DATA MANIPULATION WITH PANDAS

DATACAMP

Kolawole David

PANDAS

pandas is built on NumPy and Matplotlib

Introducing DataFrames

pandas DataFrames

```
print(dogs)
```

```
      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
5   Stella    Chihuahua    Tan         18          2    2015-04-20
6   Bernie  St. Bernard  White         77         74    2018-02-27
```

Introducing DataFrames

Exploring a DataFrame: .head()

```
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
```

Exploring a DataFrame: .info()

```
dogs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
 --  ------         --------------  -----
 0   name           7 non-null      object
 1   breed          7 non-null      object
 2   color          7 non-null      object
 3   height_cm      7 non-null      int64
 4   weight_kg      7 non-null      int64
 5   date_of_birth  7 non-null      object
dtypes: int64(2), object(4)
memory usage: 464.0+ bytes
```

Introducing DataFrames

Exploring a DataFrame: .describe()

```
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
```

PANDAS

Components of a DataFrame: .values

```
dogs.values
```

```
array([['Bella', 'Labrador', 'Brown', 56, 24, '2013-07-01'],
       ['Charlie', 'Poodle', 'Black', 43, 24, '2016-09-16'],
       ['Lucy', 'Chow Chow', 'Brown', 46, 24, '2014-08-25'],
       ['Cooper', 'Schnauzer', 'Gray', 49, 17, '2011-12-11'],
       ['Max', 'Labrador', 'Black', 59, 29, '2017-01-20'],
       ['Stella', 'Chihuahua', 'Tan', 18, 2, '2015-04-20'],
       ['Bernie', 'St. Bernard', 'White', 77, 74, '2018-02-27']],
      dtype=object)
```

Introducing DataFrames

Components of a DataFrame: .columns and .index

```
dogs.columns
```

```
Index(['name', 'breed', 'color', 'height_cm', 'weight_kg', 'date_of_birth'],
dtype='object')
```

```
dogs.index
```

```
RangeIndex(start=0, stop=7, step=1)
```

Introducing DataFrames

pandas Philosophy

> There should be one -- and preferably only one -- obvious way to do it.

- *The Zen of Python* by Tim Peters, Item 13

Learn / Courses / **Data Manipulation with pandas**

Course Outline

Exercise

Inspecting a DataFrame

When you get a new DataFrame to work with, the first thing you need to do is explore it and see what it contains. There are several useful methods and attributes for this.

- `.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.

`homelessness` is a DataFrame containing estimates of homelessness in each U.S. state in 2018. The `individual` column is the number of homeless individuals not part of a family with children. The `family_members` column is the number of homeless individuals part of a family with children. The `state_pop` column is the state's total population.

script.py

```
# Print the head of the homelessness data
print(homelessness.head())

# Print information about homelessness
print(homelessness.info())

# Print the shape of homelessness
print(homelessness.shape)

# Print a description of homelessness
print(homelessness.describe())
```

PANDAS

Exercise

Parts of a DataFrame

To better understand DataFrame objects, it's useful to know that they consist of three components, stored as attributes:

- `.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.

You can usually think of indexes as a list of strings or numbers, though the pandas `Index` data type allows for more sophisticated options. (These will be covered later in the course.)

`homelessness` is available.

script.py

```python
# Import pandas using the alias pd
import pandas as pd

# Print the values of homelessness
print(homelessness.values)

# Print the column index of homelessness
print(homelessness.columns)

# Print the row index of homelessness
print(homelessness.index)
```

PANDAS : SORTING

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

Sorting in descending order

```
dogs.sort_values("weight_kg", ascending=False)
```

	name	breed	color	height_cm	weight_kg	date_of_birth
6	Bernie	St. Bernard	White	77	74	2018-02-27
4	Max	Labrador	Black	59	29	2017-01-20
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
5	Stella	Chihuahua	Tan	18	2	2015-04-20

Sorting and subsetting

Sorting by multiple variables

```
dogs.sort_values(["weight_kg", "height_cm"])
```

	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
1	Charlie	Poodle	Black	43	24	2016-09-16
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
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

Sorting by multiple variables

```
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

Subsetting columns

```
dogs["name"]
```

```
0      Bella
1    Charlie
2       Lucy
3     Cooper
4        Max
5     Stella
6     Bernie
Name: name, dtype: object
```

Subsetting multiple columns

```
dogs[["breed", "height_cm"]]
```

	breed	height_cm
0	Labrador	56
1	Poodle	43
2	Chow Chow	46
3	Schnauzer	49
4	Labrador	59
5	Chihuahua	18
6	St. Bernard	77

PANDAS: SUBSETTING

Subsetting rows

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

```
0     True
1    False
2    False
3    False
4     True
5    False
6     True
Name: height_cm, dtype: bool
```

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 dates

```
dogs[dogs["date_of_birth"] < "2015-01-01"]
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	24	2013-07-01
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
3	Cooper	Schnauzer	Gray	49	17	2011-12-11

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

PANDAS

New columns 50 XP

Adding a new column

```
dogs["height_m"] = dogs["height_cm"] / 100
print(dogs)
```

```
     name        breed  color  height_cm  weight_kg date_of_birth  height_m
0   Bella     Labrador  Brown         56         24    2013-07-01      0.56
1 Charlie       Poodle  Black         43         24    2016-09-16      0.43
2    Lucy    Chow Chow  Brown         46         24    2014-08-25      0.46
3  Cooper    Schnauzer   Gray         49         17    2011-12-11      0.49
4     Max     Labrador  Black         59         29    2017-01-20      0.59
5  Stella    Chihuahua    Tan         18          2    2015-04-20      0.18
6  Bernie  St. Bernard  White         77         74    2018-02-27      0.77
```

New columns 50 XP

Doggy mass index

$$\text{BMI} = \text{weight in kg}/(\text{height in m})^2$$

```
dogs["bmi"] = dogs["weight_kg"] / dogs["height_m"] ** 2
print(dogs.head())
```

```
     name      breed  color  height_cm  weight_kg date_of_birth  height_m         bmi
0   Bella   Labrador  Brown         56         24    2013-07-01      0.56   76.530612
1 Charlie     Poodle  Black         43         24    2016-09-16      0.43  129.799892
2    Lucy  Chow Chow  Brown         46         24    2014-08-25      0.46  113.421550
                      Gray         49         17    2011-12-11      0.49   70.803832
                      Black        59         29    2017-01-20      0.59   83.309394
```

New columns

Multiple manipulations

```
bmi_lt_100 = dogs[dogs["bmi"] < 100]
bmi_lt_100_height  = bmi_lt_100.sort_values("height_cm", ascending=False)
bmi_lt_100_height[["name", "height_cm", "bmi"]]
```

```
     name  height_cm        bmi
4     Max         59  83.309394
0   Bella         56  76.530612
3  Cooper         49  70.803832
5  Stella         18  61.728395
```

Exercise

Sorting rows

Finding interesting bits of data in a DataFrame is often easier if you change the order of the rows. You can sort the rows by passing a column name to `.sort_values()`.

In cases where rows have the same value (this is common if you sort on a categorical variable), you may wish to break the ties by sorting on another column. You can sort on multiple columns in this way by passing a list of column names.

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

script.py Light Mode

```
# Sort homelessness by region, then descending family members
homelessness_reg_fam = homelessness.sort_values(['region','family_members'],
ascending=[True,False])

# Print the top few rows
print(homelessness_reg_fam.head())
```

Run Code Submit Answer

PANDAS

Exercise

Combo-attack!

You've seen the four most common types of data manipulation: sorting rows, subsetting columns, subsetting rows, and adding new columns. In a real-life data analysis, you can mix and match these four manipulations to answer a multitude of questions.

In this exercise, you'll answer the question, "Which state has the highest number of homeless individuals per 10,000 people in the state?" Combine your new `pandas` skills to find out.

Instructions

100 XP

- Add a column to `homelessness`, `indiv_per_10k`, containing the number of homeless individuals per ten thousand people in each state.
- Subset rows where `indiv_per_10k` is higher than `20`, assigning to `high_homelessness`.
- Sort `high_homelessness` by descending `indiv_per_10k`, assigning to `high_homelessness_srt`.
- Select only the `state` and `indiv_per_10k` columns of `high_homelessness_srt` and save as `result`. *Look at the `result`.*

Take Hint (-30 XP)

script.py

Light Mode

```python
# Create indiv_per_10k col as homeless individuals per 10k state pop
homelessness["indiv_per_10k"] = 10000 * homelessness['individuals'] / homelessness['state_pop']

# Subset rows for indiv_per_10k greater than 20
high_homelessness = homelessness[homelessness['indiv_per_10k']>20]

# Sort high_homelessness by descending indiv_per_10k
high_homelessness_srt = high_homelessness.sort_values('indiv_per_10k', ascending=False)

# From high_homelessness_srt, select the state and indiv_per_10k cols
result = high_homelessness_srt[['state', 'indiv_per_10k']]

# See the result
print(result)
```

Run Code

Submit Answer

IPython Shell

Slides

```
                 state  indiv_per_10k
8   District of Columbia         53.738
11               Hawaii         29.079
4             California         27.624
37               Oregon         26.636
28               Nevada         23.314
47           Washington         21.829
32             New York         20.392
```

PANDAS

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)
```

```
22.599999999999998
```

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
```

PANDAS : COUNTING & SUBSETTING FOR CATEGORICAL VARIABLES

Counting

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
```

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
```

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
```

PANDAS

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

50 XP

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
```

Different statistics

```
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
```

Pivot tables

50 XP

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

The mean is calculated by default

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

The all column represent mean of the records for each row and column

PANDAS

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
```

Setting a column as the index

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

```
                breed  color  height_cm  weight_kg
name
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
```

PANDAS :after converting names to index the 2nd snapshot appears easier

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"]]
```

```
            breed  color  height_cm  weight_kg
name
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)
```

```
                name  color  height_cm  weight_kg
breed
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
```

Subsetting on duplicated index values

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

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

PANDAS

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

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

```
                     name  height_cm  weight_kg
breed       color
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"]]
```

```
                   name  height_cm  weight_kg
breed     color
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")]]
```

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

PANDAS

Sorting by index values

```
dogs_ind3.sort_index()
```

```
                     name  height_cm  we
breed       color
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
```

Explicit Indexes

Controlling sort_index

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

```
                     name  height_cm  weight_kg
breed       color
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
```

Now you have two problems

- Index values are just data
- Indexes violate "tidy data" principles
- You need to learn two syntaxes

PANDAS

Learn / Courses / **Data Manipulation with pandas**

Exercise

Setting and removing indexes

pandas allows you to designate columns as an *index*. This enables cleaner code when taking subsets (as well as providing more efficient lookup under some circumstances).

In this chapter, you'll be exploring `temperatures`, a DataFrame of average temperatures in cities around the world. `pandas` is loaded as `pd`.

Instructions 0 XP

- *Look at* `temperatures`.
- Set the index of `temperatures` to `"city"`, assigning to `temperatures_ind`.
- *Look at* `temperatures_ind`. *How is it different from* `temperatures`*?*
- Reset the index of `temperatures_ind`, keeping its contents.
- Reset the index of `temperatures_ind`, dropping its contents.

```
# Look at temperatures
print(temperatures)

# Set the index of temperatures to city
temperatures_ind = temperatures.set_index("city")

# Look at temperatures_ind
print(temperatures_ind)

# Reset the temperatures_ind index, keeping its contents
print(temperatures_ind.reset_index())

# Reset the temperatures_ind index, dropping its contents
print(temperatures_ind.reset_index(drop=True)
```

PANDAS

Exercise

Subsetting with .loc[]

The killer feature for indexes is `.loc[]`: a subsetting method that accepts index values. When you pass it a single argument, it will take a subset of rows.

The code for subsetting using `.loc[]` can be easier to read than standard square bracket subsetting, which can make your code less burdensome to maintain.

`pandas` is loaded as `pd`. `temperatures` and `temperatures_ind` are available; the latter is indexed by `city`.

Instructions 100 XP

- Create a list called `cities` that contains "Moscow" and "Saint Petersburg".
- Use `[]` subsetting to filter `temperatures` for rows where the `city` column takes a value in the `cities` list.
- Use `.loc[]` subsetting to filter `temperatures_ind` for rows where the city is in the `cities` list.

```
# Make a list of cities to subset on
cities = ["Moscow","Saint Petersburg"]

# Subset temperatures using square brackets
print(temperatures[temperatures["city"].isin(cities)])

# Subset temperatures_ind using .loc[]
print(temperatures_ind.loc[["Moscow","Saint Petersburg"]])
```

PANDAS

Exercise

Subsetting with .loc[]

The killer feature for indexes is `.loc[]`: a subsetting method that accepts index values. When you pass it a single argument, it will take a subset of rows.

The code for subsetting using `.loc[]` can be easier to read than standard square bracket subsetting, which can make your code less burdensome to maintain.

`pandas` is loaded as `pd`. `temperatures` and `temperatures_ind` are available; the latter is indexed by `city`.

Instructions 100 XP

- Create a list called `cities` that contains "Moscow" and "Saint Petersburg".
- Use `[]` subsetting to filter `temperatures` for rows where the `city` column takes a value in the `cities` list.
- Use `.loc[]` subsetting to filter `temperatures_ind` for rows where the city is in the `cities` list.

```
# Make a list of cities to subset on
cities = ["Moscow","Saint Petersburg"]

# Subset temperatures using square brackets
print(temperatures[temperatures["city"].isin(cities)])

# Subset temperatures_ind using .loc[]
print(temperatures_ind.loc[["Moscow","Saint Petersburg"]])
```

PANDAS

Exercise

Setting multi-level indexes

Indexes can also be made out of multiple columns, forming a *multi-level index* (sometimes called a *hierarchical index*). There is a trade-off to using these.

The benefit is that multi-level indexes make it more natural to reason about nested categorical variables. For example, in a clinical trial, you might have control and treatment groups. Then each test subject belongs to one or another group, and we can say that a test subject is nested inside the treatment group. Similarly, in the temperature dataset, the city is located in the country, so we can say a city is nested inside the country.

The main downside is that the code for manipulating indexes is different from the code for manipulating columns, so you have to learn two syntaxes and keep track of how your data is represented.

`pandas` is loaded as `pd`. `temperatures` is available.

Instructions 100 XP

- Set the index of `temperatures` to the `"country"` and `"city"` columns, and assign this to `temperatures_ind`.
- Specify two country/city pairs to keep: `"Brazil"` / `"Rio De Janeiro"` and `"Pakistan"` / `"Lahore"`, assigning to `rows_to_keep`.
- Print and subset `temperatures_ind` for `rows_to_keep` using `.loc[]`.

Take Hint (-30 XP)

```
# Index temperatures by country & city
temperatures_ind = temperatures.set_index(["country","city"])

# List of tuples: Brazil, Rio De Janeiro & Pakistan, Lahore
rows_to_keep = [("Brazil","Rio De Janeiro"),("Pakistan","Lahore")]

# Subset for rows to keep
print(temperatures_ind.loc[rows_to_keep])
```

PANDAS

Sorting by index values

Previously, you changed the order of the rows in a DataFrame by calling `.sort_values()`. It's also useful to be able to sort by elements in the index. For this, you need to use `.sort_index()`.

`pandas` is loaded as `pd`. `temperatures_ind` has a multi-level index of `country` and `city`, and is available.

Instructions

70 XP

- Sort `temperatures_ind` by the index values.
- Sort `temperatures_ind` by the index values at the `"city"` level.
- Sort `temperatures_ind` by ascending country then descending city.

```
# Sort temperatures_ind by index values
print(temperatures_ind.sort_index())

# Sort temperatures_ind by index values at the city level
print(temperatures_ind.sort_index(level= ["city"]))

# Sort temperatures_ind by country then descending city
print(temperatures_ind.sort_index(level = ["country", "city"],ascending = [True,False]))
```

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']
```

PANDAS

Slicing and subsetting with .loc and .iloc

Sort the index before you slice

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

```
                      name  height_cm  weight_kg
breed       color
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"]
```

		name	height_cm	weight_kg
breed	color			
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

		name	height_cm	weight_kg
breed	color			
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

		name	height_cm	weight_kg
breed	color			
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

This approach is wrong : be careful

Slicing the inner index levels correctly

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

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

Full dataset

		name	height_cm	weight_kg
breed	color			
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"]
```

		name	height_cm
breed	color		
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

		name	height_cm	weight_kg
breed	color			
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

Dog days

```
dogs = dogs.set_index("date_of_birth").sort_index()
print(dogs)
```

```
                 name        breed  color  height_cm  weight_kg
date_of_birth
2011-12-11     Cooper    Schanuzer   Grey         49         17
2013-07-01      Bella     Labrador  Brown         56         25
2014-08-25       Lucy    Chow Chow  Brown         46         22
2015-04-20     Stella    Chihuahua    Tan         18          2
2016-09-16    Charlie       Poodle  Black         43         23
2017-01-20        Max     Labrador  Black         59         29
2018-02-27     Bernie  St. Bernard  White         77         74
```

Slicing by dates

```
# Get dogs with date_of_birth between 2014-08-25 and 2016-09-16
dogs.loc["2014-08-25":"2016-09-16"]
```

```
                name      breed  color  height_cm  weight_kg
date_of_birth
2014-08-25      Lucy  Chow Chow  Brown         46         22
2015-04-20    Stella  Chihuahua    Tan         18          2
2016-09-16   Charlie     Poodle  Black         43         23
```

Slicing by partial dates

```
# Get dogs with date_of_birth between 2014-01-01 and 2016-12-31
dogs.loc["2014":"2016"]
```

```
                name      breed  color  height_cm  weight_kg
date_of_birth
2014-08-25       Lucy  Chow Chow  Brown         46         22
2015-04-20     Stella  Chihuahua    Tan         18          2
2016-09-16    Charlie     Poodle  Black         43         23
```

Subsetting by row/column number

```
print(dogs.iloc[2:5, 1:4])
```

```
       breed  color  height_cm
2  Chow Chow  Brown         46
3  Schnauzer   Grey         49
4   Labrador  Black         59
```

Full dataset

```
      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
```

PANDAS

Slicing index values

Slicing lets you select consecutive elements of an object using `first:last` syntax. DataFrames can be sliced by index values or by row/column number; we'll start with the first case. This involves slicing inside the `.loc[]` method.

Compared to slicing lists, there are a few things to remember.

- You can only slice an index if the index is sorted (using `.sort_index()`).
- To slice at the outer level, `first` and `last` can be strings.
- To slice at inner levels, `first` and `last` should be tuples.
- If you pass a single slice to `.loc[]`, it will slice the rows.

`pandas` is loaded as `pd`. `temperatures_ind` has country and city in the index, and is available.

Instructions

100 XP

- Sort the index of `temperatures_ind`.
- Use slicing with `.loc[]` to get these subsets:
  - from Pakistan to Russia.
  - from Lahore to Moscow. (*This will return nonsense.*)
  - from Pakistan, Lahore to Russia, Moscow.

```
# Sort the index of temperatures_ind
temperatures_srt = temperatures_ind.sort_index()

# Subset rows from Pakistan to Russia
print(temperatures_srt.loc["Pakistan":"Russia"])

# Try to subset rows from Lahore to Moscow
print(temperatures_srt.loc["Lahore":"Moscow"])

# Subset rows from Pakistan, Lahore to Russia, Moscow
print(temperatures_srt.loc[("Pakistan", "Lahore"):("Russia", "M
oscow")])
```

PANDAS

Exercise

Slicing in both directions

You've seen slicing DataFrames by rows and by columns, but since DataFrames are two-dimensional objects, it is often natural to slice both dimensions at once. That is, by passing two arguments to `.loc[]`, you can subset by rows and columns in one go.

`pandas` is loaded as `pd`. `temperatures_srt` is indexed by country and city, has a sorted index, and is available.

Instructions 100 XP

- Use `.loc[]` slicing to subset rows from India, Hyderabad to Iraq, Baghdad.
- Use `.loc[]` slicing to subset columns from `date` to `avg_temp_c`.
- Slice in both directions at once from Hyderabad to Baghdad, and `date` to `avg_temp_c`.

```
# Subset rows from India, Hyderabad to Iraq, Baghdad
print(temperatures_srt.loc[("India", "Hyderabad"):("Iraq", "Baghdad")])

# Subset columns from date to avg_temp_c
print(temperatures_srt.loc[:, "date" : "avg_temp_c"])

# Subset in both directions at once
print(temperatures_srt.loc[("India","Hyderabad"):("Iraq","Baghdad"),"date" : "avg_temp_c" ])
```

Slicing time series

Slicing is particularly useful for time series since it's a common thing to want to filter for data within a date range. Add the `date` column to the index, then use `.loc[]` to perform the subsetting. The important thing to remember is to keep your dates in ISO 8601 format, that is, `"yyyy-mm-dd"` for year-month-day, `"yyyy-mm"` for year-month, and `"yyyy"` for year.

Recall from Chapter 1 that you can combine multiple Boolean conditions using logical operators, such as `&`. To do so in one line of code, you'll need to add parentheses `()` around each condition.

`pandas` is loaded as `pd` and `temperatures`, with no index, is available.

Instructions

0 XP

- Use Boolean conditions, not `.isin()` or `.loc[]`, and the full date `"yyyy-mm-dd"`, to subset `temperatures` for rows where the `date` column is in 2010 and 2011 and print the results.
- Set the index of `temperatures` to the `date` column and sort it.
- Use `.loc[]` to subset `temperatures_ind` for rows in 2010 and 2011.
- Use `.loc[]` to subset `temperatures_ind` for rows from August 2010 to February 2011.

PANDAS

```
# Use Boolean conditions to subset temperatures for rows in 2010 and 2011
temperatures_bool = temperatures[(temperatures["date"] >= "2010-01-01") & (temperatures["date"] <= "2011-12-31")]
print(temperatures_bool)

# Set date as the index and sort the index
temperatures_ind = temperatures.set_index("date").sort_index()

# Use .loc[] to subset temperatures_ind for rows in 2010 and 2011
print(temperatures_ind.loc["2010":"2011"])

# Use .loc[] to subset temperatures_ind for rows from Aug 2010 to Feb 2011
print(temperatures_ind.loc["2010-08":"2011-02"])
```

PANDAS

Exercise

Subsetting by row/column number

The most common ways to subset rows are the ways we've previously discussed: using a Boolean condition or by index labels. However, it is also occasionally useful to pass row numbers.

This is done using `.iloc[]`, and like `.loc[]`, it can take two arguments to let you subset by rows and columns.

`pandas` is loaded as `pd`. `temperatures` (without an index) is available.

Instructions

100 XP

Use `.iloc[]` on `temperatures` to take subsets.

- Get the 23rd row, 2nd column (index positions 22 and 1).
- Get the first 5 rows (index positions 0 to 5).
- Get all rows, columns 3 and 4 (index positions 2 to 4).
- Get the first 5 rows, columns 3 and 4.

PANDAS

```
# Get 23rd row, 2nd column (index 22, 1)
print(temperatures.iloc[22,1])

# Use slicing to get the first 5 rows
print(temperatures.iloc[0:5])

# Use slicing to get columns 3 to 4
print(temperatures.iloc[:,2:4])

# Use slicing in both directions at once
print(temperatures.iloc[0:5,2:4])
```

PANDAS

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
```

PANDAS

Working with pivot tables

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

Working with pivot tables

.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

The sort_index learnt earlier can be used to perform similar pivot table actions

PANDAS

Working with pivot tables

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
```

The sort_index learnt earlier can be used to perform similar pivot table actions

PANDAS

Pivot temperature by city and year

It's interesting to see how temperatures for each city change over time—looking at every month results in a big table, which can be tricky to reason about. Instead, let's look at how temperatures change by year.

You can access the components of a date (year, month and day) using code of the form `dataframe["column"].dt.component`. For example, the month component is `dataframe["column"].dt.month`, and the year component is `dataframe["column"].dt.year`.

Once you have the year column, you can create a pivot table with the data aggregated by city and year, which you'll explore in the coming exercises.

`pandas` is loaded as `pd`. `temperatures` is available.

Exercise

Pivot temperature by city and year

It's interesting to see how temperatures for each city change over time—looking at every month results in a big table, which can be tricky to reason about. Instead, let's look at how temperatures change by year.

You can access the components of a date (year, month and day) using code of the form `dataframe["column"].dt.component`. For example, the month component is `dataframe["column"].dt.month`, and the year component is `dataframe["column"].dt.year`.

Once you have the year column, you can create a pivot table with the data aggregated by city and year, which you'll explore in the coming exercises.

`pandas` is loaded as `pd`. `temperatures` is available.

Instructions 70 XP

- Add a `year` column to `temperatures`, from the `year` component of the `date` column.
- Make a pivot table of the `avg_temp_c` column, with `country` and `city` as rows, and `year` as columns. Assign to `temp_by_country_city_vs_year`, and *look at the result.*

The sort_index learnt earlier can be used to perform similar pivot table actions

PANDAS

```
# Add a year column to temperatures
temperatures["year"] = temperatures["date"].dt.year

# Pivot avg_temp_c by country and city vs year
temp_by_country_city_vs_year = temperatures.pivot_table( "avg_temp_c" , index =["c
ountry","city"], columns = "year" )

# See the result
print(temp_by_country_city_vs_year)
```

The sort_index learnt earlier can be used to perform similar pivot table actions

PANDAS

Exercise

Subsetting pivot tables

A pivot table is just a DataFrame with sorted indexes, so the techniques you have learned already can be used to subset them. In particular, the `.loc[]` + slicing combination is often helpful.

`pandas` is loaded as `pd`. `temp_by_country_city_vs_year` is available.

Instructions 0 XP

Use `.loc[]` on `temp_by_country_city_vs_year` to take subsets.

- From Egypt to India.
- From Egypt, Cairo to India, Delhi.
- From Egypt, Cairo to India, Delhi, and 2005 to 2010.

```
# Subset for Egypt to India
temp_by_country_city_vs_year.loc["Egypt":"India"]

# Subset for Egypt, Cairo to India, Delhi
temp_by_country_city_vs_year.loc[("Egypt", "Cairo"):("India", "Delhi")]

# Subset for Egypt, Cairo to India, Delhi, and 2005 to 2010
temp_by_country_city_vs_year.loc[("Egypt", "Cairo"):("India", "Delhi"),
"2005":"2010"]
```

The sort_index learnt earlier can be used to perform similar pivot table actions

PANDAS

Recall from Chapter 1 that you can easily subset a Series or DataFrame to find rows of interest using a logical condition inside of square brackets. For example: `series[series > value]`.

`pandas` is loaded as `pd` and the DataFrame `temp_by_country_city_vs_year` is available. The `.head()` for this DataFrame is shown below, with only a few of the year columns displayed:

country	city	2000	2001	2002	...	2013
Afghanistan	Kabul	15.823	15.848	15.715	...	16.206
Angola	Luanda	24.410	24.427	24.791	...	24.554
Australia	Melbourne	14.320	14.180	14.076	...	14.742
	Sydney	17.567	17.854	17.734	...	18.090
Bangladesh	span translate="no">Dhaka	25.905	25.931	26.095	...	26.587

Instructions

70 XP

- Calculate the mean temperature for each year, assigning to `mean_temp_by_year`.
- Filter `mean_temp_by_year` for the year that had the highest mean temperature.
- Calculate the mean temperature for each city (across columns), assigning to `mean_temp_by_city`.
- Filter `mean_temp_by_city` for the city that had the lowest mean temperature.

PANDAS

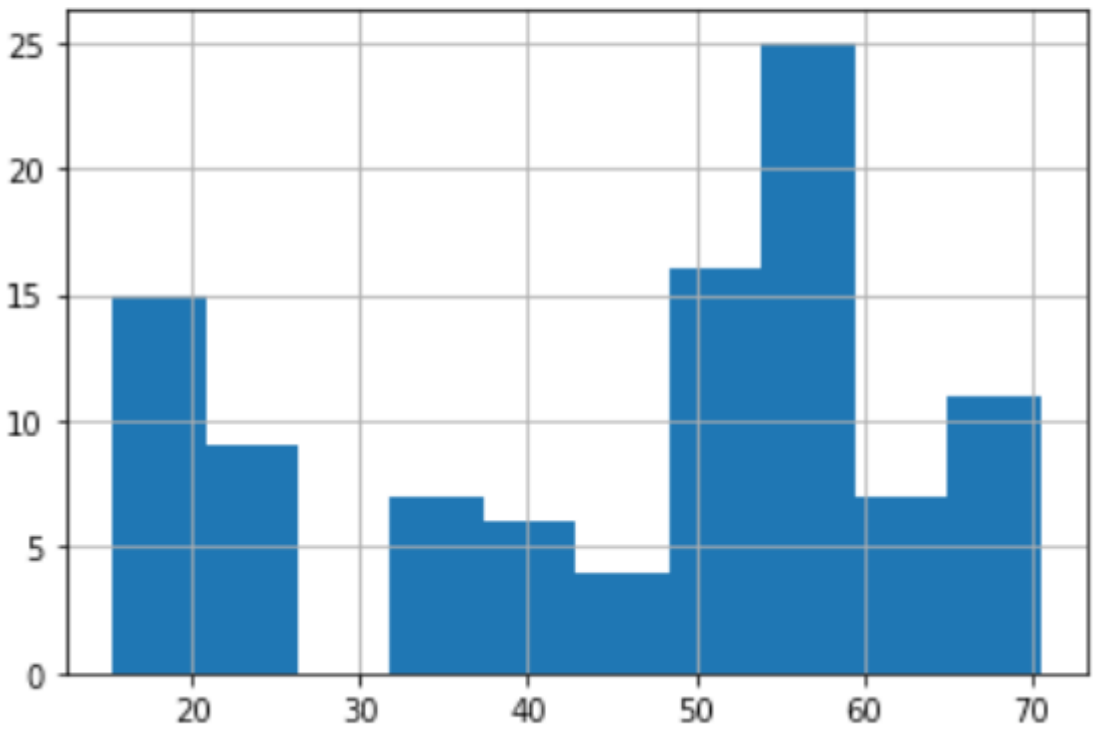
```
# Get the worldwide mean temp by year
mean_temp_by_year = temp_by_country_city_vs_year.mean()

# Filter for the year that had the highest mean temp
print(mean_temp_by_year[mean_temp_by_year == mean_temp_by_year.max()])

# Get the mean temp by city
mean_temp_by_city = temp_by_country_city_vs_year.mean(axis ="columns")

# Filter for the city that had the lowest mean temp

print(mean_temp_by_city[mean_temp_by_city == mean_temp_by_city.min()])
```

Histograms

```
import matplotlib.pyplot as plt
```

```
dog_pack["height_cm"].hist()
```

```
plt.show()
```

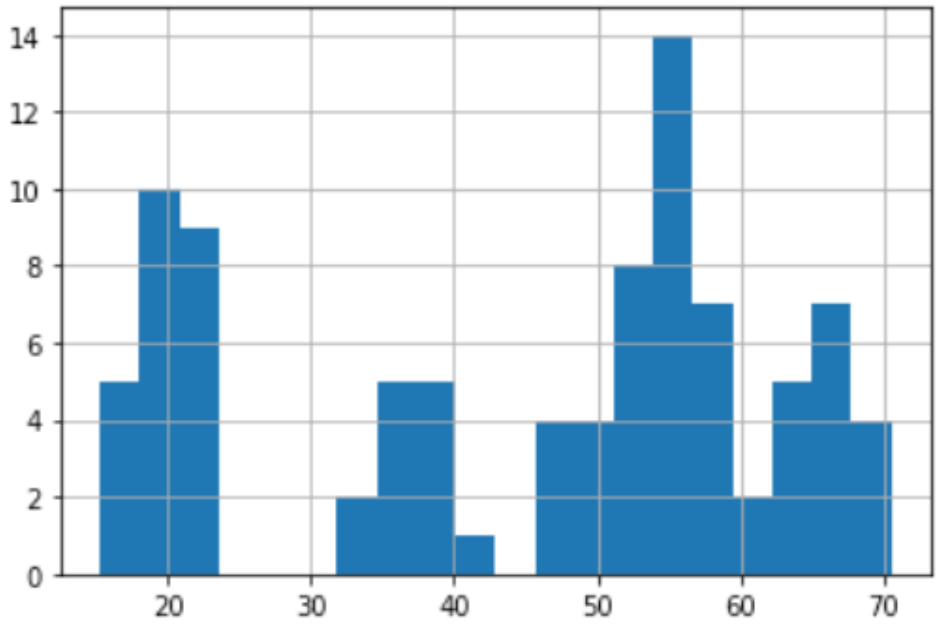


PANDAS

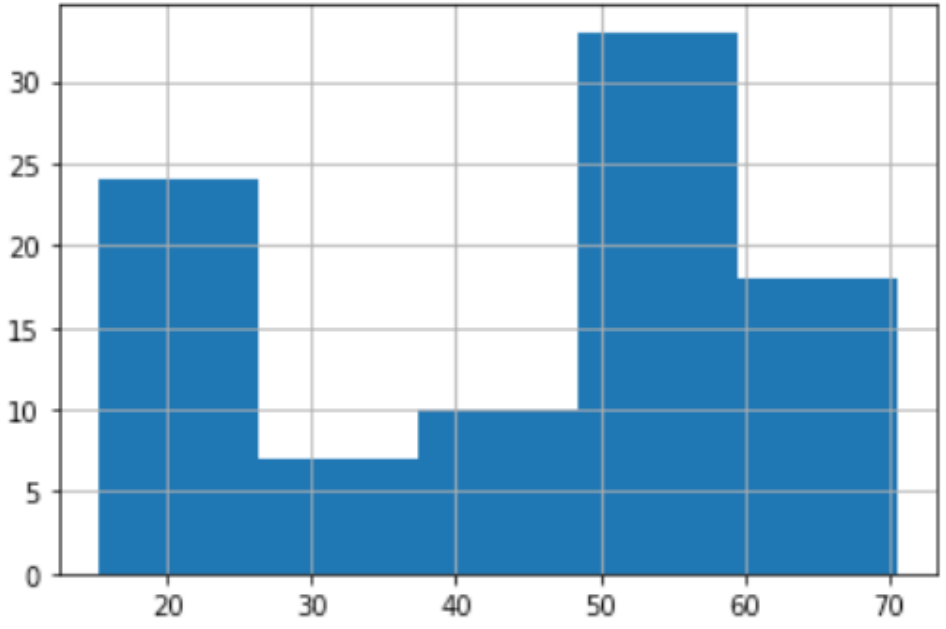
Visualizing your data

Histograms

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




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



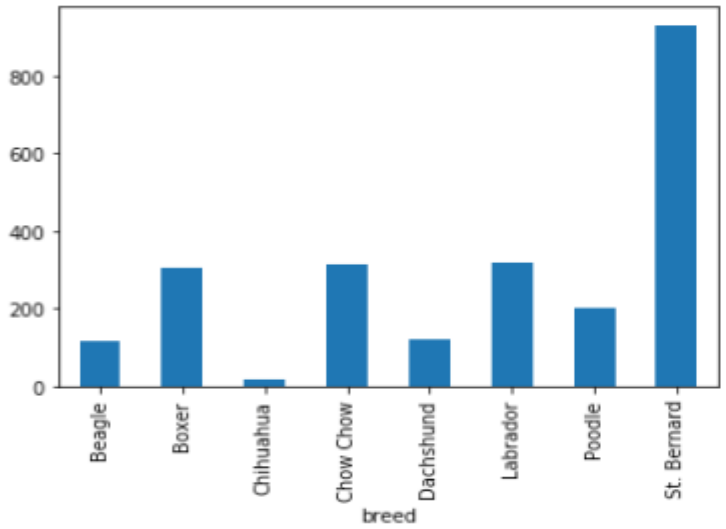
Bar plots

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

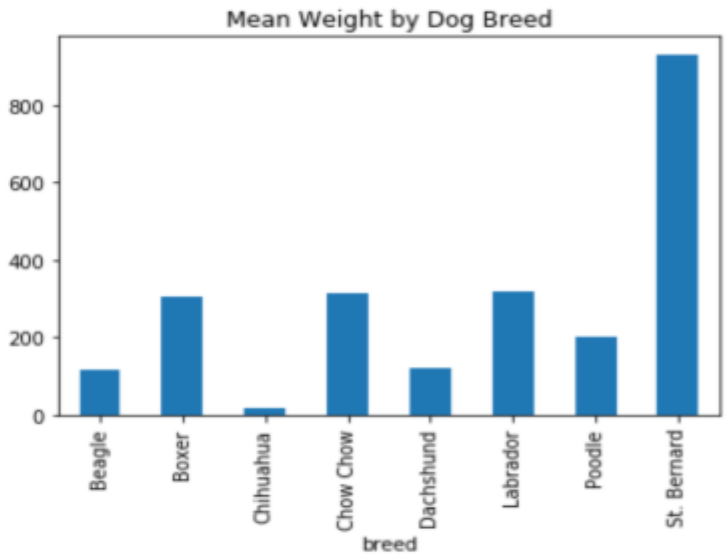
```
breed
Beagle         10.636364
Boxer          30.620000
Chihuahua       1.491667
Chow Chow      22.535714
Dachshund       9.975000
Labrador       31.850000
Poodle         20.400000
St. Bernard    71.576923
Name: weight_kg, dtype: float64
```

Bar plots

```
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
```

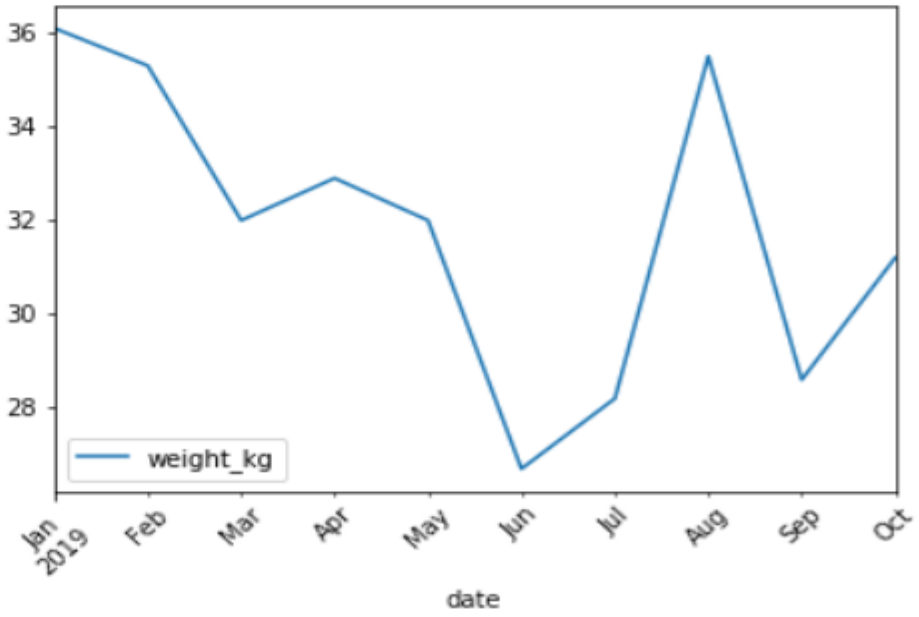
```
sully.plot(x="date",
           y="weight_kg",
           kind="line")
plt.show()
```

PANDAS

Visualizing your data

Rotating axis labels

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

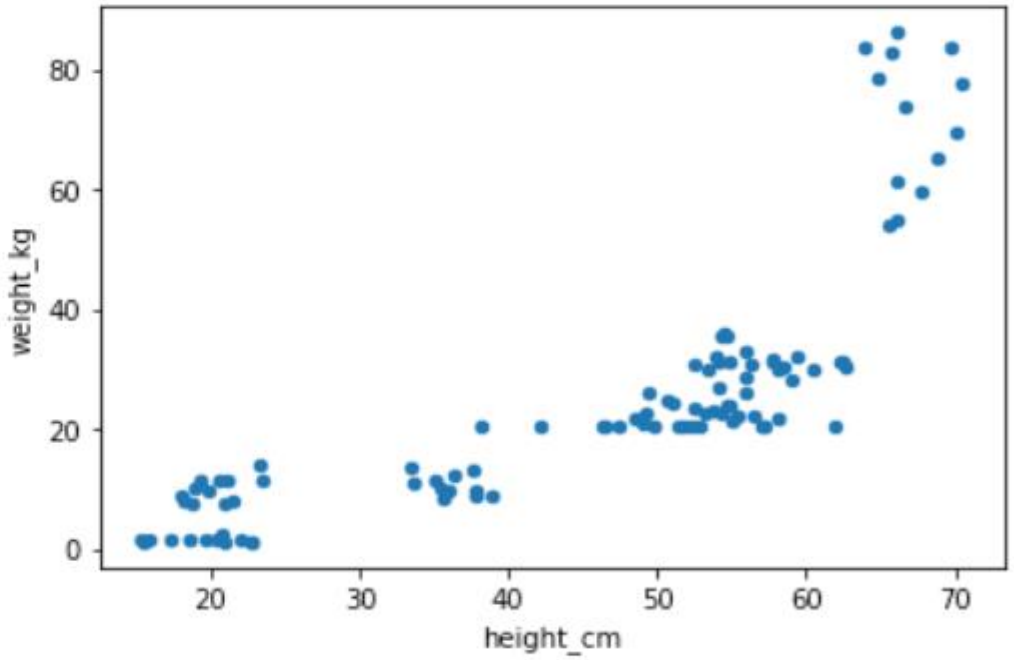


PANDAS

Visualizing your data

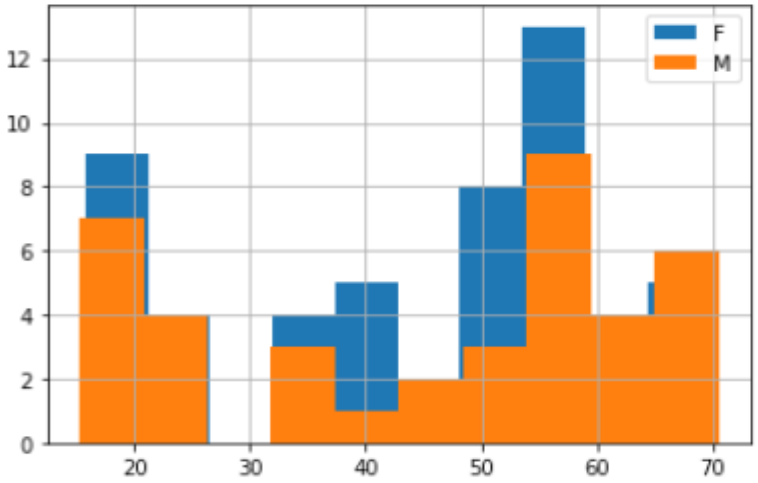
Scatter plots

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



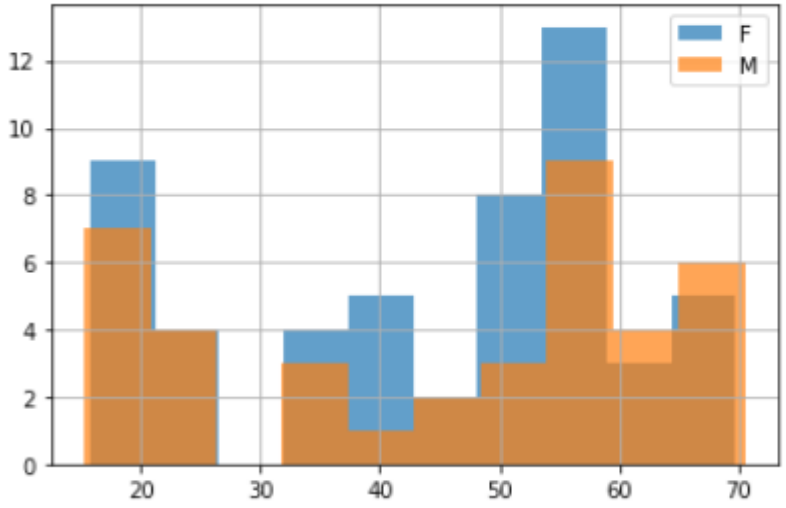
Add a legend

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



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()
```



PANDAS

Which avocado size is most popular?

Avocados are increasingly popular and delicious in guacamole and on toast. The Hass Avocado Board keeps track of avocado supply and demand across the USA, including the sales of three different sizes of avocado. In this exercise, you'll use a bar plot to figure out which size is the most popular.

Bar plots are great for revealing relationships between categorical (size) and numeric (number sold) variables, but you'll often have to manipulate your data first in order to get the numbers you need for plotting.

`pandas` has been imported as `pd`, and `avocados` is available.

Instructions 100 XP

- Print the head of the `avocados` dataset. *What columns are available?*
- For each avocado size group, calculate the total number sold, storing as `nb_sold_by_size`.
- Create a bar plot of the number of avocados sold by size.
- Show the plot.

```
# Import matplotlib.pyplot with alias plt
import matplotlib.pyplot as plt

# Look at the first few rows of data
print(avocados.head())

# Get the total number of avocados sold of each size
nb_sold_by_size = avocados.groupby("size")["nb_sold"].sum()

# Create a bar plot of the number of avocados sold by size
nb_sold_by_size.plot(kind ="bar")

# Show the plot
plt.show()
```

PANDAS

```
# Import matplotlib.pyplot with alias plt
import matplotlib.pyplot as plt

# Look at the first few rows of data
print(avocados.head())

# Get the total number of avocados sold of each size
nb_sold_by_size = avocados.groupby("size")["nb_sold"].sum()

# Create a bar plot of the number of avocados sold by size
nb_sold_by_size.plot(kind ="bar")

# Show the plot
plt.show()
```

PANDAS

Changes in sales over time

Line plots are designed to visualize the relationship between two numeric variables, where each data values is connected to the next one. They are especially useful for visualizing the change in a number over time since each time point is naturally connected to the next time point. In this exercise, you'll visualize the change in avocado sales over three years.

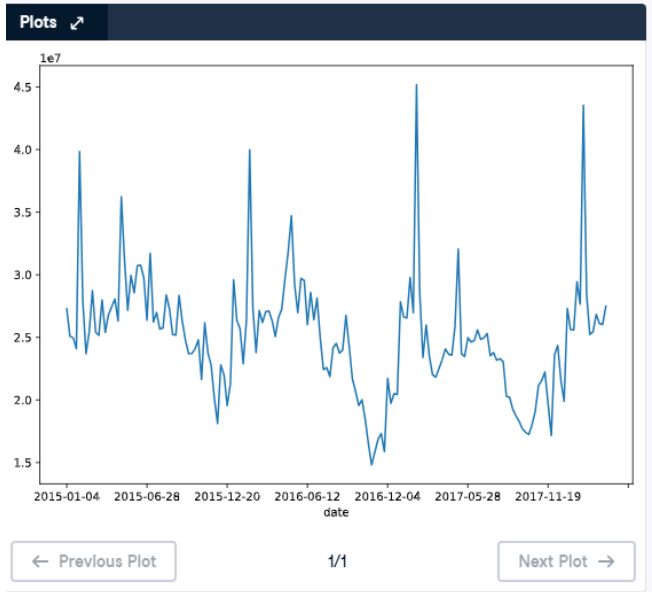
`pandas` has been imported as `pd`, and `avocados` is available.

Instructions

0 XP

- Get the total number of avocados sold on each date. *The DataFrame has two rows for each date—one for organic, and one for conventional.* Save this as `nb_sold_by_date`.
- Create a line plot of the number of avocados sold.
- Show the plot.

```
# Import matplotlib.pyplot with alias plt
import matplotlib.pyplot as plt

# Get the total number of avocados sold on each date
nb_sold_by_date = avocados.groupby("date")["nb_sold"].sum
()

# Create a line plot of the number of avocados sold by date
nb_sold_by_date.plot(kind="line")

# Show the plot
plt.show()
```



PANDAS

Avocado supply and demand

Scatter plots are ideal for visualizing relationships between numerical variables. In this exercise, you'll compare the number of avocados sold to average price and see if they're at all related. If they're related, you may be able to use one number to predict the other.

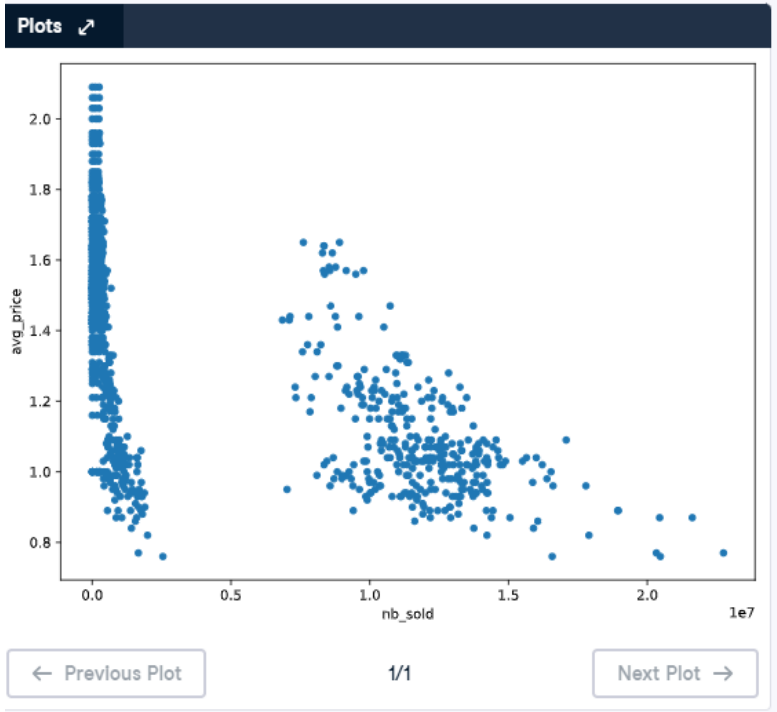
`matplotlib.pyplot` has been imported as `plt`, `pandas` has been imported as `pd`, and `avocados` is available.

Instructions

100 XP

- Create a scatter plot with `nb_sold` on the x-axis and `avg_price` on the y-axis. Title it `"Number of avocados sold vs. average price"`.
- Show the plot.

```
# Scatter plot of avg_price vs. nb_sold with title
avocados.plot(x ='nb_sold',y ='avg_price', kind = 'scatter', title
="Number of avocados sold vs. average price" )

# Show the plot
plt.show()
```



PANDAS

Price of conventional vs. organic avocados

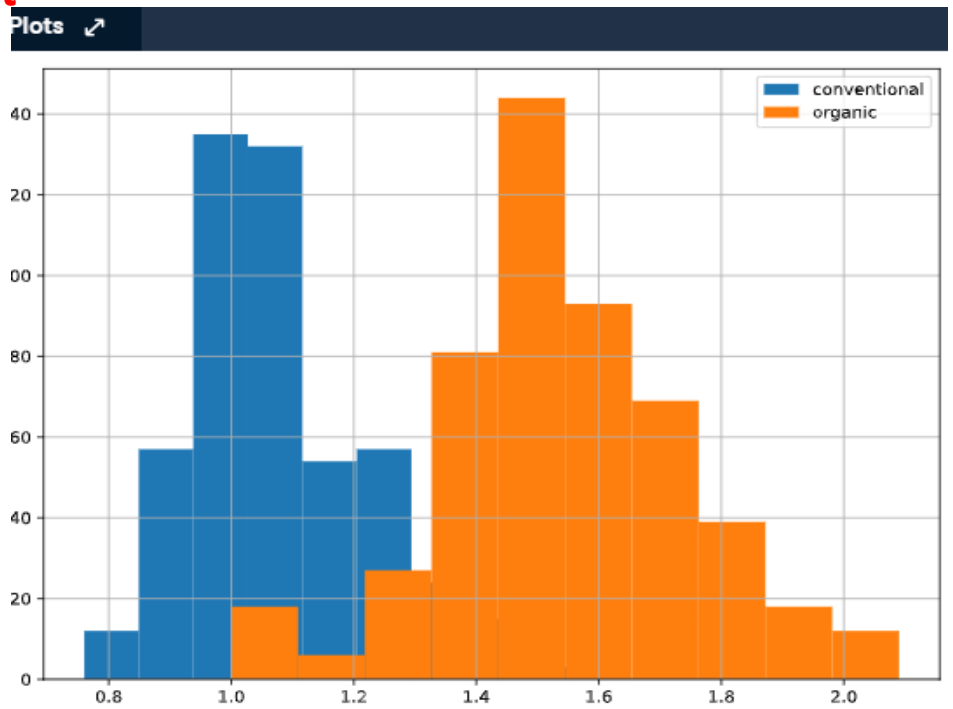
Creating multiple plots for different subsets of data allows you to compare groups. In this exercise, you'll create multiple histograms to compare the prices of conventional and organic avocados.

`matplotlib.pyplot` has been imported as `plt` and `pandas` has been imported as `pd`.

Instructions 1/3 1 XP 1 2 3

- Subset `avocados` for the `"conventional"` `type` and create a histogram of the `avg_price` column.
- Create a histogram of `avg_price` for `"organic"` `type` avocados.
- Add a legend to your plot, with the names `"conventional"` and `"organic"`.
- Show your plot.

```
# Histogram of conventional avg_price
avocados[avocados["type"] ==
"conventional"]["avg_price"].hist()

# Histogram of organic avg_price
avocados[avocados["type"] == "organic"]["avg_price"].hist()

# Add a legend
plt.legend(["conventional", "organic"])

# Show the plot
plt.show()
```



PANDAS

Price of conventional vs. organic avocados

Creating multiple plots for different subsets of data allows you to compare groups. In this exercise, you'll create multiple histograms to compare the prices of conventional and organic avocados.

`matplotlib.pyplot` has been imported as `plt` and `pandas` has been imported as `pd`.

Instructions 2/3 11 XP 2 3

- Modify your code to adjust the transparency of both histograms to `0.5` to see how much overlap there is between the two distributions.

```
# Modify histogram transparency to 0.5
avocados[avocados["type"] ==
"conventional"]["avg_price"].hist(alpha=0.5)

# Modify histogram transparency to 0.5
avocados[avocados["type"] ==
"organic"]["avg_price"].hist(alpha=0.5)

# Add a legend
plt.legend(["conventional", "organic"])

# Show the plot
plt.show()
```

PANDAS

Price of conventional vs. organic avocados

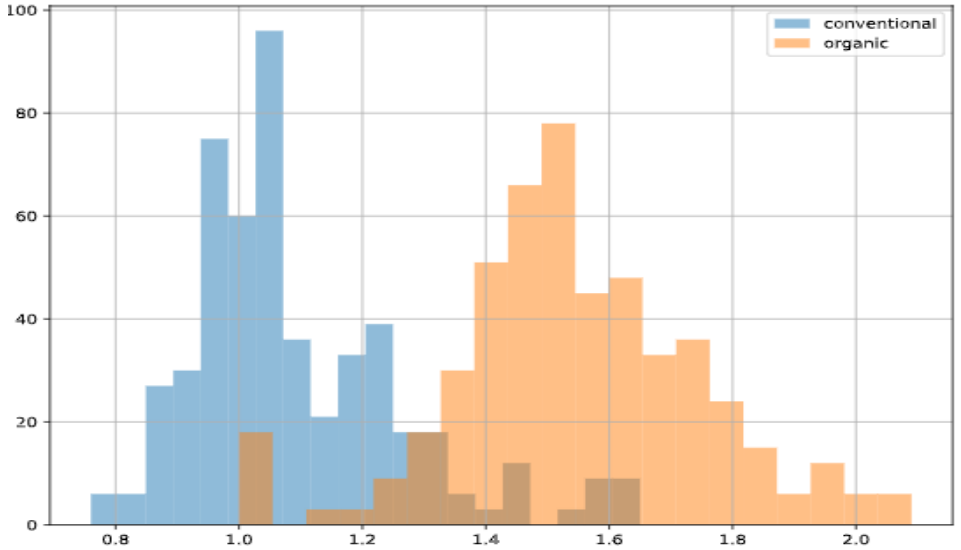
Creating multiple plots for different subsets of data allows you to compare groups. In this exercise, you'll create multiple histograms to compare the prices of conventional and organic avocados.

`matplotlib.pyplot` has been imported as `plt` and `pandas` has been imported as `pd`.

Instructions 3/3 0 XP 3

- Modify your code to use 20 bins in both histograms.

Plots



← Previous Plot 4/4 Next Plot →

```
# Modify bins to 20
avocados[avocados["type"] ==
"conventional"]["avg_price"].hist(alpha=0.5, bins=20)

# Modify bins to 20
avocados[avocados["type"] ==
"organic"]["avg_price"].hist(alpha=0.5, bins=20)

# Add a legend
plt.legend(["conventional", "organic"])

# Show the plot
plt.show()
```

What's a missing value?

Name	Breed	Color	Height (cm)	Weight (kg)	Date of Birth
Bella	Labrador	Brown	56	25	2013-07-01
Charlie	Poodle	Black	43	23	2016-09-16
Lucy	Chow Chow	Brown	46	22	2014-08-25
Cooper	Schnauzer	Gray	49	17	2011-12-11
Max	Labrador	Black	59	29	2017-01-20
Stella	Chihuahua	Tan	18	2	2015-04-20
Bernie	St. Bernard	White	77	74	2018-02-27

Missing values

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

PANDAS

Missing values

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
```

Missing values

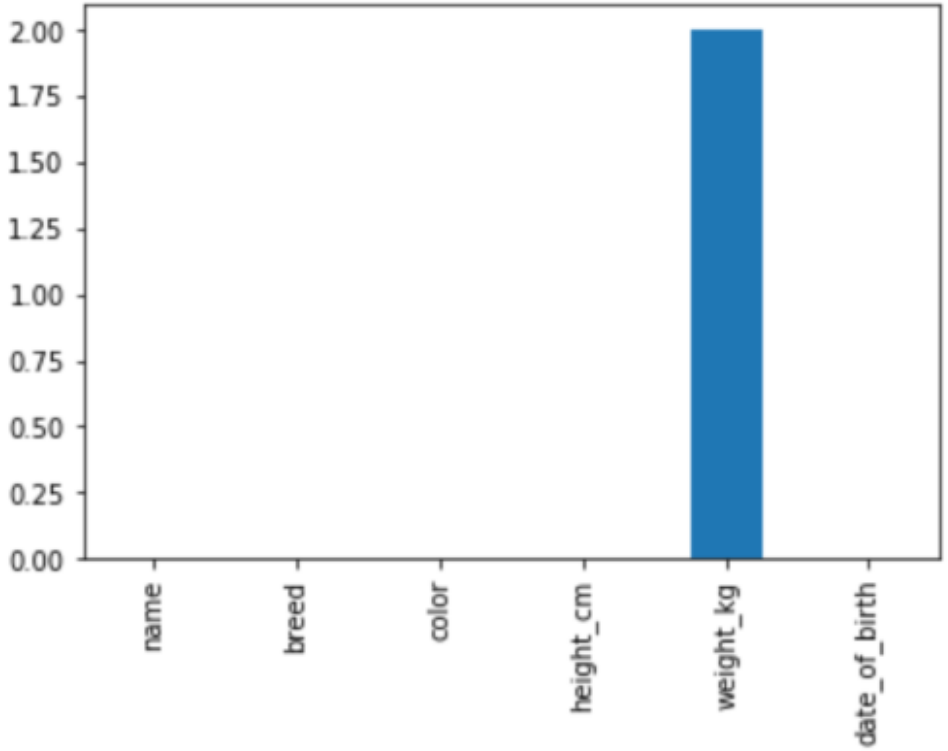
Counting missing values

```
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()
```



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

PANDAS

Finding missing values

Missing values are everywhere, and you don't want them interfering with your work. Some functions ignore missing data by default, but that's not always the behavior you might want. Some functions can't handle missing values at all, so these values need to be taken care of before you can use them. If you don't know where your missing values are, or if they exist, you could make mistakes in your analysis. In this exercise, you'll determine if there are missing values in the dataset, and if so, how many.

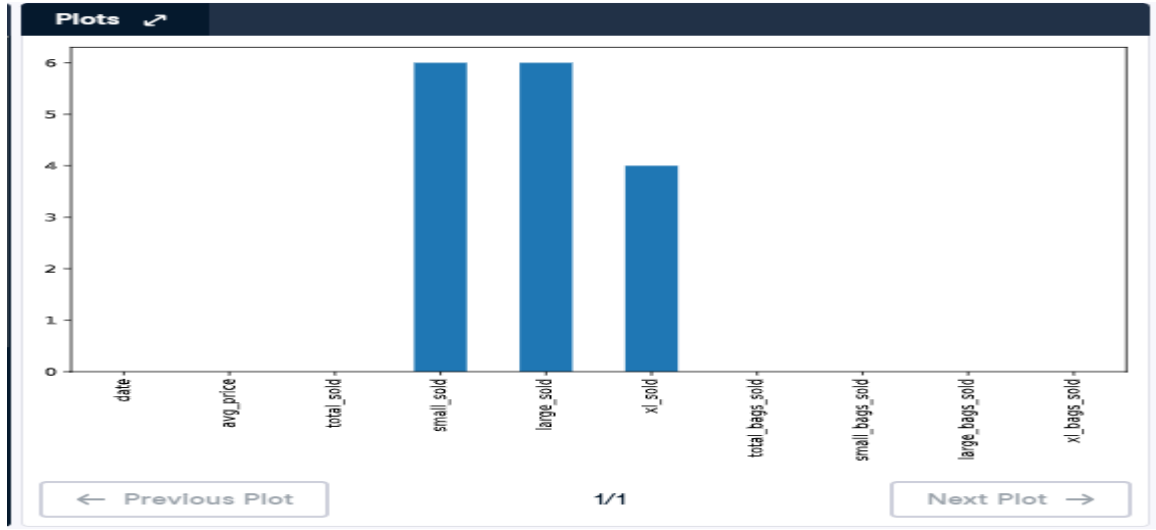
`pandas` has been imported as `pd` and `avocados_2016`, a subset of `avocados` that contains only sales from 2016, is available.

Instructions 100 XP

- Print a DataFrame that shows whether each value in `avocados_2016` is missing or not.
- Print a summary that shows whether *any* value in each column is missing or not.
- Create a bar plot of the total number of missing values in each column.

Take Hint (-30 XP)

```
# Import matplotlib.pyplot with alias plt
import matplotlib.pyplot as plt

# Check individual values for missing values
print(avocados_2016.isna())

# Check each column for missing values
print(avocados_2016.isna().any())

# Bar plot of missing values by variable
avocados_2016.isna().sum().plot(kind = "bar")

# Show plot
plt.show()
```



PANDAS

Removing missing values

Now that you know there are some missing values in your DataFrame, you have a few options to deal with them. One way is to remove them from the dataset completely. In this exercise, you'll remove missing values by removing all rows that contain missing values.

`pandas` has been imported as `pd` and `avocados_2016` is available.

Instructions

0 XP

- Remove the rows of `avocados_2016` that contain missing values and store the remaining rows in `avocados_complete`.
- Verify that all missing values have been removed from `avocados_complete`. Calculate each column that has NAs and print.

```
# Remove rows with missing values
avocados_complete = avocados_2016.dropna()

# Check if any columns contain missing values
print(avocados_complete.isna().any())
```

PANDAS

Exercise

Replacing missing values

Another way of handling missing values is to replace them all with the same value. For numerical variables, one option is to replace values with 0— you'll do this here. However, when you replace missing values, you make assumptions about what a missing value means. In this case, you will assume that a missing number sold means that no sales for that avocado type were made that week.

In this exercise, you'll see how replacing missing values can affect the distribution of a variable using histograms. You can plot histograms for multiple variables at a time as follows:

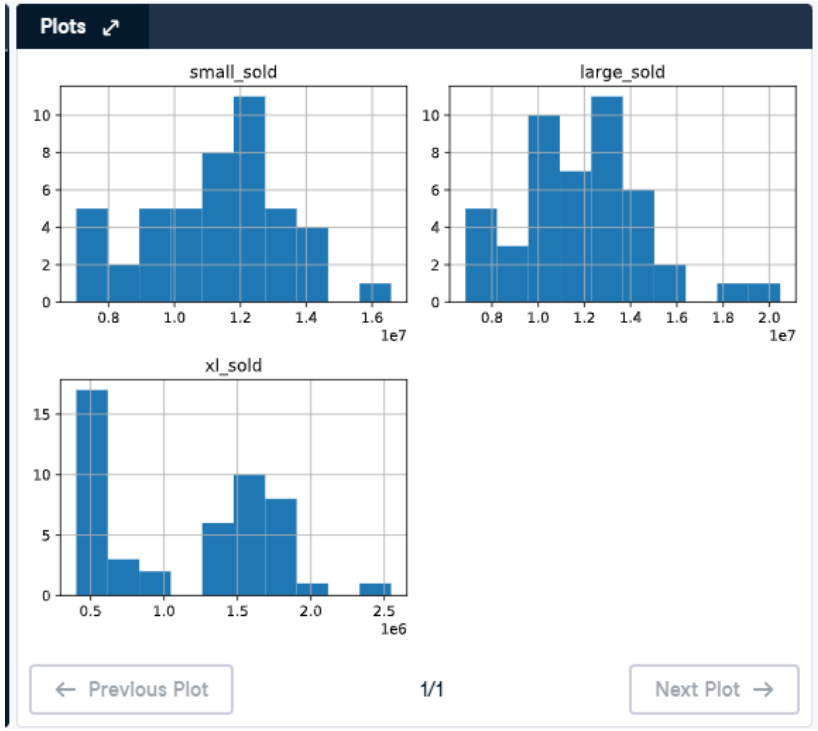
```
dogs[["height_cm", "weight_kg"]].hist()
```

`pandas` has been imported as `pd` and `matplotlib.pyplot` has been imported as `plt`. The `avocados_2016` dataset is available.

Instructions 1/2 50 XP 2

- A list has been created, `cols_with_missing`, containing the names of columns with missing values: `"small_sold"`, `"large_sold"`, and `"xl_sold"`.
- Create a histogram of those columns.
- Show the plot.

```
# List the columns with missing values
cols_with_missing = ["small_sold", "large_sold", "xl_sold"]

# Create histograms showing the distributions
cols_with_missing
avocados_2016[cols_with_missing].hist()

# Show the plot
plt.show()
```



PANDAS

Exercise

Replacing missing values

Another way of handling missing values is to replace them all with the same value. For numerical variables, one option is to replace values with 0— you'll do this here. However, when you replace missing values, you make assumptions about what a missing value means. In this case, you will assume that a missing number sold means that no sales for that avocado type were made that week.

In this exercise, you'll see how replacing missing values can affect the distribution of a variable using histograms. You can plot histograms for multiple variables at a time as follows:

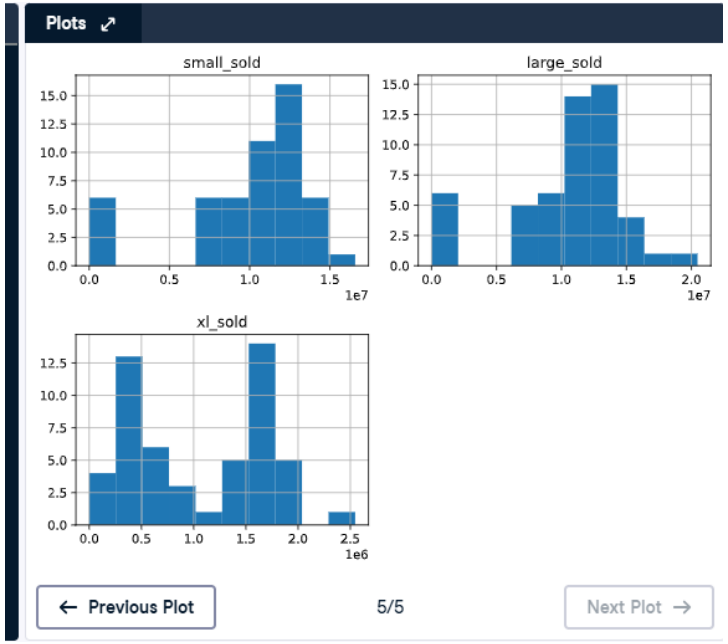
```
dogs[["height_cm", "weight_kg"]].hist()
```

`pandas` has been imported as `pd` and `matplotlib.pyplot` has been imported as `plt`. The `avocados_2016` dataset is available.

Instructions 2/2 50 XP

- Replace the missing values of `avocados_2016` with `0`s and store the result as `avocados_filled`.
- Create a histogram of the `cols_with_missing` columns of `avocados_filled`.

```
# From previous step
cols_with_missing = ["small_sold", "large_sold", "xl_sold"]
avocados_2016[cols_with_missing].hist()
plt.show()

# Fill in missing values with 0
avocados_filled = avocados_2016.fillna(0)

# Create histograms of the filled columns
avocados_filled[cols_with_missing].hist()

# Show the plot
plt.show()
```



PANDAS

Creating DataFrames

Dictionaries

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

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

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

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

```
value1
```

```
Charlotte's Web
```

PANDAS

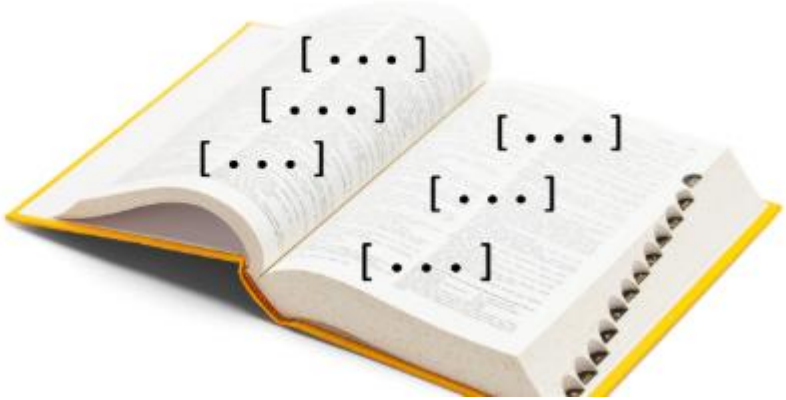
Creating DataFrames

From a list of dictionaries

- Constructed row by row

From a dictionary of lists

- Constructed column by column



PANDAS

Creating DataFrames

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"}
]
```

PANDAS

Creating DataFrames

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)
```

PANDAS

Creating DataFrames

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
```

PANDAS

List of dictionaries

You recently got some new avocado data from 2019 that you'd like to put in a DataFrame using the list of dictionaries method. Remember that with this method, you go through the data row by row.

date	small_sold	large_sold
"2019-11-03"	10376832	7835071
"2019-11-10"	10717154	8561348

`pandas` as `pd` is imported.

Instructions 0 XP

- Create a list of dictionaries with the new data called `avocados_list`.
- Convert the list into a DataFrame called `avocados_2019`.
- Print your new DataFrame.

```
# Create a list of dictionaries with new data
avocados_list = [
    {"date": "2019-11-03", "small_sold": 10376832, "large_sold": 7835071},
    {"date": "2019-11-10", "small_sold": 10717154, "large_sold": 8561348},
]

# Convert list into DataFrame
avocados_2019 = pd.DataFrame(avocados_list)

# Print the new DataFrame
print(avocados_2019)
```

PANDAS

Exercise

Dictionary of lists

Some more data just came in! This time, you'll use the dictionary of lists method, parsing the data column by column.

date	small_sold	large_sold
"2019-11-17"	10859987	7674135
"2019-12-01"	9291631	6238096

`pandas` as `pd` is imported.

Instructions 0 XP

- Create a dictionary of lists with the new data called `avocados_dict`.
- Convert the dictionary to a DataFrame called `avocados_2019`.
- Print your new DataFrame.

```
# Create a dictionary of lists with new data
avocados_dict = {
  "date": ["2019-11-17", "2019-12-01"],
  "small_sold": [10859987, 9291631],
  "large_sold": [7674135, 6238096]
}

# Convert dictionary into DataFrame
avocados_2019 = pd.DataFrame(avocados_dict)

# Print the new DataFrame
print(avocados_2019))
```

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

PANDAS

Reading and writing CSVs

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
```

PANDAS

Reading and writing CSVs

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
```

Reading and writing CSVs

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
```

PANDAS

Exercise

CSV to DataFrame

You work for an airline, and your manager has asked you to do a competitive analysis and see how often passengers flying on other airlines are involuntarily bumped from their flights. You got a CSV file (`airline_bumping.csv`) from the Department of Transportation containing data on passengers that were involuntarily denied boarding in 2016 and 2017, but it doesn't have the exact numbers you want. In order to figure this out, you'll need to get the CSV into a pandas DataFrame and do some manipulation!

`pandas` is imported for you as `pd` . `"airline_bumping.csv"` is in your working directory.

Instructions 1/4

25 XP 2 3 4

- Read the CSV file `"airline_bumping.csv"` and store it as a DataFrame called `airline_bumping` .
- Print the first few rows of `airline_bumping` .

```
# Read CSV as DataFrame called airline_bumping
airline_bumping = pd.read_csv("airline_bumping.csv")

# Take a look at the DataFrame
print(airline_bumping.head())
```

PANDAS

CSV to DataFrame

'ou work for an airline, and your manager has asked you to do a competitive analysis and see how often ›assengers flying on other airlines are involuntarily bumped from their flights. You got a CSV file (`airline_bumping.csv`) from the Department of Transportation containing data on passengers that were ıvoluntarily denied boarding in 2016 and 2017, but it doesn't have the exact numbers you want. In order to igure this out, you'll need to get the CSV into a pandas DataFrame and do some manipulation!

`pandas` is imported for you as `pd` . `"airline_bumping.csv"` is in your working directory.

Instructions 2/4

1 XP ✓ 2 3 4

For each airline group, select the `nb_bumped` , and `total_passengers` columns, and calculate the sum (for both years). Store this as `airline_totals` .

```
# From previous step
airline_bumping =
pd.read_csv("airline_bumping.csv")
print(airline_bumping.head())

# For each airline, select nb_bumped and
total_passengers and sum
airline_totals =
airline_bumping.groupby("airline")[["nb_bum
ped", "total_passengers"]].sum()
```

CSV to DataFrame

You work for an airline, and your manager has asked you to do a competitive analysis and see how often passengers flying on other airlines are involuntarily bumped from their flights. You got a CSV file (`airline_bumping.csv`) from the Department of Transportation containing data on passengers that were involuntarily denied boarding in 2016 and 2017, but it doesn't have the exact numbers you want. In order to figure this out, you'll need to get the CSV into a pandas DataFrame and do some manipulation!

`pandas` is imported for you as `pd` . `"airline_bumping.csv"` is in your working directory.

Instructions 3/4

1 XP ✓ ✓ 3 4

- Create a new column of `airline_totals` called `bumps_per_10k` , which is the number of passengers bumped per 10,000 passengers in 2016 and 2017.

```
# From previous steps
airline_bumping = pd.read_csv("airline_bumping.csv")
print(airline_bumping.head())
airline_totals =
airline_bumping.groupby("airline")[["nb_bumped",
"total_passengers"]].sum()

# Create new col, bumps_per_10k: no. of bumps per 10k
passengers for each airline
airline_totals["bumps_per_10k"] =
airline_totals["nb_bumped"] /
airline_totals["total_passengers"] * 10000
```

PANDAS

CSV to DataFrame

You work for an airline, and your manager has asked you to do a competitive analysis and see how often passengers flying on other airlines are involuntarily bumped from their flights. You got a CSV file (`airline_bumping.csv`) from the Department of Transportation containing data on passengers that were involuntarily denied boarding in 2016 and 2017, but it doesn't have the exact numbers you want. In order to figure this out, you'll need to get the CSV into a pandas DataFrame and do some manipulation!

`pandas` is imported for you as `pd` . `"airline_bumping.csv"` is in your working directory.

Instructions 4/4 25 XP

- Print `airline_totals` to see the results of your manipulations.

```
# From previous steps
airline_bumping = pd.read_csv("airline_bumping.csv")
print(airline_bumping.head())
airline_totals =
airline_bumping.groupby("airline")[["nb_bumped",
"total_passengers"]].sum()
airline_totals["bumps_per_10k"] =
airline_totals["nb_bumped"] /
airline_totals["total_passengers"] * 10000

# Print airline_totals
print(airline_totals)
```

PANDAS

Exercise

DataFrame to CSV

You're almost there! To make things easier to read, you'll need to sort the data and export it to CSV so that your colleagues can read it.

`pandas` as `pd` has been imported for you.

Instructions 0 XP

- Sort `airline_totals` by the values of `bumps_per_10k` from highest to lowest, storing as `airline_totals_sorted`.
- Print your sorted DataFrame.
- Save the sorted DataFrame as a CSV called `"airline_totals_sorted.csv"`.

```
# Create airline_totals_sorted
airline_totals_sorted = airline_totals.sort_values("bumps_per_10k", ascending=False)

# Print airline_totals_sorted
print(airline_totals_sorted)

# Save as airline_totals_sorted.csv
airline_totals_sorted.to_csv("airline_totals_sorted.csv")
```

Wrap-up

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