.0 |

5. Pivot on two variables

You also previously computed the mean weight grouped by two variables: color and breed. We can also do this using the `pivot_table` method. To group by two variables, we can pass a second variable name into the `columns` argument. While the result looks a little different than what we had before, it contains the same numbers. There are NaNs, or missing values, because there are no black Chihuahuas or gray Labradors in our dataset, for example.

Filling missing values in pivot tables

```
dogs.pivot_table(values="weight_kg", index="color", columns="breed", fill_value=0)
```

| breed | Chihuahua | Chow Chow | Labrador | Poodle | Schnauzer | St. Bernard |
|-------|-----------|-----------|----------|--------|-----------|-------------|
| color | | | | | | |
| Black | 0 | 0 | 29 | 24 | 0 | 0 |
| Brown | 0 | 24 | 24 | 0 | 0 | 0 |
| Gray | 0 | 0 | 0 | 0 | 17 | 0 |
| Tan | 2 | 0 | 0 | 0 | 0 | 0 |
| White | 0 | 0 | 0 | 0 | 0 | 74 |

Summing with pivot tables

```
dogs.pivot_table(values="weight_kg", index="color", columns="breed",  
                  fill_value=0, margins=True)
```

| breed | Chihuahua | Chow Chow | Labrador | Poodle | Schnauzer | St. Bernard | All |
|-------|-----------|-----------|----------|--------|-----------|-------------|-----------|
| color | | | | | | | |
| Black | 0 | 0 | 29 | 24 | 0 | 0 | 26.500000 |
| Brown | 0 | 24 | 24 | 0 | 0 | 0 | 24.000000 |
| Gray | 0 | 0 | 0 | 0 | 17 | 0 | 17.000000 |
| Tan | 2 | 0 | 0 | 0 | 0 | 0 | 2.000000 |
| White | 0 | 0 | 0 | 0 | 0 | 74 | 74.000000 |
| All | 2 | 24 | 26 | 24 | 17 | 74 | 27.714286 |

If we set the margins argument to True, the last row and last column of the pivot table contain the mean of all the values in the column or row, not including the missing values that were filled in with Os.

Let's practice!

DATA MANIPULATION WITH PANDAS