Data Manipulation with Pandas

Recap

- Chapter 1
  - Subsetting and sorting
  - Adding new columns
- Chapter 2
  - Aggregating and grouping
  - Summary statistics
- Chapter 3
  - Indexing
  - Slicing
- Chapter 4
  - Visualizations
  - Reading and writing CSVs

Pandas is built on NumPy and Matplotlib

Exploring a DataFrame

```
dogs.head()
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	24	2013-07-01
1	Charlie	Poodle	Black	43	24	2016-09-16
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
3	Cooper	Schnauzer	Gray	49	17	2011-12-11
4	Max	Labrador	Black	59	29	2017-01-20

The first is head, which returns the first few rows of the DataFrame.

```
dogs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 6 columns):
name            7 non-null object
breed           7 non-null object
color           7 non-null object
height_cm       7 non-null int64
weight_kg       7 non-null int64
date_of_birth   7 non-null object
dtypes: int64(2), object(4)
```

The info method displays the names of columns, the data types they contain, and whether they have any missing values.

```
dogs.shape
```

```
(7, 6)
```

A DataFrame's shape attribute contains a tuple that holds the number of rows followed by the number of columns.

```
dogs.describe()
```

```
       height_cm  weight_kg
count   7.000000   7.000000
mean   49.714286  27.428571
std    17.960274  22.292429
min    18.000000   2.000000
25%    44.500000  19.500000
50%    49.000000  23.000000
75%    57.500000  27.000000
max    77.000000  74.000000
```

The describe method computes some summary statistics for numerical columns, like mean and median.

- `.head()` returns the first few rows (the "head" of the DataFrame).
- `.info()` shows information on each of the columns, such as the data type and number of missing values.
- `.shape` returns the number of rows and columns of the DataFrame.
- `.describe()` calculates a few summary statistics for each column.

- `.values` : A two-dimensional NumPy array of values.
- `.columns` : An index of columns: the column names.
- `.index` : An index for the rows: either row numbers or row names.

Sorting

```
dogs.sort_values("weight_kg")
```

	name	breed	color	height_cm	weight_kg	date_of_birth
5	Stella	Chihuahua	Tan	18	2	2015-04-20
3	Cooper	Schnauzer	Gray	49	17	2011-12-11
0	Bella	Labrador	Brown	56	24	2013-07-01
1	Charlie	Poodle	Black	43	24	2016-09-16
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
4	Max	Labrador	Black	59	29	2017-01-20
6	Bernie	St. Bernard	White	77	74	2018-02-27

```
dogs.sort_values(["weight_kg", "height_cm"], ascending=[True, False])
```

	name	breed	color	height_cm	weight_kg	date_of_birth
5	Stella	Chihuahua	Tan	18	2	2015-04-20
3	Cooper	Schnauzer	Gray	49	17	2011-12-11
0	Bella	Labrador	Brown	56	24	2013-07-01
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
1	Charlie	Poodle	Black	43	24	2016-09-16
4	Max	Labrador	Black	59	29	2017-01-20
6	Bernie	St. Bernard	White	77	74	2018-02-27

Sort on ...	Syntax
one column	`df.sort_values("breed")`
multiple columns	`df.sort_values(["breed", "weight_kg"])`

Subsetting

Subsetting rows

```
dogs[dogs["height_cm"] > 50]
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	24	2013-07-01
4	Max	Labrador	Black	59	29	2017-01-20
6	Bernie	St. Bernard	White	77	74	2018-02-27

Subsetting based on text data

```
dogs[dogs["breed"] == "Labrador"]
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	24	2013-07-01
4	Max	Labrador	Black	59	29	2017-01-20

Subsetting based on multiple conditions

```
is_lab = dogs["breed"] == "Labrador"
is_brown = dogs["color"] == "Brown"
dogs[is_lab & is_brown]
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	24	2013-07-01

```
dogs[ (dogs["breed"] == "Labrador") & (dogs["color"] == "Brown") ]
```

Subsetting using .isin()

```
is_black_or_brown = dogs["color"].isin(["Black", "Brown"])
dogs[is_black_or_brown]
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	24	2013-07-01
1	Charlie	Poodle	Black	43	24	2016-09-16
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
4	Max	Labrador	Black	59	29	2017-01-20

There are many ways to subset a DataFrame, perhaps the most common is to use relational operators to return `True` or `False` for each row, then pass that inside square brackets.

```
dogs[dogs["height_cm"] > 60]
dogs[dogs["color"] == "tan"]
```

You can filter for multiple conditions at once by using the "bitwise and" operator, `&`.

```
dogs[(dogs["height_cm"] > 60) & (dogs["color"] == "tan")]
```

Subsetting data based on a categorical variable often involves using the "or" operator (`|`) to select rows from multiple categories. This can get tedious when you want all states in one of three different regions, for example. Instead, use the `.isin()` method, which will allow you to tackle this problem by writing one condition instead of three separate ones.

```
colors = ["brown", "black", "tan"]
condition = dogs["color"].isin(colors)
dogs[condition]
```

Adding a new column

```
dogs["height_m"] = dogs["height_cm"] / 100
```

Summary statistics

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

```
49.714285714285715
```

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

```
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
```

.agg( ) allows you to compute summary statistics

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.

```
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
```

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

Counting

Dropping duplicate names

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

Dropping duplicate pairs

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

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
```

```
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
```

Proportions

```
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
```

Grouped Summary Statistics

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
```

Multiple grouped summaries

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

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

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
```

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
```

Pivot tables

- Pivot tables are the standard way of aggregating data in spreadsheets. In pandas, pivot tables are essentially just another way of performing grouped calculations. That is, the .pivot_table() method is just an alternative to .groupby().

Group by to pivot table

```
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
```

Multiple statistics

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

```
          mean    median
     weight_kg weight_kg
color
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
```

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

Contain the mean of all the values in the column or row, not including the missing values that we're filled in with 0s.

Marin=true will give you a summary statistics of your dataset.

Explicit indexes

Original dataset:

The dog dataset, revisited

```
print(dogs)
```

	name	breed	color	height_cm	weight_kg
0	Bella	Labrador	Brown	56	25
1	Charlie	Poodle	Black	43	23
2	Lucy	Chow Chow	Brown	46	22
3	Cooper	Schnauzer	Gray	49	17
4	Max	Labrador	Black	59	29
5	Stella	Chihuahua	Tan	18	2
6	Bernie	St. Bernard	White	77	74

Explicit index functions:

Setting a column as the index

```
dogs_ind = dogs.set_index("name")
print(dogs_ind)
```

name	breed	color	height_cm	weight_kg
Bella	Labrador	Brown	56	25
Charlie	Poodle	Black	43	23
Lucy	Chow Chow	Brown	46	22
Cooper	Schnauzer	Grey	49	17
Max	Labrador	Black	59	29
Stella	Chihuahua	Tan	18	2
Bernie	St. Bernard	White	77	74

Removing an index

```
dogs_ind.reset_index()
```

	name	breed	color	height_cm	weight_kg
0	Bella	Labrador	Brown	56	25
1	Charlie	Poodle	Black	43	23
2	Lucy	Chow Chow	Brown	46	22
3	Cooper	Schnauzer	Grey	49	17
4	Max	Labrador	Black	59	29
5	Stella	Chihuahua	Tan	18	2
6	Bernie	St. Bernard	White	77	74

Dropping an index

```
dogs_ind.reset_index(drop=True)
```

	breed	color	height_cm	weight_kg
0	Labrador	Brown	56	25
1	Poodle	Black	43	23
2	Chow Chow	Brown	46	22
3	Schnauzer	Grey	49	17
4	Labrador	Black	59	29
5	Chihuahua	Tan	18	2
6	St. Bernard	White	77	74

Indexes make subsetting simpler

```
dogs[dogs["name"].isin(["Bella", "Stella"])]
```

	name	breed	color	height_cm	weight_kg
0	Bella	Labrador	Brown	56	25
5	Stella	Chihuahua	Tan	18	2

```
dogs_ind.loc[["Bella", "Stella"]]
```

name	breed	color	height_cm	weight_kg
Bella	Labrador	Brown	56	25
Stella	Chihuahua	Tan	18	2

Index values don't need to be unique

```
dogs_ind2 = dogs.set_index("breed")
print(dogs_ind2)
```

breed	name	color	height_cm	weight_kg
Labrador	Bella	Brown	56	25
Poodle	Charlie	Black	43	23
Chow Chow	Lucy	Brown	46	22
Schnauzer	Cooper	Grey	49	17
Labrador	Max	Black	59	29
Chihuahua	Stella	Tan	18	2
St. Bernard	Bernie	White	77	74

More than one Labrador in the index

Subsetting on duplicated index values

```
dogs_ind2.loc["Labrador"]
```

breed	name	color	height_cm	weight_kg
Labrador	Bella	Brown	56	25
Labrador	Max	Black	59	29

Multi-level indexes a.k.a. hierarchical indexes

```
dogs_ind3 = dogs.set_index(["breed", "color"])
print(dogs_ind3)
```

breed	color	name	height_cm	weight_kg
Labrador	Brown	Bella	56	25
Poodle	Black	Charlie	43	23
Chow Chow	Brown	Lucy	46	22
Schnauzer	Grey	Cooper	49	17
Labrador	Black	Max	59	29
Chihuahua	Tan	Stella	18	2
St. Bernard	White	Bernie	77	74

Subset the outer level with a list

```
dogs_ind3.loc[["Labrador", "Chihuahua"]]
```

breed	color	name	height_cm	weight_kg
Labrador	Brown	Bella	56	25
	Black	Max	59	29
Chihuahua	Tan	Stella	18	2

Subset inner levels with a list of tuples

```
dogs_ind3.loc[[("Labrador", "Brown"), ("Chihuahua", "Tan")]]
```

breed	color	name	height_cm	weight_kg
Labrador	Brown	Bella	56	25
Chihuahua	Tan	Stella	18	2

Sorting by index values

```
dogs_ind3.sort_index()
```

breed	color	name	height_cm	weight_kg
Chihuahua	Tan	Stella	18	2
Chow Chow	Brown	Lucy	46	22
Labrador	Black	Max	59	29
	Brown	Bella	56	25
Poodle	Black	Charlie	43	23
Schnauzer	Grey	Cooper	49	17
St. Bernard	White	Bernie	77	74

By default, it sorts all index levels from outer to inner, in ascending order.

Controlling sort_index

```
dogs_ind3.sort_index(level=["color", "breed"], ascending=[True, False])
```

breed	color	name	height_cm	weight_kg
Poodle	Black	Charlie	43	23
Labrador	Black	Max	59	29
	Brown	Bella	56	25
Chow Chow	Brown	Lucy	46	22
Schanuzer	Grey	Cooper	49	17
Chihuahua	Tan	Stella	18	2
St. Bernard	White	Bernie	77	74

You can control the sorting by passing lists to the level and ascending arguments.

Slicing and subsetting with .loc and .iloc

Slicing lists

```
breeds = ["Labrador", "Poodle",
          "Chow Chow", "Schnauzer",
          "Labrador", "Chihuahua",
          "St. Bernard"]
```

```
['Labrador',
 'Poodle',
 'Chow Chow',
 'Schnauzer',
 'Labrador',
 'Chihuahua',
 'St. Bernard']
```

```
breeds[2:5]
```

```
['Chow Chow', 'Schnauzer', 'Labrador']
```

```
breeds[:3]
```

```
['Labrador', 'Poodle', 'Chow Chow']
```

```
breeds[:]
```

```
['Labrador','Poodle','Chow Chow','Schnauzer',
 'Labrador','Chihuahua','St. Bernard']
```

Sort the index before you slice

```
dogs_srt = dogs.set_index(["breed", "color"]).sort_index()
print(dogs_srt)
```

breed	color	name	height_cm	weight_kg
Chihuahua	Tan	Stella	18	2
Chow Chow	Brown	Lucy	46	22
Labrador	Black	Max	59	29
	Brown	Bella	56	25
Poodle	Black	Charlie	43	23
Schnauzer	Grey	Cooper	49	17
St. Bernard	White	Bernie	77	74

Slicing the outer index level

```
dogs_srt.loc["Chow Chow":"Poodle"]
```

breed	color	name	height_cm	weight_kg
Chow Chow	Brown	Lucy	46	22
Labrador	Black	Max	59	29
	Brown	Bella	56	25
Poodle	Black	Charlie	43	23

The final value `"Poodle"` is included

Full dataset

breed	color	name	height_cm	weight_kg
Chihuahua	Tan	Stella	18	2
Chow Chow	Brown	Lucy	46	22
Labrador	Black	Max	59	29
	Brown	Bella	56	25
Poodle	Black	Charlie	43	23
Schnauzer	Grey	Cooper	49	17
St. Bernard	White	Bernie	77	74

Slicing the inner index levels badly

```
dogs_srt.loc["Tan":"Grey"]
```

```
Empty DataFrame
Columns: [name, height_cm, weight_kg]
Index: []
```

Full dataset

breed	color	name	height_cm	weight_kg
Chihuahua	Tan	Stella	18	2
Chow Chow	Brown	Lucy	46	22
Labrador	Black	Max	59	29
	Brown	Bella	56	25
Poodle	Black	Charlie	43	23
Schnauzer	Grey	Cooper	49	17
St. Bernard	White	Bernie	77	74

Slicing the inner index levels correctly

```
dogs_srt.loc[
    ("Labrador", "Brown"):("Schnauzer", "Grey")]
```

breed	color	name	height_cm	weight_kg
Labrador	Brown	Bella	56	25
Poodle	Black	Charlie	43	23
Schnauzer	Grey	Cooper	49	17

Full dataset

breed	color	name	height_cm	weight_kg
Chihuahua	Tan	Stella	18	2
Chow Chow	Brown	Lucy	46	22
Labrador	Black	Max	59	29
	Brown	Bella	56	25
Poodle	Black	Charlie	43	23
Schnauzer	Grey	Cooper	49	17
St. Bernard	White	Bernie	77	74

Slicing columns

```
dogs_srt.loc[:, "name":"height_cm"]
```

breed	color	name	height_cm
Chihuahua	Tan	Stella	18
Chow Chow	Brown	Lucy	46
Labrador	Black	Max	59
	Brown	Bella	56
Poodle	Black	Charlie	43
Schnauzer	Grey	Cooper	49
St. Bernard	White	Bernie	77

Full dataset

breed	color	name	height_cm	weight_kg
Chihuahua	Tan	Stella	18	2
Chow Chow	Brown	Lucy	46	22
Labrador	Black	Max	59	29
	Brown	Bella	56	25
Poodle	Black	Charlie	43	23
Schnauzer	Grey	Cooper	49	17
St. Bernard	White	Bernie	77	74

Slice twice

```
dogs_srt.loc[
    ("Labrador", "Brown"):("Schnauzer", "Grey"),
    "name":"height_cm"]
```

breed	color	name	height_cm
Labrador	Brown	Bella	56
Poodle	Black	Charlie	43
Schanuzer	Grey	Cooper	49

Full dataset

breed	color	name	height_cm	weight_kg
Chihuahua	Tan	Stella	18	2
Chow Chow	Brown	Lucy	46	22
Labrador	Black	Max	59	29
	Brown	Bella	56	25
Poodle	Black	Charlie	43	23
Schnauzer	Grey	Cooper	49	17
St. Bernard	White	Bernie	77	74

Working with Pivot tables

A bigger dog dataset

```
print(dog_pack)
```

	breed	color	height_cm	weight_kg
0	Boxer	Brown	62.64	30.4
1	Poodle	Black	46.41	20.4
2	Beagle	Brown	36.39	12.4
3	Chihuahua	Tan	19.70	1.6
4	Labrador	Tan	54.44	36.1
..	...	...	...	...
87	Boxer	Gray	58.13	29.9
88	St. Bernard	White	70.13	69.4
89	Poodle	Gray	51.30	20.4
90	Beagle	White	38.81	8.8
91	Beagle	Black	33.40	13.5

Pivoting the dog pack

```
dogs_height_by_breed_vs_color = dog_pack.pivot_table(
        "height_cm", index="breed", columns="color")
print(dogs_height_by_breed_vs_color)
```

color	Black	Brown	Gray	Tan	White
breed					
Beagle	34.500000	36.4500	36.313333	35.740000	38.810000
Boxer	57.203333	62.6400	58.280000	62.310000	56.360000
Chihuahua	18.555000	NaN	21.660000	20.096667	17.933333
Chow Chow	51.262500	50.4800	NaN	53.497500	54.413333
Dachshund	21.186667	19.7250	NaN	19.375000	20.660000
Labrador	57.125000	NaN	NaN	55.190000	55.310000
Poodle	48.036000	57.1300	56.645000	NaN	44.740000
St. Bernard	63.920000	65.8825	67.640000	68.334000	67.495000

.loc[] + slicing is a power combo

```
dogs_height_by_breed_vs_color.loc["Chow Chow":"Poodle"]
```

color	Black	Brown	Gray	Tan	White
breed					
Chow Chow	51.262500	50.480	NaN	53.4975	54.413333
Dachshund	21.186667	19.725	NaN	19.3750	20.660000
Labrador	57.125000	NaN	NaN	55.1900	55.310000
Poodle	48.036000	57.130	56.645	NaN	44.740000

In particular, the loc and slicing combination is ideal for subsetting pivot tables, like so.

The axis argument

```
dogs_height_by_breed_vs_color.mean(axis="index")
```

```
color
Black    43.973563
Brown    48.717917
Gray     48.107667
Tan      44.934738
White    44.465208
dtype: float64
```

The default value is "index," which means "calculate the statistic across rows."

Calculating summary stats across columns

```
dogs_height_by_breed_vs_color.mean(axis="columns")
```

```
breed
Beagle          36.362667
Boxer           59.358667
Chihuahua       19.561250
Chow Chow       52.413333
Dachshund       20.236667
Labrador        55.875000
Poodle          51.637750
St. Bernard     66.654300
dtype: float64
```

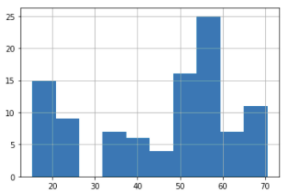
To calculate a summary statistic for each row, that is, "across the columns," you set axis to "columns."

Creating and Visualizing DataFrames

Visualizing your data

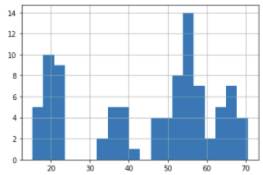
Histograms

```
import matplotlib.pyplot as plt

dog_pack["height_cm"].hist()

plt.show()
```

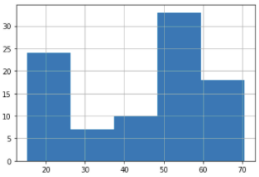


Histograms

```
dog_pack["height_cm"].hist(bins=20)
plt.show()
```



```
dog_pack["height_cm"].hist(bins=5)
plt.show()
```



Bar plots

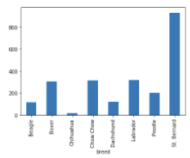
```
avg_weight_by_breed = dog_pack.groupby("breed")["weight_kg"].mean()
print(avg_weight_by_breed)
```

```
breed
Beagle          10.636364
Boxer           30.620000
Chihuahua        1.491667
Chow Chow       22.535714
Dachshund        9.975000
Labrador        31.850000
Poodle          20.400000
St. Bernard     71.576923
Name: weight_kg, dtype: float64
```

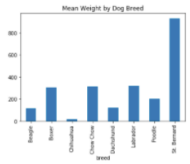
Bar plots can reveal relationships between a categorical variable and a numeric variable, like breed and weight.

Bar plots

```
avg_weight_by_breed.plot(kind="bar")
plt.show()
```



```
avg_weight_by_breed.plot(kind="bar",
    title="Mean Weight by Dog Breed")
plt.show()
```

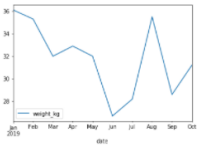


Line plots

```
sully.head()
```

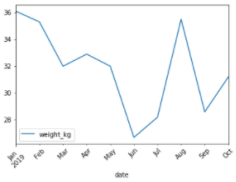
	date	weight_kg
0	2019-01-31	36.1
1	2019-02-28	35.3
2	2019-03-31	32.0
3	2019-04-30	32.9
4	2019-05-31	32.0

```
sully.plot(x="date",
           y="weight_kg",
           kind="line")
plt.show()
```



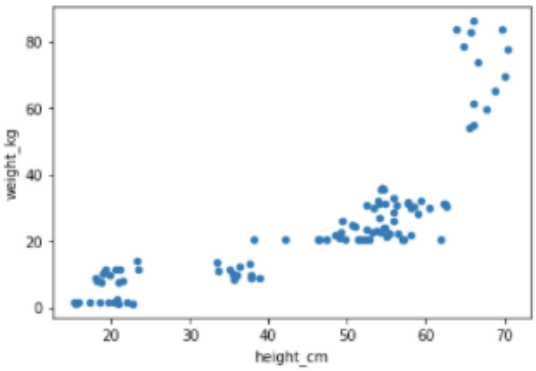
Rotating axis labels

```
sully.plot(x="date", y="weight_kg", kind="line", rot=45)
plt.show()
```



Scatter plots

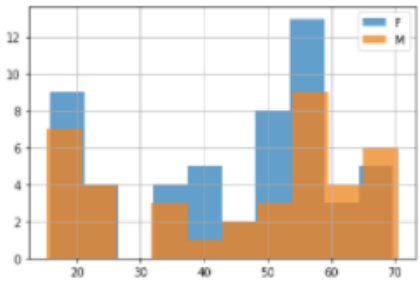
```
dog_pack.plot(x="height_cm", y="weight_kg", kind="scatter")
plt.show()
```



Scatter plots are great for visualizing relationships between two numeric variables.

Transparency

```
dog_pack[dog_pack["sex"]=="F"]["height_cm"].hist(alpha=0.7)
dog_pack[dog_pack["sex"]=="M"]["height_cm"].hist(alpha=0.7)
plt.legend(["F", "M"])
plt.show()
```



0 means completely transparent that is, invisible, and 1 means completely opaque.

Missing values

In a pandas DataFrame, missing values are indicated with N-a-N, which stands for "not a number."

When you first get a DataFrame, it's a good idea to get a sense of whether it contains any missing values, and if so, how many. That's where the isna method comes in. When we call isna on a DataFrame, we get a Boolean for every single value indicating whether the value is missing or not, but this isn't very helpful when you're working with a lot of data.

Detecting missing values

```
dogs.isna()
```

```
    name  breed  color  height_cm  weight_kg  date_of_birth
0  False  False  False      False       True          False
1  False  False  False      False      False          False
2  False  False  False      False      False          False
3  False  False  False      False       True          False
4  False  False  False      False      False          False
5  False  False  False      False      False          False
6  False  False  False      False      False          False
```

Detecting any missing values

```
dogs.isna().any()
```

```
name             False
breed            False
color            False
height_cm        False
weight_kg         True
date_of_birth    False
dtype: bool
```

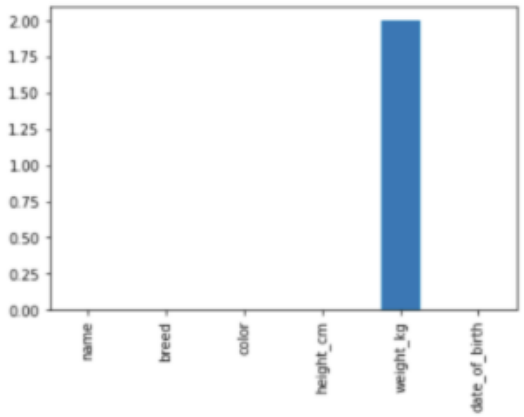
Counting missing values

```
dogs.isna().sum()
```

```
name            0
breed           0
color           0
height_cm       0
weight_kg       2
date_of_birth   0
dtype: int64
```

Plotting missing values

```
import matplotlib.pyplot as plt
dogs.isna().sum().plot(kind="bar")
plt.show()
```



We can use those counts to visualize the missing values in the dataset using a bar plot.

Removing missing values

```
dogs.dropna()
```

	name	breed	color	height_cm	weight_kg	date_of_birth
1	Charlie	Poodle	Black	43	24.0	2016-09-16
2	Lucy	Chow Chow	Brown	46	24.0	2014-08-25
4	Max	Labrador	Black	59	29.0	2017-01-20
5	Stella	Chihuahua	Tan	18	2.0	2015-04-20
6	Bernie	St. Bernard	White	77	74.0	2018-02-27

One option is to remove the rows in the DataFrame that contain missing values.

Replacing missing values

```
dogs.fillna(0)
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	0.0	2013-07-01
1	Charlie	Poodle	Black	43	24.0	2016-09-16
2	Lucy	Chow Chow	Brown	46	24.0	2014-08-25
3	Cooper	Schnauzer	Gray	49	0.0	2011-12-11
4	Max	Labrador	Black	59	29.0	2017-01-20
5	Stella	Chihuahua	Tan	18	2.0	2015-04-20
6	Bernie	St. Bernard	White	77	74.0	2018-02-27

Another option is to replace missing values with another value.

Creating DataFrames

Dictionaries

```
my_dict = {
    "key1": value1,
    "key2": value2,
    "key3": value3
}
```

```
my_dict["key1"]
```

```
value1
```

```
my_dict = {
    "title": "Charlotte's Web",
    "author": "E.B. White",
    "published": 1952
}
```

```
my_dict["title"]
```

```
Charlotte's Web
```

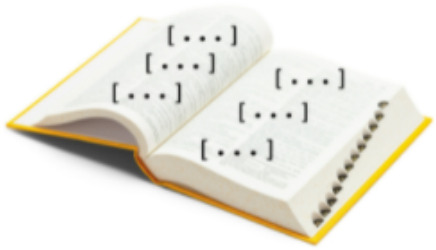
Creating DataFrames

From a list of dictionaries

- Constructed row by row

From a dictionary of lists

- Constructed column by column



while in the second method, the DataFrame is built up column by column.

List of dictionaries - by row

name	breed	height (cm)	weight (kg)	date of birth
Ginger	Dachshund	22	10	2019-03-14
Scout	Dalmatian	59	25	2019-05-09

```
list_of_dicts = [
    {"name": "Ginger", "breed": "Dachshund", "height_cm": 22,
    "weight_kg": 10, "date_of_birth": "2019-03-14"},
    {"name": "Scout", "breed": "Dalmatian", "height_cm": 59,
    "weight_kg": 25, "date_of_birth": "2019-05-09"}
]
```

List of dictionaries - by row

name	breed	height (cm)	weight (kg)	date of birth
Ginger	Dachshund	22	10	2019-03-14
Scout	Dalmatian	59	25	2019-05-09

```
new_dogs = pd.DataFrame(list_of_dicts)
print(new_dogs)
```

```
    name      breed  height_cm  weight_kg date_of_birth
0  Ginger  Dachshund         22         10    2019-03-14
1   Scout  Dalmatian         59         25    2019-05-09
```

Dictionary of lists - by column

name	breed	height	weight	date of birth
Ginger	Dachshund	22	10	2019-03-14
Scout	Dalmatian	59	25	2019-05-09

- **Key** = column name
- **Value** = list of column values

```
dict_of_lists = {
    "name": ["Ginger", "Scout"],
    "breed": ["Dachshund", "Dalmatian"],
    "height_cm": [22, 59],
    "weight_kg": [10, 25],
    "date_of_birth": ["2019-03-14",
    "2019-05-09"]
}
new_dogs = pd.DataFrame(dict_of_lists)
```

Dictionary of lists - by column

name	breed	height (cm)	weight (kg)	date of birth
Ginger	Dachshund	22	10	2019-03-14
Scout	Dalmatian	59	25	2019-05-09

```
print(new_dogs)
```

```
    name      breed  height_cm  weight_kg date_of_birth
0  Ginger  Dachshund         22         10    2019-03-14
1   Scout  Dalmatian         59         25    2019-05-09
```

If we print the new DataFrame, we can see that it's exactly what we wanted.

Reading and writing CSVs

What's a CSV file?

- CSV = comma-separated values
- Designed for DataFrame-like data
- Most database and spreadsheet programs can use them or create them

CSV to DataFrame

```
import pandas as pd
new_dogs = pd.read_csv("new_dogs.csv")
print(new_dogs)
```

```
     name      breed  height_cm  weight_kg date_of_birth
0  Ginger  Dachshund         22         10    2019-03-14
1   Scout  Dalmatian         59         25    2019-05-09
```

DataFrame manipulation

```
new_dogs["bmi"] = new_dogs["weight_kg"] / (new_dogs["height_cm"] / 100) ** 2
print(new_dogs)
```

```
     name      breed  height_cm  weight_kg date_of_birth         bmi
0  Ginger  Dachshund         22         10    2019-03-14  206.611570
1   Scout  Dalmatian         59         25    2019-05-09   71.818443
```

DataFrame to CSV

```
new_dogs.to_csv("new_dogs_with_bmi.csv")
```

new_dogs_with_bmi.csv

```
name,breed,height_cm,weight_kg,d_o_b,bmi
Ginger,Dachshund,22,10,2019-03-14,206.611570
Scout,Dalmatian,59,25,2019-05-09,71.818443
```