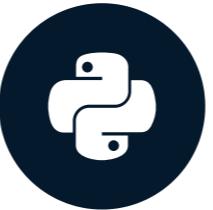


Hello Python!

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

How you will learn

The screenshot shows a DataCamp exercise interface for "Calculations with variables".

Instructions:

- Create a variable `growth_multiplier`, equal to `1.1`.
- Create a variable `result`, equal to the amount of money you saved after `7` years.
- Print out the value of `result`.

Code Snippet:

```
savings = 100
growth_multiplier = 1.1
result = savings *
# Print out result
```

Python Shell:

```
In [1]:
```

Buttons:

- Course Outline
- Daily XP 0
- Run Code
- Submit Answer

Python



- General purpose: build anything
- Open source! Free!
- Python packages, also for data science
 - Many applications and fields
- Version 3.x - <https://www.python.org/downloads/>

IPython Shell

Execute Python commands

The screenshot shows a DataCamp exercise interface for "Calculations with variables".

Exercise Overview: Daily XP 100

Instructions (100 XP):

- Create a variable `growth_multiplier`, equal to `1.1`.
- Create a variable, `result`, equal to the amount of money you saved after `7` years.
- Print out the value of `result`.

Code Editor: script.py

```
1 # Create a variable savings
2 savings = 100
3
4 # Create a variable growth_multiplier
5 growth_multiplier = 1.1
6
7 # Calculate result
8 result = savings * growth_multiplier ** 7
9
10 # Print out result
11 print(result)
12
13
```

Run Code button

IPython Shell: In [1]:

IPython Shell

Execute Python commands

The screenshot shows a DataCamp exercise interface for "Calculations with variables". The main area displays a script named "script.py" with the following code:

```
script.py
1 # Create a variable savings
2 savings = 100
3
4 # Create a variable growth_multiplier
5 growth_multiplier = 1.1
6
7 # Calculate result
8 result = savings * growth_multiplier ** 7
9
10 # Print out result
11 print(result)
12
13
```

To the left, there's an "Instructions" section with a "100 XP" badge, containing the following tasks:

- Create a variable `growth_multiplier`, equal to `1.1`.
- Create a variable, `result`, equal to the amount of money you saved after `7` years.
- Print out the value of `result`.

At the bottom, there are "Run Code" and "Submit Answer" buttons, and an "IPython Shell" tab which shows the prompt "In [1]:".

IPython Shell

The screenshot shows a DataCamp exercise interface. At the top, there's a navigation bar with a logo, 'Course Outline', and 'Daily XP 100'. Below it, a sidebar on the left is titled 'Exercise' and contains the section title 'Calculations with variables'. The main area has a dark background with white text. It displays a Python script named 'script.py' with the following code:

```
script.py
1 # Create a variable savings
2 savings = 100
3
4 # Create a variable growth_multiplier
5
6
7 # Calculate result
8
9
10 # Print out result
11
```

Below the code editor, there are three buttons: 'Run Code', 'Submit Answer', and a yellow '100 XP' button. On the far left, under 'Instructions', there's a list of tasks:

- Create a variable `growth_multiplier`, equal to `1.1`.
- Create a variable, `result`, equal to the amount of money you saved after `7` years.
- Print out the value of `result`.

At the bottom, there's a 'Take Hint (-30 XP)' button. The bottom right corner of the main area says 'Python Shell'.

Python Script

- Text files - `.py`
- List of Python commands
- Similar to typing in IPython Shell

The screenshot shows a DataCamp exercise interface for a Python script named `script.py`. The code in the editor is:

```
script.py
1 # Create a variable savings
2 savings = 100
3
4 # Create a variable growth_multiplier
5 growth_multiplier = 1.1
6
7 # Calculate result
8 result = savings * growth_multiplier ** 7
9
10 # Print out result
11 print(result)
12
13
```

The exercise title is "Calculations with variables". The instructions state: "Remember how you calculated the money you ended up with after 7 years of investing \$100? You did something like this: `100 * 1.1 ** 7`". It then explains: "Instead of calculating with the actual values, you can use variables instead. The `savings` variable you've created in the previous exercise represents the \$100 you started with. It's up to you to create a new variable to represent `1.1` and then redo the calculations!"

The instructions list three tasks:

- Create a variable `growth_multiplier`, equal to `1.1`.
- Create a variable, `result`, equal to the amount of money you saved after 7 years.
- Print out the value of `result`.

Buttons at the bottom include "Run Code" and "Submit Answer". Below the code editor is an IPython Shell tab showing "In [1]:".

Python Script

The screenshot shows a Python script exercise interface. At the top, there's a navigation bar with icons for back, forward, course outline, and daily XP (100). Below the navigation is a sidebar titled "Exercise" with the section "Calculations with variables". The main content area has a heading "Remember how you calculated the money you ended up with after 7 years of investing \$100? You did something like this:" followed by a code snippet: `100 * 1.1 ** 7`. A note below explains: "Instead of calculating with the actual values, you can use variables instead. The `savings` variable you've created in the previous exercise represents the \$100 you started with. It's up to you to create a new variable to represent `1.1` and then redo the calculation!" On the left, under "Instructions", there are three bullet points: "Create a variable `growth_multiplier`, equal to `1.1`", "Create a variable, `result`, equal to the amount of money you saved after 7 years.", and "Print out the value of `result`". A "Take Hint (-30 XP)" button is available. On the right, there's a code editor window titled "script.py" with the line `1` at the top. Below it is a "Run Code" button. At the bottom, a Python shell shows the prompt `In [1]:`.

Python Script

The screenshot shows a DataCamp Python script exercise interface. At the top, there's a navigation bar with icons for back, forward, course outline, and other course metrics like Daily XP (100). Below the navigation is a sidebar titled "Exercises" with a section for "Calculations with variables". The main area contains a code editor window titled "scr1.py" with the following code:

```
1
```

Below the code editor, there's a text block explaining how to calculate compound interest using variables instead of hard-coded values. It mentions a previous exercise where a variable `savings` was created to represent the initial amount of \$100.

The exercise instructions are listed as follows:

- Create a variable `growth_multiplier`, equal to `1.1`.
- Create a variable `result`, equal to the amount of money you saved after 7 years.
- Print out the value of `result`.

At the bottom of the interface, there are "Run Code" and "Submit Answer" buttons, along with a Python Shell tab showing the command `In [1]:`.

- Use `print()` to generate output from script

DataCamp Interface

The screenshot shows the DataCamp Python exercise interface. On the left, there's an 'Exercise' sidebar with a title 'Calculations with variables'. It contains a code snippet: `100 * 1.1 ** 7`. Below it, instructions mention using variables instead of hard-coded values. A list of tasks includes creating a variable `growth_multiplier`, calculating a result, and printing it. A 'Take Hint (-30 XP)' button is present. On the right, the main workspace shows a code editor with `script.py` containing the following code:

```
script.py
1 # Create a variable savings
2 savings = 100
3
4 # Create a variable growth_multiplier
5
6
7 # Calculate result
8
9
10 # Print out result
11
```

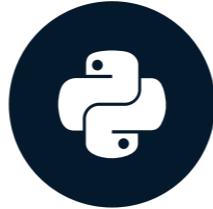
The workspace also includes a terminal shell tab labeled 'IPython Shell' with the prompt 'In [1]:'.

Let's practice!

INTRODUCTION TO PYTHON

Variables and Types

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Variable

- Specific, case-sensitive name
- Call up value through variable name
- 1.79 m - 68.7 kg

```
height = 1.79
```

```
weight = 68.7
```

```
height
```

```
1.79
```

Calculate BMI

```
height = 1.79
```

```
weight = 68.7
```

```
height
```

```
1.79
```

$$\text{BMI} = \frac{\text{weight}}{\text{height}^2}$$

```
68.7 / 1.79 ** 2
```

```
21.4413
```

```
weight / height ** 2
```

```
21.4413
```

```
bmi = weight / height ** 2
```

```
bmi
```

```
21.4413
```

Reproducibility

```
height = 1.79  
weight = 68.7  
bmi = weight / height ** 2  
print(bmi)
```

```
21.4413
```

Reproducibility

```
height = 1.79  
weight = 74.2 # <-  
bmi = weight / height ** 2  
print(bmi)
```

23.1578

Python Types

```
type(bmi)
```

```
float
```

```
day_of_week = 5  
type(day_of_week)
```

```
int
```

Python Types (2)

```
x = "body mass index"  
y = 'this works too'  
type(y)
```

str

```
z = True  
type(z)
```

bool

Python Types (3)

```
2 + 3
```

```
5
```

```
'ab' + 'cd'
```

```
'abcd'
```

- Different type = different behavior!

Let's practice!

INTRODUCTION TO PYTHON

Python Lists

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Python Data Types

- float - real numbers
- int - integer numbers
- str - string, text
- bool - True, False

```
height = 1.73  
tall = True
```

- Each variable represents single value

Problem

- Data Science: many data points
- Height of entire family

```
height1 = 1.73  
height2 = 1.68  
height3 = 1.71  
height4 = 1.89
```

- Inconvenient

Python List

- [a, b, c]

```
[1.73, 1.68, 1.71, 1.89]
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
fam = [1.73, 1.68, 1.71, 1.89]  
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

- Name a collection of values
- Contain any type
- Contain different types

Python List

- [a, b, c]

```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]
```

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam2 = [[{"name": "liz", "height": 1.73},  
         {"name": "emma", "height": 1.68},  
         {"name": "mom", "height": 1.71},  
         {"name": "dad", "height": 1.89}]]
```

```
fam2
```

```
[["liz", 1.73], ["emma", 1.68], ["mom", 1.71], ["dad", 1.89]]
```

List type

```
type(fam)
```

```
list
```

```
type(fam2)
```

```
list
```

- Specific functionality
- Specific behavior

Let's practice!

INTRODUCTION TO PYTHON

Subsetting Lists

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Subsetting lists

```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]  
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[3]
```

```
1.68
```

Subsetting lists

```
[liz', 1.73, emma', 1.68, mom', 1.71, dad', 1.89]
```

```
fam[6]
```

```
'dad'
```

```
fam[-1]
```

```
1.89
```

```
fam[7]
```

```
1.89
```

Subsetting lists

```
[liz', 1.73, emma', 1.68, mom', 1.71, dad', 1.89]
```

```
fam[6]
```

```
'dad'
```

```
fam[-1] # <-
```

```
1.89
```

```
fam[7] # <-
```

```
1.89
```

List slicing

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[3:5]
```

```
[1.68, 'mom']
```

```
fam[1:4]
```

```
[1.73, 'emma', 1.68]
```

[start : end]

inclusive

exclusive

List slicing

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[:4]
```

```
['liz', 1.73, 'emma', 1.68]
```

```
fam[5:]
```

```
[1.71, 'dad', 1.89]
```

Let's practice!

INTRODUCTION TO PYTHON

Manipulating Lists

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

List Manipulation

- Change list elements
- Add list elements
- Remove list elements

Changing list elements

```
fam = ["liz", 1.73, "emma", 1.68, "mom", 1.71, "dad", 1.89]  
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam[7] = 1.86  
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```

```
fam[0:2] = ["lisa", 1.74]  
fam
```

```
['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86]
```

Adding and removing elements

```
fam + ["me", 1.79]
```

```
['lisa', 1.74, 'emma', 1.68, 'mom', 1.71, 'dad', 1.86, 'me', 1.79]
```

```
fam_ext = fam + ["me", 1.79]
```

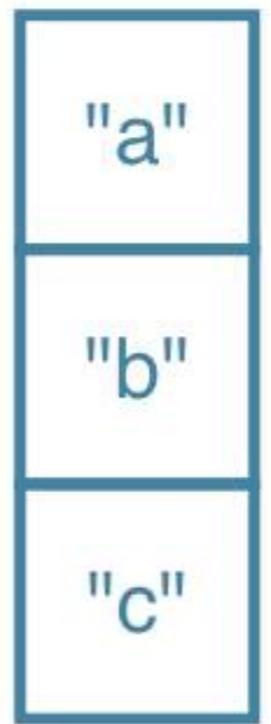
```
del(fam[2])
```

```
fam
```

```
['lisa', 1.74, 1.68, 'mom', 1.71, 'dad', 1.86]
```

Behind the scenes (1)

```
x = ["a", "b", "c"]
```



Behind the scenes (1)

```
x = ["a", "b", "c"]
```

```
y = x
```

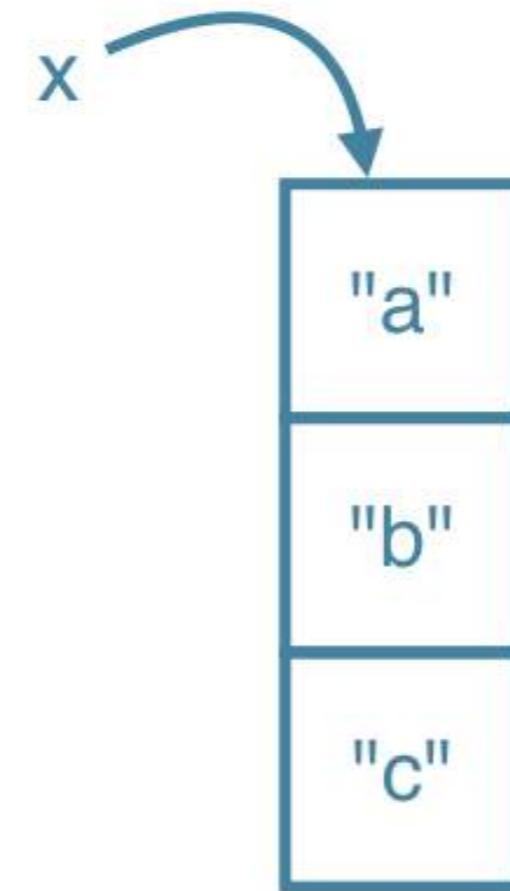
```
y[1] = "z"
```

```
y
```

```
['a', 'z', 'c']
```

```
x
```

```
['a', 'z', 'c']
```



Behind the scenes (1)

```
x = ["a", "b", "c"]
```

```
y = x
```

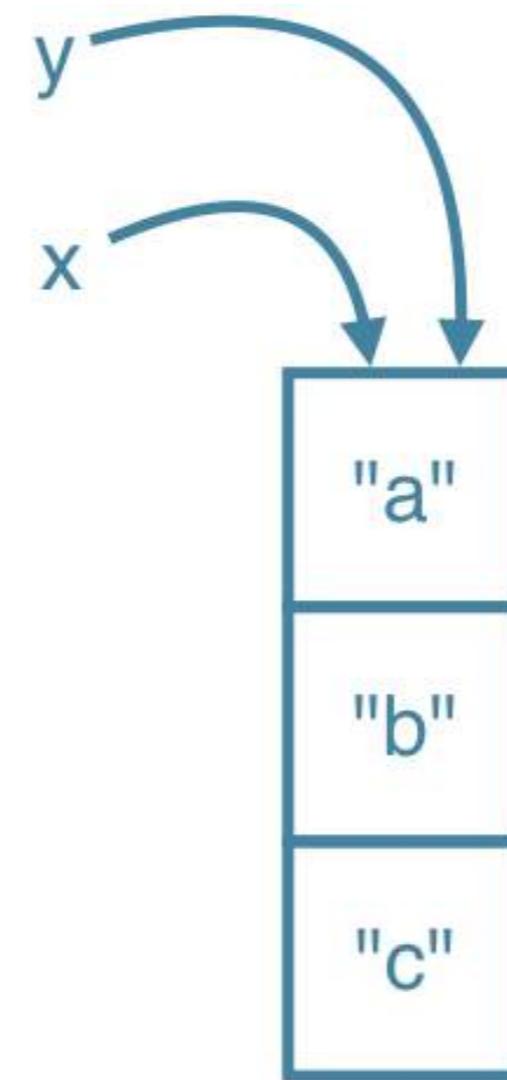
```
y[1] = "z"
```

```
y
```

```
['a', 'z', 'c']
```

```
x
```

```
['a', 'z', 'c']
```



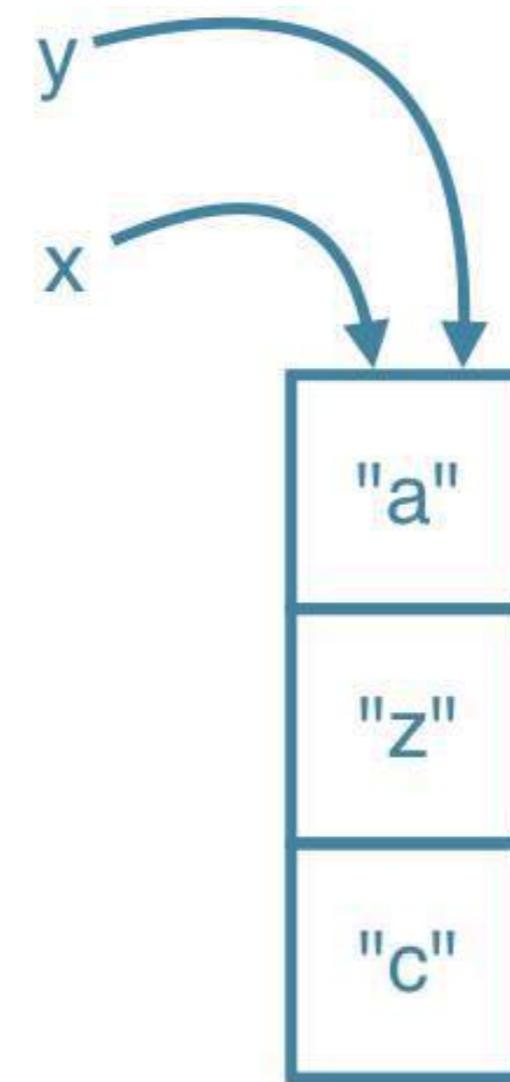
Behind the scenes (1)

```
x = ["a", "b", "c"]  
y = x  
y[1] = "z"  
y
```

```
['a', 'z', 'c']
```

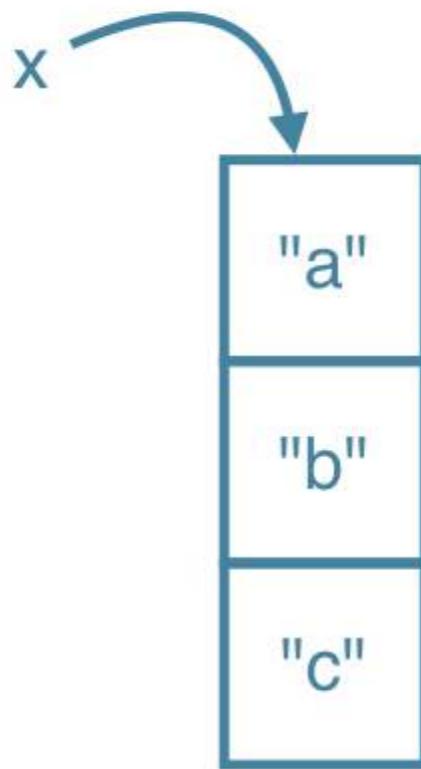
```
x
```

```
['a', 'z', 'c']
```



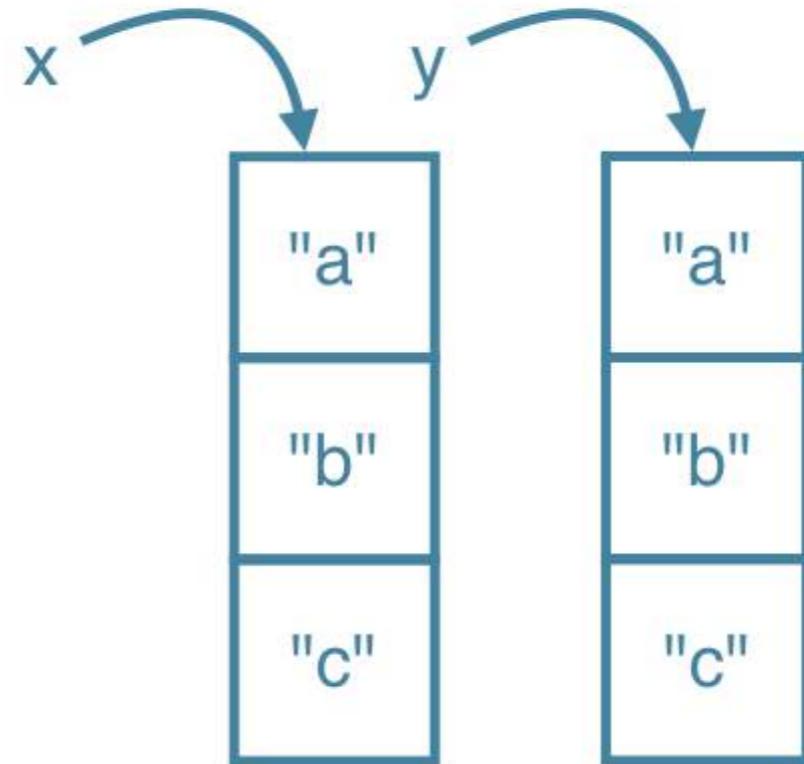
Behind the scenes (2)

```
x = ["a", "b", "c"]
```



Behind the scenes (2)

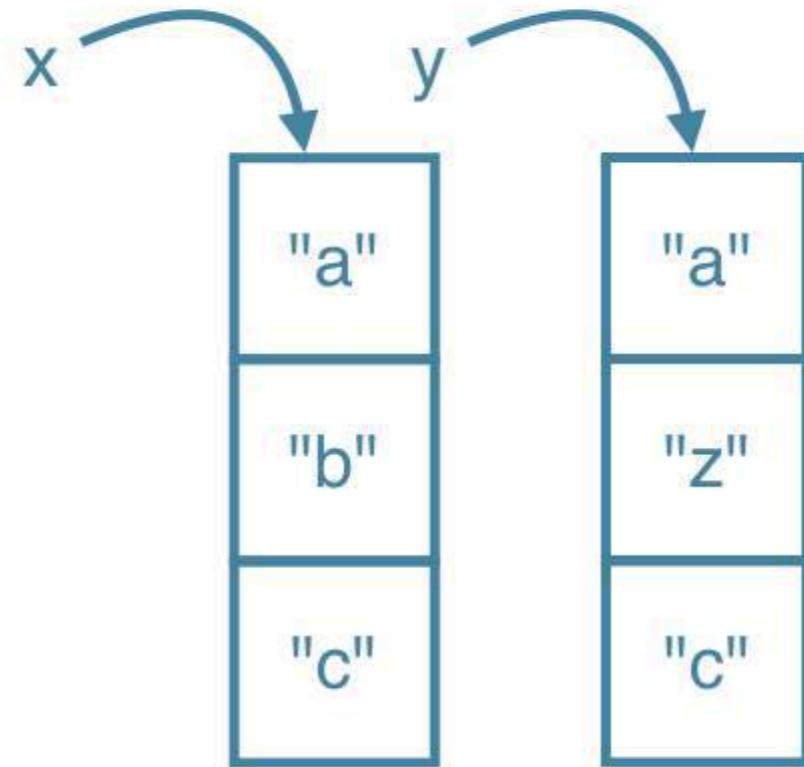
```
x = ["a", "b", "c"]  
y = list(x)  
y = x[:]
```



Behind the scenes (2)

```
x = ["a", "b", "c"]
y = list(x)
y = y[:]
y[1] = "z"
x
```

```
['a', 'b', 'c']
```



Let's practice!

INTRODUCTION TO PYTHON

Functions

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Functions

- Nothing new!
- `type()`
- Piece of reusable code
- Solves particular task
- Call function instead of writing code yourself

Example

```
fam = [1.73, 1.68, 1.71, 1.89]  
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
max(fam)
```

```
1.89
```

```
max()
```

Example

```
fam = [1.73, 1.68, 1.71, 1.89]  
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
max(fam)
```

```
1.89
```

[1.73, 1.68, 1.71, 1.89] →

max()

Example

```
fam = [1.73, 1.68, 1.71, 1.89]  
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
max(fam)
```

```
1.89
```

[1.73, 1.68, 1.71, 1.89] →  → 1.89

Example

```
fam = [1.73, 1.68, 1.71, 1.89]  
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
max(fam)
```

```
1.89
```

```
tallest = max(fam)  
tallest
```

```
1.89
```

round()

```
round(1.68, 1)
```

```
1.7
```

```
round(1.68)
```

```
2
```

```
help(round) # Open up documentation
```

```
Help on built-in function round in module builtins:
```

```
round(number, ndigits=None)
```

```
    Round a number to a given precision in decimal digits.
```

```
The return value is an integer if ndigits is omitted or None.
```

```
Otherwise the return value has the same type as the number. ndigits may be negative.
```

round()

```
help(round)
```

Help on built-in function round in module builtins:

```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

round()

round()

```
help(round)
```

Help on built-in function round in module builtins:

```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

```
round(1.68, 1)
```

round()



round()

```
help(round)
```

Help on built-in function round in module builtins:

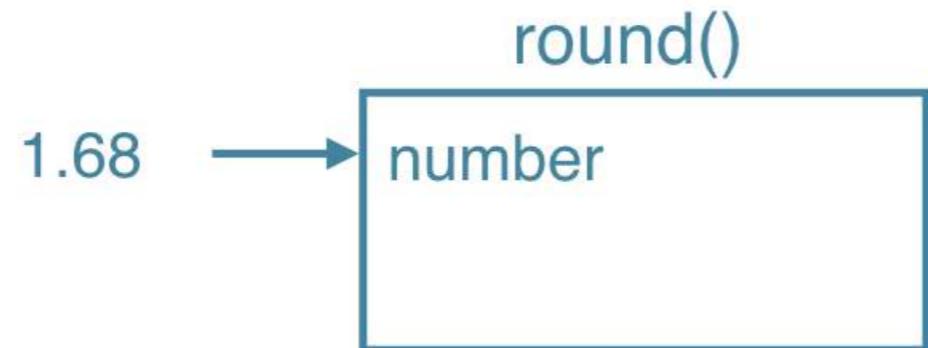
```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

```
round(1.68, 1)
```



round()

```
help(round)
```

Help on built-in function round in module builtins:

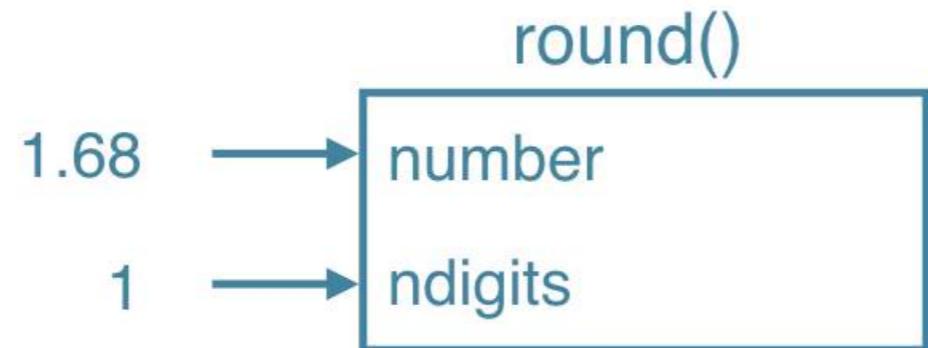
```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

```
round(1.68, 1)
```



round()

```
help(round)
```

Help on built-in function round in module builtins:

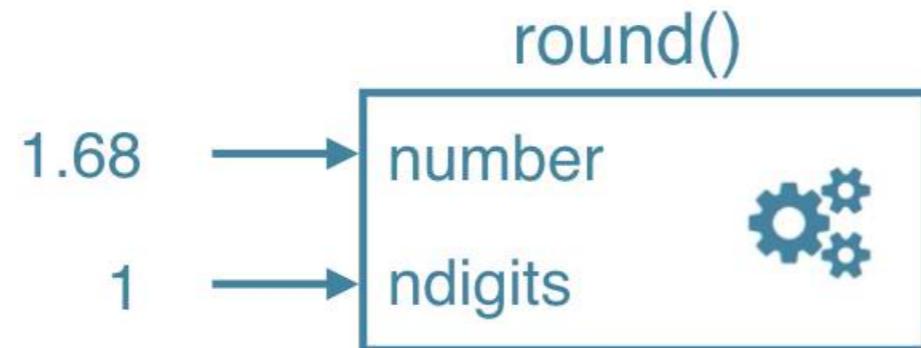
```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

```
round(1.68, 1)
```



round()

```
help(round)
```

Help on built-in function round in module builtins:

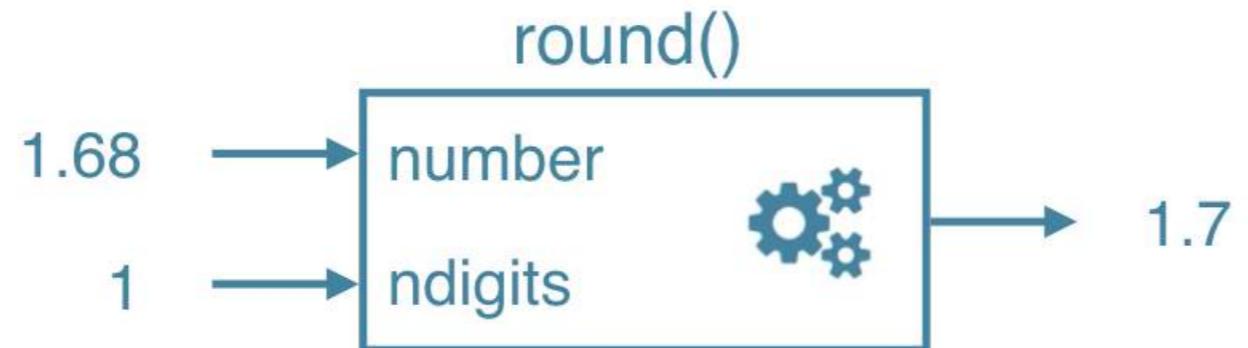
```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

```
round(1.68, 1)
```



round()

```
help(round)
```

Help on built-in function round in module builtins:

```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

round()

round()

```
help(round)
```

Help on built-in function round in module builtins:

```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

```
round(1.68)
```

round()



round()

```
help(round)
```

Help on built-in function round in module builtins:

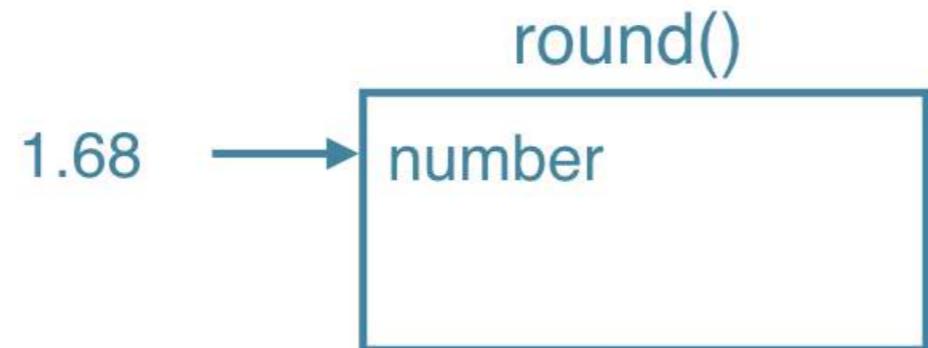
```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

```
round(1.68)
```



round()

```
help(round)
```

Help on built-in function round in module builtins:

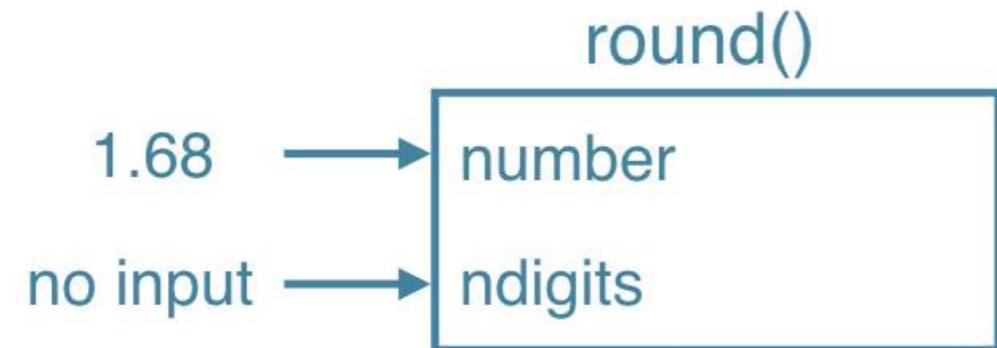
```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

```
round(1.68)
```



round()

```
help(round)
```

Help on built-in function round in module builtins:

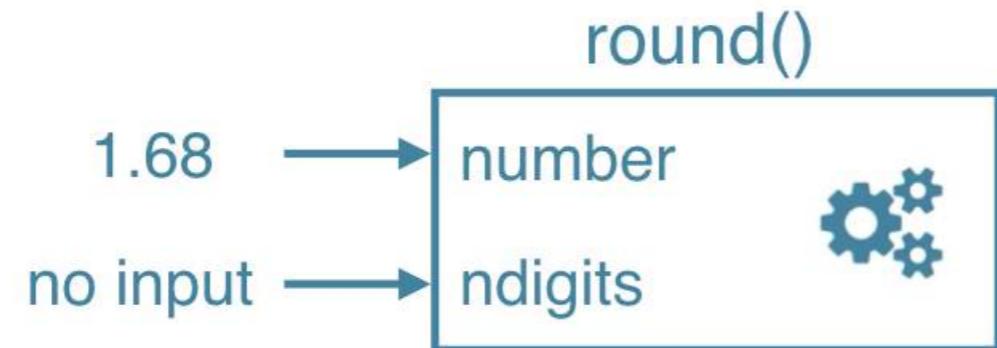
```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

```
round(1.68)
```



round()

```
help(round)
```

Help on built-in function round in module builtins:

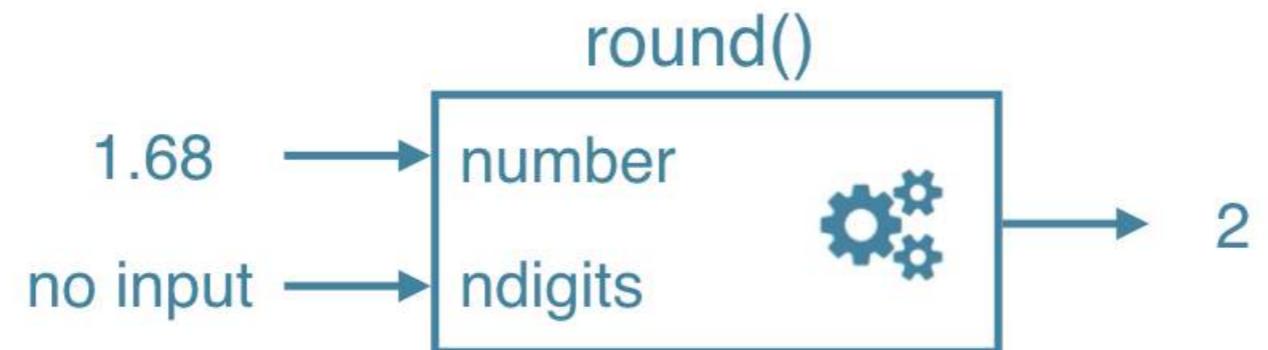
```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

```
round(1.68)
```



round()

```
help(round)
```

Help on built-in function round in module builtins:

```
round(number, ndigits=None)
```

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.

Otherwise the return value has the same type as the number. ndigits may be negative.

- `round(number)`
- `round(number, ndigits)`

Find functions

- How to know?
- Standard task -> probably function exists!
- The internet is your friend

Let's practice!

INTRODUCTION TO PYTHON

Methods

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Built-in Functions

- Maximum of list: `max()`
- Length of list or string: `len()`
- Get index in list: ?
- Reversing a list: ?

Back 2 Basics

```
sister = "liz"
```

Object

```
height = 1.73
```

Object

```
fam = ["liz", 1.73, "emma", 1.68,  
       "mom", 1.71, "dad", 1.89]
```

Object

Back 2 Basics

```
sister = "liz"
```

```
height = 1.73
```

```
fam = ["liz", 1.73, "emma", 1.68,  
       "mom", 1.71, "dad", 1.89]
```

type

Object str

Object float

Object list

- Methods: Functions that belong to objects

Back 2 Basics

```
sister = "liz"
```

```
height = 1.73
```

```
fam = ["liz", 1.73, "emma", 1.68,  
       "mom", 1.71, "dad", 1.89]
```

	type	examples of methods
Object	str	capitalize() replace()

Object	float	bit_length() conjugate()
--------	-------	-----------------------------

Object	list	index() count()
--------	------	--------------------

- Methods: Functions that belong to objects

list methods

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.index("mom") # "Call method index() on fam"
```

```
4
```

```
fam.count(1.73)
```

```
1
```

str methods

```
sister
```

```
'liz'
```

```
sister.capitalize()
```

```
'Liz'
```

```
sister.replace("z", "sa")
```

```
'lisa'
```

Methods

- Everything = object
- Object have methods associated, depending on type

```
sister.replace("z", "sa")
```

```
'lisa'
```

```
fam.replace("mom", "mommy")
```

```
AttributeError: 'list' object has no attribute 'replace'
```

Methods

```
sister.index("z")
```

2

```
fam.index("mom")
```

4

Methods (2)

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.append("me")
```

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me']
```

```
fam.append(1.79)
```

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me', 1.79]
```

Summary

Functions

```
type(fam)
```

```
list
```

Methods: call functions on objects

```
fam.index("dad")
```

```
6
```

Let's practice!

INTRODUCTION TO PYTHON

Packages

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Motivation

- Functions and methods are powerful
- All code in Python distribution?
 - Huge code base: messy
 - Lots of code you won't use
 - Maintenance problem

Packages

- Directory of Python Scripts
- Each script = module
- Specify functions, methods, types
- Thousands of packages available
 - NumPy
 - Matplotlib
 - scikit-learn

```
pkg/  
    mod1.py  
    mod2.py  
    ...
```

Install package

- <http://pip.readthedocs.org/en/stable/installing/>
- Download `get-pip.py`
- Terminal:
 - `python3 get-pip.py`
 - `pip3 install numpy`

Import package

```
import numpy  
array([1, 2, 3])
```

```
NameError: name 'array' is not defined
```

```
numpy.array([1, 2, 3])
```

```
array([1, 2, 3])
```

```
import numpy as np  
np.array([1, 2, 3])
```

```
array([1, 2, 3])
```

```
from numpy import array  
array([1, 2, 3])
```

```
array([1, 2, 3])
```

from numpy import array

- my_script.py

```
from numpy import array

fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]

...
fam_ext = fam + ["me", 1.79]

...
print(str(len(fam_ext)) + " elements in fam_ext")

...
np_fam = array(fam_ext)
```

- Using NumPy, but not very clear

import numpy

```
import numpy as np

fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]

...
fam_ext = fam + ["me", 1.79]

...
print(str(len(fam_ext)) + " elements in fam_ext")

...
np_fam = np.array(fam_ext) # Clearly using NumPy
```

Let's practice!

INTRODUCTION TO PYTHON

NumPy

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Lists Recap

- Powerful
- Collection of values
- Hold different types
- Change, add, remove
- Need for Data Science
 - Mathematical operations over collections
 - Speed

Illustration

```
height = [1.73, 1.68, 1.71, 1.89, 1.79]  
height
```

```
[1.73, 1.68, 1.71, 1.89, 1.79]
```

```
weight = [65.4, 59.2, 63.6, 88.4, 68.7]  
weight
```

```
[65.4, 59.2, 63.6, 88.4, 68.7]
```

```
weight / height ** 2
```

```
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Solution: NumPy

- Numeric Python
- Alternative to Python List: NumPy Array
- Calculations over entire arrays
- Easy and Fast
- Installation
 - In the terminal: `pip3 install numpy`

NumPy

```
import numpy as np  
np_height = np.array(height)  
np_height
```

```
array([1.73, 1.68, 1.71, 1.89, 1.79])
```

```
np_weight = np.array(weight)  
np_weight
```

```
array([65.4, 59.2, 63.6, 88.4, 68.7])
```

```
bmi = np_weight / np_height ** 2  
bmi
```

```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 , 21.44127836])
```

Comparison

```
height = [1.73, 1.68, 1.71, 1.89, 1.79]  
weight = [65.4, 59.2, 63.6, 88.4, 68.7]  
weight / height ** 2
```

```
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

```
np_height = np.array(height)  
np_weight = np.array(weight)  
np_weight / np_height ** 2
```

```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 , 21.44127836])
```

NumPy: remarks

```
np.array([1.0, "is", True])
```

```
array(['1.0', 'is', 'True'], dtype='<U32')
```

- NumPy arrays: contain only one type

NumPy: remarks

```
python_list = [1, 2, 3]  
numpy_array = np.array([1, 2, 3])
```

```
python_list + python_list
```

```
[1, 2, 3, 1, 2, 3]
```

```
numpy_array + numpy_array
```

```
array([2, 4, 6])
```

- Different types: different behavior!

NumPy Subsetting

```
bmi
```

```
array([21.85171573, 20.97505669, 21.75028214, 24.7473475 , 21.44127836])
```

```
bmi[1]
```

```
20.975
```

```
bmi > 23
```

```
array([False, False, False, True, False])
```

```
bmi[bmi > 23]
```

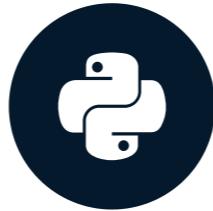
```
array([24.7473475])
```

Let's practice!

INTRODUCTION TO PYTHON

2D NumPy Arrays

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Type of NumPy Arrays

```
import numpy as np  
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])  
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
```

```
type(np_height)
```

```
numpy.ndarray
```

```
type(np_weight)
```

```
numpy.ndarray
```

2D NumPy Arrays

```
np_2d = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],  
                 [65.4, 59.2, 63.6, 88.4, 68.7]])  
  
np_2d
```

```
array([[ 1.73,  1.68,  1.71,  1.89,  1.79],  
       [65.4 , 59.2 , 63.6 , 88.4 , 68.7 ]])
```

```
np_2d.shape
```

```
(2, 5) # 2 rows, 5 columns
```

```
np.array([[1.73, 1.68, 1.71, 1.89, 1.79],  
         [65.4, 59.2, 63.6, 88.4, "68.7"]])
```

```
array([['1.73', '1.68', '1.71', '1.89', '1.79'],  
      ['65.4', '59.2', '63.6', '88.4', '68.7']], dtype='|<U32')
```

Subsetting

```
0      1      2      3      4
```

```
array([[ 1.73,   1.68,   1.71,   1.89,   1.79],  
       [ 65.4,   59.2,   63.6,   88.4,   68.7]])
```

```
np_2d[0]
```

```
array([1.73, 1.68, 1.71, 1.89, 1.79])
```

Subsetting

```
0      1      2      3      4
```

```
array([[ 1.73,   1.68,   1.71,   1.89,   1.79],  
       [ 65.4,   59.2,   63.6,   88.4,   68.7]])
```

```
np_2d[0][2]
```

```
1.71
```

```
np_2d[0, 2]
```

```
1.71
```

Subsetting

```
0      1      2      3      4
```

```
array([[ 1.73,   1.68,   1.71,   1.89,   1.79],  
       [ 65.4,   59.2,   63.6,   88.4,   68.7]])
```

```
np_2d[:, 1:3]
```

```
array([[ 1.68,   1.71],  
       [59.2 ,  63.6 ]])
```

```
np_2d[1, :]
```

```
array([65.4, 59.2, 63.6, 88.4, 68.7])
```

Let's practice!

INTRODUCTION TO PYTHON

NumPy: Basic Statistics

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Data analysis

- Get to know your data
- Little data -> simply look at it
- Big data -> ?

City-wide survey

```
import numpy as np  
np_city = ... # Implementation left out  
np_city
```

```
array([[1.64, 71.78],  
       [1.37, 63.35],  
       [1.6 , 55.09],  
       ...,  
       [2.04, 74.85],  
       [2.04, 68.72],  
       [2.01, 73.57]])
```

NumPy

```
np.mean(np_city[:, 0])
```

```
1.7472
```

```
np.median(np_city[:, 0])
```

```
1.75
```

NumPy

```
np.corrcoef(np_city[:, 0], np_city[:, 1])
```

```
array([[ 1.        , -0.01802],
       [-0.01803,  1.        ]])
```

```
np.std(np_city[:, 0])
```

```
0.1992
```

- sum(), sort(), ...
- Enforce single data type: speed!

Generate data

- Arguments for `np.random.normal()`
 - distribution mean
 - distribution standard deviation
 - number of samples

```
height = np.round(np.random.normal(1.75, 0.20, 5000), 2)
```

```
weight = np.round(np.random.normal(60.32, 15, 5000), 2)
```

```
np_city = np.column_stack((height, weight))
```

Let's practice!

INTRODUCTION TO PYTHON

Basic plots with Matplotlib

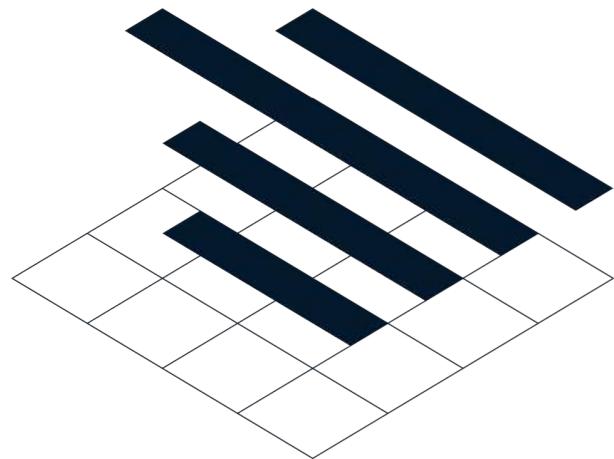
INTERMEDIATE PYTHON



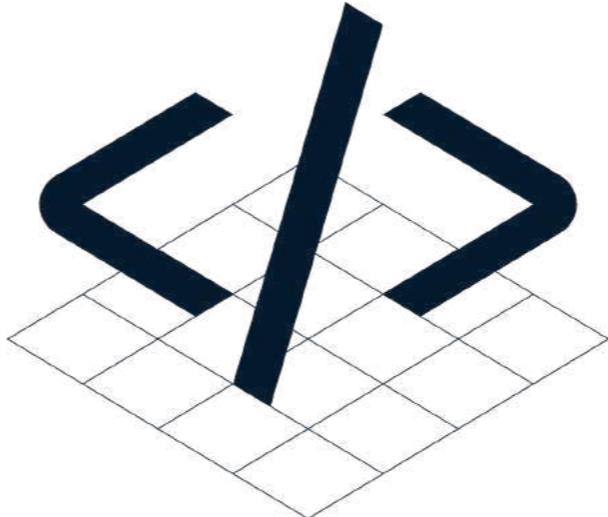
Hugo Bowne-Anderson
Data Scientist at DataCamp

Basic plots with Matplotlib

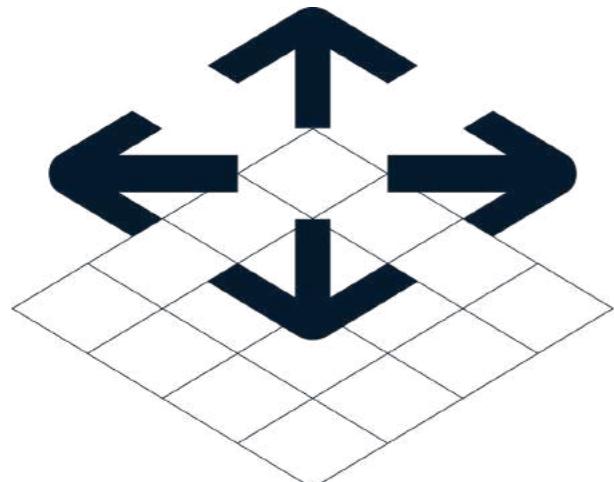
- Visualization



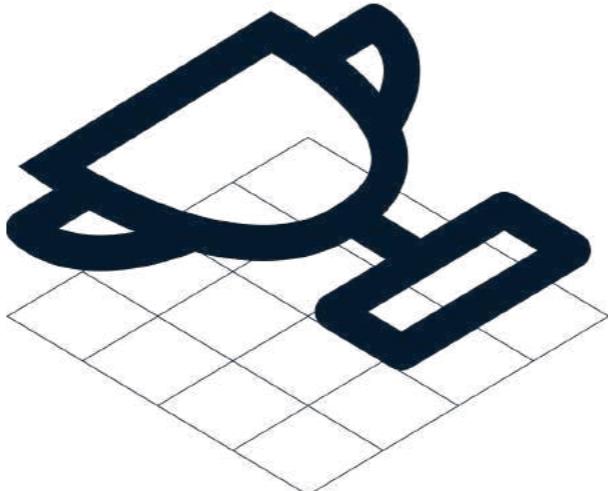
- Data Structure



- Control Structures

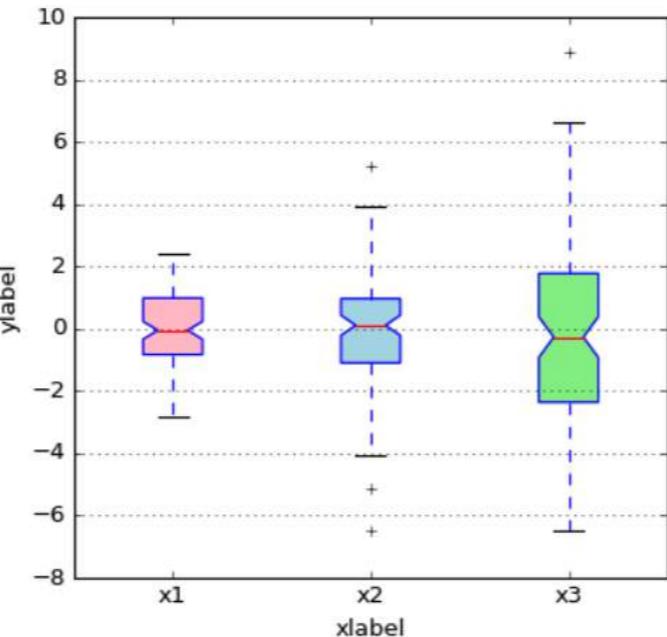
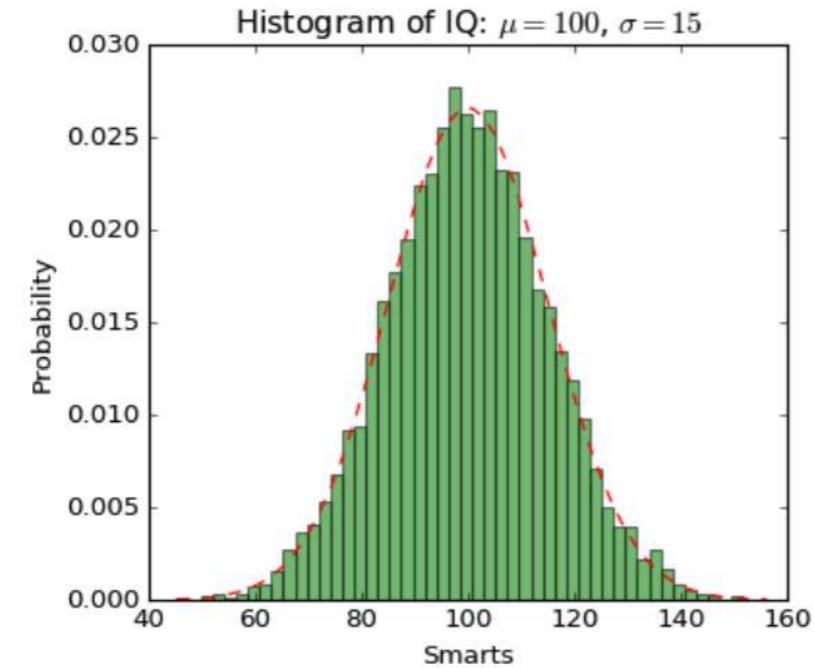


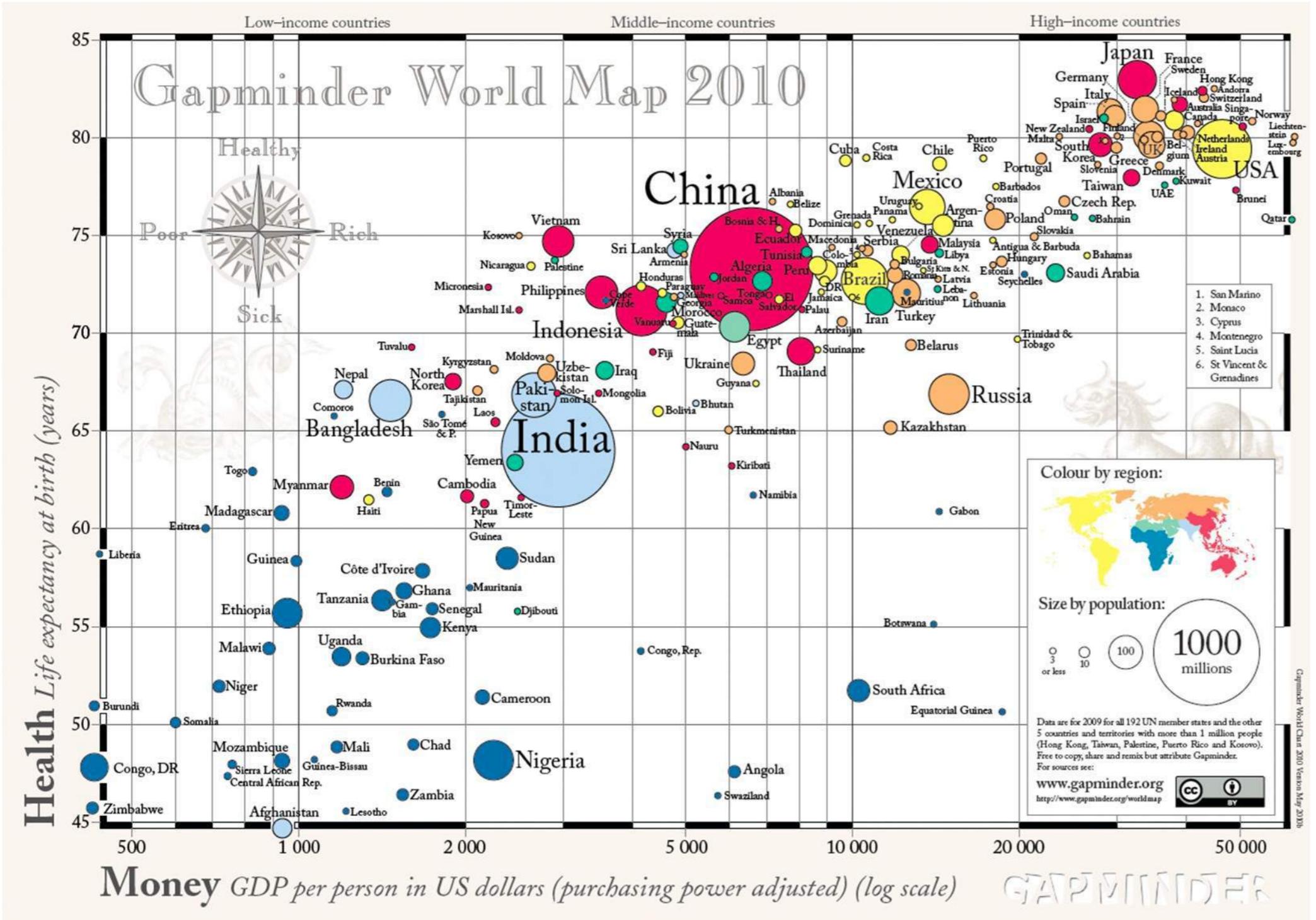
- Case Study



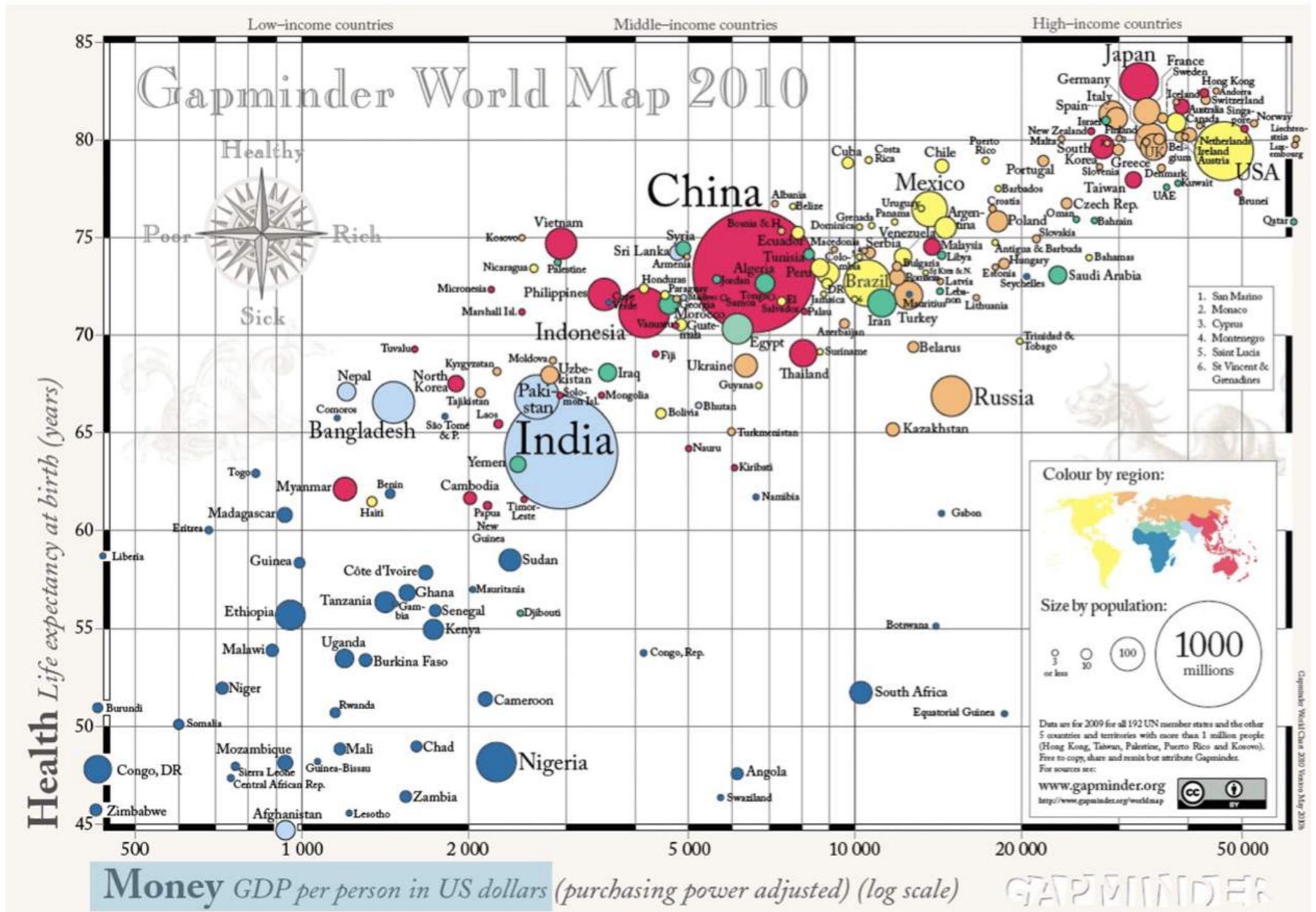
Data visualization

- Very important in Data Analysis
 - Explore data
 - Report insights

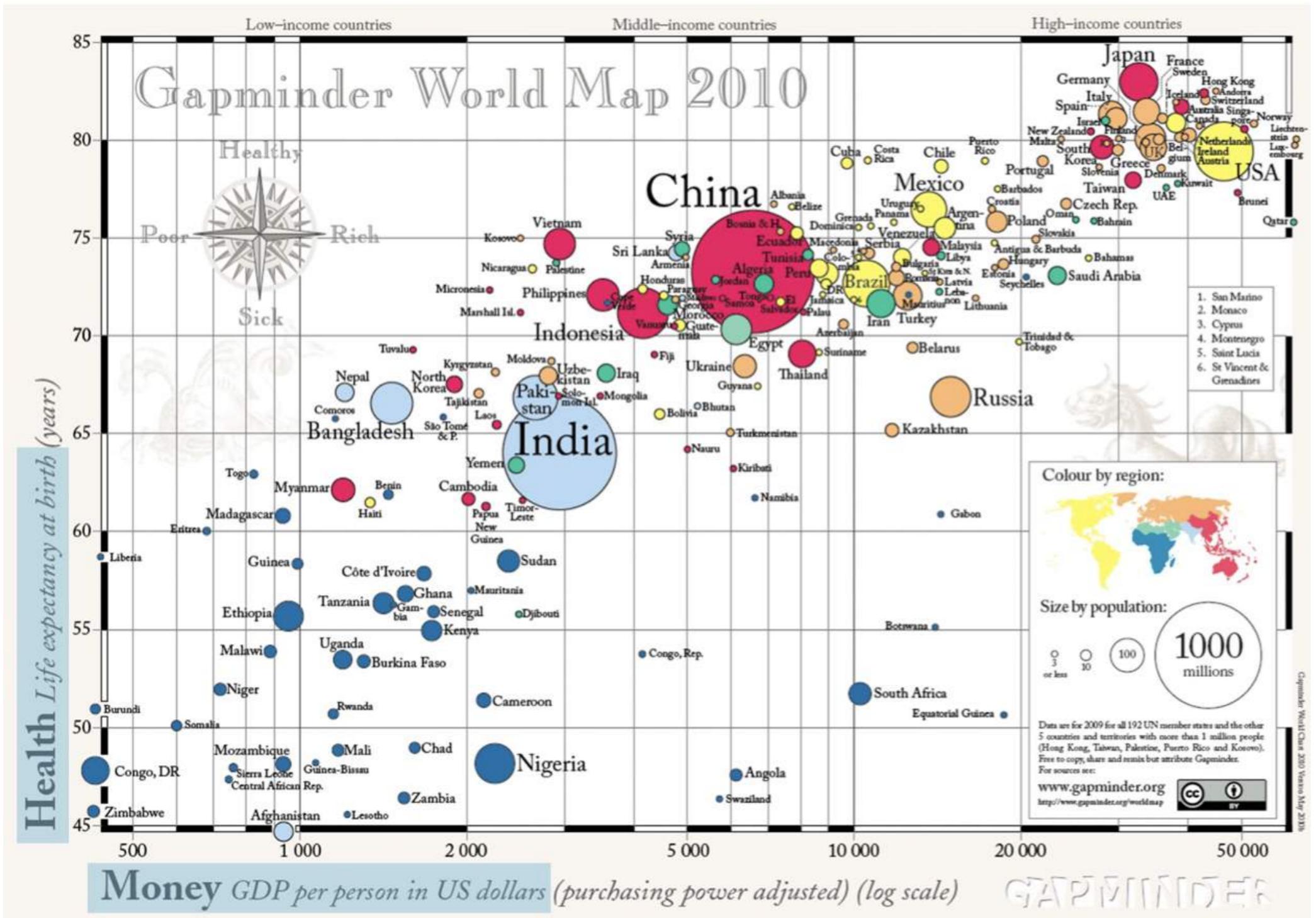




¹ Source: GapMinder, Wealth and Health of Nations



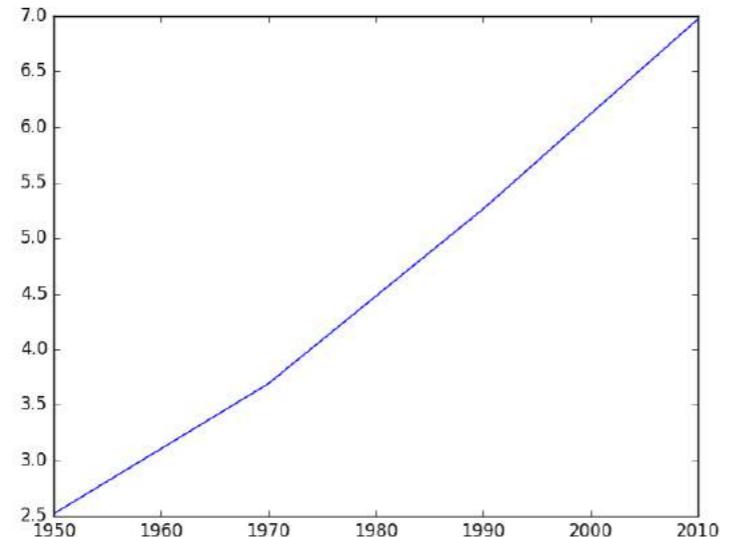
¹ Source: GapMinder, Wealth and Health of Nations



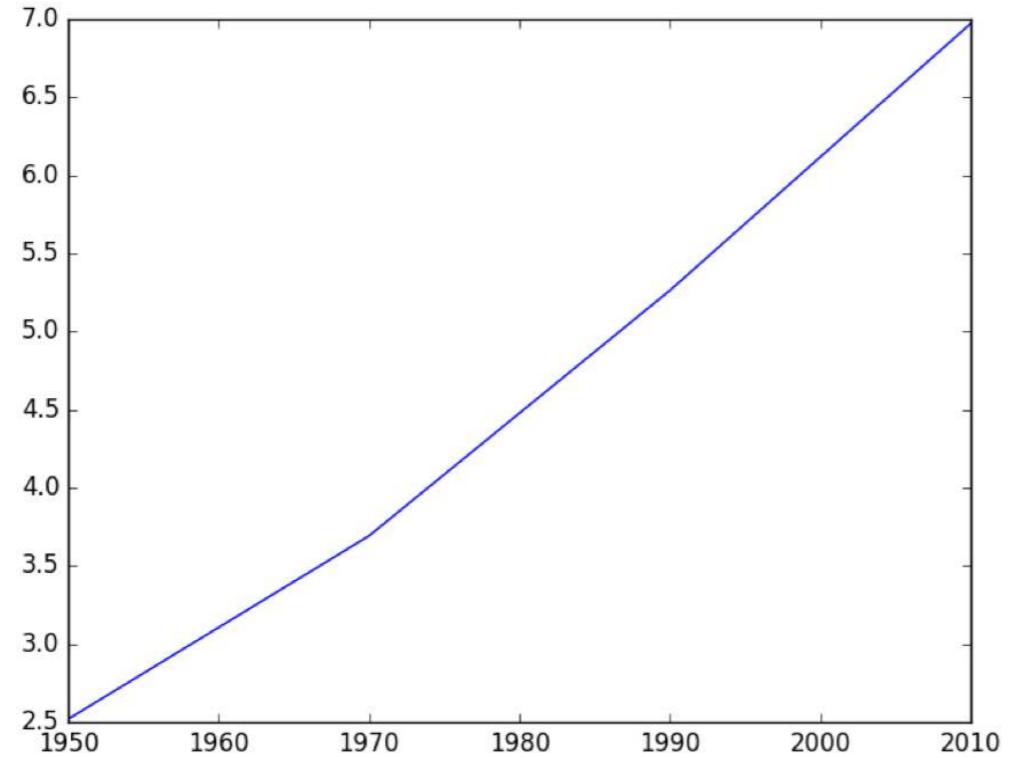
¹ Source: GapMinder, Wealth and Health of Nations

Matplotlib

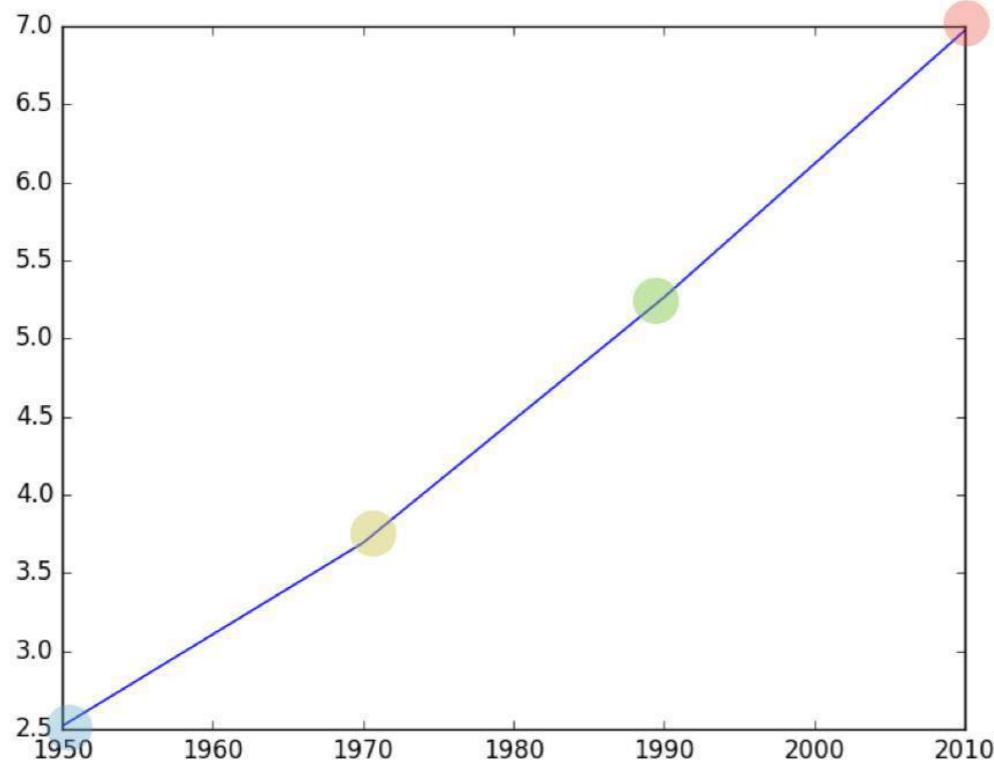
```
import matplotlib.pyplot as plt  
year = [1950, 1970, 1990, 2010]  
pop = [2.519, 3.692, 5.263, 6.972]  
plt.plot(year, pop)  
plt.show()
```



Matplotlib



Matplotlib



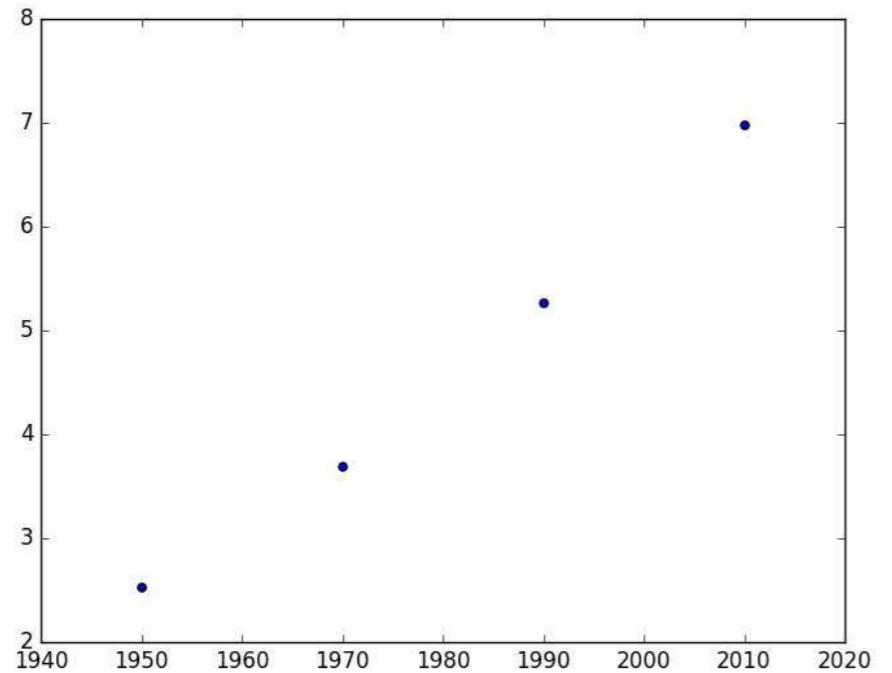
```
year = [1950 , 1970 , 1990 , 2010]  
pop = [2.519, 3.692, 5.263, 6.972]
```

Scatter plot

```
import matplotlib.pyplot as plt  
year = [1950, 1970, 1990, 2010]  
pop = [2.519, 3.692, 5.263, 6.972]  
plt.plot(year, pop)  
plt.show()
```

Scatter plot

```
import matplotlib.pyplot as plt  
year = [1950, 1970, 1990, 2010]  
pop = [2.519, 3.692, 5.263, 6.972]  
plt.scatter(year, pop)  
plt.show()
```

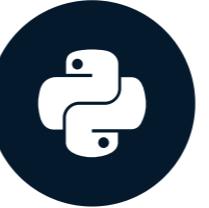


Let's practice!

INTERMEDIATE PYTHON

Histogram

INTERMEDIATE PYTHON



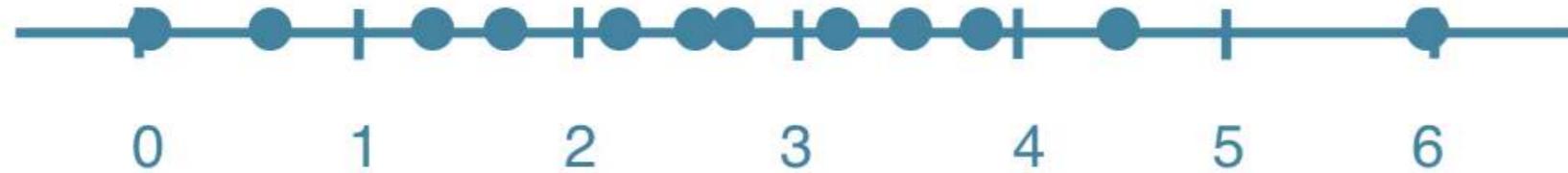
Hugo Bowne-Anderson
Data Scientist at DataCamp

Histogram

- Explore dataset
- Get idea about distribution

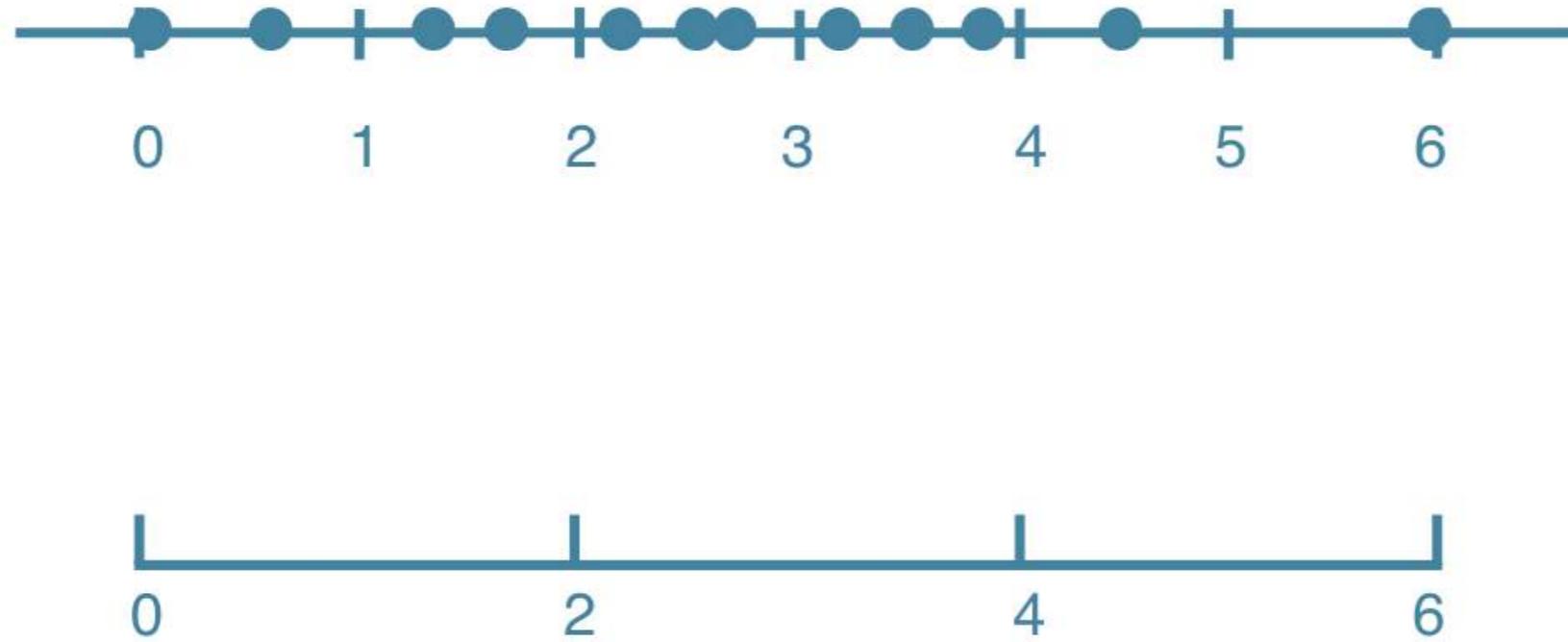
Histogram

- Explore dataset
- Get idea about distribution



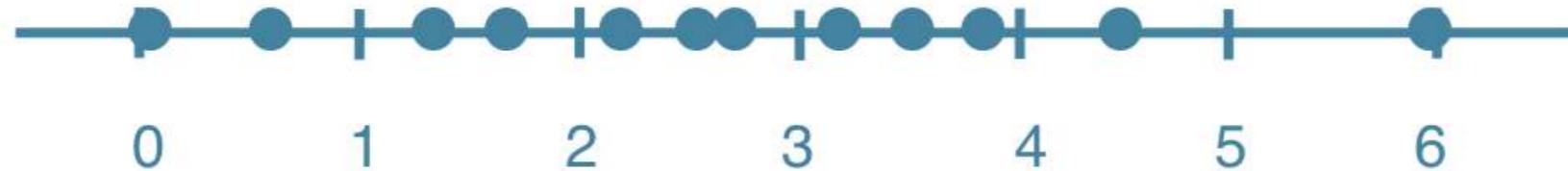
Histogram

- Explore dataset
- Get idea about distribution



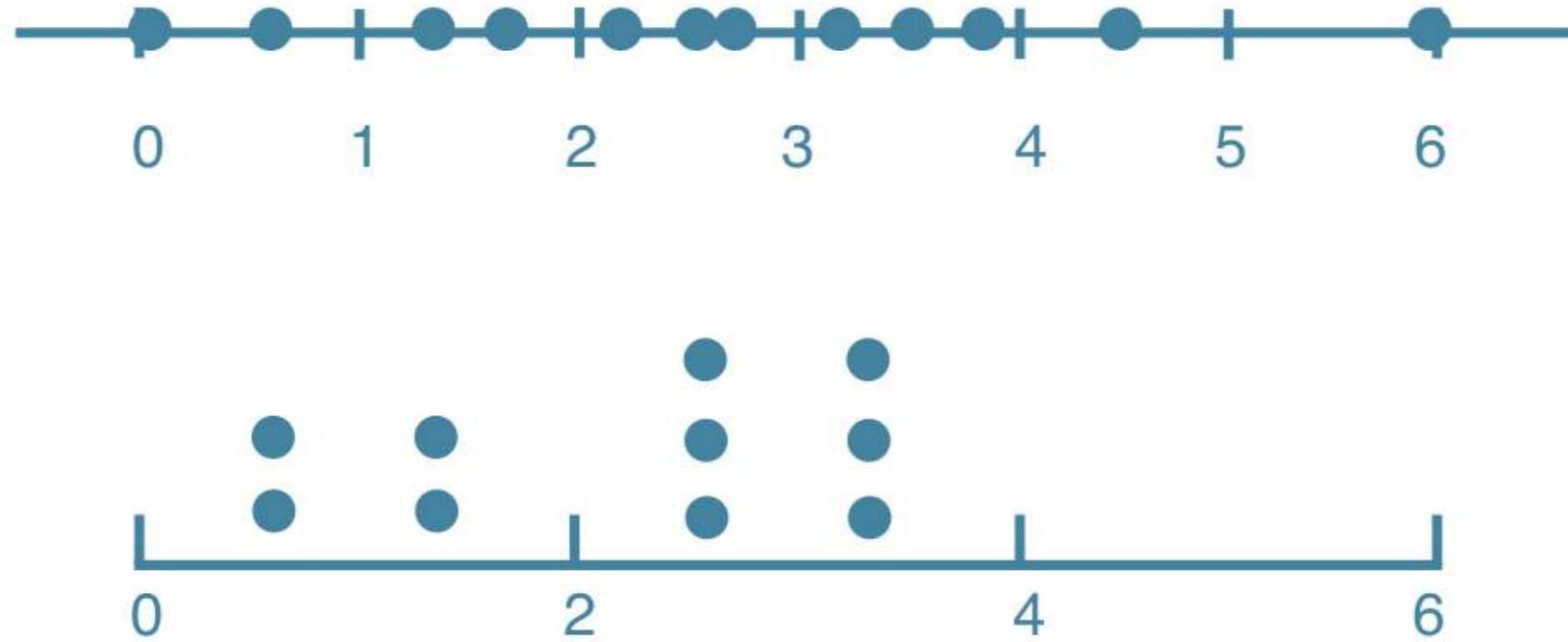
Histogram

- Explore dataset
- Get idea about distribution



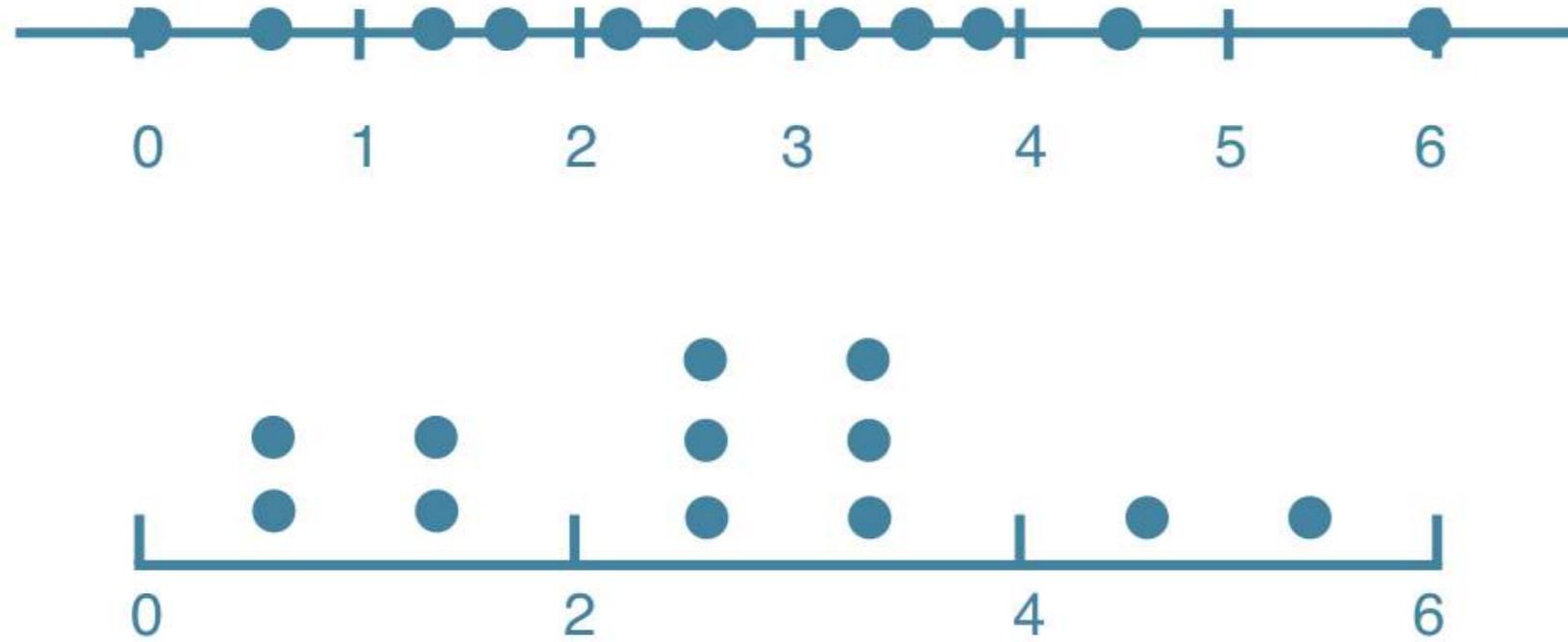
Histogram

- Explore dataset
- Get idea about distribution



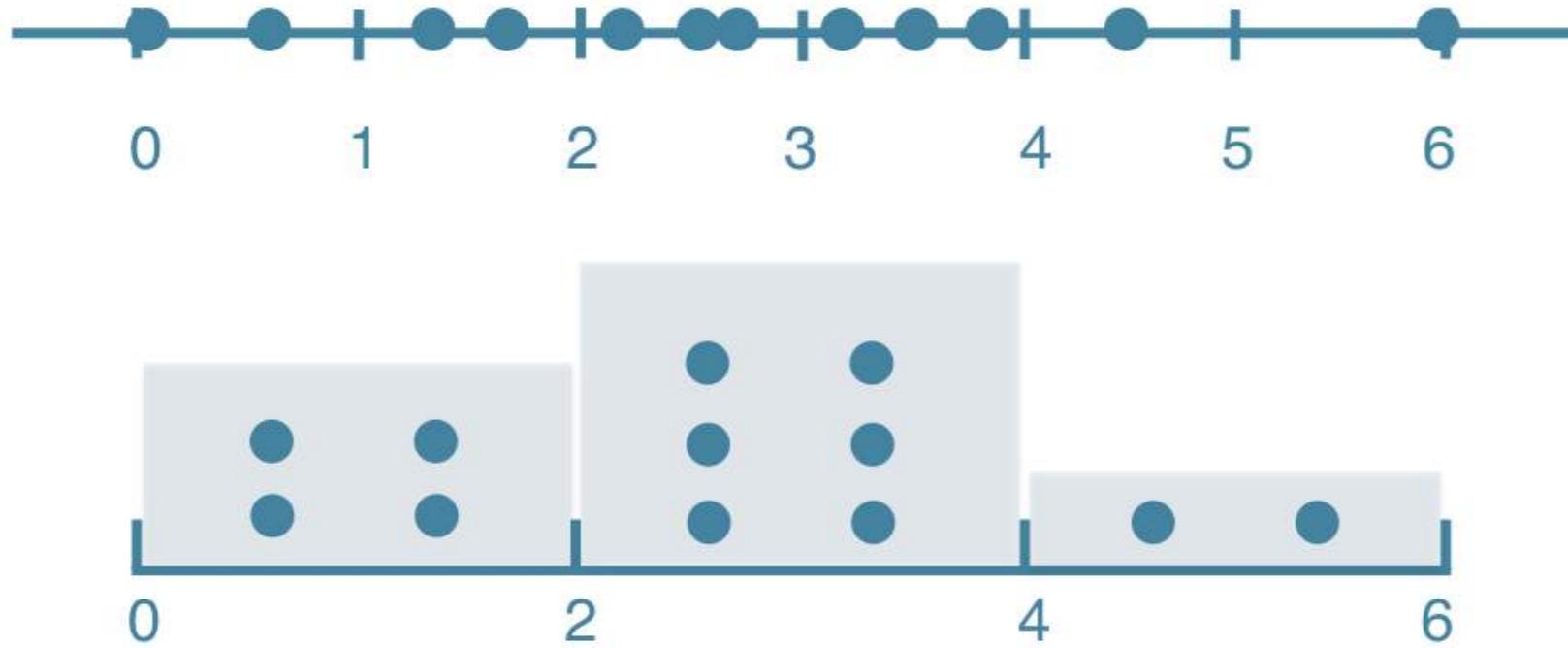
Histogram

- Explore dataset
- Get idea about distribution



Histogram

- Explore dataset
- Get idea about distribution



Matplotlib

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

```
help(plt.hist)
```

Help on function hist in module matplotlib.pyplot:

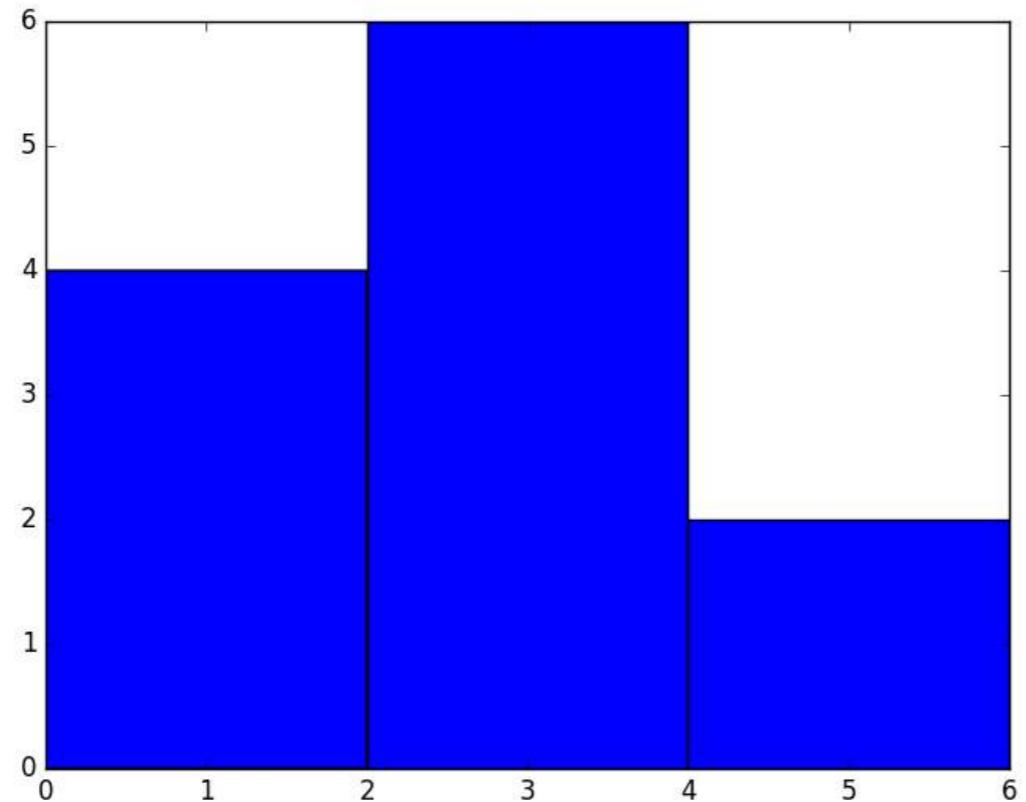
```
hist(x, bins=None, range=None, density=False, weights=None,  
cumulative=False, bottom=None, histtype='bar', align='mid',  
orientation='vertical', rwidth=None, log=False, color=None,  
label=None, stacked=False, *, data=None, **kwargs)
```

Plot a histogram.

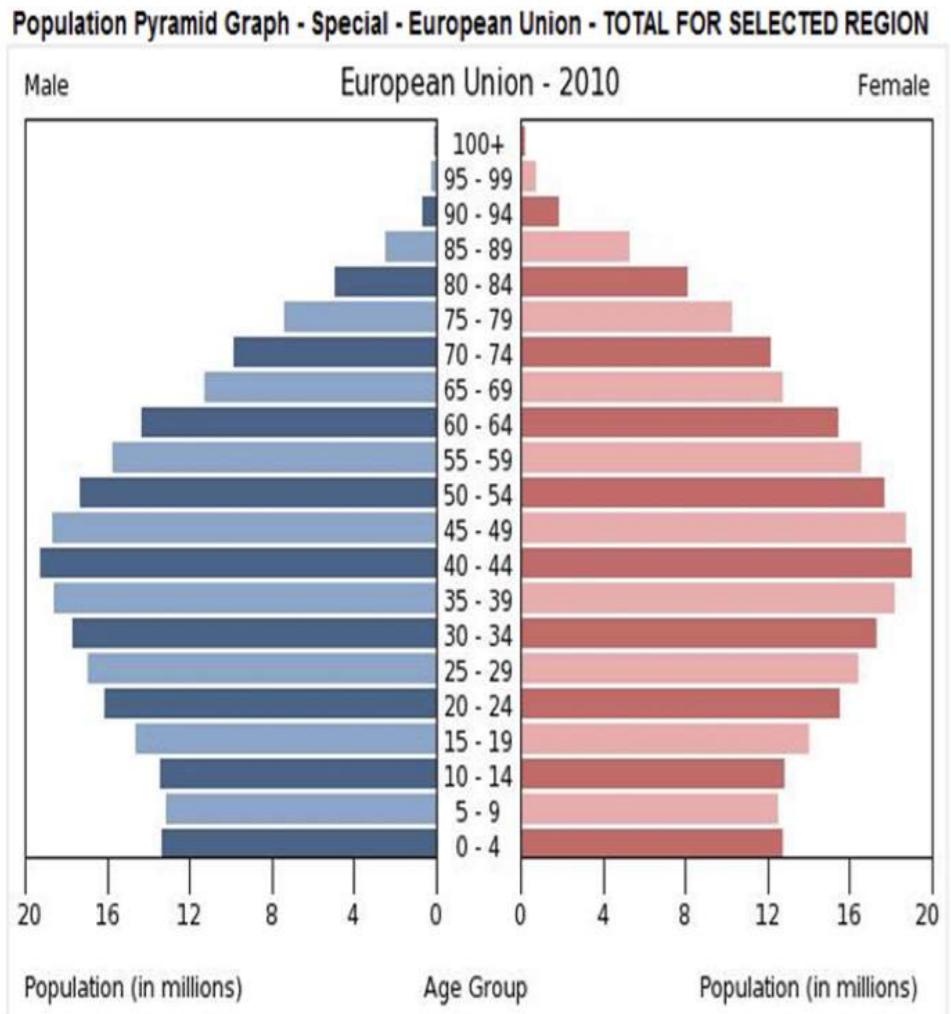
Compute and draw the histogram of `*x*`. The return value is a tuple `(*n*, *bins*, *patches*)` or `([*n0*, *n1*, ...], *bins*, [*patches0*, *patches1*, ...])` if the input contains multiple data.

Matplotlib example

```
values = [0,0.6,1.4,1.6,2.2,2.5,2.6,3.2,3.5,3.9,4.2,6]  
plt.hist(values, bins=3)  
plt.show()
```

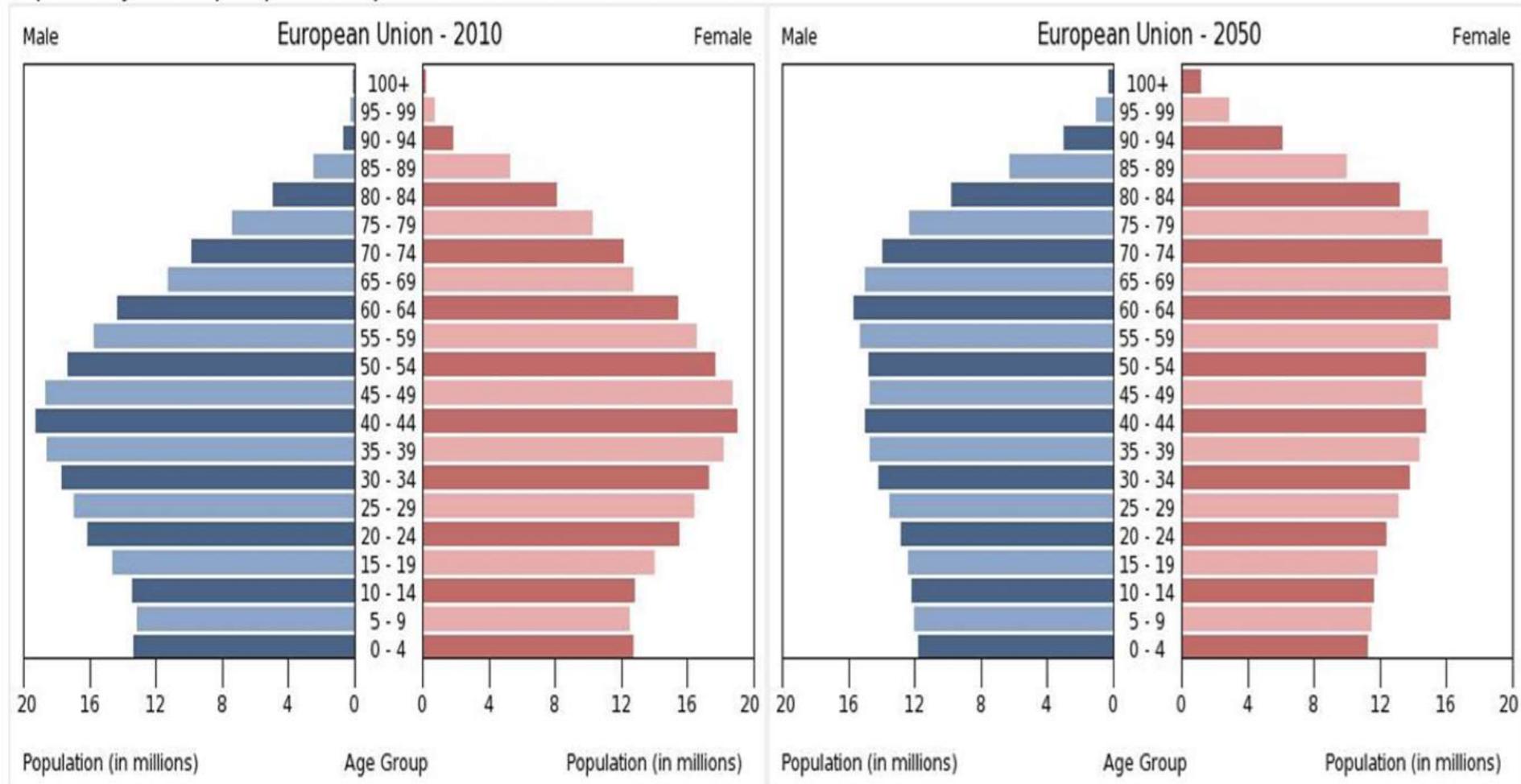


Population pyramid



Population pyramid

Population Pyramid Graph - Special - European Union - TOTAL FOR SELECTED REGION



Let's practice!

INTERMEDIATE PYTHON

Customization

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Data visualization

- Many options
 - Different plot types
 - Many customizations
- Choice depends on
 - Data
 - Story you want to tell

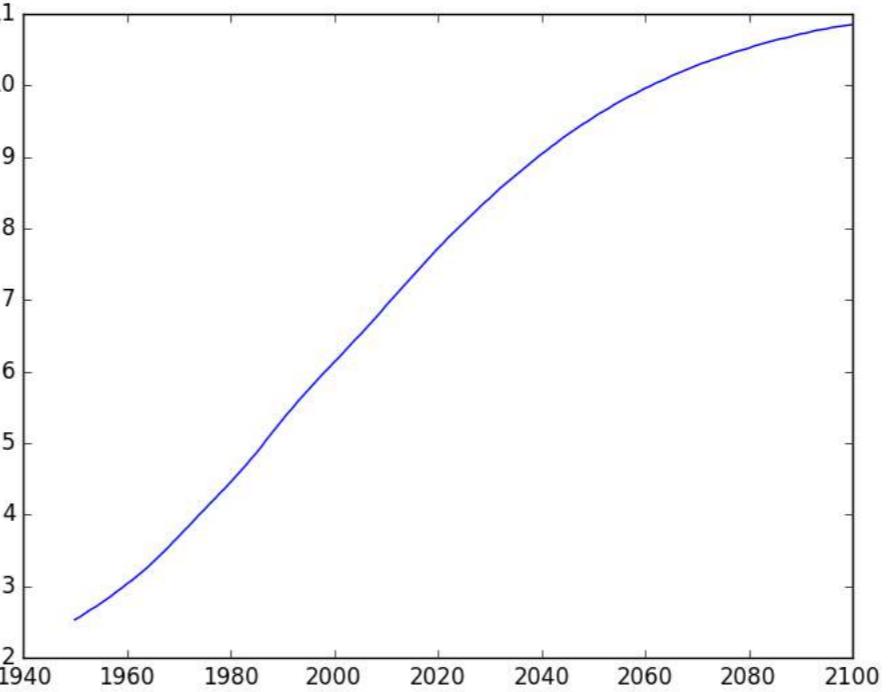
Basic plot

population.py

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.show()
```



Axis labels

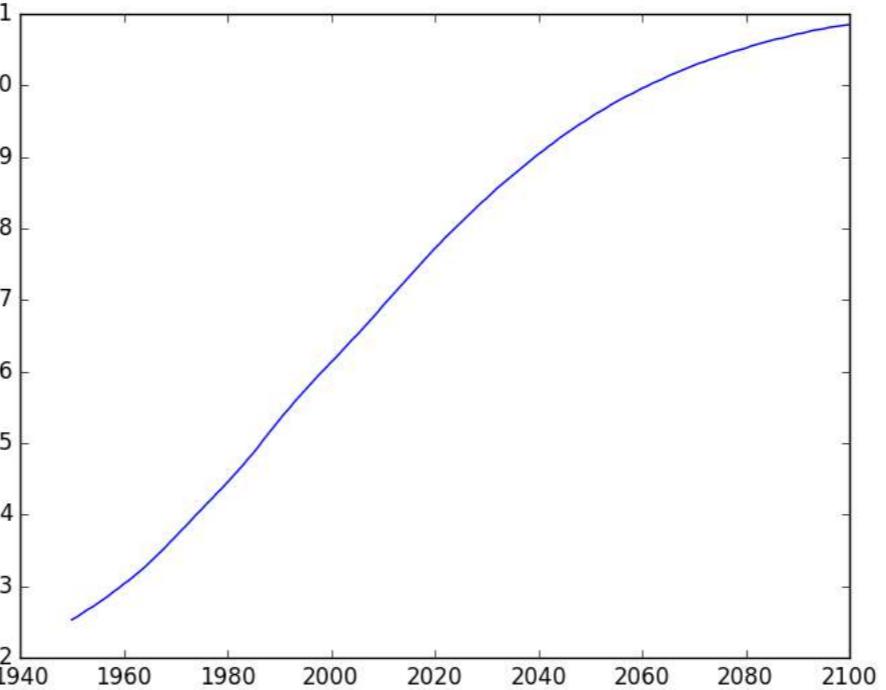
population.py

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')

plt.show()
```



Axis labels

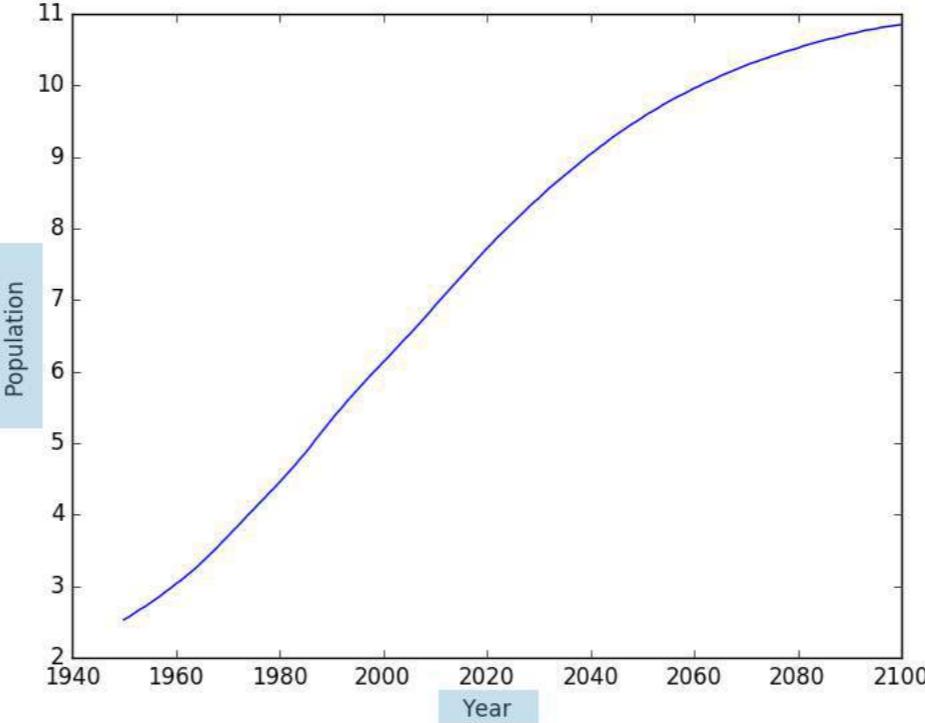
population.py

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')

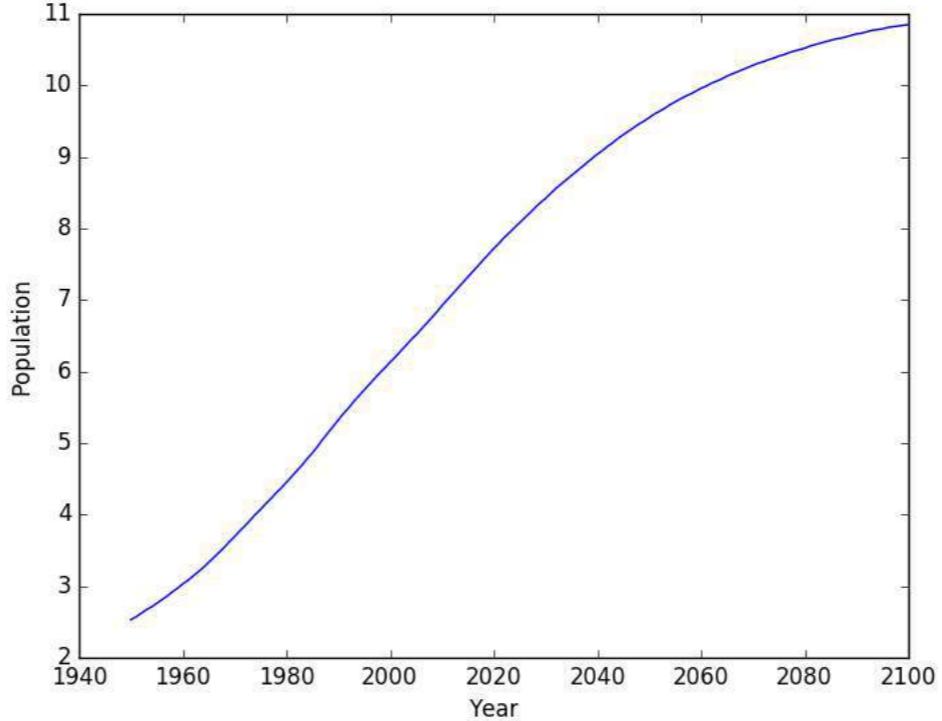
plt.show()
```



Title

population.py

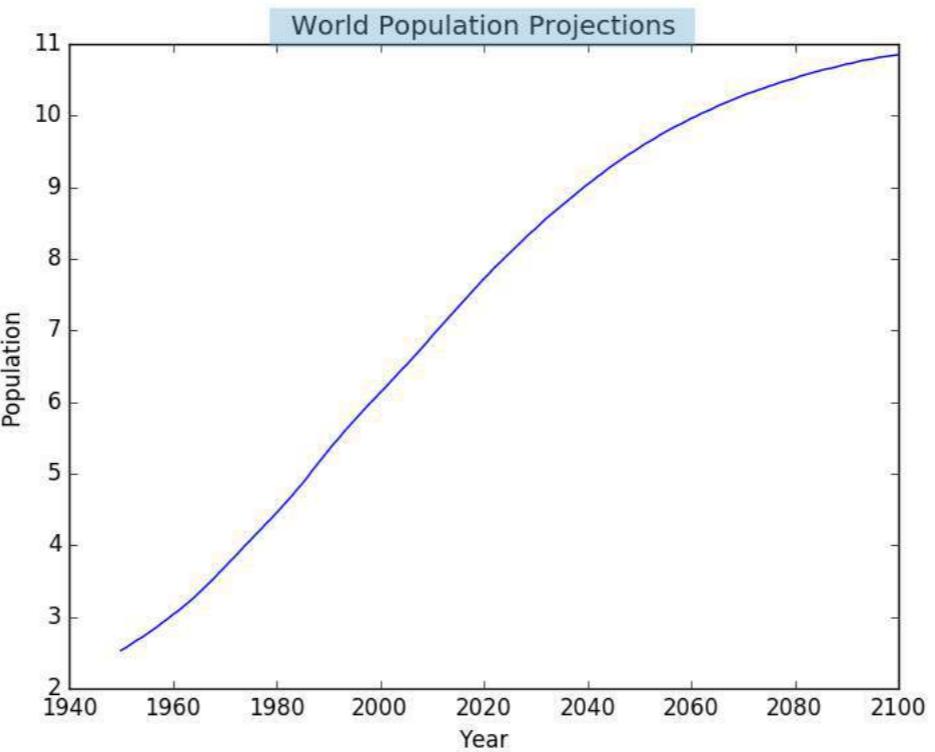
```
import matplotlib.pyplot as plt  
year = [1950, 1951, 1952, ..., 2100]  
pop = [2.538, 2.57, 2.62, ..., 10.85]  
  
plt.plot(year, pop)  
  
plt.xlabel('Year')  
plt.ylabel('Population')  
plt.title('World Population Projections')  
  
plt.show()
```



Title

population.py

```
import matplotlib.pyplot as plt  
year = [1950, 1951, 1952, ..., 2100]  
pop = [2.538, 2.57, 2.62, ..., 10.85]  
  
plt.plot(year, pop)  
  
plt.xlabel('Year')  
plt.ylabel('Population')  
plt.title('World Population Projections')  
  
plt.show()
```



Ticks

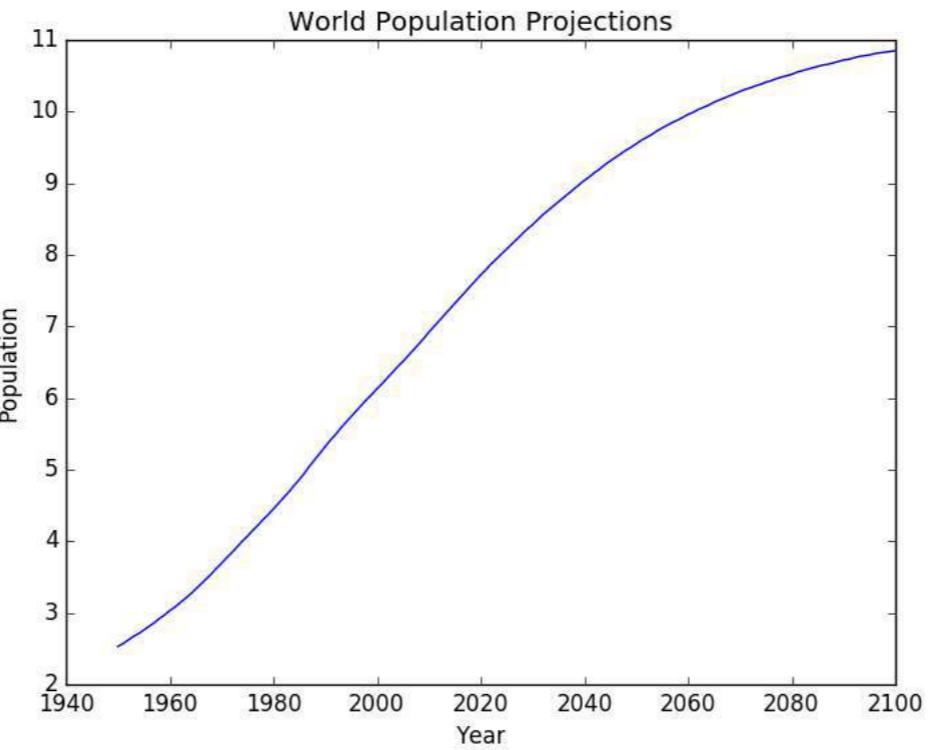
population.py

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10])

plt.show()
```



Ticks

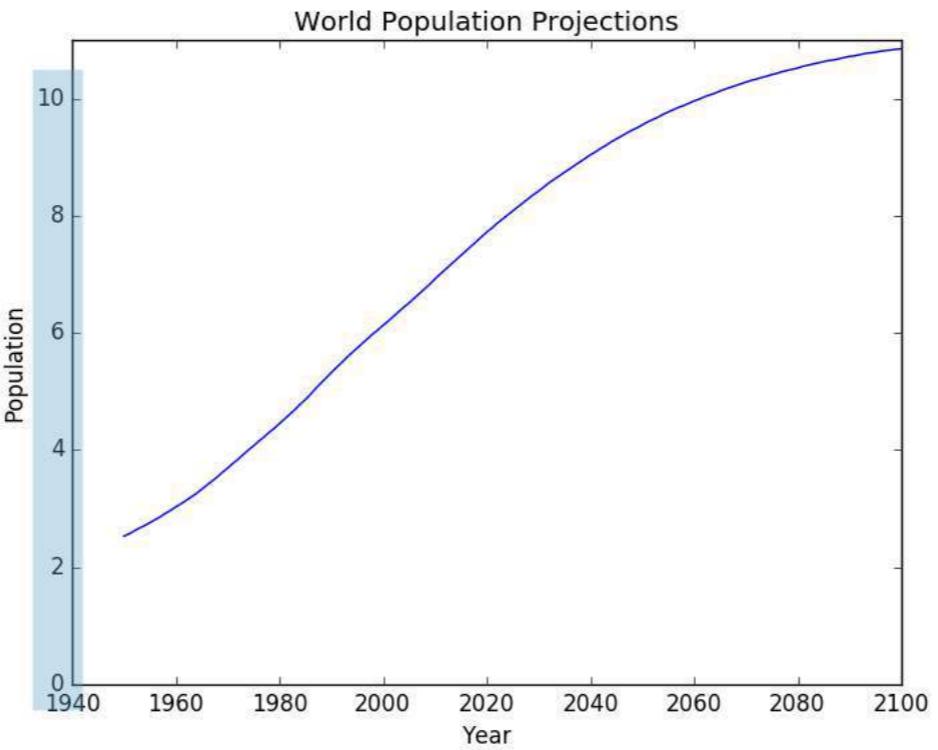
population.py

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10])

plt.show()
```



Ticks (2)

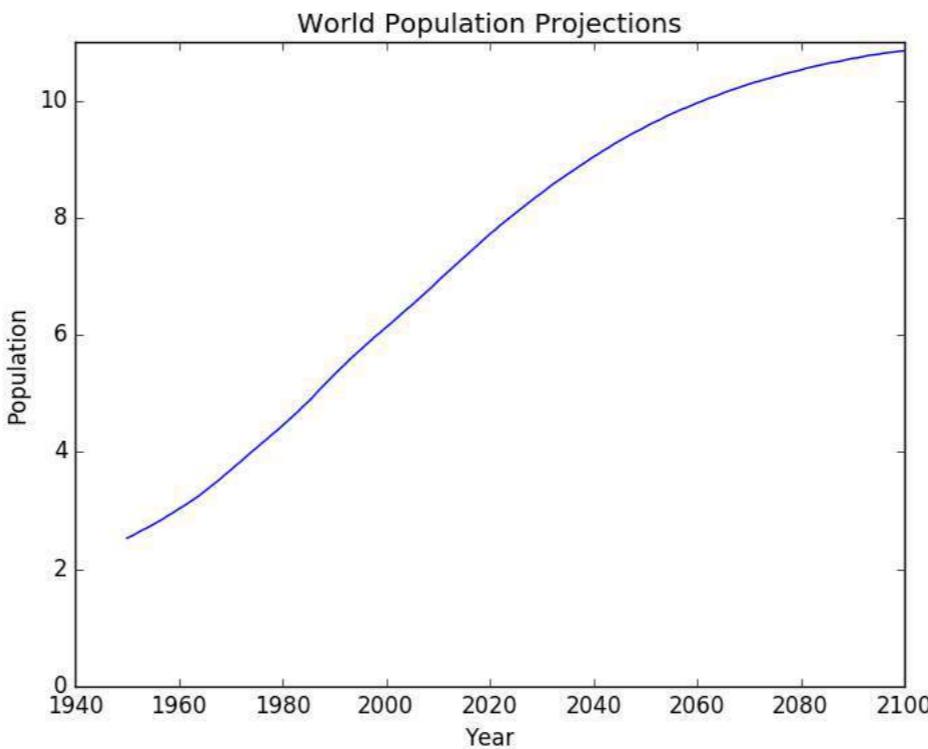
population.py

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10],
           ['0', '2B', '4B', '6B', '8B', '10B'])

plt.show()
```



Ticks (2)

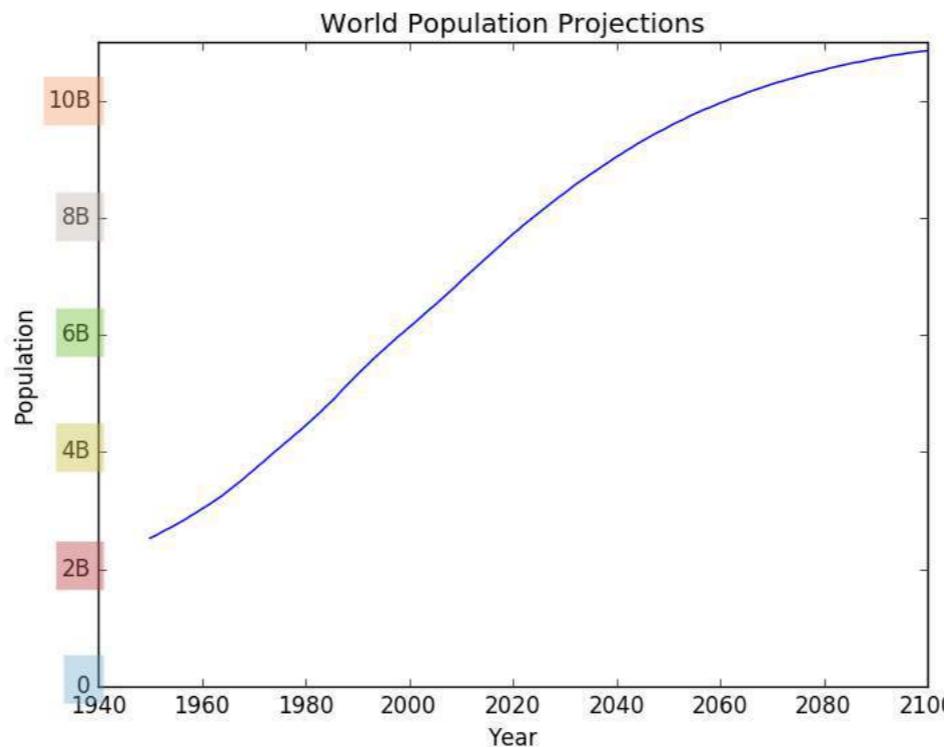
population.py

```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10],
           ['0', '2B', '4B', '6B', '8B', '10B'])

plt.show()
```



Add historical data

population.py

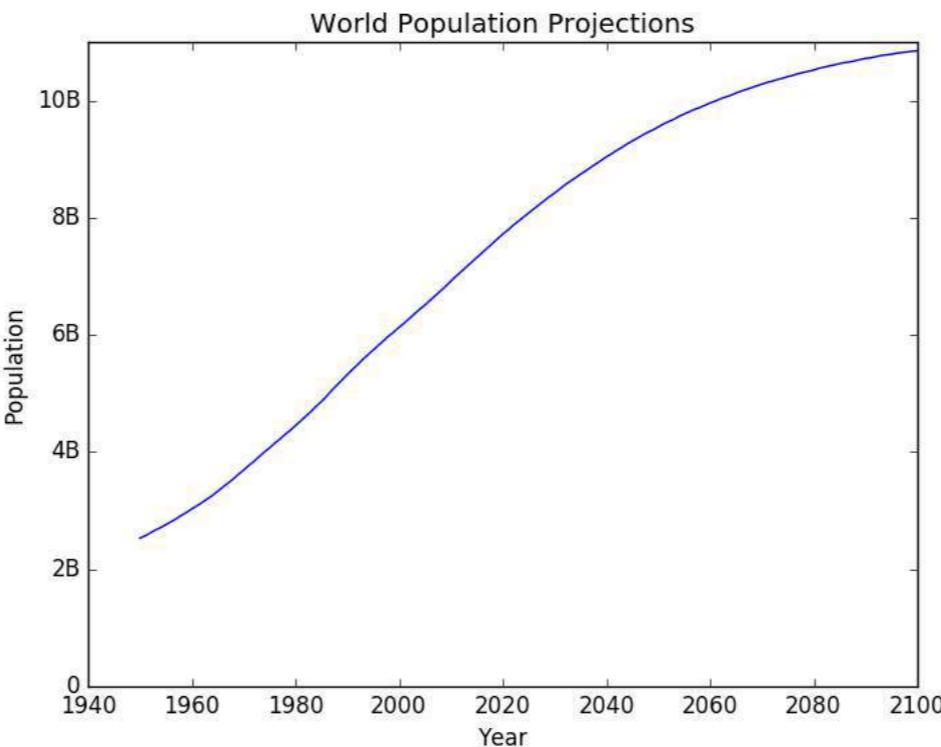
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

# Add more data
year = [1800, 1850, 1900] + year
pop = [1.0, 1.262, 1.650] + pop

plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10],
           ['0', '2B', '4B', '6B', '8B', '10B'])

plt.show()
```



Add historical data

population.py

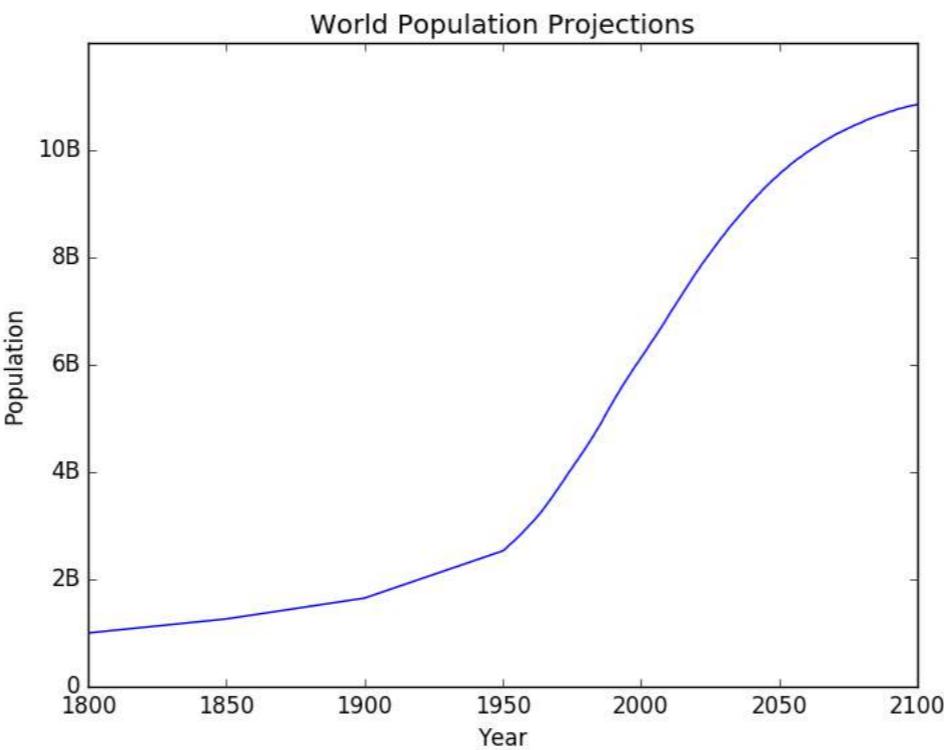
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

# Add more data
year = [1800, 1850, 1900] + year
pop = [1.0, 1.262, 1.650] + pop

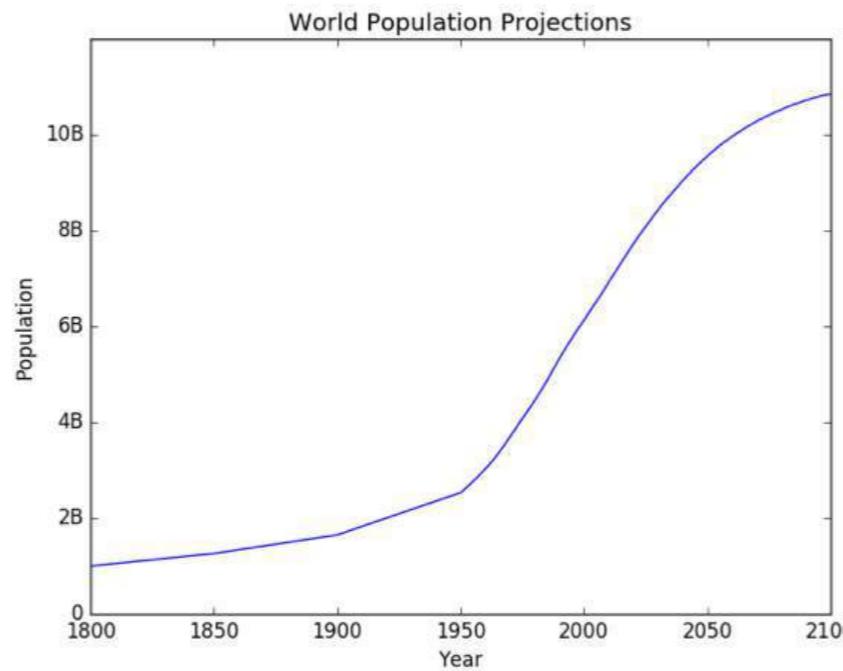
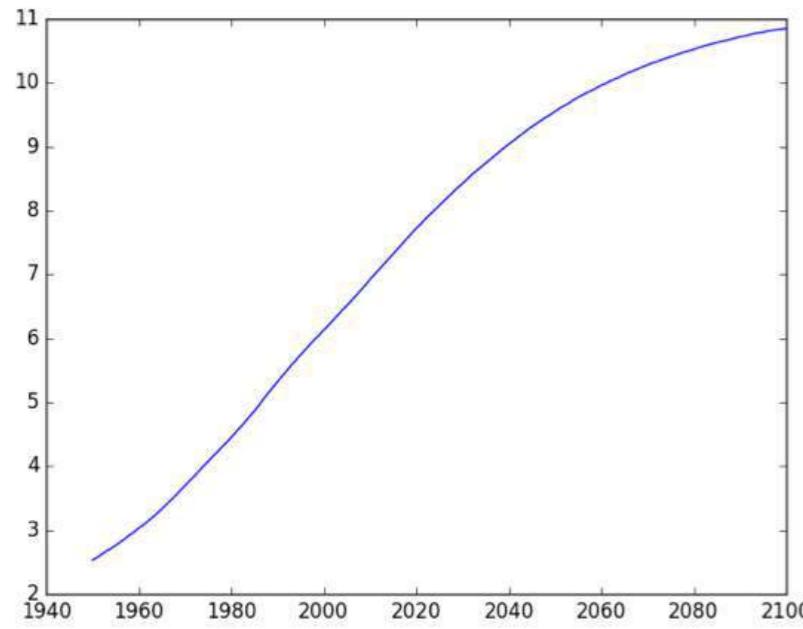
plt.plot(year, pop)

plt.xlabel('Year')
plt.ylabel('Population')
plt.title('World Population Projections')
plt.yticks([0, 2, 4, 6, 8, 10],
           ['0', '2B', '4B', '6B', '8B', '10B'])

plt.show()
```



Before vs. after

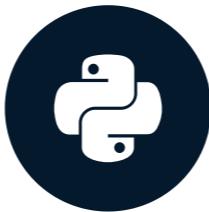


Let's practice!

INTERMEDIATE PYTHON

Dictionaries, Part 1

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

List

```
pop = [30.55, 2.77, 39.21]
countries = ["afghanistan", "albania", "algeria"]
ind_alb = countries.index("albania")
ind_alb
```

1

```
pop[ind_alb]
```

2.77

- Not convenient
- Not intuitive

Dictionary

```
pop = [30.55, 2.77, 39.21]  
countries = ["afghanistan", "albania", "algeria"]
```

...

{

}

Dictionary

```
pop = [30.55, 2.77, 39.21]  
countries = ["afghanistan", "albania", "algeria"]
```

...

```
{"afghanistan":30.55, }
```

Dictionary

```
pop = [30.55, 2.77, 39.21]
countries = ["afghanistan", "albania", "algeria"]

...
world = {"afghanistan":30.55, "albania":2.77, "algeria":39.21}
world["albania"]
```

```
2.77
```

Let's practice!

INTERMEDIATE PYTHON

Dictionaries, Part 2

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Recap

```
world = {"afghanistan":30.55, "albania":2.77, "algeria":39.21}  
world["albania"]
```

```
2.77
```

```
world = {"afghanistan":30.55, "albania":2.77,  
         "algeria":39.21, "albania":2.81}  
world
```

```
{'afghanistan': 30.55, 'albania': 2.81, 'algeria': 39.21}
```

Recap

- Keys have to be "immutable" objects

```
{0:"hello", True:"dear", "two":"world"}
```

```
{0: 'hello', True: 'dear', 'two': 'world'}
```

```
{"just", "to", "test": "value"}
```

```
TypeError: unhashable type: 'list'
```

Principality of Sealand



¹ Source: Wikipedia

Dictionary

```
world["sealand"] = 0.000027  
world
```

```
{'afghanistan': 30.55, 'albania': 2.81,  
'algeria': 39.21, 'sealand': 2.7e-05}
```

```
"sealand" in world
```

```
True
```

Dictionary

```
world["sealand"] = 0.000028  
world
```

```
{'afghanistan': 30.55, 'albania': 2.81,  
 'algeria': 39.21, 'sealand': 2.8e-05}
```

```
del(world["sealand"])  
world
```

```
{'afghanistan': 30.55, 'albania': 2.81, 'algeria': 39.21}
```

List vs. Dictionary

List vs. Dictionary

List vs. Dictionary

List	Dictionary
Select, update, and remove with <code>[]</code>	Select, update, and remove with <code>[]</code>

List vs. Dictionary

List	Dictionary
Select, update, and remove with <code>[]</code>	Select, update, and remove with <code>[]</code>

List vs. Dictionary

List	Dictionary
Select, update, and remove with <code>[]</code>	Select, update, and remove with <code>[]</code>
Indexed by range of numbers	

List vs. Dictionary

List	Dictionary
Select, update, and remove with <code>[]</code>	Select, update, and remove with <code>[]</code>
Indexed by range of numbers	Indexed by unique keys

List vs. Dictionary

List	Dictionary
Select, update, and remove with <code>[]</code>	Select, update, and remove with <code>[]</code>
Indexed by range of numbers	Indexed by unique keys
Collection of values — order matters, for selecting entire subsets	

List vs. Dictionary

List	Dictionary
Select, update, and remove with <code>[]</code>	Select, update, and remove with <code>[]</code>
Indexed by range of numbers	Indexed by unique keys
Collection of values — order matters, for selecting entire subsets	Lookup table with unique keys

Let's practice!

INTERMEDIATE PYTHON

Pandas, Part 1

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Tabular dataset examples

temperature	measured_at	location
76	2016-01-01 14:00:01	valve
86	2016-01-01 14:00:01	compressor
72	2016-01-01 15:00:01	valve
88	2016-01-01 15:00:01	compressor
68	2016-01-01 16:00:01	valve
78	2016-01-01 16:00:01	compressor

Tabular dataset examples

temperature	measured_at	location
76	2016-01-01 14:00:01	valve
86	2016-01-01 14:00:01	compressor
72	2016-01-01 15:00:01	valve
88	2016-01-01 15:00:01	compressor
68	2016-01-01 16:00:01	valve
78	2016-01-01 16:00:01	compressor

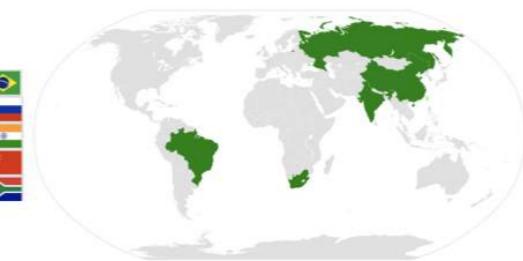
row = observations
column = variable

Tabular dataset examples

temperature	measured_at	location
76	2016-01-01 14:00:01	valve
86	2016-01-01 14:00:01	compressor
72	2016-01-01 15:00:01	valve
88	2016-01-01 15:00:01	compressor
68	2016-01-01 16:00:01	valve
78	2016-01-01 16:00:01	compressor

row = observations
column = variable

country	capital	area	population
Brazil	Brasilia	8.516	200.4
Russia	Moscow	17.10	143.5
India	New Delhi	3.286	1252
China	Beijing	9.597	1357
South Africa	Pretoria	1.221	52.98



Datasets in Python

- 2D NumPy array?
 - One data type

Datasets in Python

country	capital	area	population
Brazil	Brasilia	8.516	200.4
Russia	Moscow	17.10	143.5
India	New Delhi	3.286	1252
China	Beijing	9.597	1357
South	Pretoria	1.221	52.98

float float

Datasets in Python

country	capital	area	population
Brazil	Brasilia	8.516	200.4
Russia	Moscow	17.10	143.5
India	New Delhi	3.286	1252
China	Beijing	9.597	1357
South	Pretoria	1.221	52.98

str str float float

- pandas!
 - High level data manipulation tool
 - Wes McKinney
 - Built on NumPy
 - DataFrame

DataFrame

brics

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

DataFrame from Dictionary

```
dict = {  
    "country": ["Brazil", "Russia", "India", "China", "South Africa"],  
    "capital": ["Brasilia", "Moscow", "New Delhi", "Beijing", "Pretoria"],  
    "area": [8.516, 17.10, 3.286, 9.597, 1.221]  
    "population": [200.4, 143.5, 1252, 1357, 52.98] }
```

- keys (column labels)
- values (data, column by column)

```
import pandas as pd  
brics = pd.DataFrame(dict)
```

DataFrame from Dictionary (2)

```
brics
```

```
    area      capital      country  population
0   8.516    Brasilia     Brazil       200.40
1  17.100    Moscow      Russia      143.50
2   3.286  New Delhi    India      1252.00
3   9.597    Beijing     China      1357.00
4   1.221  Pretoria  South Africa     52.98
```

```
brics.index = ["BR", "RU", "IN", "CH", "SA"]
brics
```

```
    area      capital      country  population
BR   8.516    Brasilia     Brazil       200.40
RU  17.100    Moscow      Russia      143.50
IN   3.286  New Delhi    India      1252.00
CH   9.597    Beijing     China      1357.00
SA   1.221  Pretoria  South Africa     52.98
```

DataFrame from CSV file

brics.csv

```
,country,capital,area,population  
BR,Brazil,Brasilia,8.516,200.4  
RU,Russia,Moscow,17.10,143.5  
IN,India,New Delhi,3.286,1252  
CH,China,Beijing,9.597,1357  
SA,South Africa,Pretoria,1.221,52.98
```

- CSV = comma-separated values

DataFrame from CSV file

- `brics.csv`

```
,country,capital,area,population  
BR,Brazil,Brasilia,8.516,200.4  
RU,Russia,Moscow,17.10,143.5  
IN,India,New Delhi,3.286,1252  
CH,China,Beijing,9.597,1357  
SA,South Africa,Pretoria,1.221,52.98
```

```
brics = pd.read_csv("path/to/brics.csv")  
brics
```

```
  Unnamed: 0      country    capital     area  population  
0        BR        Brazil   Brasilia  8.516      200.40  
1        RU       Russia   Moscow  17.100      143.50  
2        IN        India  New Delhi  3.286     1252.00  
3        CH        China   Beijing  9.597     1357.00  
4        SA  South Africa  Pretoria  1.221      52.98
```

DataFrame from CSV file

```
brics = pd.read_csv("path/to/brics.csv", index_col = 0)  
brics
```

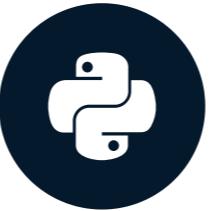
	country	population	area	capital
BR	Brazil	200	8515767	Brasilia
RU	Russia	144	17098242	Moscow
IN	India	1252	3287590	New Delhi
CH	China	1357	9596961	Beijing
SA	South Africa	55	1221037	Pretoria

Let's practice!

INTERMEDIATE PYTHON

Pandas, Part 2

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

brics

```
import pandas as pd  
brics = pd.read_csv("path/to/brics.csv", index_col = 0)  
brics
```

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Index and select data

- Square brackets
- Advanced methods
 - loc
 - iloc

Column Access []

```
country    capital     area  population
BR          Brazil      Brasilia   8.516      200.40
RU          Russia      Moscow    17.100      143.50
IN          India       New Delhi  3.286      1252.00
CH          China       Beijing   9.597      1357.00
SA  South Africa  Pretoria   1.221       52.98
```

```
brics["country"]
```

```
BR          Brazil
RU          Russia
IN          India
CH          China
SA  South Africa
Name: country, dtype: object
```

Column Access []

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

```
type(brics["country"])
```

```
pandas.core.series.Series
```

- 1D labelled array

Column Access []

```
country    capital     area  population  
BR         Brazil      Brasilia  8.516      200.40  
RU         Russia     Moscow   17.100      143.50  
IN         India      New Delhi 3.286      1252.00  
CH         China      Beijing   9.597      1357.00  
SA         South Africa Pretoria 1.221      52.98
```

```
brics[["country"]]
```

```
country  
BR         Brazil  
RU         Russia  
IN         India  
CH         China  
SA         South Africa
```

Column Access []

```
country    capital     area  population
BR          Brazil      Brasilia   8.516      200.40
RU          Russia      Moscow    17.100      143.50
IN          India       New Delhi  3.286      1252.00
CH          China       Beijing   9.597      1357.00
SA  South Africa  Pretoria   1.221      52.98
```

```
type(brics[["country"]])
```

```
pandas.core.frame.DataFrame
```

Column Access []

```
country    capital    area   population  
BR         Brazil     Brasilia  8.516    200.40  
RU         Russia    Moscow   17.100   143.50  
IN         India     New Delhi 3.286    1252.00  
CH         China     Beijing  9.597    1357.00  
SA         South Africa Pretoria 1.221    52.98
```

```
brics[["country", "capital"]]
```

```
country    capital  
BR         Brazil     Brasilia  
RU         Russia    Moscow  
IN         India     New Delhi  
CH         China     Beijing  
SA         South Africa Pretoria
```

Row Access []

```
country    capital     area  population
BR         Brazil      Brasilia   8.516      200.40
RU         Russia      Moscow    17.100      143.50
IN         India       New Delhi  3.286      1252.00
CH         China       Beijing   9.597      1357.00
SA         South Africa Pretoria  1.221      52.98
```

```
brics[1:4]
```

```
country    capital     area  population
RU         Russia      Moscow    17.100      143.5
IN         India       New Delhi  3.286      1252.0
CH         China       Beijing   9.597      1357.0
```

Row Access []

```
country    capital     area  population
BR         Brazil      Brasilia  8.516      200.40    * 0 *
RU         Russia      Moscow   17.100     143.50    * 1 *
IN         India       New Delhi 3.286      1252.00   * 2 *
CH         China       Beijing  9.597      1357.00   * 3 *
SA         South Africa Pretoria 1.221      52.98    * 4 *
```

```
brics[1:4]
```

```
country    capital     area  population
RU         Russia      Moscow   17.100     143.5
IN         India       New Delhi 3.286      1252.0
CH         China       Beijing  9.597      1357.0
```

Discussion []

- Square brackets: limited functionality
- Ideally
 - 2D NumPy arrays
 - `my_array[rows, columns]`
- pandas
 - `loc` (label-based)
 - `iloc` (integer position-based)

Row Access loc

```
country    capital     area  population
BR         Brazil      Brasilia   8.516      200.40
RU         Russia      Moscow    17.100      143.50
IN         India       New Delhi  3.286      1252.00
CH         China       Beijing   9.597      1357.00
SA         South Africa Pretoria  1.221      52.98
```

```
brics.loc["RU"]
```

```
country        Russia
capital        Moscow
area           17.1
population    143.5
Name: RU, dtype: object
```

- Row as pandas Series

Row Access loc

```
country    capital   area  population  
BR         Brazil    Brasilia  8.516      200.40  
RU         Russia   Moscow   17.100     143.50  
IN         India    New Delhi 3.286      1252.00  
CH         China    Beijing   9.597      1357.00  
SA  South Africa Pretoria  1.221      52.98
```

```
brics.loc[["RU"]]
```

```
country    capital   area  population  
RU         Russia   Moscow   17.1        143.5
```

- DataFrame

Row Access loc

```
country      capital     area  population
BR          Brazil      Brasilia   8.516      200.40
RU          Russia      Moscow    17.100      143.50
IN          India       New Delhi  3.286      1252.00
CH          China       Beijing   9.597      1357.00
SA  South Africa  Pretoria   1.221       52.98
```

```
brics.loc[["RU", "IN", "CH"]]
```

```
country      capital     area  population
RU  Russia      Moscow    17.100      143.5
IN  India       New Delhi  3.286      1252.0
CH  China       Beijing   9.597      1357.0
```

Row & Column loc

```
country      capital     area  population
BR          Brazil      Brasilia   8.516      200.40
RU          Russia      Moscow    17.100      143.50
IN          India       New Delhi  3.286      1252.00
CH          China       Beijing   9.597      1357.00
SA  South Africa  Pretoria   1.221       52.98
```

```
brics.loc[["RU", "IN", "CH"], ["country", "capital"]]
```

```
country      capital
RU  Russia      Moscow
IN  India       New Delhi
CH  China       Beijing
```

Row & Column loc

```
country    capital     area  population
BR         Brazil      Brasilia   8.516      200.40
RU         Russia      Moscow    17.100      143.50
IN         India       New Delhi  3.286      1252.00
CH         China       Beijing   9.597      1357.00
SA         South Africa Pretoria  1.221      52.98
```

```
brics.loc[:, ["country", "capital"]]
```

```
country    capital
BR         Brazil      Brasilia
RU         Russia      Moscow
IN         India       New Delhi
CH         China       Beijing
SA         South Africa Pretoria
```

Recap

- Square brackets
 - Column access `brics[["country", "capital"]]`
 - Row access: only through slicing `brics[1:4]`
- `loc` (label-based)
 - Row access `brics.loc[["RU", "IN", "CH"]]`
 - Column access `brics.loc[:, ["country", "capital"]]`
 - Row & Column access

```
brics.loc[  
    ["RU", "IN", "CH"],  
    ["country", "capital"]]  
]
```

Row Access iloc

```
country    capital   area  population
BR          Brazil    Brasilia  8.516      200.40
RU          Russia    Moscow   17.100     143.50
IN          India     New Delhi 3.286      1252.00
CH          China     Beijing  9.597      1357.00
SA  South Africa Pretoria 1.221       52.98
```

```
brics.loc[["RU"]]
```

```
country  capital   area  population
RU      Russia    Moscow  17.1       143.5
```

```
brics.iloc[[1]]
```

```
country  capital   area  population
RU      Russia    Moscow  17.1       143.5
```

Row Access iloc

```
country      capital     area  population
BR          Brazil      Brasilia   8.516      200.40
RU          Russia      Moscow    17.100      143.50
IN          India       New Delhi  3.286      1252.00
CH          China       Beijing   9.597      1357.00
SA  South Africa Pretoria  1.221      52.98
```

```
brics.loc[['RU', 'IN', 'CH']]
```

```
country      capital     area  population
RU          Russia      Moscow    17.100      143.5
IN          India       New Delhi  3.286      1252.0
CH          China       Beijing   9.597      1357.0
```

Row Access iloc

```
country      capital     area  population
BR          Brazil      Brasilia   8.516      200.40
RU          Russia      Moscow    17.100      143.50
IN          India       New Delhi  3.286      1252.00
CH          China       Beijing   9.597      1357.00
SA  South Africa  Pretoria   1.221      52.98
```

```
brics.iloc[[1,2,3]]
```

```
country      capital     area  population
RU          Russia      Moscow    17.100      143.5
IN          India       New Delhi  3.286      1252.0
CH          China       Beijing   9.597      1357.0
```

Row & Column iloc

```
country    capital     area  population
BR         Brazil      Brasilia   8.516      200.40
RU         Russia      Moscow    17.100      143.50
IN         India       New Delhi  3.286      1252.00
CH         China       Beijing   9.597      1357.00
SA         South Africa Pretoria  1.221      52.98
```

```
brics.loc[["RU", "IN", "CH"], ["country", "capital"]]
```

```
country    capital
RU         Russia      Moscow
IN         India       New Delhi
CH         China       Beijing
```

Row & Column iloc

```
country    capital     area  population
BR         Brazil      Brasilia   8.516      200.40
RU         Russia      Moscow    17.100      143.50
IN         India       New Delhi  3.286      1252.00
CH         China       Beijing   9.597      1357.00
SA         South Africa Pretoria  1.221      52.98
```

```
brics.iloc[[1,2,3], [0, 1]]
```

```
country    capital
RU         Russia      Moscow
IN         India       New Delhi
CH         China       Beijing
```

Row & Column iloc

```
country    capital     area  population
BR          Brazil    Brasilia   8.516      200.40
RU          Russia    Moscow    17.100      143.50
IN          India     New Delhi  3.286      1252.00
CH          China     Beijing   9.597      1357.00
SA  South Africa Pretoria  1.221       52.98
```

```
brics.loc[:, ["country", "capital"]]
```

```
country    capital
BR          Brazil    Brasilia
RU          Russia    Moscow
IN          India     New Delhi
CH          China     Beijing
SA  South Africa Pretoria
```

Row & Column iloc

```
country    capital     area  population
BR          Brazil    Brasilia   8.516      200.40
RU          Russia    Moscow    17.100      143.50
IN          India     New Delhi  3.286      1252.00
CH          China     Beijing   9.597      1357.00
SA  South Africa Pretoria  1.221       52.98
```

```
brics.iloc[:, [0,1]]
```

```
country    capital
BR          Brazil    Brasilia
RU          Russia    Moscow
IN          India     New Delhi
CH          China     Beijing
SA  South Africa Pretoria
```

Let's practice!

INTERMEDIATE PYTHON

Comparison Operators

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

NumPy recap

```
# Code from Intro to Python for Data Science, Chapter 4
import numpy as np
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
bmi = np_weight / np_height ** 2
bmi
```

```
array([ 21.852,  20.975,  21.75 ,  24.747,  21.441])
```

```
bmi > 23
```

```
array([False, False, False, True, False], dtype=bool)
```

```
bmi[bmi > 23]
```

```
array([ 24.747])
```

- Comparison operators: how Python values relate

Numeric comparisons

```
2 < 3
```

```
True
```

```
2 == 3
```

```
False
```

```
2 <= 3
```

```
True
```

```
3 <= 3
```

```
True
```

```
x = 2  
y = 3  
x < y
```

```
True
```

Other comparisons

```
"carl" < "chris"
```

```
True
```

```
3 < "chris"
```

```
TypeError: unorderable types: int() < str()
```

```
3 < 4.1
```

```
True
```

Other comparisons

```
bmi
```

```
array([21.852, 20.975, 21.75 , 24.747, 21.441])
```

```
bmi > 23
```

```
array([False, False, False, True, False], dtype=bool)
```

Comparators

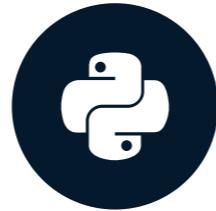
Comparator	Meaning
<	Strictly less than
<=	Less than or equal
>	Strictly greater than
>=	Greater than or equal
==	Equal
!=	Not equal

Let's practice!

INTERMEDIATE PYTHON

Boolean Operators

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Boolean Operators

- `and`
- `or`
- `not`

and

True **and** True

True

```
x = 12  
x > 5 and x < 15  
# True      True
```

True

False **and** True

False

True **and** False

False

False **and** False

False

or

True **or** True

True

False **or** True

True

True **or** False

True

False **or** False

False

y = 5
y < 7 **or** y > 13

True

not

not True

False

not False

True

NumPy

```
bmi      # calculation of bmi left out
```

```
array([21.852, 20.975, 21.75 , 24.747, 21.441])
```

```
bmi > 21
```

```
array([True, False, True, True, True], dtype=bool)
```

```
bmi < 22
```

```
array([True, True, True, False, True], dtype=bool)
```

```
bmi > 21 and bmi < 22
```

```
ValueError: The truth value of an array with more than one element is  
ambiguous. Use a.any() or a.all()
```

NumPy

- `logical_and()`
- `logical_or()`
- `logical_not()`

```
np.logical_and(bmi > 21, bmi < 22)
```

```
array([True, False, True, False, True], dtype=bool)
```

```
bmi[np.logical_and(bmi > 21, bmi < 22)]
```

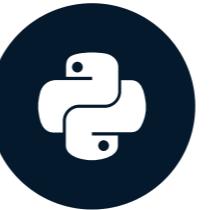
```
array([21.852, 21.75, 21.441])
```

Let's practice!

INTERMEDIATE PYTHON

if, elif, else

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Overview

- Comparison Operators
 - < , > , >= , <= , == , !=
- Boolean Operators
 - and , or , not
- Conditional Statements
 - if , else , elif

if

```
if condition :  
    expression
```

control.py

```
z = 4  
if z % 2 == 0 :      # True  
    print("z is even")
```

z is even

if

```
if condition :  
    expression
```

- expression not part of if

control.py

```
z = 4  
if z % 2 == 0 :      # True  
    print("z is even")
```

z is even

if

```
if condition :  
    expression
```

control.py

```
z = 4  
if z % 2 == 0 :  
    print("checking " + str(z))  
    print("z is even")
```

```
checking 4  
z is even
```

if

```
if condition :  
    expression
```

control.py

```
z = 5  
if z % 2 == 0 :      # False  
    print("checking " + str(z))  
    print("z is even")
```

else

```
if condition :  
    expression  
else :  
    expression
```

control.py

```
z = 5  
if z % 2 == 0 :      # False  
    print("z is even")  
else :  
    print("z is odd")
```

z is odd

elif

```
if condition :  
    expression  
elif condition :  
    expression  
else :  
    expression
```

control.py

```
z = 3  
if z % 2 == 0 :  
    print("z is divisible by 2")      # False  
elif z % 3 == 0 :  
    print("z is divisible by 3")      # True  
else :  
    print("z is neither divisible by 2 nor by 3")
```

```
z is divisible by 3
```

elif

```
if condition :  
    expression  
elif condition :  
    expression  
else :  
    expression
```

control.py

```
z = 6  
if z % 2 == 0 :  
    print("z is divisible by 2")      # True  
elif z % 3 == 0 :  
    print("z is divisible by 3")      # Never reached  
else :  
    print("z is neither divisible by 2 nor by 3")
```

```
z is divisible by 2
```

Let's practice!

INTERMEDIATE PYTHON

Filtering pandas DataFrames

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

brics

```
import pandas as pd  
brics = pd.read_csv("path/to/brics.csv", index_col = 0)  
brics
```

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

Goal

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

- Select countries with area over 8 million km²
- 3 steps
 - Select the area column
 - Do comparison on area column
 - Use result to select countries

Step 1: Get column

```
country    capital     area  population
BR          Brazil      Brasilia   8.516      200.40
RU          Russia      Moscow    17.100      143.50
IN          India       New Delhi  3.286      1252.00
CH          China       Beijing   9.597      1357.00
SA  South Africa  Pretoria   1.221       52.98
```

```
brics["area"]
```

```
BR      8.516
RU     17.100
IN      3.286
CH      9.597
SA      1.221
Name: area, dtype: float64    # - Need Pandas Series
```

- Alternatives:

```
brics.loc[:, "area"]
brics.iloc[:, 2]
```

Step 2: Compare

```
brics["area"]
```

```
BR      8.516
RU     17.100
IN      3.286
CH      9.597
SA      1.221
Name: area, dtype: float64
```

```
brics["area"] > 8
```

```
BR      True
RU      True
IN     False
CH      True
SA     False
Name: area, dtype: bool
```

```
is_huge = brics["area"] > 8
```

Step 3: Subset DF

```
is_huge
```

```
BR      True
RU      True
IN      False
CH      True
SA      False
Name: area, dtype: bool
```

```
brics[is_huge]
```

```
country    capital     area  population
BR      Brazil    Brasilia   8.516        200.4
RU      Russia     Moscow  17.100        143.5
CH      China      Beijing  9.597      1357.0
```

Summary

```
country    capital    area  population
BR         Brazil     Brasilia  8.516      200.40
RU         Russia    Moscow   17.100     143.50
IN         India     New Delhi 3.286      1252.00
CH         China     Beijing  9.597      1357.00
SA  South Africa Pretoria 1.221      52.988
```

```
is_huge = brics["area"] > 8
brics[is_huge]
```

```
country    capital    area  population
BR  Brazil     Brasilia  8.516      200.4
RU  Russia    Moscow   17.100     143.5
CH  China     Beijing  9.597      1357.0
```

```
brics[brics["area"] > 8]
```

```
country    capital    area  population
BR  Brazil     Brasilia  8.516      200.4
RU  Russia    Moscow   17.100     143.5
CH  China     Beijing  9.597      1357.0
```

Boolean operators

```
country    capital     area   population
BR          Brazil      Brasilia  8.516      200.40
RU          Russia      Moscow   17.100     143.50
IN          India       New Delhi 3.286      1252.00
CH          China       Beijing  9.597      1357.00
SA  South Africa  Pretoria  1.221      52.98
```

```
import numpy as np
np.logical_and(brics["area"] > 8, brics["area"] < 10)
```

```
BR      True
RU      False
IN      False
CH      True
SA      False
Name: area, dtype: bool
```

```
brics[np.logical_and(brics["area"] > 8, brics["area"] < 10)]
```

```
country    capital     area   population
BR  Brazil      Brasilia  8.516      200.4
CH  China       Beijing  9.597      1357.0
```

Let's practice!

INTERMEDIATE PYTHON

while loop

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

if-elif-else

control.py

- Goes through construct only once!

```
z = 6
if z % 2 == 0 : # True
    print("z is divisible by 2") # Executed
elif z % 3 == 0 :
    print("z is divisible by 3")
else :
    print("z is neither divisible by 2 nor by 3")

... # Moving on
```

- While loop = repeated if statement

While

```
while condition :  
    expression
```

- Numerically calculating model
- "repeating action until condition is met"
- Example
 - Error starts at 50
 - Divide error by 4 on every run
 - Continue until error no longer > 1

While

```
while condition :  
    expression
```

while_loop.py

```
error = 50.0  
  
while error > 1:  
    error = error / 4  
    print(error)
```

- Error starts at 50
- Divide error by 4 on every run
- Continue until error no longer > 1

While

```
while condition :  
    expression
```

while_loop.py

```
error = 50.0  
#      50  
while error > 1:      # True  
    error = error / 4  
    print(error)
```

12.5

While

```
while condition :  
    expression
```

while_loop.py

```
error = 50.0  
#      12.5  
while error > 1:      # True  
    error = error / 4  
    print(error)
```

```
12.5  
3.125
```

While

```
while condition :  
    expression
```

while_loop.py

```
error = 50.0  
#      3.125  
while error > 1:      # True  
    error = error / 4  
    print(error)
```

```
12.5  
3.125  
0.78125
```

While

```
while condition :  
    expression
```

while_loop.py

```
error = 50.0  
#      0.78125  
while error > 1:      # False  
    error = error / 4  
    print(error)
```

```
12.5  
3.125  
0.78125
```

While

```
while condition :  
    expression
```

while_loop.py

```
error = 50.0
while error > 1 :      # always True
    # error = error / 4
    print(error)
```

- DataCamp: session disconnected
 - Local system: Control + C

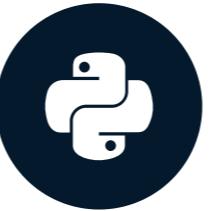
50 50 50 50 50 50 50 50

Let's practice!

INTERMEDIATE PYTHON

for loop

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

for loop

```
for var in seq :  
    expression
```

- "for each var in seq, execute expression"

fam

family.py

```
fam = [1.73, 1.68, 1.71, 1.89]  
print(fam)
```

```
[1.73, 1.68, 1.71, 1.89]
```

fam

family.py

```
fam = [1.73, 1.68, 1.71, 1.89]
print(fam[0])
print(fam[1])
print(fam[2])
print(fam[3])
```

```
1.73
1.68
1.71
1.89
```

for loop

```
for var in seq :  
    expression
```

family.py

```
fam = [1.73, 1.68, 1.71, 1.89]  
for height in fam :  
    print(height)
```

for loop

```
for var in seq :  
    expression
```

family.py

```
fam = [1.73, 1.68, 1.71, 1.89]  
for height in fam :  
    print(height)  
    # first iteration  
    # height = 1.73
```

1.73

for loop

```
for var in seq :  
    expression
```

family.py

```
fam = [1.73, 1.68, 1.71, 1.89]  
for height in fam :  
    print(height)  
    # second iteration  
    # height = 1.68
```

```
1.73  
1.68
```

for loop

```
for var in seq :  
    expression
```

family.py

```
fam = [1.73, 1.68, 1.71, 1.89]  
for height in fam :  
    print(height)
```

```
1.73  
1.68  
1.71  
1.89
```

- No access to indexes

for loop

```
for var in seq :  
    expression
```

family.py

```
fam = [1.73, 1.68, 1.71, 1.89]
```

- ???

```
index 0: 1.73  
index 1: 1.68  
index 2: 1.71  
index 3: 1.89
```

enumerate

```
for var in seq :  
    expression
```

family.py

```
fam = [1.73, 1.68, 1.71, 1.89]  
for index, height in enumerate(fam) :  
    print("index " + str(index) + ": " + str(height))
```

```
index 0: 1.73  
index 1: 1.68  
index 2: 1.71  
index 3: 1.89
```

Loop over string

```
for var in seq :  
    expression
```

strloop.py

```
for c in "family" :  
    print(c.capitalize())
```

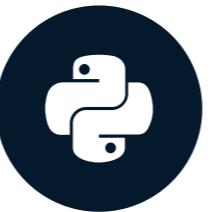
F
A
M
I
L
Y

Let's practice!

INTERMEDIATE PYTHON

Loop Data Structures Part 1

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Dictionary

```
for var in seq :  
    expression
```

dictloop.py

```
world = { "afghanistan":30.55,  
          "albania":2.77,  
          "algeria":39.21 }  
  
for key, value in world :  
    print(key + " -- " + str(value))
```

```
ValueError: too many values to  
        unpack (expected 2)
```

Dictionary

```
for var in seq :  
    expression
```

dictloop.py

```
world = { "afghanistan":30.55,  
          "albania":2.77,  
          "algeria":39.21 }  
  
for key, value in world.items() :  
    print(key + " -- " + str(value))
```

```
algeria -- 39.21  
afghanistan -- 30.55  
albania -- 2.77
```

Dictionary

```
for var in seq :  
    expression
```

dictloop.py

```
world = { "afghanistan":30.55,  
          "albania":2.77,  
          "algeria":39.21 }  
  
for k, v in world.items() :  
    print(k + " -- " + str(v))
```

```
algeria -- 39.21  
afghanistan -- 30.55  
albania -- 2.77
```

NumPy Arrays

```
for var in seq :  
    expression
```

nploop.py

```
import numpy as np  
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])  
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])  
bmi = np_weight / np_height ** 2  
for val in bmi :  
    print(val)
```

```
21.852  
20.975  
21.750  
24.747  
21.441
```

2D NumPy Arrays

nploop.py

```
import numpy as np
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
meas = np.array([np_height, np_weight])
for val in meas :
    print(val)
```

```
[ 1.73  1.68  1.71  1.89  1.79]
[ 65.4   59.2   63.6   88.4   68.7]
```

2D NumPy Arrays

nploop.py

```
import numpy as np
np_height = np.array([1.73, 1.68, 1.71, 1.89, 1.79])
np_weight = np.array([65.4, 59.2, 63.6, 88.4, 68.7])
meas = np.array([np_height, np_weight])
for val in np.nditer(meas) :
    print(val)
```

```
1.73
1.68
1.71
1.89
1.79
65.4
...
```

Recap

- Dictionary
 - `for key, val in my_dict.items() :`
- NumPy array
 - `for val in np.nditer(my_array) :`

Let's practice!

INTERMEDIATE PYTHON

Loop Data Structures Part 2

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

brics

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.40
RU	Russia	Moscow	17.100	143.50
IN	India	New Delhi	3.286	1252.00
CH	China	Beijing	9.597	1357.00
SA	South Africa	Pretoria	1.221	52.98

dfloop.py

```
import pandas as pd  
brics = pd.read_csv("brics.csv", index_col = 0)
```

for, first try

dfloop.py

```
import pandas as pd  
brics = pd.read_csv("brics.csv", index_col = 0)  
for val in brics :  
    print(val)
```

```
country  
capital  
area  
population
```

iterrows

dfloop.py

```
import pandas as pd  
brics = pd.read_csv("brics.csv", index_col = 0)  
for lab, row in brics.iterrows():  
    print(lab)  
    print(row)
```

```
BR  
country      Brazil  
capital      Brasilia  
area         8.516  
population   200.4  
Name: BR, dtype: object  
...  
RU  
country      Russia  
capital      Moscow  
area         17.1  
population   143.5  
Name: RU, dtype: object  
IN ...
```

Selective print

dfloop.py

```
import pandas as pd  
brics = pd.read_csv("brics.csv", index_col = 0)  
for lab, row in brics.iterrows():  
    print(lab + ": " + row["capital"])
```

BR: Brasilia

RU: Moscow

IN: New Delhi

CH: Beijing

SA: Pretoria

Add column

dfloop.py

```
import pandas as pd  
brics = pd.read_csv("brics.csv", index_col = 0)  
for lab, row in brics.iterrows() :  
    # - Creating Series on every iteration  
    brics.loc[lab, "name_length"] = len(row["country"])  
print(brics)
```

	country	capital	area	population	name_length
BR	Brazil	Brasilia	8.516	200.40	6
RU	Russia	Moscow	17.100	143.50	6
IN	India	New Delhi	3.286	1252.00	5
CH	China	Beijing	9.597	1357.00	5
SA	South Africa	Pretoria	1.221	52.98	12

apply

dfloop.py

```
import pandas as pd  
brics = pd.read_csv("brics.csv", index_col = 0)  
brics["name_length"] = brics["country"].apply(len)  
print(brics)
```

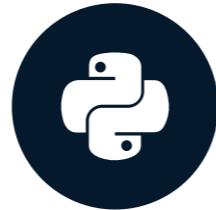
	country	capital	area	population	name_length
BR	Brazil	Brasilia	8.516	200.40	6
RU	Russia	Moscow	17.100	143.50	6
IN	India	New Delhi	3.286	1252.00	5
CH	China	Beijing	9.597	1357.00	5
SA	South Africa	Pretoria	1.221	52.98	12

Let's practice!

INTERMEDIATE PYTHON

Random Numbers

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

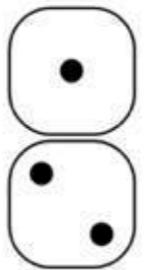


100 x





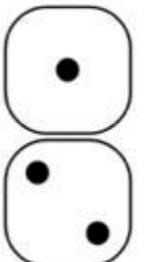
100 x



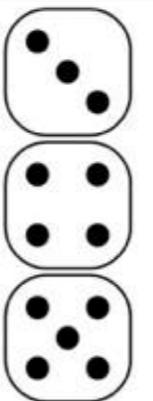
-1



100 x



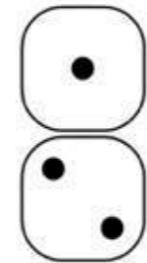
-1



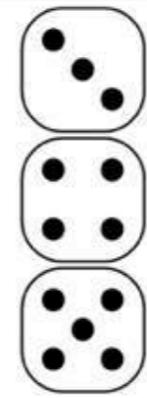
+1



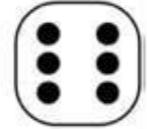
100 x



-1



+1



+1 +2 +3 +4 +5 +6

100 x



-1



+1



+1 +2 +3 +4 +5 +6

- Can't go below step 0
- 0.1 % chance of falling down the stairs
- Bet: you'll reach step 60

How to solve?

- Analytical
- Simulate the process
 - Hacker statistics!

Random generators

```
import numpy as np  
np.random.rand()      # Pseudo-random numbers
```

```
0.9535543896720104    # Mathematical formula
```

```
np.random.seed(123)    # Starting from a seed  
np.random.rand()
```

```
0.6964691855978616
```

```
np.random.rand()
```

```
0.28613933495037946
```

Random generators

```
np.random.seed(123)  
np.random.rand()
```

```
0.696469185597861 # Same seed: same random numbers!
```

```
np.random.rand() # Ensures "reproducibility"
```

```
0.28613933495037946
```

Coin toss

game.py

```
import numpy as np
np.random.seed(123)
coin = np.random.randint(0,2) # Randomly generate 0 or 1
print(coin)
```

0

Coin toss

game.py

```
import numpy as np
np.random.seed(123)
coin = np.random.randint(0,2) # Randomly generate 0 or 1
print(coin)
if coin == 0:
    print("heads")
else:
    print("tails")
```

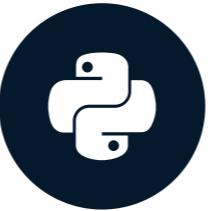
```
0
heads
```

Let's practice!

INTERMEDIATE PYTHON

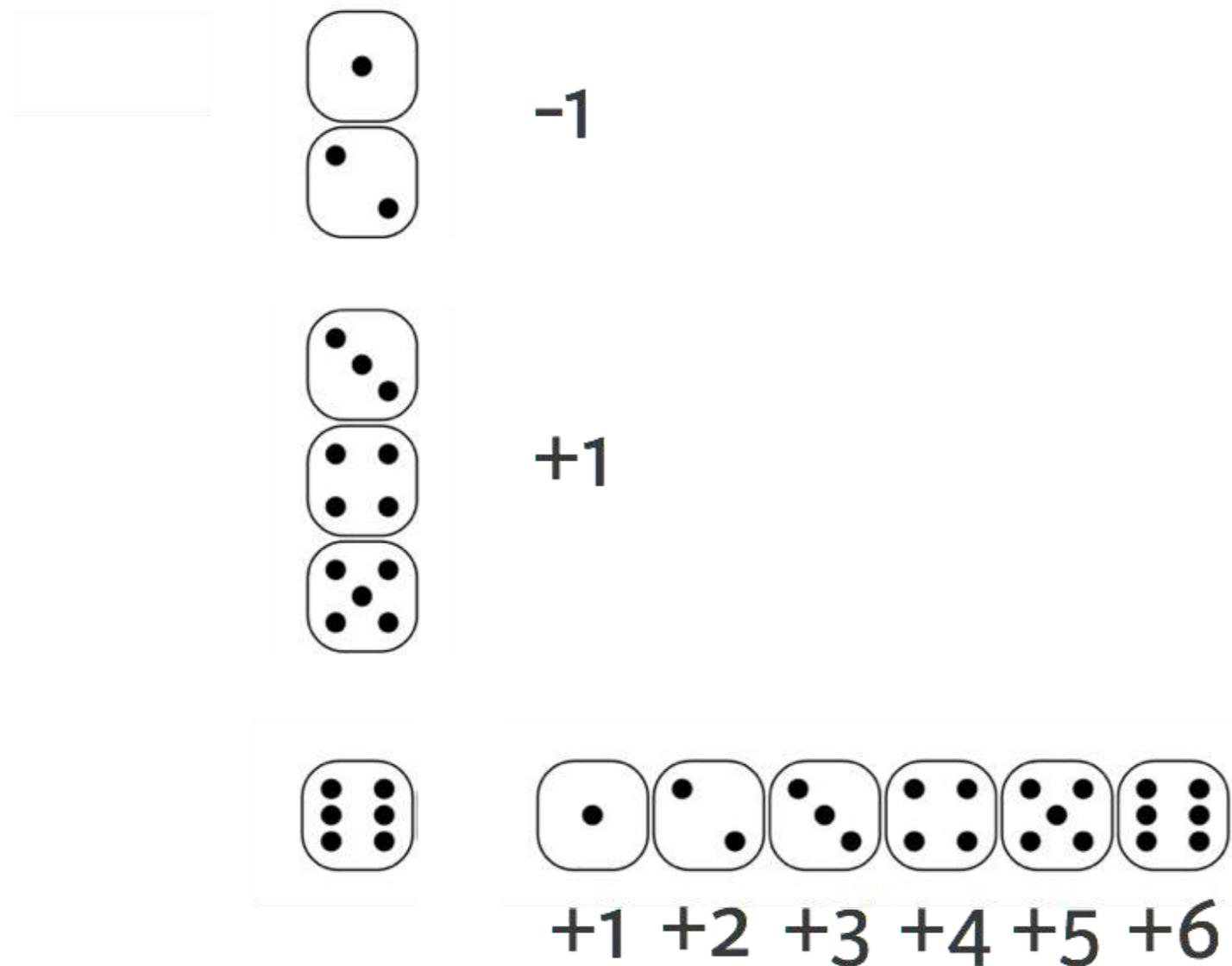
Random Walk

INTERMEDIATE PYTHON

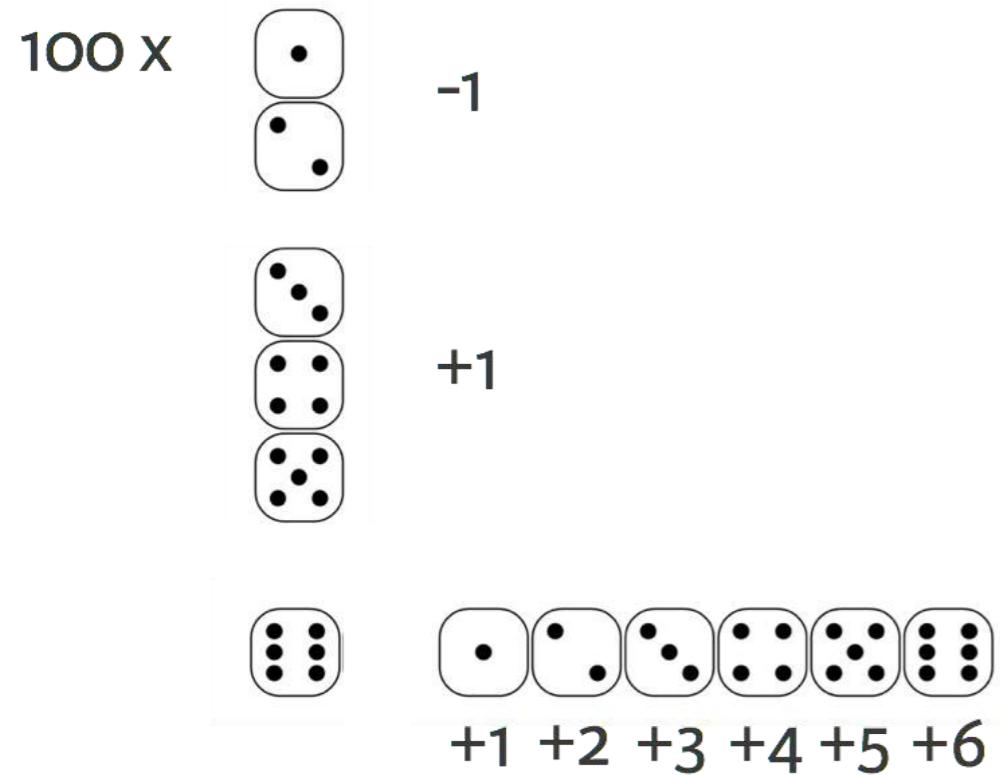


Hugo Bowne-Anderson
Data Scientist at DataCamp

Random Step



Random Walk



Known in Science

- Path of molecules
- Gambler's financial status

Heads or Tails

headtails.py

```
import numpy as np
np.random.seed(123)
outcomes = []
for x in range(10) :
    coin = np.random.randint(0, 2)
    if coin == 0 :
        outcomes.append("heads")
    else :
        outcomes.append("tails")
print(outcomes)
```

```
['heads', 'tails', 'heads', 'heads', 'heads',
 'heads', 'heads', 'tails', 'tails', 'heads']
```

Heads or Tails: Random Walk

headtailsrw.py

```
import numpy as np
np.random.seed(123)
tails = [0]
for x in range(10) :
    coin = np.random.randint(0, 2)
    tails.append(tails[x] + coin)
print(tails)
```

```
[0, 0, 1, 1, 1, 1, 1, 1, 2, 3, 3]
```

Step to Walk

outcomes

```
['heads', 'tails', 'heads', 'heads', 'heads',
 'heads', 'heads', 'tails', 'tails', 'heads']
```

tails

```
[0, 0, 1, 1, 1, 1, 1, 1, 2, 3, 3]
```

Let's practice!

INTERMEDIATE PYTHON

Distribution

INTERMEDIATE PYTHON

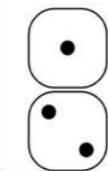


Hugo Bowne-Anderson
Data Scientist at DataCamp

Distribution

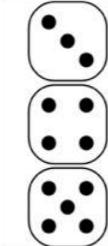


100 x



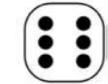
-1

Each random walk has an end point



+1

Distribution!



+1 +2 +3 +4 +5 +6

Calculate chances!

Simulate 10,000 times: 10,000 end points

Random Walk

headtailsrw.py

```
import numpy as np
np.random.seed(123)
tails = [0]
for x in range(10) :
    coin = np.random.randint(0, 2)
    tails.append(tails[x] + coin)
```

100 runs

distribution.py

```
import numpy as np
np.random.seed(123)
final_tails = []
for x in range(100) :
    tails = [0]
    for x in range(10) :
        coin = np.random.randint(0, 2)
        tails.append(tails[-1] + coin)
    final_tails.append(tails[-1])
print(final_tails)
```

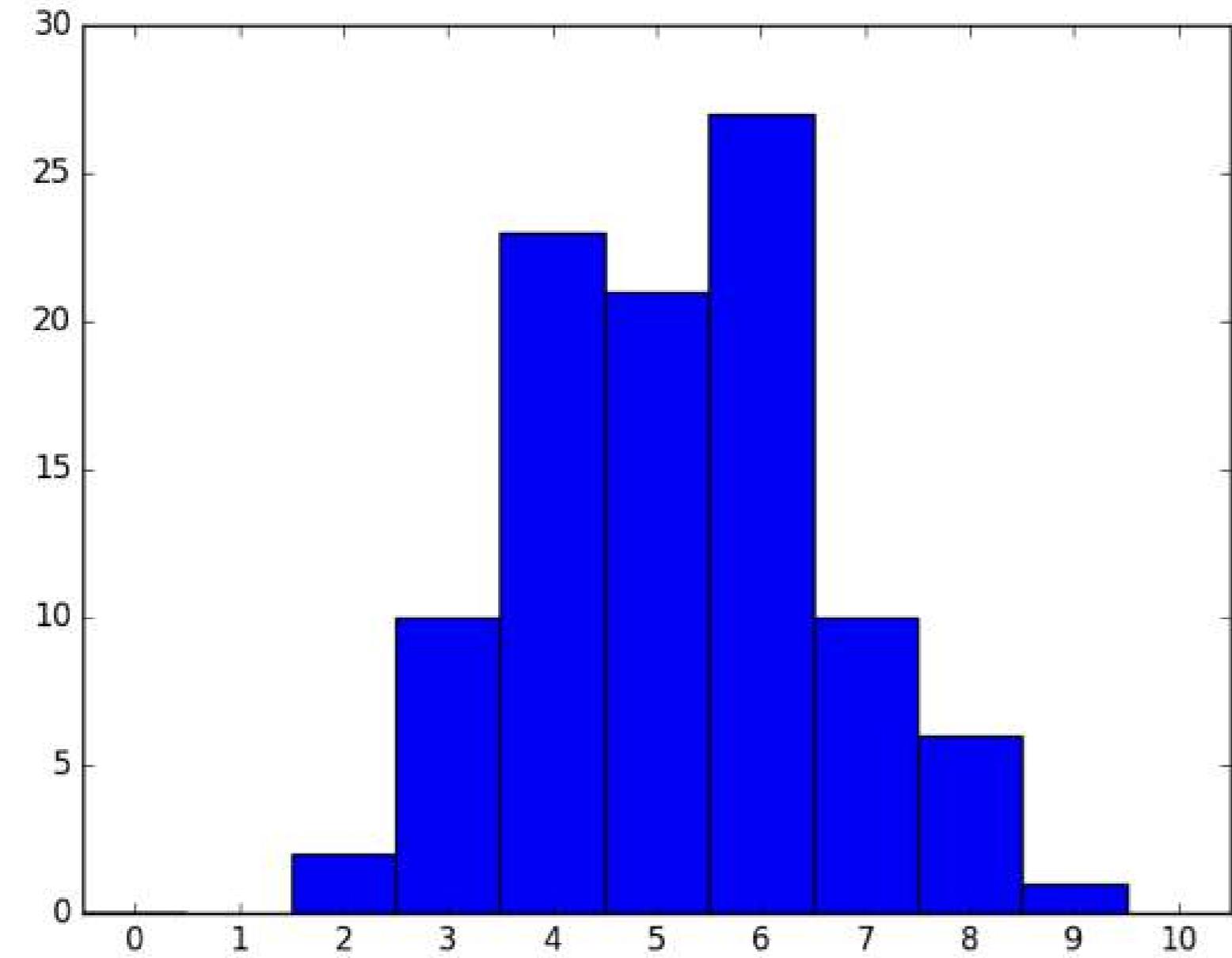
```
[3, 6, 4, 5, 4, 5, 3, 5, 4, 6, 6, 8, 6, 4, 7, 5, 7, 4, 3, 3, ..., 4]
```

Histogram, 100 runs

distribution.py

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(123)
final_tails = []
for x in range(100) :
    tails = [0]
    for x in range(10) :
        coin = np.random.randint(0, 2)
        tails.append(tails[-1] + coin)
    final_tails.append(tails[-1])
plt.hist(final_tails, bins = 10)
plt.show()
```

Histogram, 100 runs

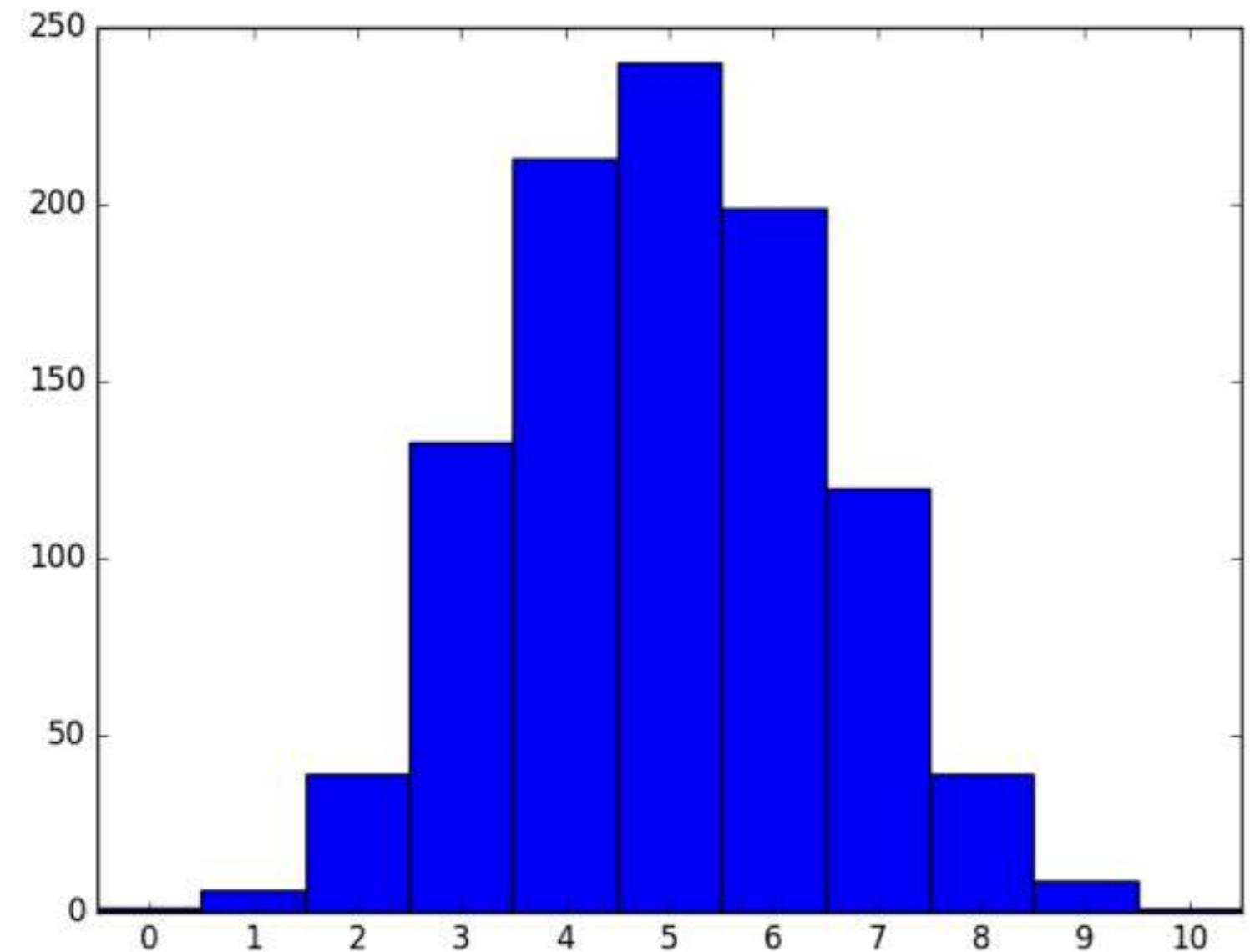


Histogram, 1,000 runs

distribution.py

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(123)
final_tails = []
for x in range(1000) : # <--
    tails = [0]
    for x in range(10) :
        coin = np.random.randint(0, 2)
        tails.append(tails[-1] + coin)
    final_tails.append(tails[-1])
plt.hist(final_tails, bins = 10)
plt.show()
```

Histogram, 1,000 runs

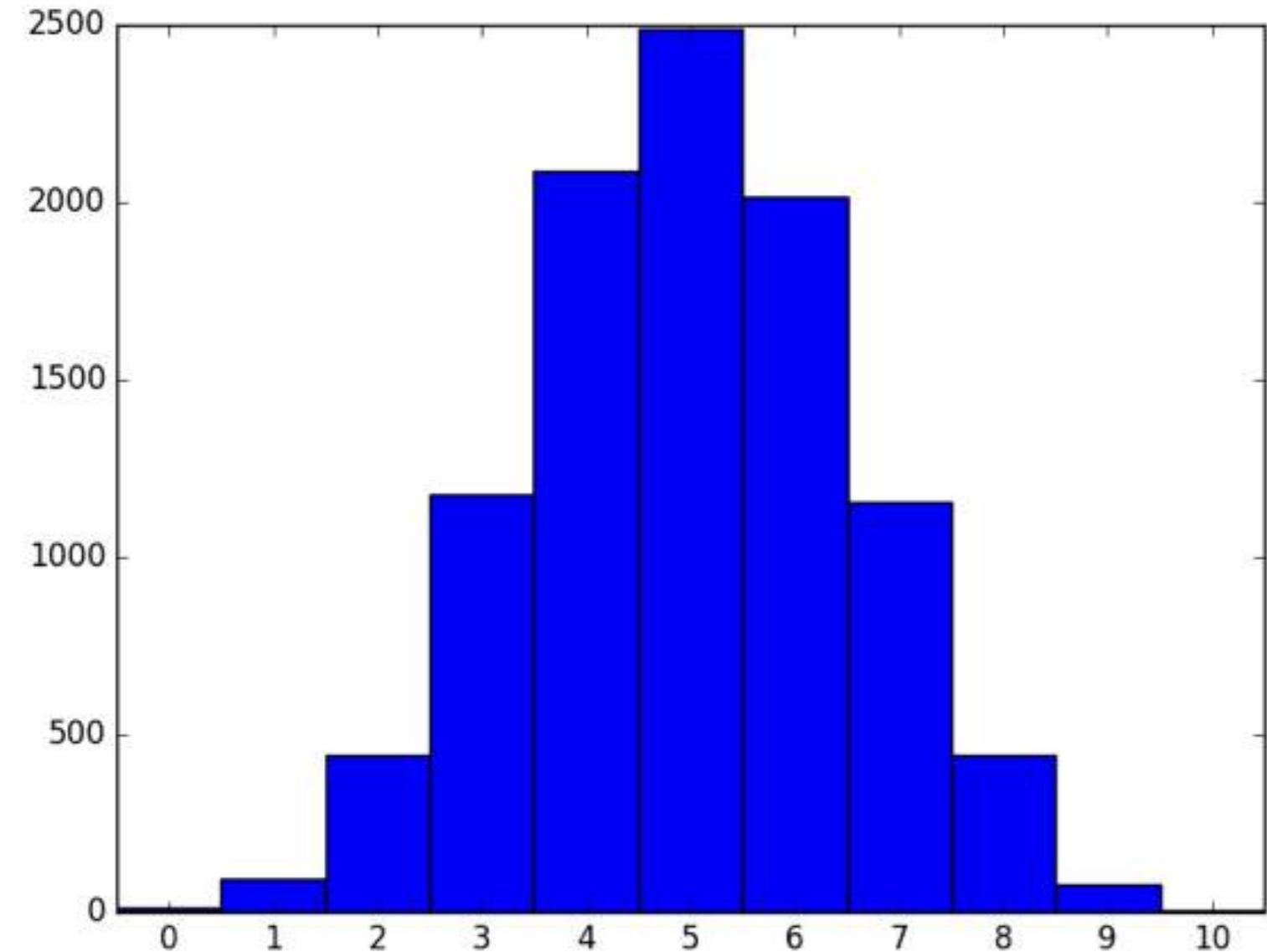


Histogram, 10,000 runs

distribution.py

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(123)
final_tails = []
for x in range(10000) : # <--
    tails = [0]
    for x in range(10) :
        coin = np.random.randint(0, 2)
        tails.append(tails[-1] + coin)
    final_tails.append(tails[-1])
plt.hist(final_tails, bins = 10)
plt.show()
```

Histogram, 10,000 runs



Let's practice!

INTERMEDIATE PYTHON

Introducing DataFrames

DATA MANIPULATION WITH PANDAS



Richie Cotton

Learning Solutions Architect at
DataCamp

What's the point of pandas?

- Data Manipulation skill track
- Data Visualization skill track

Course outline

- **Chapter 1: DataFrames**
 - Sorting and subsetting
 - Creating new columns
- **Chapter 2: Aggregating Data**
 - Summary statistics
 - Counting
 - Grouped summary statistics
- **Chapter 3: Slicing and Indexing Data**
 - Subsetting using slicing
 - Indexes and subsetting using indexes
- **Chapter 4: Creating and Visualizing Data**
 - Plotting
 - Handling missing data
 - Reading data into a DataFrame

pandas is built on NumPy and Matplotlib



pandas is popular

According to PyPI, pandas is one of the most popular Python packages.¹

¹ <https://pypistats.org/packages/pandas>

Rectangular data

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

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

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
```

Exploring a DataFrame: .shape

```
dogs.shape
```

```
(7, 6)
```

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
```

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)
```

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)
```

pandas Philosophy

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

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



¹ <https://www.python.org/dev/peps/pep-0020/>

Let's practice!

DATA MANIPULATION WITH PANDAS

Sorting and subsetting

DATA MANIPULATION WITH PANDAS



Richie Cotton

Learning Solutions Architect at
DataCamp

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 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
```

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

```
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
```

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]
```

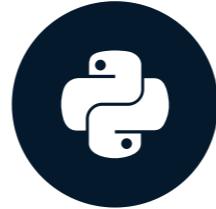
	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	24	2013-07-01
1	Charlie	Poodle	Black	43	24	2016-09-16
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
4	Max	Labrador	Black	59	29	2017-01-20

Let's practice!

DATA MANIPULATION WITH PANDAS

New columns

DATA MANIPULATION WITH PANDAS



Richie Cotton

Learning Solutions Architect at
DataCamp

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

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
3	Cooper	Schnauzer	Gray	49	17	2011-12-11	0.49	70.803832
4	Max	Labrador	Black	59	29	2017-01-20	0.59	83.309394

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
```

Let's practice!

DATA MANIPULATION WITH PANDAS

Summary statistics

DATA MANIPULATION WITH PANDAS



Maggie Matsui

Senior Content Developer at DataCamp

Summarizing numerical data

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

```
49.714285714285715
```

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

Summarizing dates

Oldest dog:

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

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

Youngest dog:

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

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

The .agg() method

```
def pct30(column):  
    return column.quantile(0.3)
```

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

```
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
```

Cumulative statistics

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

Walmart

```
sales.head()
```

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

Let's practice!

DATA MANIPULATION WITH PANDAS

Counting

DATA MANIPULATION WITH PANDAS



Maggie Matsui

Senior Content Developer at DataCamp

Avoiding double counting



Vet visits

```
print(vet_visits)
```

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

Dropping duplicate names

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

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

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
```

Let's practice!

DATA MANIPULATION WITH PANDAS

Grouped summary statistics

DATA MANIPULATION WITH PANDAS



Maggie Matsui

Senior Content Developer at DataCamp

Summaries by group

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

```
26.0  
24.0  
74.0  
17.0  
2.0
```

Grouped summaries

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

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

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

Let's practice!

DATA MANIPULATION WITH PANDAS

Pivot tables

DATA MANIPULATION WITH PANDAS



Maggie Matsui

Senior Content Developer at DataCamp

Group by to pivot table

```
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
```

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

Let's practice!

DATA MANIPULATION WITH PANDAS

Explicit indexes

DATA MANIPULATION WITH PANDAS



Richie Cotton

Learning Solutions Architect at
DataCamp

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
```

.columns and .index

dogs.columns

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

dogs.index

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

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
```

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
```

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

Sorting by index values

```
dogs_ind3.sort_index()
```

			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

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

Temperature dataset

	date	city	country	avg_temp_c
0	2000-01-01	Abidjan	Côte D'Ivoire	27.293
1	2000-02-01	Abidjan	Côte D'Ivoire	27.685
2	2000-03-01	Abidjan	Côte D'Ivoire	29.061
3	2000-04-01	Abidjan	Côte D'Ivoire	28.162
4	2000-05-01	Abidjan	Côte D'Ivoire	27.547

Let's practice!

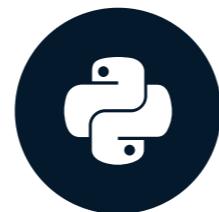
DATA MANIPULATION WITH PANDAS

Slicing and subsetting with .loc and .iloc

DATA MANIPULATION WITH PANDAS

Richie Cotton

Learning Solutions Architect at
DataCamp



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)
```

			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"]
```

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"))]
```

			name	height_cm	weight_kg
breed	color				
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"]
```

			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

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"]]
```

			name	height_cm
breed	color			
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

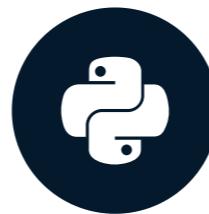
	name	breed	color	height_cm	weight_kg
0	Bella	Labrador	Brown	56	25
1	Charlie	Poodle	Black	43	23
2	Lucy	Chow Chow	Brown	46	22
3	Cooper	Schnauzer	Grey	49	17
4	Max	Labrador	Black	59	29
5	Stella	Chihuahua	Tan	18	2
6	Bernie	St. Bernard	White	77	74

Let's practice!

DATA MANIPULATION WITH PANDAS

Working with pivot tables

DATA MANIPULATION WITH PANDAS



Richie Cotton

Learning Solutions Architect at
DataCamp

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

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
```

Calculating summary stats across columns

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

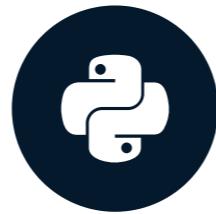
```
breed
Beagle          36.362667
Boxer           59.358667
Chihuahua       19.561250
Chow Chow        52.413333
Dachshund        20.236667
Labrador         55.875000
Poodle           51.637750
St. Bernard       66.654300
dtype: float64
```

Let's practice!

DATA MANIPULATION WITH PANDAS

Visualizing your data

DATA MANIPULATION WITH PANDAS

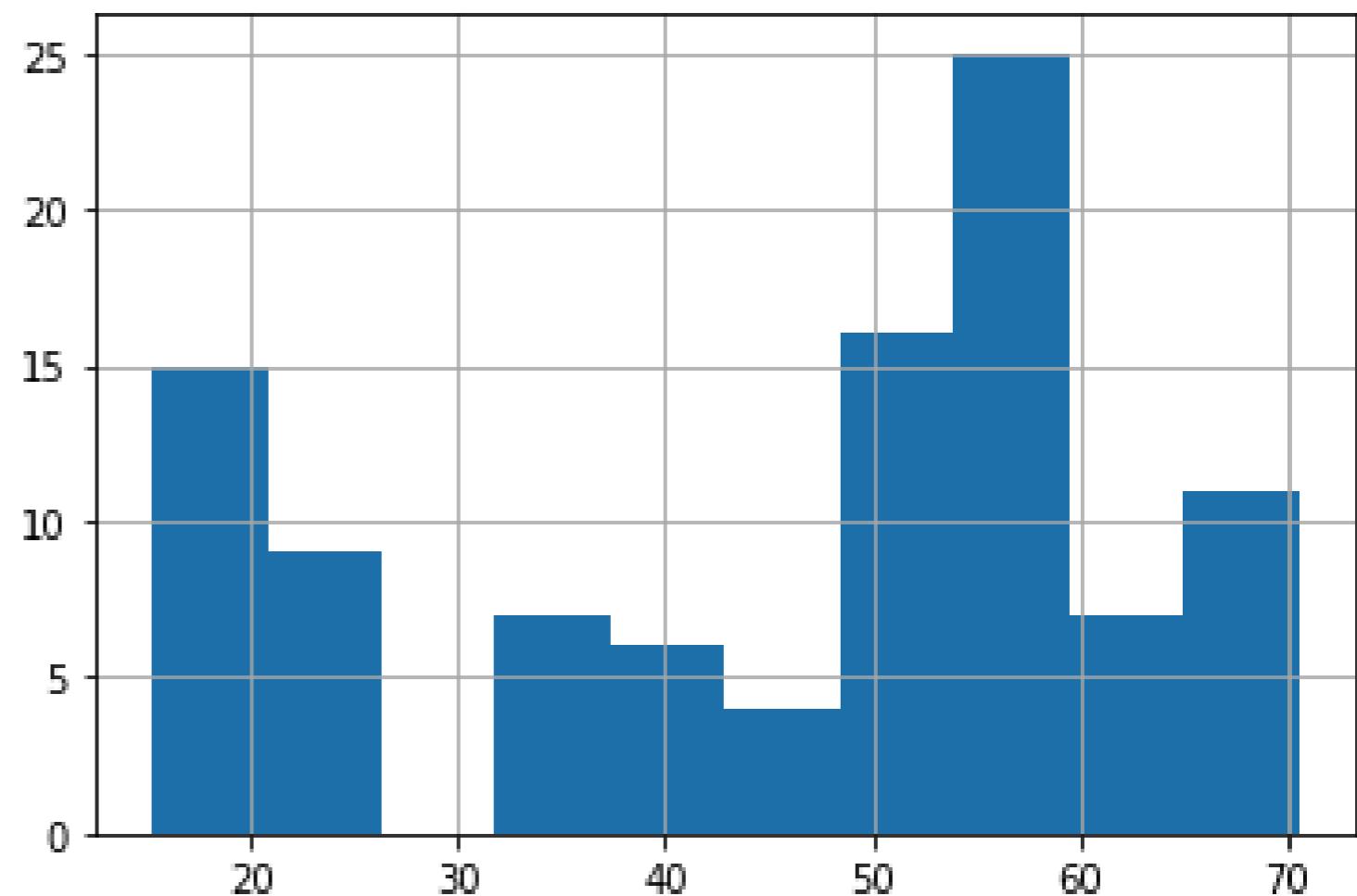


Maggie Matsui

Senior Content Developer at DataCamp

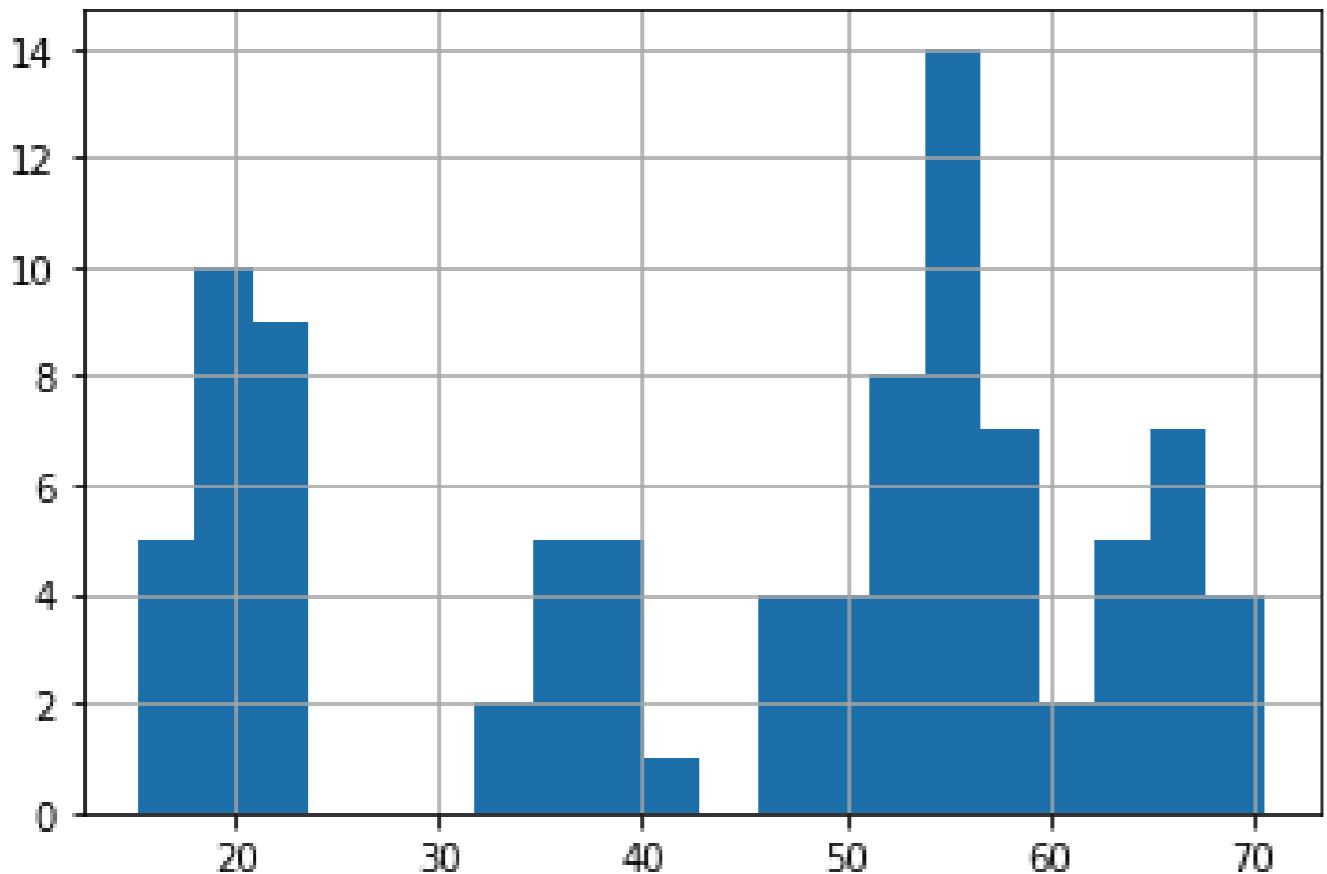
Histograms

```
import matplotlib.pyplot as plt  
  
dog_pack["height_cm"].hist()  
  
plt.show()
```

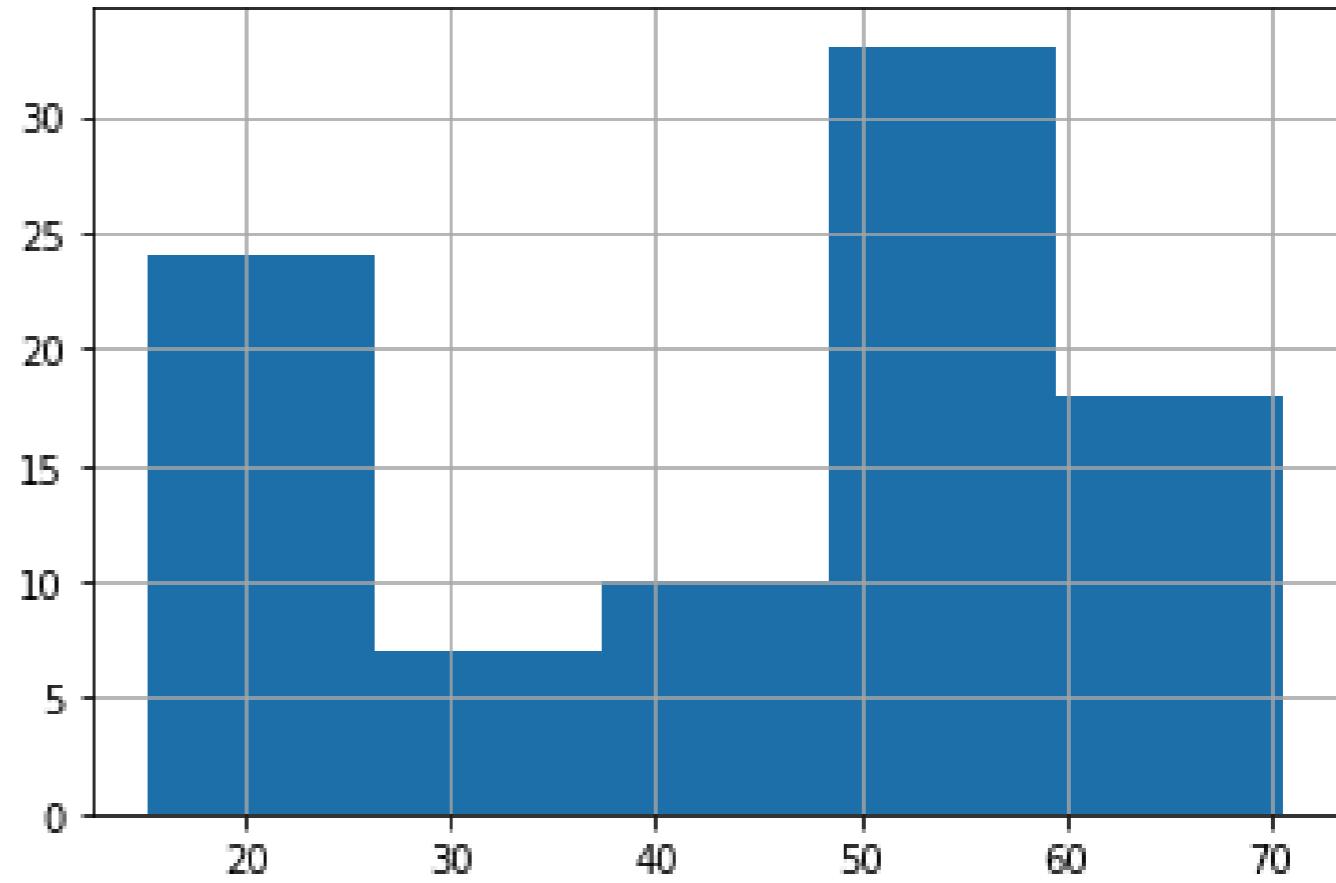


Histograms

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



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



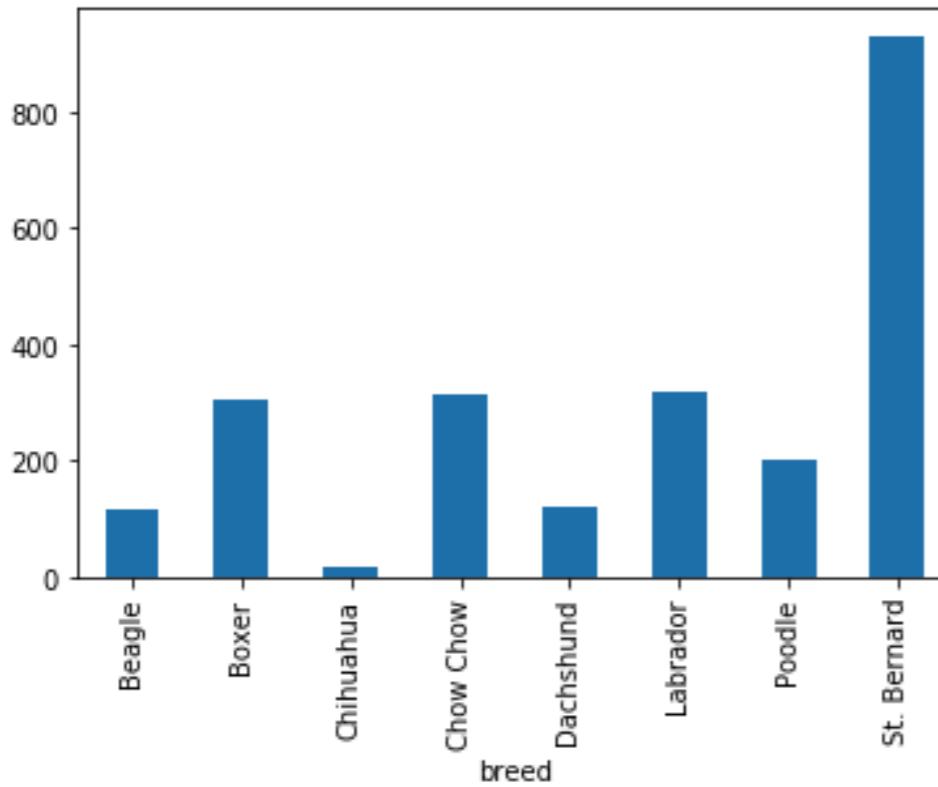
Bar plots

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

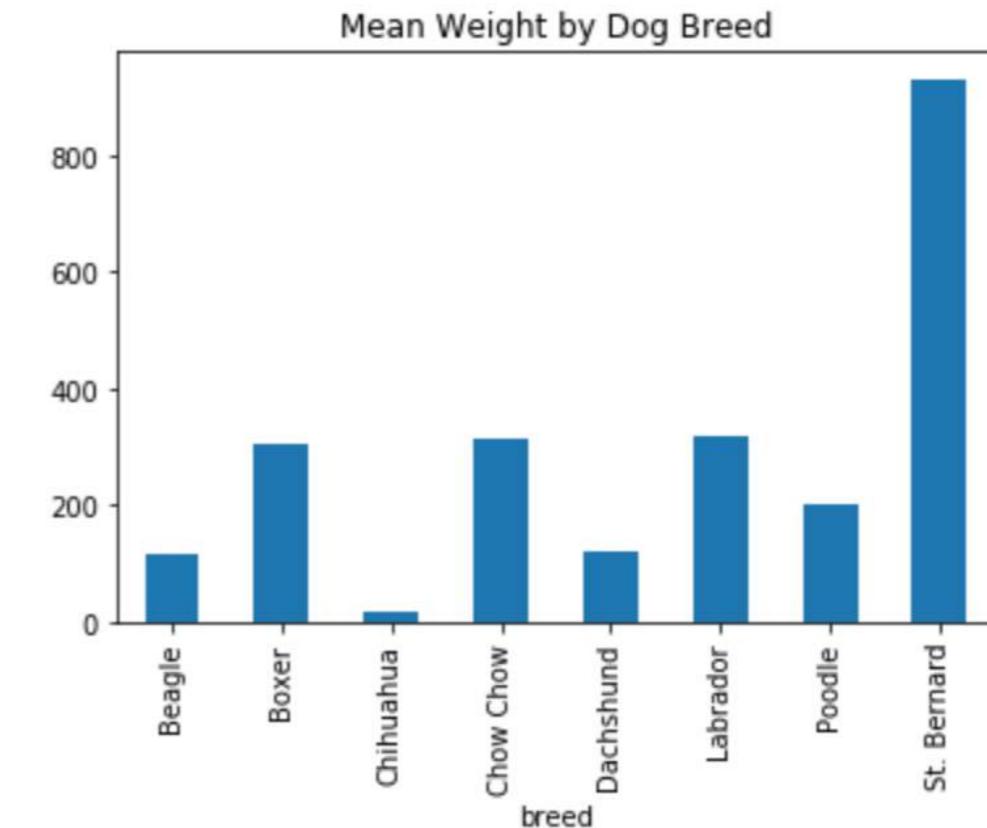
```
breed  
Beagle      10.636364  
Boxer       30.620000  
Chihuahua   1.491667  
Chow Chow    22.535714  
Dachshund   9.975000  
Labrador    31.850000  
Poodle      20.400000  
St. Bernard  71.576923  
Name: weight_kg, dtype: float64
```

Bar plots

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



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

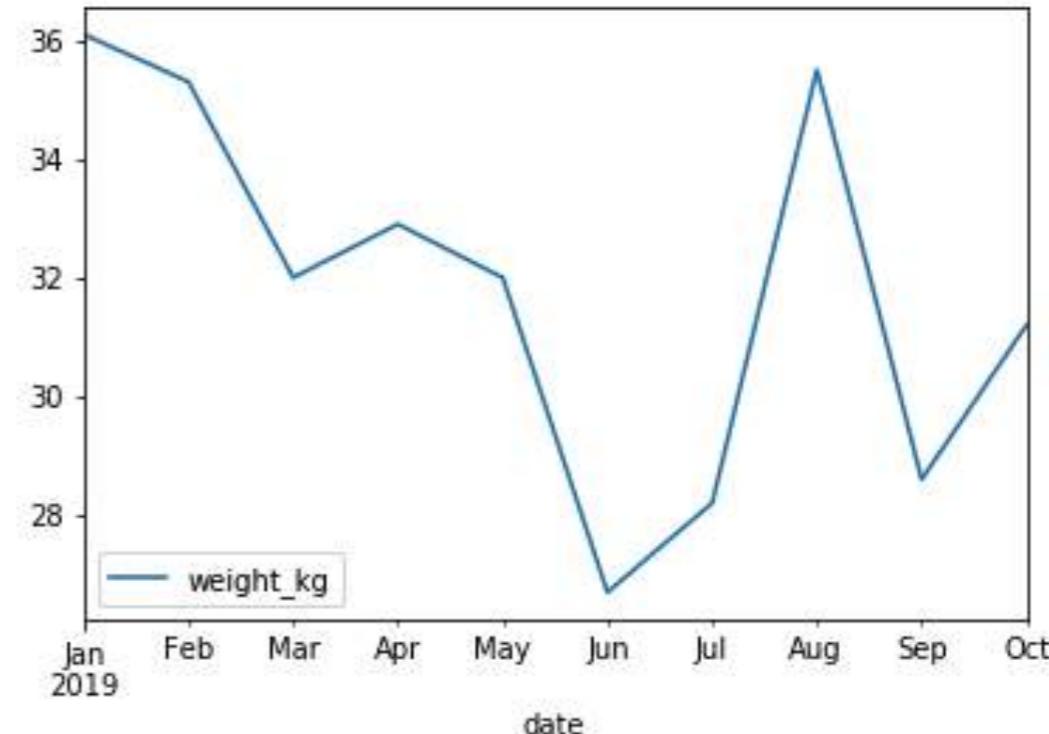


Line plots

```
sully.head()
```

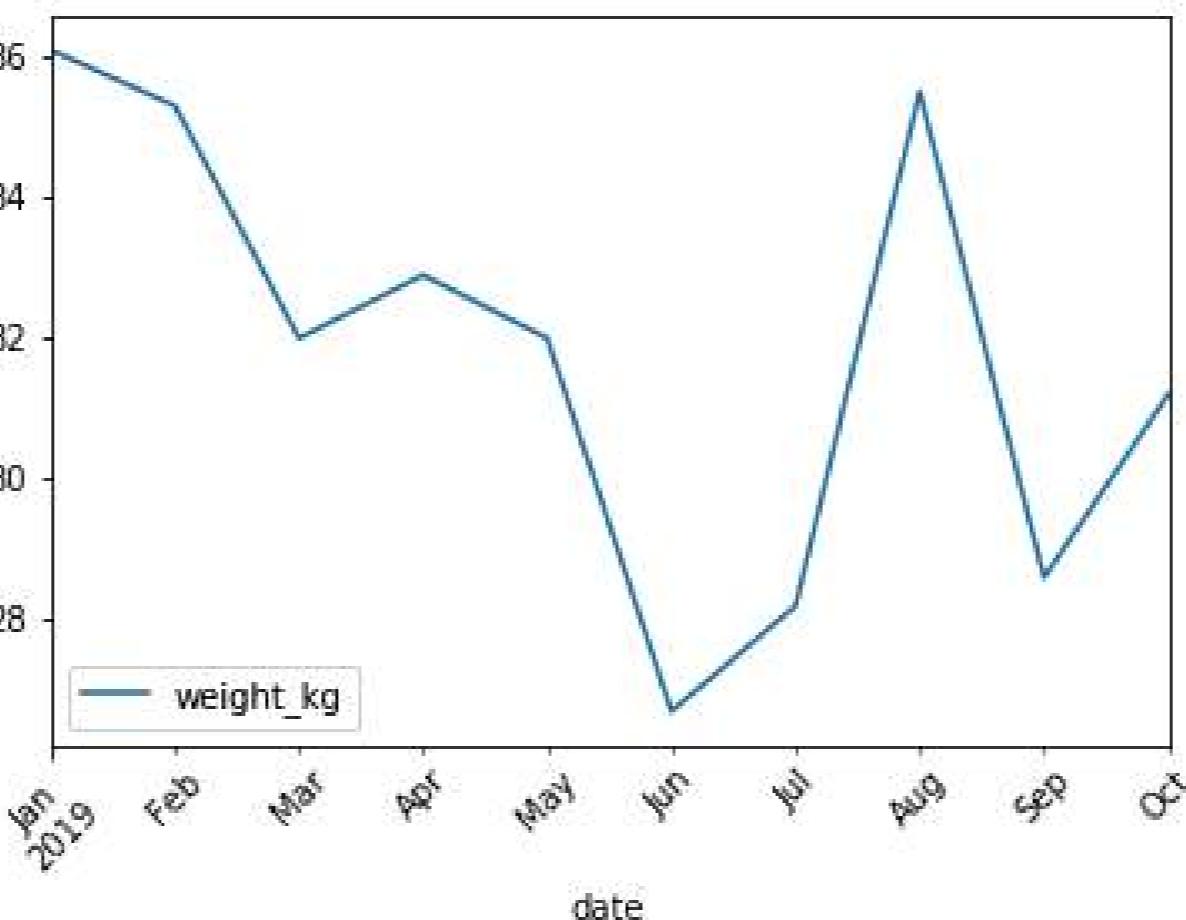
```
      date    weight_kg
0 2019-01-31        36.1
1 2019-02-28        35.3
2 2019-03-31        32.0
3 2019-04-30        32.9
4 2019-05-31        32.0
```

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



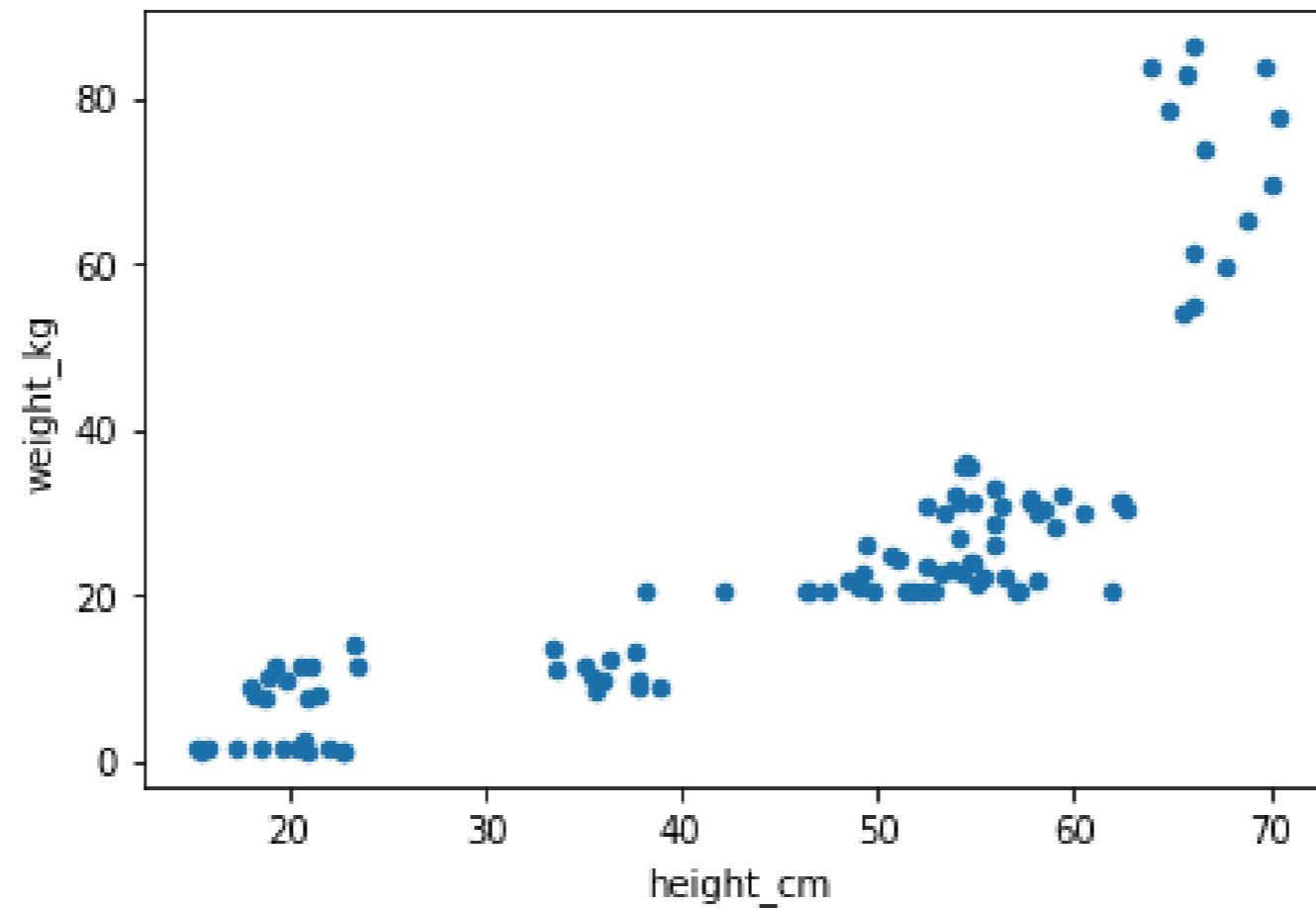
Rotating axis labels

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



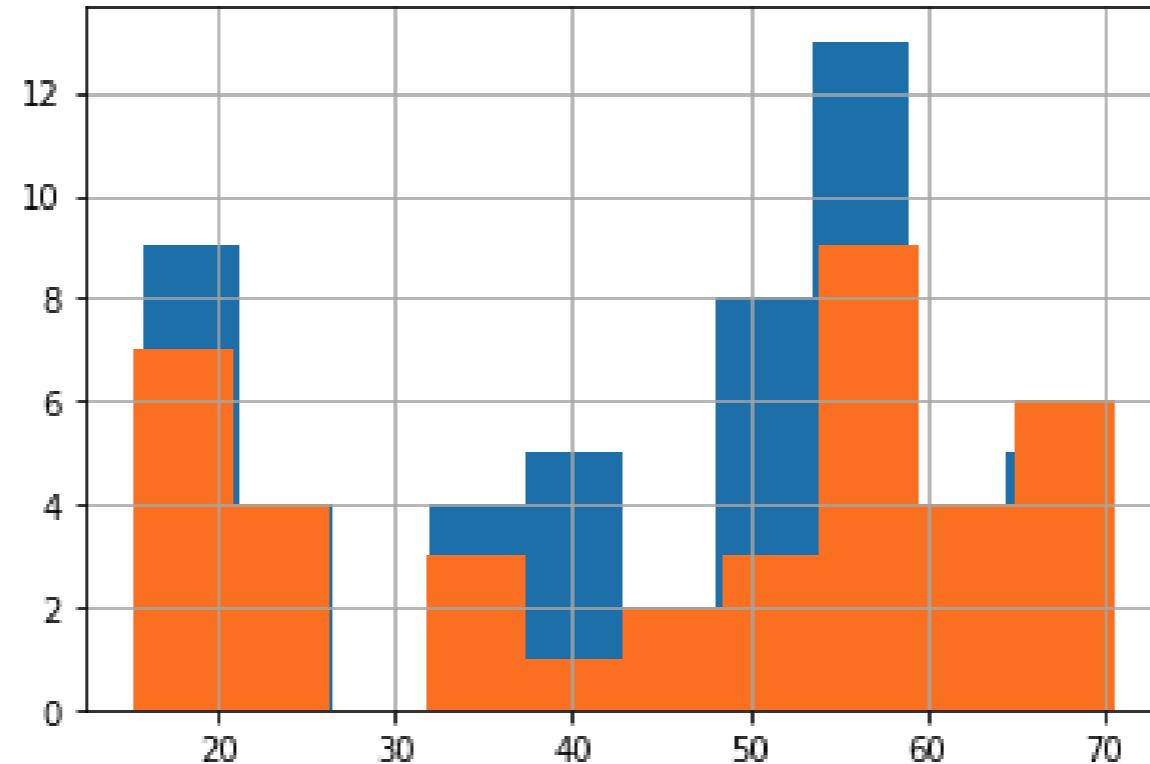
Scatter plots

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



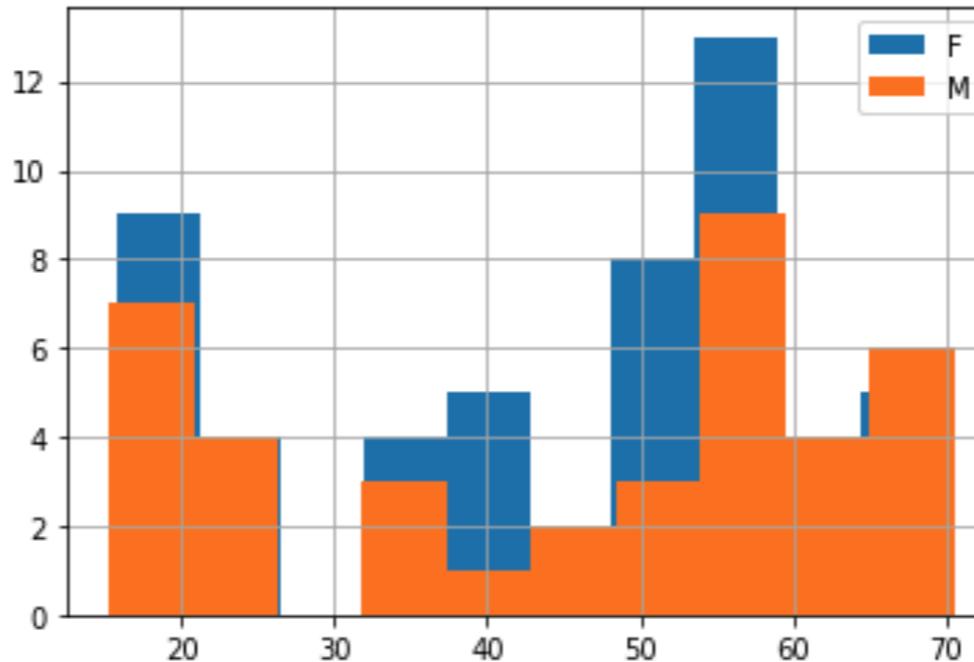
Layering plots

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



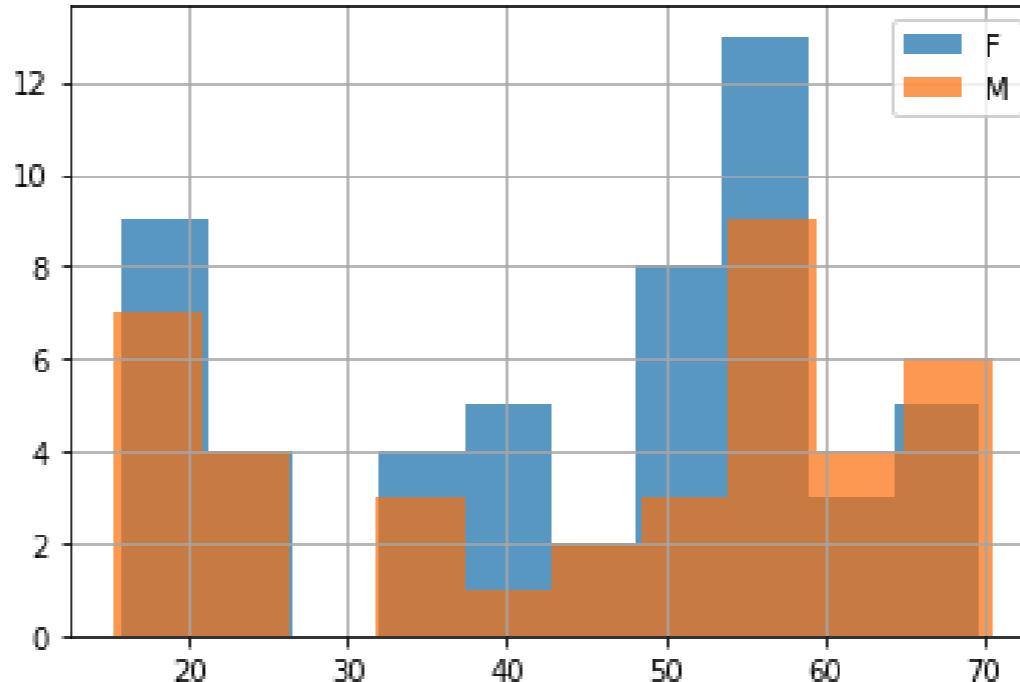
Add a legend

```
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()
```



Avocados

```
print(avocados)
```

	date	type	year	avg_price	size	nb_sold
0	2015-12-27	conventional	2015	0.95	small	9626901.09
1	2015-12-20	conventional	2015	0.98	small	8710021.76
2	2015-12-13	conventional	2015	0.93	small	9855053.66
...
1011	2018-01-21	organic	2018	1.63	extra_large	1490.02
1012	2018-01-14	organic	2018	1.59	extra_large	1580.01
1013	2018-01-07	organic	2018	1.51	extra_large	1289.07

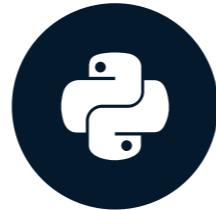
[1014 rows x 6 columns]

Let's practice!

DATA MANIPULATION WITH PANDAS

Missing values

DATA MANIPULATION WITH PANDAS



Maggie Matsui

Senior Content Developer at DataCamp

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

What's a missing value?

Name	Breed	Color	Height (cm)	Weight (kg)	Date of Birth
Bella	Labrador	Brown	56	?	2013-07-01
Charlie	Poodle	Black	43	23	2016-09-16
Lucy	Chow Chow	Brown	46	22	2014-08-25
Cooper	Schnauzer	Gray	49	?	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 in pandas DataFrames

```
print(dogs)
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	NaN	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	NaN	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

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
```

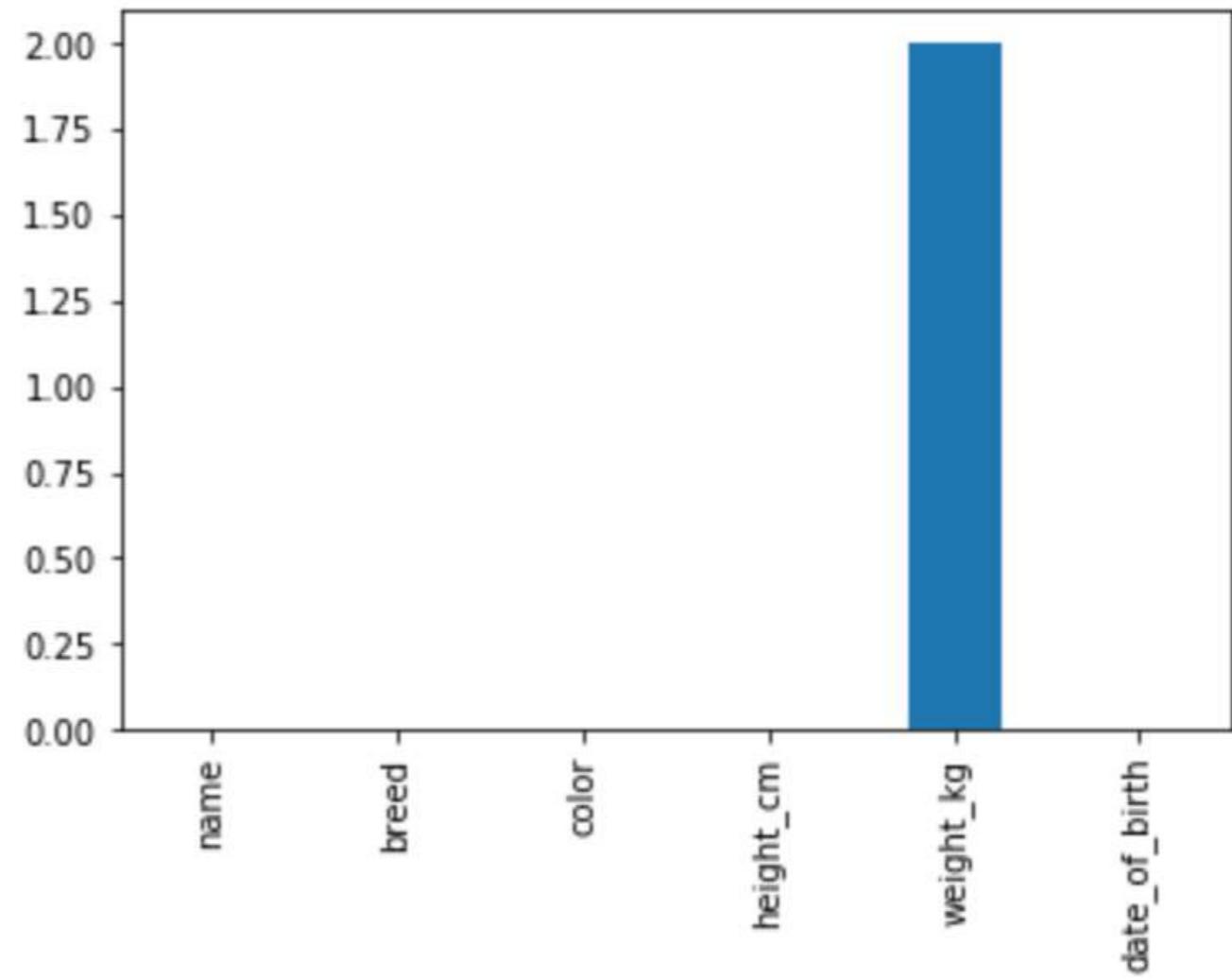
Counting missing values

```
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()
```



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

Replacing missing values

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

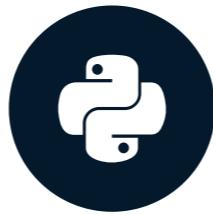
	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	0.0	2013-07-01
1	Charlie	Poodle	Black	43	24.0	2016-09-16
2	Lucy	Chow Chow	Brown	46	24.0	2014-08-25
3	Cooper	Schnauzer	Gray	49	0.0	2011-12-11
4	Max	Labrador	Black	59	29.0	2017-01-20
5	Stella	Chihuahua	Tan	18	2.0	2015-04-20
6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

Let's practice!

DATA MANIPULATION WITH PANDAS

Creating DataFrames

DATA MANIPULATION WITH PANDAS



Maggie Matsui

Senior Content Developer at DataCamp

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

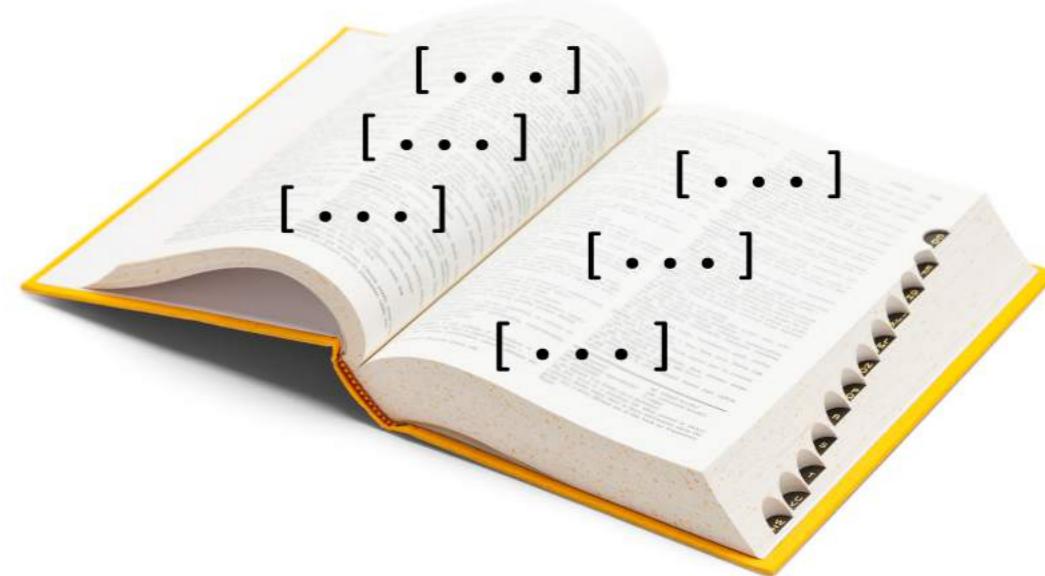
From a list of dictionaries

- Constructed row by row



From a dictionary of lists

- Constructed column by column



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)
```

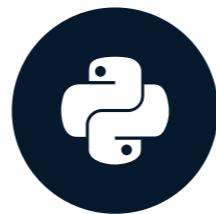
```
      name      breed  height_cm  weight_kg  date_of_birth
0  Ginger  Dachshund        22         10  2019-03-14
1    Scout   Dalmatian        59         25  2019-05-09
```

Let's practice!

DATA MANIPULATION WITH PANDAS

Reading and writing CSVs

DATA MANIPULATION WITH PANDAS

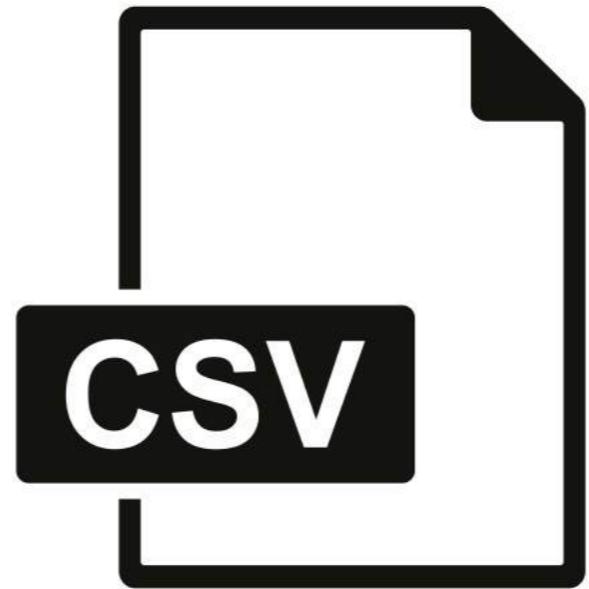


Maggie Matsui

Senior Content Developer at DataCamp

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



Example CSV file

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.csv

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

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
```

Let's practice!

DATA MANIPULATION WITH PANDAS

Wrap-up

DATA MANIPULATION WITH PANDAS



Maggie Matsui

Senior Content Developer at DataCamp

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

More to learn

- [Joining Data with pandas](#)
- [Streamlined Data Ingestion with pandas](#)
- [Analyzing Police Activity with pandas](#)
- [Analyzing Marketing Campaigns with pandas](#)

Congratulations!

DATA MANIPULATION WITH PANDAS

Inner join

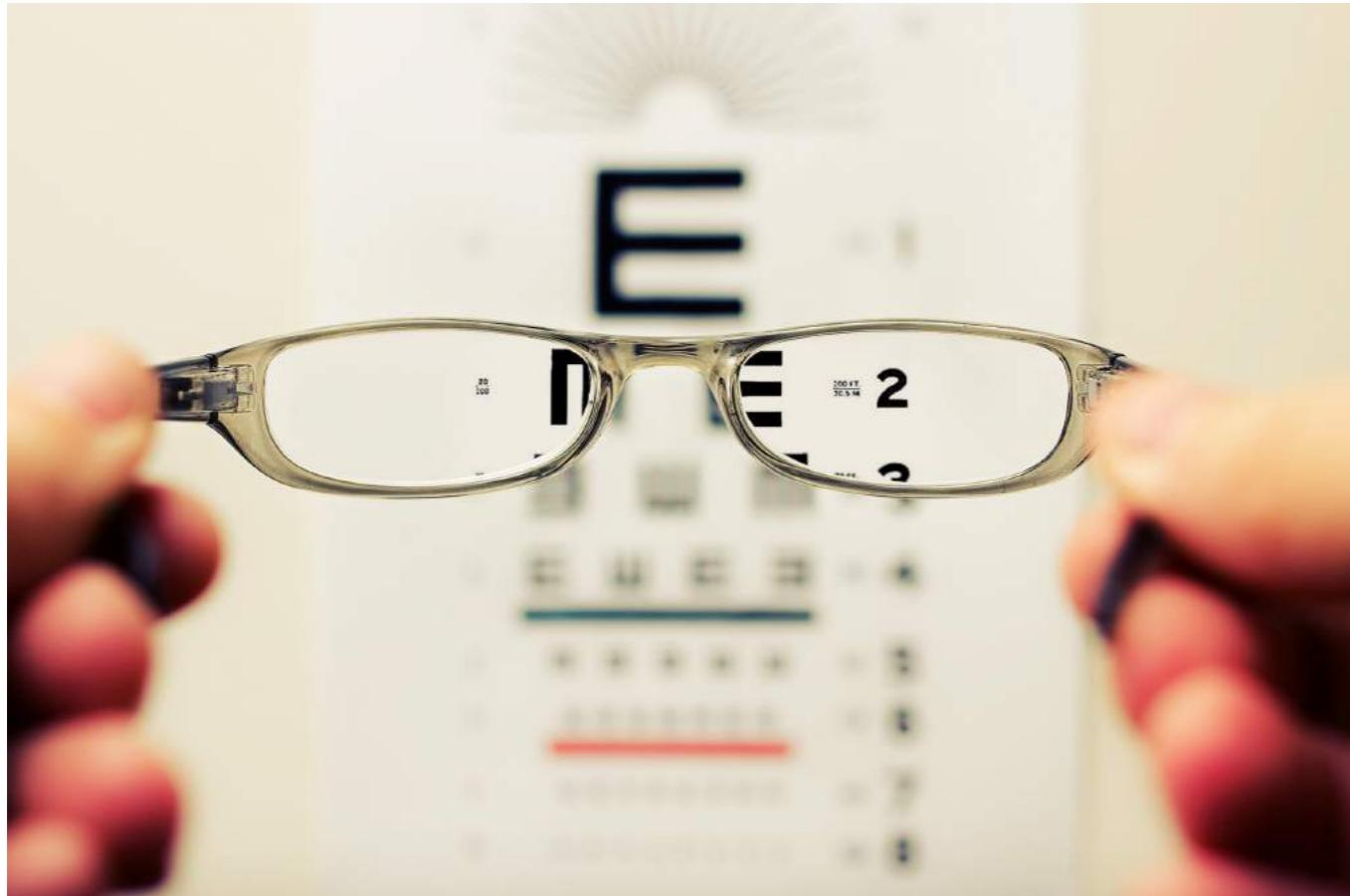
JOINING DATA WITH PANDAS



Aaren Stubberfield

Instructor

For clarity



Tables = DataFrames

Merging = Joining

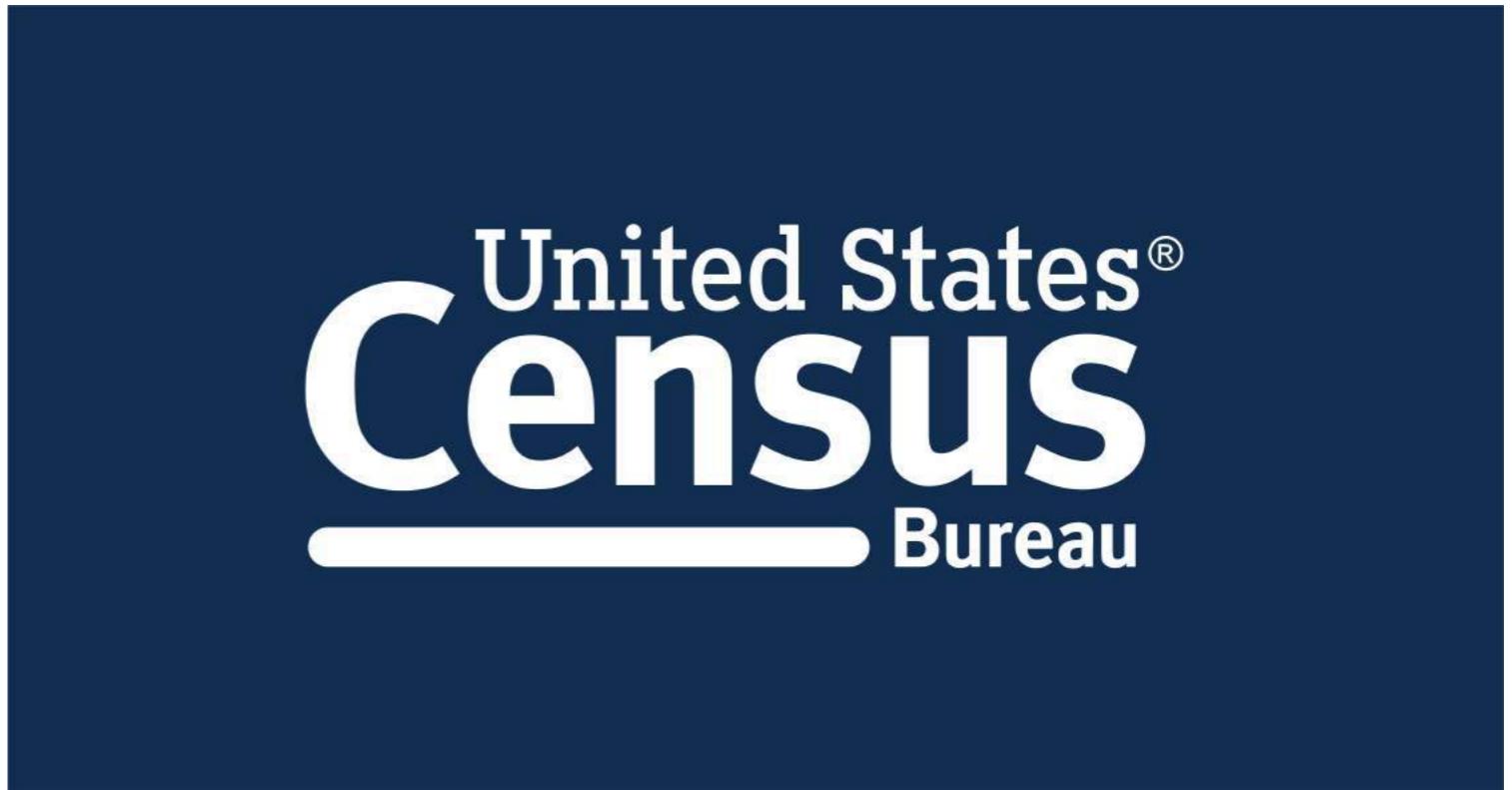
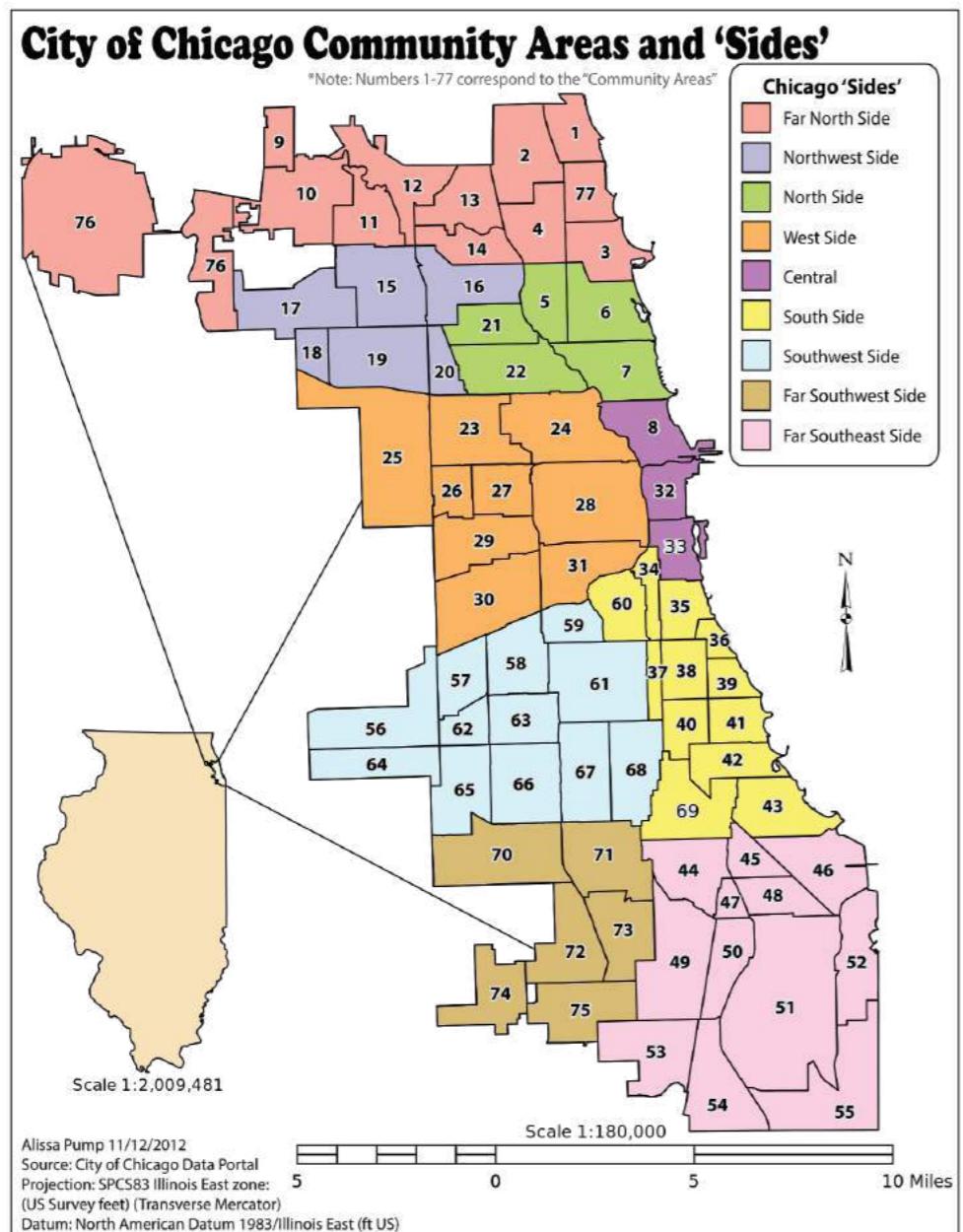
¹ Photo by David Travis on Unsplash

Chicago data portal dataset



¹ Photo by Pedro Lastra on Unsplash

Datasets for example



¹ Ward image By Alissapump, Own work, CC BY-SA 3.0

The ward data

```
wards = pd.read_csv('Ward_Offices.csv')
print(wards.head())
print(wards.shape)
```

```
   ward  alderman          address        zip
0  1    Proco "Joe" ...  2058 NORTH W...  60647
1  2    Brian Hopkins  1400 NORTH ...  60622
2  3     Pat Dowell  5046 SOUTH S...  60609
3  4  William D. B...  435 EAST 35T...  60616
4  5  Leslie A. Ha...  2325 EAST 71...  60649
(50, 4)
```

Census data

```
census = pd.read_csv('Ward_Census.csv')  
print(census.head())  
print(census.shape)
```

```
   ward  pop_2000  pop_2010  change      address          zip  
0   1       52951     56149      6%  2765 WEST SA...  60647  
1   2       54361     55805      3%    WM WASTE MAN...  60622  
2   3       40385     53039     31%  17 EAST 38TH...  60653  
3   4       51953     54589      5%   31ST ST HARB...  60653  
4   5       55302     51455     -7%  JACKSON PARK...  60637  
(50, 6)
```

Merging tables

	ward	alderman	address	zip
0	1	Proco "Joe" ...	2058 NORTH W...	60647
1	2	Brian Hopkins	1400 NORTH ...	60622
2	3	Pat Dowell	5046 SOUTH S...	60609
3	4	William D. B...	435 EAST 35T...	60616
4	5	Leslie A. Ha...	2325 EAST 71...	60649

	ward	pop_2000	pop_2010	change	address	zip
0	1	52951	56149	6%	2765 WEST SA...	60647
1	2	54361	55805	3%	WM WASTE MAN...	60622
2	3	40385	53039	31%	17 EAST 38TH...	60653
3	4	51953	54589	5%	31ST ST HARB...	60653
4	5	55302	51455	-7%	JACKSON PARK...	60637

Inner join

```
wards_census = wards.merge(census, on='ward')
print(wards_census.head(4))
```

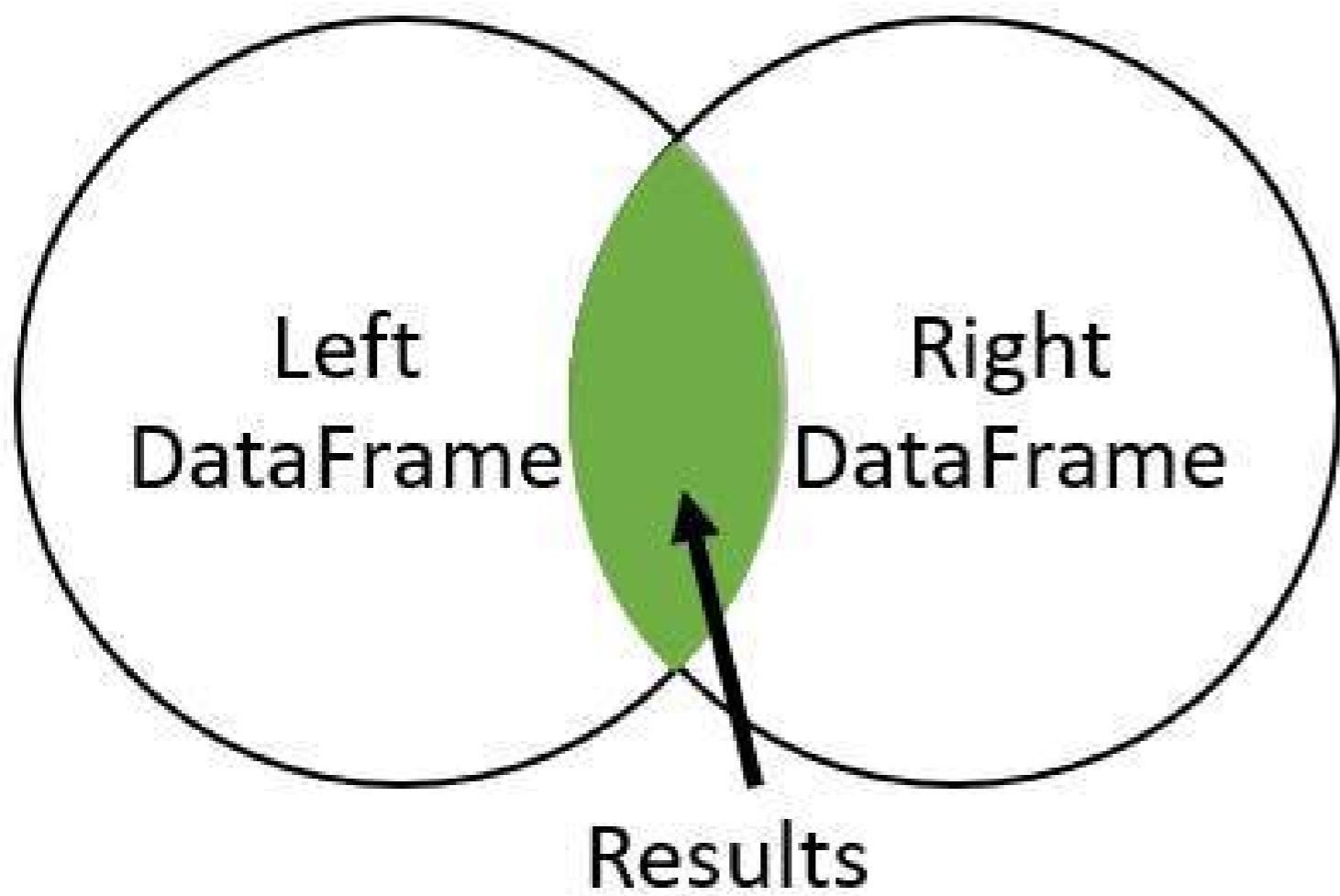
	ward	alderman	address_x	zip_x	pop_2000	pop_2010	change	address_y	zip_y
0	1	Proco "Joe" ...	2058 NORTH W...	60647	52951	56149	6%	2765 WEST SA...	60647
1	2	Brian Hopkins	1400 NORTH ...	60622	54361	55805	3%	WM WASTE MAN...	60622
2	3	Pat Dowell	5046 SOUTH S...	60609	40385	53039	31%	17 EAST 38TH...	60653
3	4	William D. B...	435 EAST 35T...	60616	51953	54589	5%	31ST ST HARB...	60653

```
print(wards_census.shape)
```

```
(50, 9)
```

Inner join

Inner Join



Suffixes

```
print(wards_census.columns)
```

```
Index(['ward', 'alderman', 'address_x', 'zip_x', 'pop_2000', 'pop_2010', 'change',
       'address_y', 'zip_y'],
      dtype='object')
```

Suffixes

```
wards_census = wards.merge(census, on='ward', suffixes=('_ward', '_cen'))  
print(wards_census.head())  
print(wards_census.shape)
```

	ward	alderman	address_ward	zip_ward	pop_2000	pop_2010	change	address_cen	zi
0	1	Proco "Joe" ...	2058 NORTH W...	60647	52951	56149	6%	2765 WEST SA...	60
1	2	Brian Hopkins	1400 NORTH ...	60622	54361	55805	3%	WM WASTE MAN...	60
2	3	Pat Dowell	5046 SOUTH S...	60609	40385	53039	31%	17 EAST 38TH...	60
3	4	William D. B...	435 EAST 35T...	60616	51953	54589	5%	31ST ST HARB...	60
4	5	Leslie A. Ha...	2325 EAST 71...	60649	55302	51455	-7%	JACKSON PARK...	60
(50, 9)									

Let's practice!

JOINING DATA WITH PANDAS

One to many relationships

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

One-to-one

A	B	C	C	D
A1	B1	C1	C1	D1
A2	B2	C2	C2	D2
A3	B3	C3	C3	D3

One-To-One = Every row in the left table is related to only one row in the right table

One-to-one example

	ward	alderman	address	zip
0	1	Proco "Joe" ...	2058 NORTH W...	60647
1	2	Brian Hopkins	1400 NORTH ...	60622
2	3	Pat Dowell	5046 SOUTH S...	60609
3	4	William D. B...	435 EAST 35T...	60616
4	5	Leslie A. Ha...	2325 EAST 71...	60649

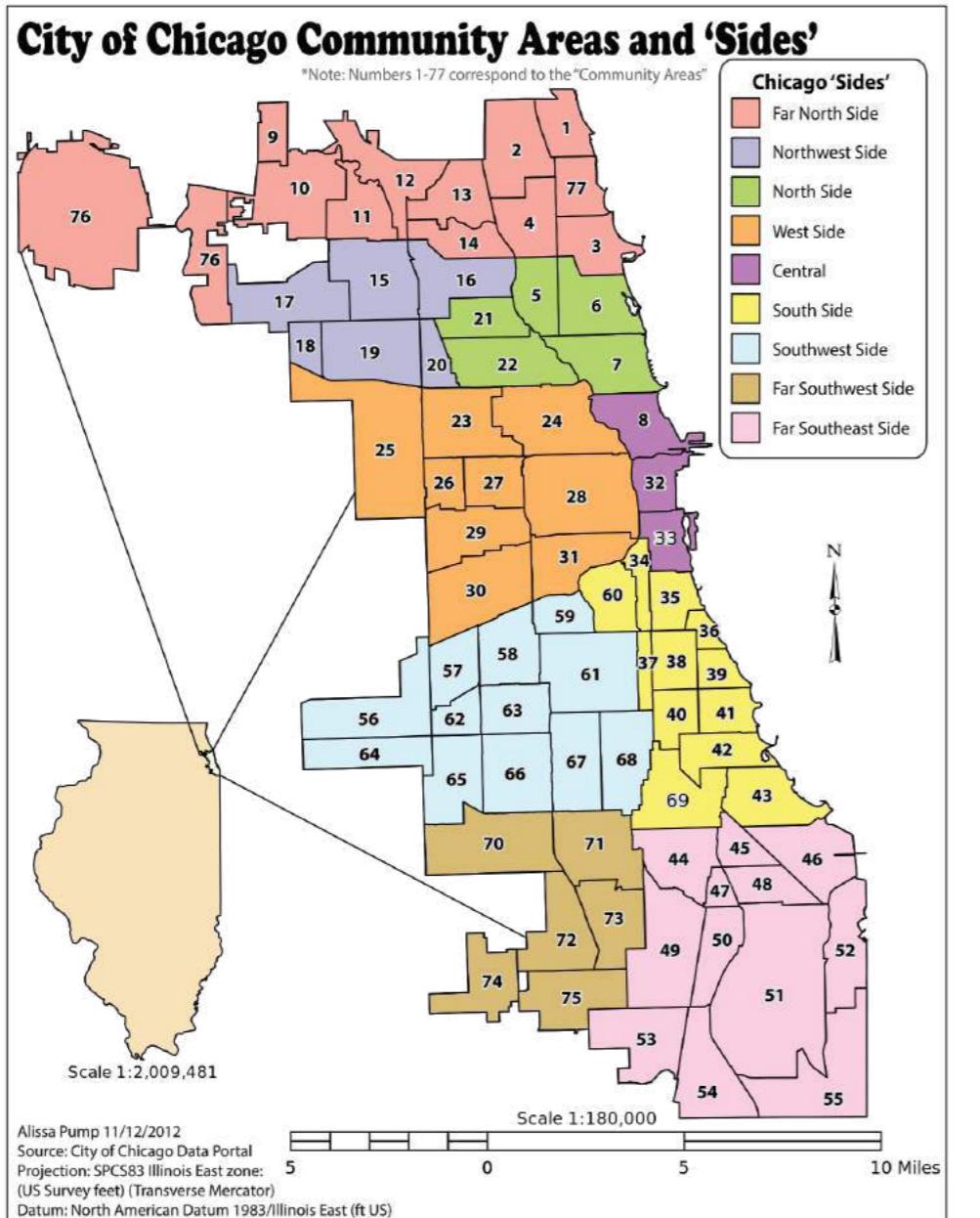
	ward	pop_2000	pop_2010	change	address	zip
0	1	52951	56149	6%	2765 WEST SA...	60647
1	2	54361	55805	3%	WM WASTE MAN...	60622
2	3	40385	53039	31%	17 EAST 38TH...	60653
3	4	51953	54589	5%	31ST ST HARB...	60653
4	5	55302	51455	-7%	JACKSON PARK...	60637

One-to-many

A	B	C	C	D
A1	B1	C1	C1	D1
A2	B2	C2	C1	D2
A3	B3	C3	C1	D3

One-To-Many = Every row in left table is related to one or more rows in the right table

One-to-many example



One-to-many example

```
licenses = pd.read_csv('Business_Licenses.csv')
print(licenses.head())
print(licenses.shape)
```

```
   account  ward  aid business          address      zip
0  307071     3  743 REGGIE'S BAR...  2105 S STATE ST  60616
1    10     10  829 HONEYBEERS  13200 S HOUS...  60633
2  10002     14  775 CELINA DELI  5089 S ARCHE...  60632
3  10005     12    nan KRAFT FOODS ...  2005 W 43RD ST  60609
4  10044     44  638 NEYBOUR'S TA...  3651 N SOUTH...  60613
(10000, 6)
```

One-to-many example

	ward	alderman	address	zip
0	1	Proco "Joe" ...	2058 NORTH W...	60647
1	2	Brian Hopkins	1400 NORTH ...	60622
2	3	Pat Dowell	5046 SOUTH S...	60609
3	4	William D. B...	435 EAST 35T...	60616
4	5	Leslie A. Ha...	2325 EAST 71...	60649

	account	ward	aid	business	address	zip
0	307071	3	743	REGGIE'S BAR...	2105 S STATE ST	60616
1	10	10	829	HONEYBEERS	13200 S HOUS...	60633
2	10002	14	775	CELINA DELI	5089 S ARCHE...	60632
3	10005	12	nan	KRAFT FOODS ...	2005 W 43RD ST	60609
4	10044	44	638	NEYBOUR'S TA...	3651 N SOUTH...	60613

One-to-many example

```
ward_licenses = wards.merge(licenses, on='ward', suffixes=('_ward', '_lic'))  
print(ward_licenses.head())
```

	ward	alderman	address_ward	zip_ward	account	aid	business	address_lic
0	1	Proco "Joe" ...	2058 NORTH W...	60647	12024	nan	DIGILOG ELEC...	1038 N ASHLA...
1	1	Proco "Joe" ...	2058 NORTH W...	60647	14446	743	EMPTY BOTTLE...	1035 N WESTE...
2	1	Proco "Joe" ...	2058 NORTH W...	60647	14624	775	LITTLE MEL'S...	2205 N CALIF...
3	1	Proco "Joe" ...	2058 NORTH W...	60647	14987	nan	MR. BROWN'S ...	2301 W CHICA...
4	1	Proco "Joe" ...	2058 NORTH W...	60647	15642	814	Beat Kitchen	2000-2100 W ...

One-to-many example

```
print(wards.shape)
```

```
(50, 4)
```

```
print(ward_licenses.shape)
```

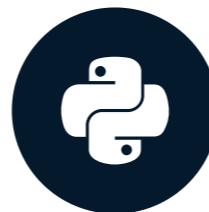
```
(10000, 9)
```

Let's practice!

JOINING DATA WITH PANDAS

Merging multiple DataFrames

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

Merging multiple tables

A	B	C	D
A1	B1	C1	C1
A2	B2	C2	D1
A3	B3	C3	D2
			D3

A	B	C	C	E	E	F	G
A1	B1	C1	C1	E1	E1	F1	G1
A2	B2	C2	C2	E2	E2	F2	G2
A3	B3	C3	C3	E3	E3	F3	G3

Remembering the licenses table

```
print(licenses.head())
```

```
account    ward    aid   business           address      zip
0 307071     3     743  REGGIE'S BAR...  2105 S STATE ST  60616
1 10          10    829  HONEYBEERS       13200 S HOUS...  60633
2 10002      14    775  CELINA DELI        5089 S ARCHE...  60632
3 10005      12    nan  KRAFT FOODS ...  2005 W 43RD ST  60609
4 10044      44    638  NEYBOUR'S TA...  3651 N SOUTH...  60613
```

Remembering the wards table

```
print(wards.head())
```

```
   ward  alderman          address      zip  
0  1    Proco "Joe" ...  2058 NORTH W...  60647  
1  2    Brian Hopkins  1400 NORTH ...  60622  
2  3    Pat Dowell    5046 SOUTH S...  60609  
3  4    William D. B...  435 EAST 35T...  60616  
4  5    Leslie A. Ha...  2325 EAST 71...  60649
```

Review new data

```
grants = pd.read_csv('Small_Business_Grant_Agreements.csv')  
print(grants.head())
```

	address	zip	grant	company
0	1000 S KOSTN...	60624	148914.50	NATIONWIDE F...
1	1000 W 35TH ST	60609	100000.00	SMALL BATCH,...
2	1000 W FULTO...	60612	34412.50	FULTON MARKE...
3	10008 S WEST...	60643	12285.32	LAW OFFICES ...
4	1002 W ARGYL...	60640	28998.75	MASALA'S IND...

Tables to merge

	address	zip	grant	company
0	1031 N CICER...	60651	150000.00	1031 HANS LLC
1	1375 W LAKE ST	60612	150000.00	1375 W LAKE ...
2	1800 W LAKE ST	60612	47700.00	1800 W LAKE LLC
3	4311 S HALST...	60609	87350.63	4311 S. HALS...
4	1747 W CARRO...	60612	50000.00	ACE STYLINE ...

	account	ward	aid	business	address	zip
0	307071	3	743	REGGIE'S BAR...	2105 S STATE ST	60616
1	10	10	829	HONEYBEERS	13200 S HOUS...	60633
2	10002	14	775	CELINA DELI	5089 S ARCHE...	60632
3	10005	12	nan	KRAFT FOODS ...	2005 W 43RD ST	60609
4	10044	44	638	NEYBOUR'S TA...	3651 N SOUTH...	60613

Theoretical merge

```
grants_licenses = grants.merge(licenses, on='zip')
print(grants_licenses.loc[grants_licenses['business']=="REGGIE'S BAR & GRILL",
                           ['grant','company','account','ward','business']])
```

```
grant      company      account      ward      business
0 136443.07  CEDARS MEDIT...  307071      3  REGGIE'S BAR...
1 39943.15    DARRYL & FYL...  307071      3  REGGIE'S BAR...
2 31250.0     JGF MANAGEMENT  307071      3  REGGIE'S BAR...
3 143427.79   HYDE PARK AN...  307071      3  REGGIE'S BAR...
4 69500.0      ZBERRY INC     307071      3  REGGIE'S BAR...
```

Single merge

```
grants.merge(licenses, on=['address', 'zip'])
```

	address	zip	grant	company	account	ward	aid	business
0	1020 N KOLMA...	60651	68309.8	TRITON INDUS...	7689	37	929	TRITON INDUS...
1	10241 S COMM...	60617	33275.5	SOUTH CHICAG...	246598	10	nan	SOUTH CHICAG...
2	11612 S WEST...	60643	30487.5	BEVERLY RECO...	3705	19	nan	BEVERLY RECO...
3	1600 S KOSTN...	60623	128513.7	CHARTER STEE...	293825	24	nan	LEELO STEEL,...
4	1647 W FULTO...	60612	5634.0	SN PECK BUIL...	85595	27	673	S.N. PECK BU...

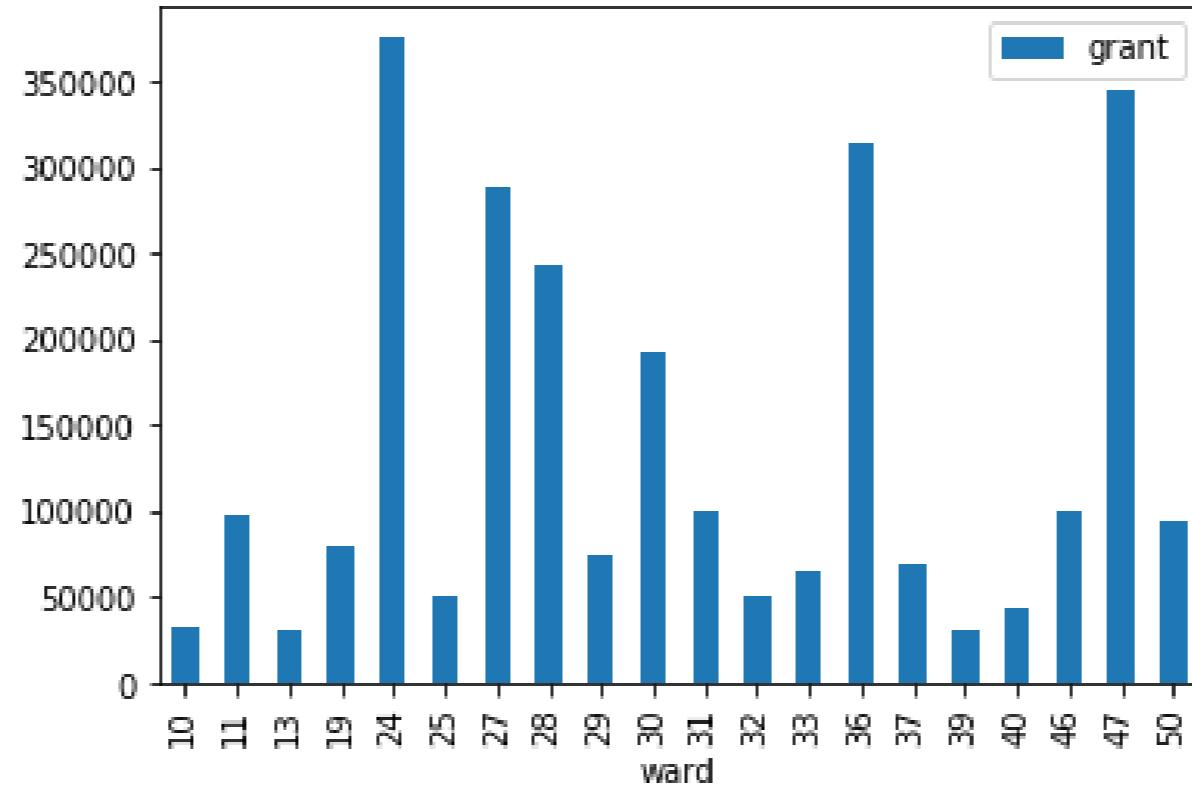
Merging multiple tables

```
grants_licenses_ward = grants.merge(licenses, on=['address','zip']) \
    .merge(wards, on='ward', suffixes=('_bus','_ward'))
grants_licenses_ward.head()
```

	address_bus	zip_bus	grant	company	account	ward	aid	business	alderma
0	1020 N KOLMA...	60651	68309.8	TRITON INDUS...	7689	37	929	TRITON INDUS...	Emma M.
1	10241 S COMM...	60617	33275.5	SOUTH CHICAG...	246598	10	nan	SOUTH CHICAG...	Susan S
2	11612 S WEST...	60643	30487.5	BEVERLY RECO...	3705	19	nan	BEVERLY RECO...	Matthew
3	3502 W 111TH ST	60655	50000.0	FACE TO FACE...	263274	19	704	FACE TO FACE	Matthew
4	1600 S KOSTN...	60623	128513.7	CHARTER STEE...	293825	24	nan	LEELO STEEL,...	Michael

Results

```
import matplotlib.pyplot as plt  
  
grant_licenses_ward.groupby('ward').agg('sum').plot(kind='bar', y='grant')  
plt.show()
```



Merging even more...

Three tables:

```
df1.merge(df2, on='col') \  
    .merge(df3, on='col')
```

Four tables:

```
df1.merge(df2, on='col') \  
    .merge(df3, on='col') \  
    .merge(df4, on='col')
```

Let's practice!

JOINING DATA WITH PANDAS

Left join

JOINING DATA WITH PANDAS

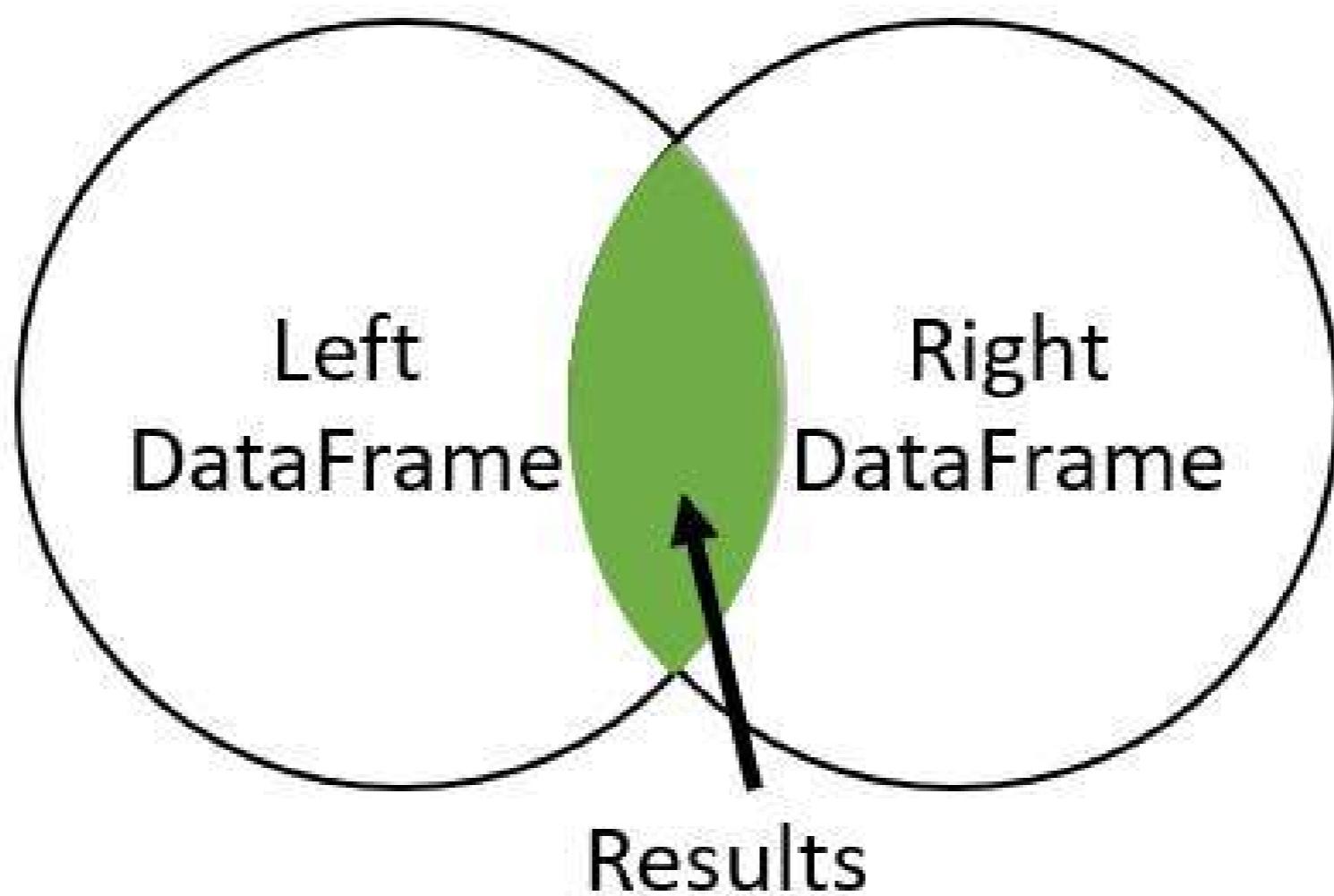


Aaren Stubberfield

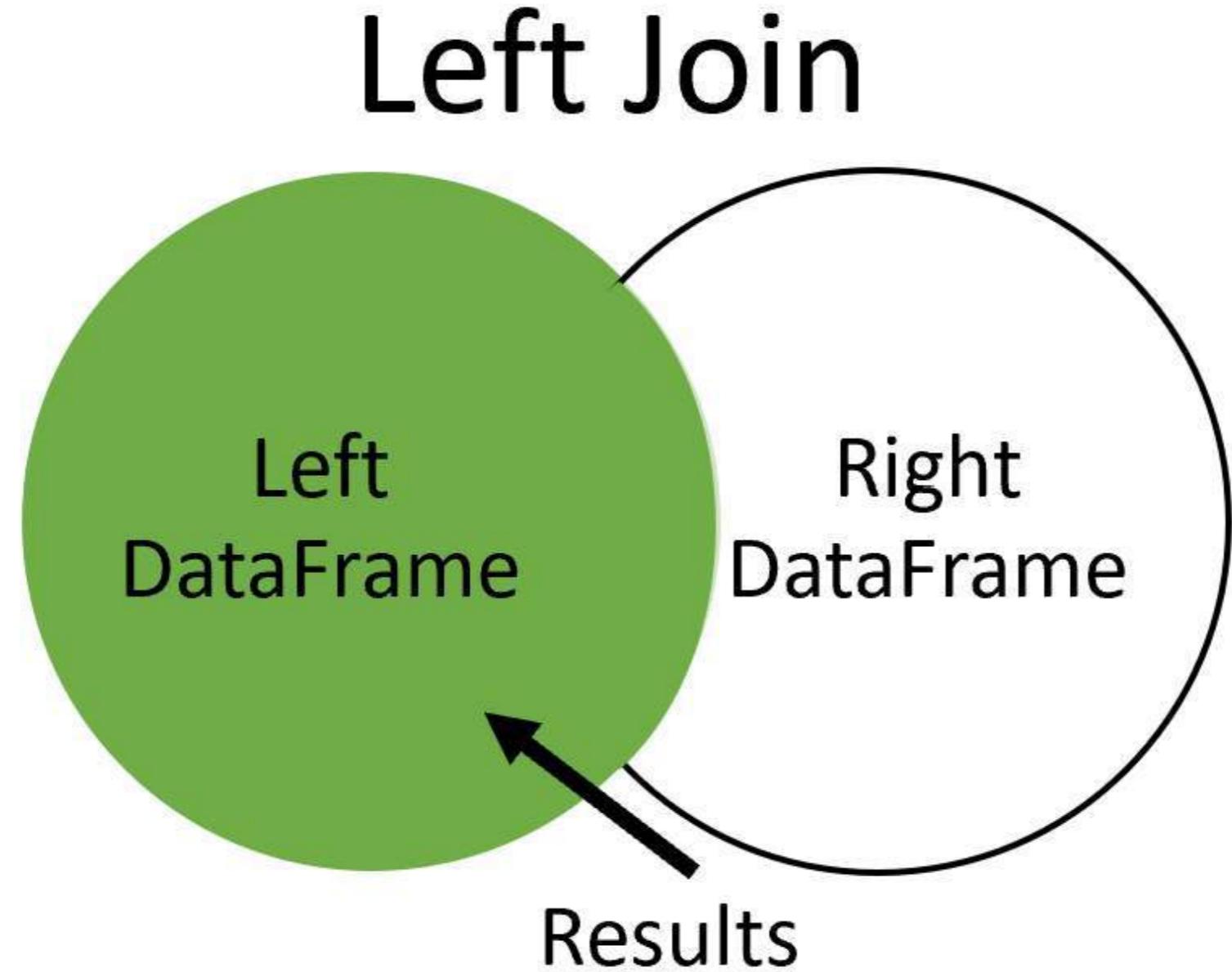
Instructor

Quick review

Inner Join



Left join



Left join

Left Table

A	B	C
A2	B2	C2
A3	B3	C3
A4	B4	C4

Right Table

C	D
C1	D1
C2	D2
C4	D4
C5	D5

Result Table

A	B	C	D
A2	B2	C2	D2
A3	B3	C3	
A4	B4	C4	D4

New dataset



Movies table

```
movies = pd.read_csv('tmdb_movies.csv')
print(movies.head())
print(movies.shape)
```

```
   id      original_title    popularity      release_date
0  257        Oliver Twist     20.415572  2005-09-23
1 14290    Better Luck ...     3.877036  2002-01-12
2  38365       Grown Ups     38.864027  2010-06-24
3  9672        Infamous  3.6808959999...
4 12819  Alpha and Omega     12.300789  2010-09-17
(4803, 4)
```

Tagline table

```
taglines = pd.read_csv('tmdb_taglines.csv')  
print(taglines.head())  
print(taglines.shape)
```

```
id      tagline  
0 19995  Enter the World of Pandora.  
1 285    At the end of the world, the adventure begins.  
2 206647 A Plan No One Escapes  
3 49026  The Legend Ends  
4 49529  Lost in our world, found in another.  
(3955, 2)
```

Merge with left join

```
movies_taglines = movies.merge(taglines, on='id', how='left')
print(movies_taglines.head())
```

	<code>id</code>	<code>original_title</code>	<code>popularity</code>	<code>release_date</code>	<code>tagline</code>
0	257	Oliver Twist	20.415572	2005-09-23	NaN
1	14290	Better Luck ...	3.877036	2002-01-12	Never undere...
2	38365	Grown Ups	38.864027	2010-06-24	Boys will be...
3	9672	Infamous	3.6808959999...	2006-11-16	There's more...
4	12819	Alpha and Omega	12.300789	2010-09-17	A Pawsome 3D...

Number of rows returned

```
print(movies_taglines.shape)
```

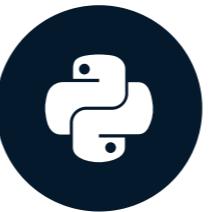
```
(4805, 5)
```

Let's practice!

JOINING DATA WITH PANDAS

Other joins

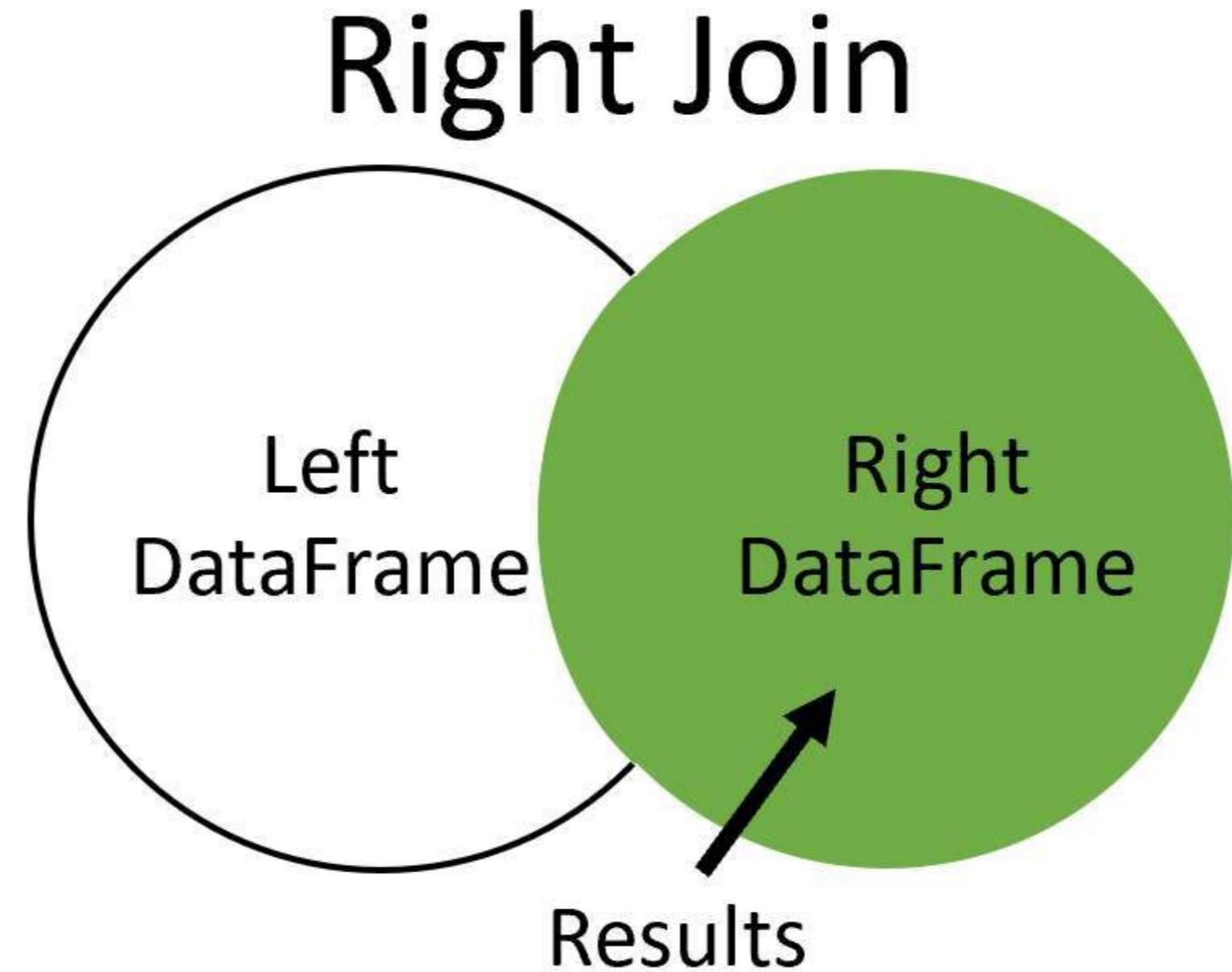
JOINING DATA WITH PANDAS



Aaren Stubberfield

Instructor

Right join



Right join

Left Table

A	B	C
A2	B2	C2
A3	B3	C3
A4	B4	C4

Right Table

C	D
C1	D1
C2	D2
C4	D4
C5	D5

Result Table

A	B	C	D
		C1	D1
A2	B2	C2	D2
A4	B4	C4	D4
		C5	D5

Looking at data

```
movie_to_genres = pd.read_csv('tmdb_movie_to_genres.csv')
tv_genre = movie_to_genres[movie_to_genres['genre'] == 'TV Movie']
print(tv_genre)
```

```
   movie_id  genre
4998    10947  TV Movie
5994    13187  TV Movie
7443    22488  TV Movie
10061   78814  TV Movie
10790   153397 TV Movie
10835   158150 TV Movie
11096   205321 TV Movie
11282   231617 TV Movie
```

Filtering the data

```
m = movie_to_genres['genre'] == 'TV Movie'  
tv_genre = movie_to_genres[m]  
print(tv_genre)
```

```
    movie_id  genre  
4998     10947  TV Movie  
5994     13187  TV Movie  
7443     22488  TV Movie  
10061    78814  TV Movie  
10790    153397 TV Movie  
10835    158150 TV Movie  
11096    205321 TV Movie  
11282    231617 TV Movie
```

Data to merge

	id	title	popularity	release_date
0	257	Oliver Twist	20.415572	2005-09-23
1	14290	Better Luck ...	3.877036	2002-01-12
2	38365	Grown Ups	38.864027	2010-06-24
3	9672	Infamous	3.6808959999...	2006-11-16
4	12819	Alpha and Omega	12.300789	2010-09-17

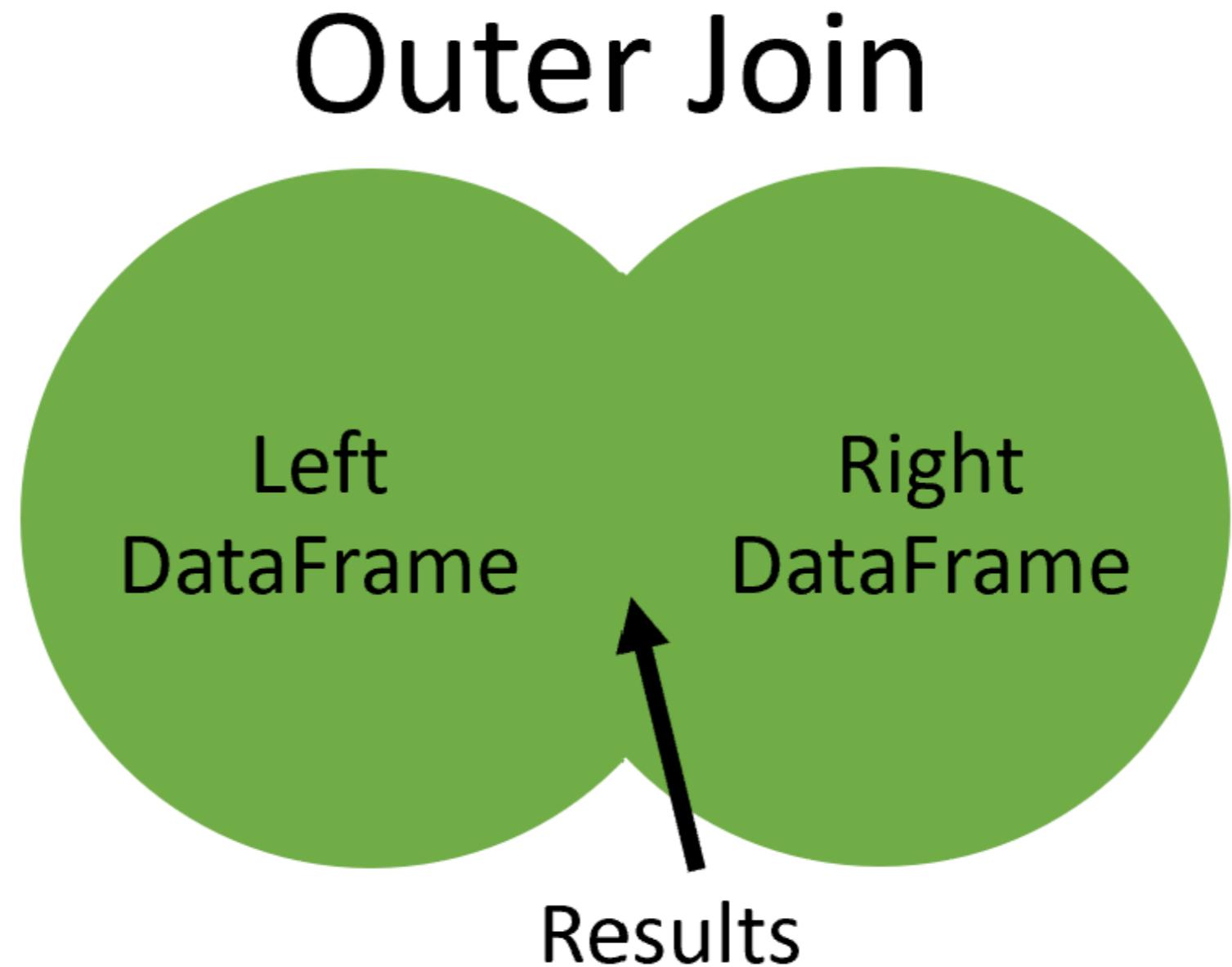
	movie_id	genre
4998	10947	TV Movie
5994	13187	TV Movie
7443	22488	TV Movie
10061	78814	TV Movie
10790	153397	TV Movie

Merge with right join

```
tv_movies = movies.merge(tv_genre, how='right',
                        left_on='id', right_on='movie_id')
print(tv_movies.head())
```

	id	title	popularity	release_date	movie_id	genre
0	153397	Restless	0.812776	2012-12-07	153397	TV Movie
1	10947	High School ...	16.536374	2006-01-20	10947	TV Movie
2	231617	Signed, Seal...	1.444476	2013-10-13	231617	TV Movie
3	78814	We Have Your...	0.102003	2011-11-12	78814	TV Movie
4	158150	How to Fall ...	1.923514	2012-07-21	158150	TV Movie

Outer join



Outer join

Left Table

A	B	C
A2	B2	C2
A3	B3	C3
A4	B4	C4

Right Table

C	D
C1	D1
C2	D2
C4	D4
C5	D5

Result Table

A	B	C	D
		C1	D1
A2	B2	C2	D2
A3	B3	C3	
A4	B4	C4	D4
		C5	D5

Datasets for outer join

```
m = movie_to_genres['genre'] == 'Family'  
family = movie_to_genres[m].head(3)
```

```
movie_id    genre  
0   12      Family  
1   35      Family  
2   105     Family
```

```
m = movie_to_genres['genre'] == 'Comedy'  
comedy = movie_to_genres[m].head(3)
```

```
movie_id    genre  
0   5       Comedy  
1   13      Comedy  
2   35      Comedy
```

Merge with outer join

```
family_comedy = family.merge(comedy, on='movie_id', how='outer',  
                             suffixes=('_fam', '_com'))  
print(family_comedy)
```

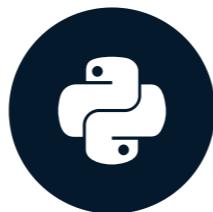
```
   movie_id  genre_fam  genre_com  
0      12     Family       NaN  
1      35     Family    Comedy  
2     105     Family       NaN  
3       5        NaN    Comedy  
4     13        NaN    Comedy
```

Let's practice!

JOINING DATA WITH PANDAS

Merging a table to itself

JOINING DATA WITH PANDAS

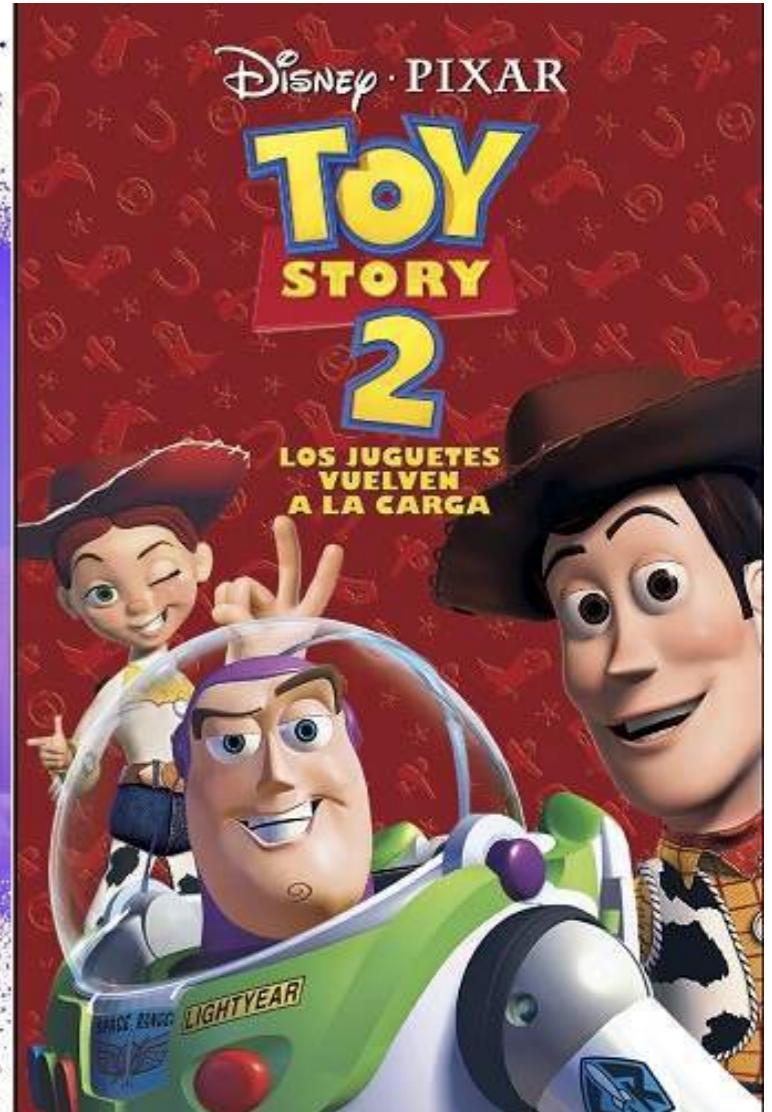
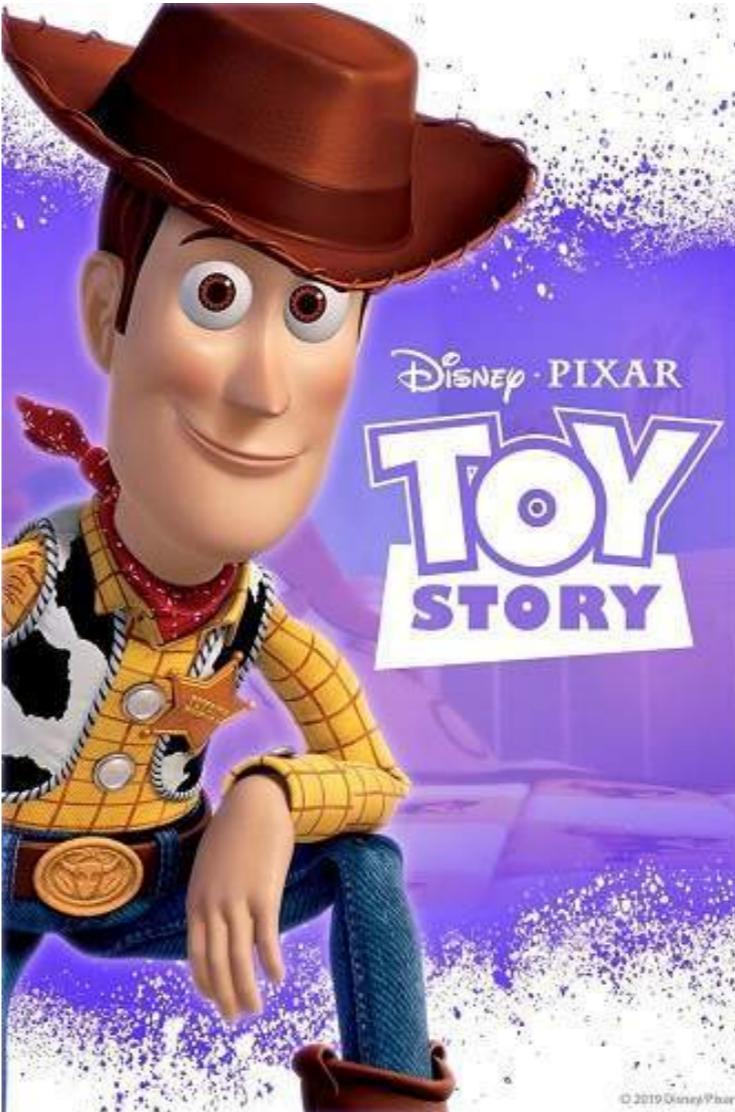


Aaren Stubberfield
Instructor

Sequel movie data

```
print(sequel.head())
```

	id	title	sequel
0	19995	Avatar	NaN
1	862	Toy Story	863
2	863	Toy Story 2	10193
3	597	Titanic	NaN
4	24428	The Avengers	NaN



Merging a table to itself

Left Table

id	title	sequel
19995	Avatar	
862	Toy Story	863
863	Toy Story 2	10193
597	Titanic	
24428	The Ave...	

Right Table

id	title	sequel
19995	Avatar	
862	Toy Story	863
863	Toy Story 2	10193
597	Titanic	
24428	The Ave...	

Result Table

id_x	title_x	sequel_x	id_y	title_y	sequel_y
862	Toy Story	863	863	Toy Story 2	10193
863	Toy Story 2	10193	10193	Toy Story 3	

Merge Columns

Merging a table to itself

```
original_sequels = sequels.merge(sequels, left_on='sequel', right_on='id',
                                  suffixes=('_org', '_seq'))
print(original_sequels.head())
```

	id_org	title_org	sequel_org	id_seq	title_seq	sequel_seq
0	862	Toy Story	863	863	Toy Story 2	10193
1	863	Toy Story 2	10193	10193	Toy Story 3	NaN
2	675	Harry Potter...	767	767	Harry Potter...	NaN
3	121	The Lord of ...	122	122	The Lord of ...	NaN
4	120	The Lord of ...	121	121	The Lord of ...	122

Continue format results

```
print(original_sequels[['title_org','title_seq']].head())
```

	title_org	title_seq
0	Toy Story	Toy Story 2
1	Toy Story 2	Toy Story 3
2	Harry Potter...	Harry Potter...
3	The Lord of ...	The Lord of ...
4	The Lord of ...	The Lord of ...

Merging a table to itself with left join

```
original_sequels = sequels.merge(sequels, left_on='sequel', right_on='id',
                                 how='left', suffixes('_org', '_seq'))
print(original_sequels.head())
```

	id_org	title_org	sequel_org	id_seq	title_seq	sequel_seq
0	19995	Avatar	NaN	NaN	NaN	NaN
1	862	Toy Story	863	863	Toy Story 2	10193
2	863	Toy Story 2	10193	10193	Toy Story 3	NaN
3	597	Titanic	NaN	NaN	NaN	NaN
4	24428	The Avengers	NaN	NaN	NaN	NaN

When to merge at table to itself

Common situations:

- Hierarchical relationships
- Sequential relationships
- Graph data

Let's practice!

JOINING DATA WITH PANDAS

Merging on indexes

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

Table with an index

	id	title	popularity	release_date
0	257	Oliver Twist	20.415572	2005-09-23
1	14290	Better Luck ...	3.877036	2002-01-12
2	38365	Grown Ups	38.864027	2010-06-24
3	9672	Infamous	3.680896	2006-11-16
4	12819	Alpha and Omega	12.300789	2010-09-17

	id	title	popularity	release_date
257	Oliver Twist	20.415572	2005-09-23	
14290	Better Luck ...	3.877036	2002-01-12	
38365	Grown Ups	38.864027	2010-06-24	
9672	Infamous	3.680896	2006-11-16	
12819	Alpha and Omega	12.300789	2010-09-17	

Setting an index

```
movies = pd.read_csv('tmdb_movies.csv', index_col=['id'])  
print(movies.head())
```

	title	popularity	release_date
id			
257	Oliver Twist	20.415572	2005-09-23
14290	Better Luck ...	3.877036	2002-01-12
38365	Grown Ups	38.864027	2010-06-24
9672	Infamous	3.680896	2006-11-16
12819	Alpha and Omega	12.300789	2010-09-17

Merge index datasets

	title	popularity	release_date
id			
257	Oliver Twist	20.415572	2005-09-23
14290	Better Luck ...	3.877036	2002-01-12
38365	Grown Ups	38.864027	2010-06-24
9672	Infamous	3.680896	2006-11-16

	tagline
id	
19995	Enter the Wo...
285	At the end o...
206647	A Plan No On...
49026	The Legend Ends

Merging on index

```
movies_taglines = movies.merge(taglines, on='id', how='left')
print(movies_taglines.head())
```

	title	popularity	release_date	tagline
id				
257	Oliver Twist	20.415572	2005-09-23	NaN
14290	Better Luck ...	3.877036	2002-01-12	Never undere...
38365	Grown Ups	38.864027	2010-06-24	Boys will be...
9672	Infamous	3.680896	2006-11-16	There's more...
12819	Alpha and Omega	12.300789	2010-09-17	A Pawsome 3D...

MultIndex datasets

```
samuel = pd.read_csv('samuel.csv',  
                     index_col=['movie_id',  
                                'cast_id'])  
  
print(samuel.head())
```

		name
movie_id	cast_id	
184	3	Samuel L. Jackson
319	13	Samuel L. Jackson
326	2	Samuel L. Jackson
329	138	Samuel L. Jackson
393	21	Samuel L. Jackson

```
casts = pd.read_csv('casts.csv',  
                     index_col=['movie_id',  
                                'cast_id'])  
  
print(casts.head())
```

		character
movie_id	cast_id	
5	22	Jezebel
	23	Diana
	24	Athena
	25	Elspeth
	26	Eva

MultIndex merge

```
samuel_casts = samuel.merge(casts, on=['movie_id','cast_id'])  
print(samuel_casts.head())  
print(samuel_casts.shape)
```

		name	character
movie_id	cast_id		
184	3	Samuel L. Jackson	Ordell Robbie
319	13	Samuel L. Jackson	Big Don
326	2	Samuel L. Jackson	Neville Flynn
329	138	Samuel L. Jackson	Arnold
393	21	Samuel L. Jackson	Rufus
(67, 2)			

Index merge with left_on and right_on

	title	popularity	release_date
id			
257	Oliver Twist	20.415572	2005-09-23
14290	Better Luck ...	3.877036	2002-01-12
38365	Grown Ups	38.864027	2010-06-24
9672	Infamous	3.680896	2006-11-16

	genre
movie_id	
5	Crime
5	Comedy
11	Science Fiction
11	Action

Index merge with left_on and right_on

```
movies_genres = movies.merge(movie_to_genres, left_on='id', left_index=True,  
                             right_on='movie_id', right_index=True)  
print(movies_genres.head())
```

```
   id  title      popularity  release_date      genre  
5    5  Four Rooms  22.876230  1995-12-09  Crime  
5    5  Four Rooms  22.876230  1995-12-09  Comedy  
11   11  Star Wars 126.393695  1977-05-25  Science Fiction  
11   11  Star Wars 126.393695  1977-05-25  Action  
11   11  Star Wars 126.393695  1977-05-25  Adventure
```

Let's practice!

JOINING DATA WITH PANDAS

Filtering joins

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

Mutating versus filtering joins

Mutating joins:

- Combines data from two tables based on matching observations in both tables

Filtering joins:

- Filter observations from table based on whether or not they match an observation in another table

What is a semi join?

Left Table

A	B	C
A2	B2	C2
A3	B3	C3
A4	B4	C4

Right Table

C	D
C1	D1
C2	D2
C4	D4
C5	D5

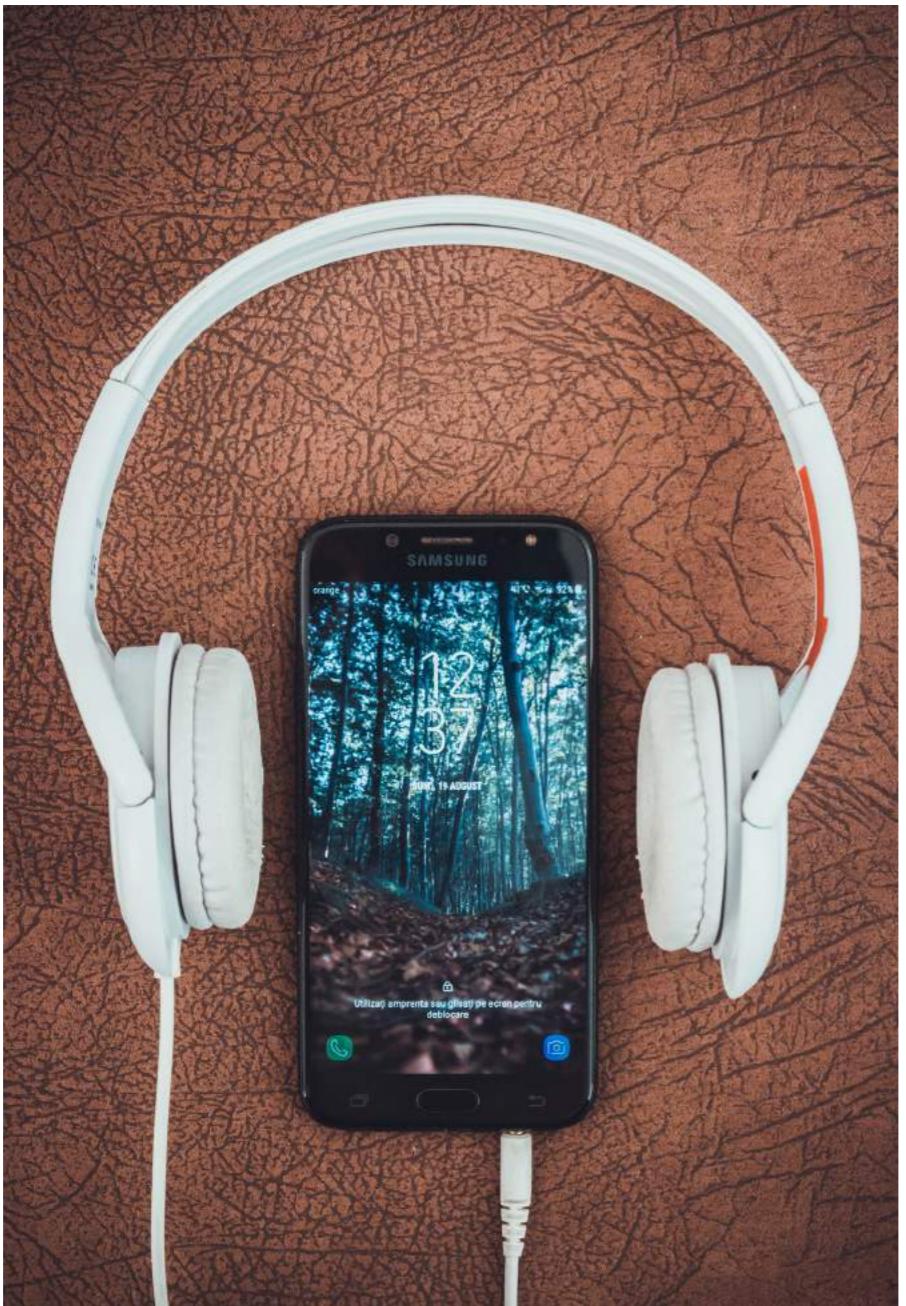
Result Table

A	B	C
A2	B2	C2
A4	B4	C4

Semi joins

- Returns the intersection, similar to an inner join
- Returns only columns from the left table and *not* the right
- No duplicates

Musical dataset



¹ Photo by Vlad Bagacian from Pexels

Example datasets

	gid	name
0	1	Rock
1	2	Jazz
2	3	Metal
3	4	Alternative ...
4	5	Rock And Roll

	tid	name	aid	mtid	gid	composer	u_price
0	1	For Those Ab...	1	1	1	Angus Young, ...	0.99
1	2	Balls to the...	2	2	1	nan	0.99
2	3	Fast As a Shark	3	2	1	F. Baltes, S...	0.99
3	4	Restless and...	3	2	1	F. Baltes, R...	0.99
4	5	Princess of ...	3	2	1	Deaffy & R.A...	0.99

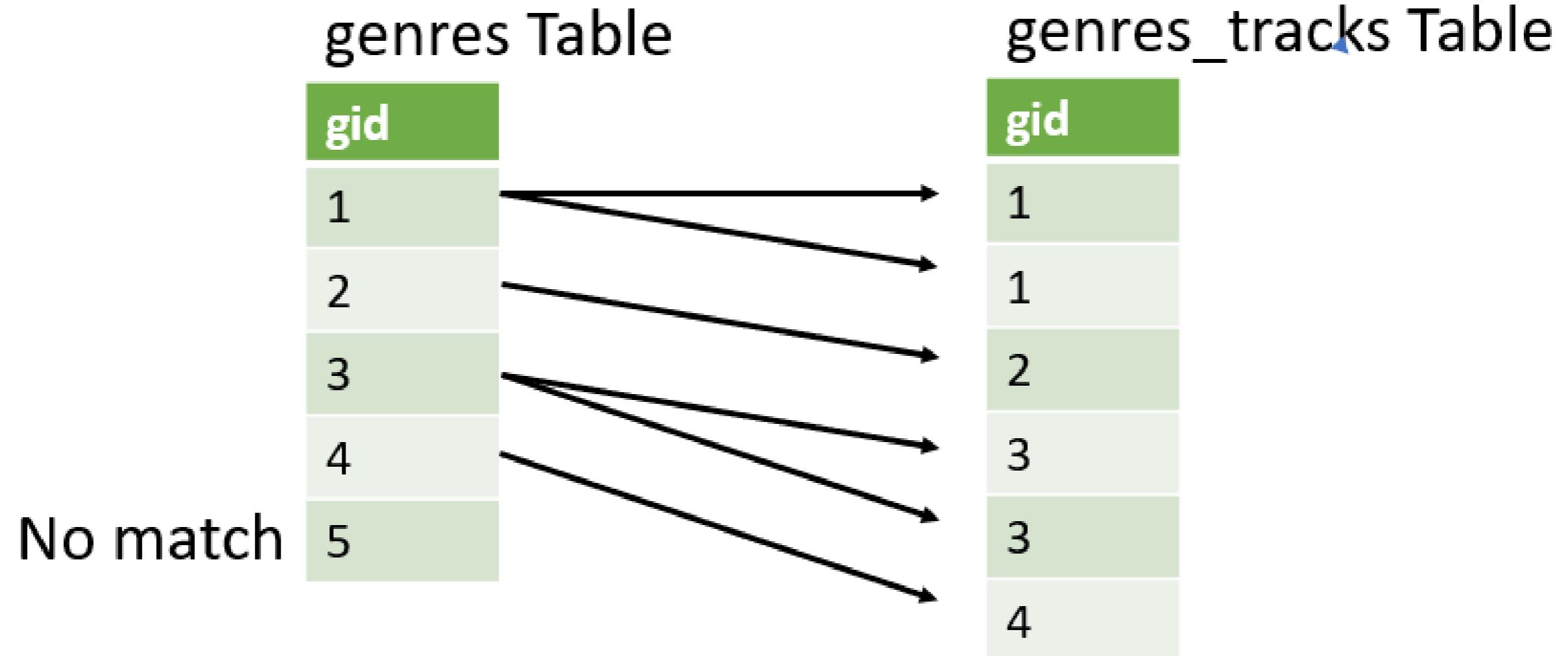
Step 1 - semi join

```
genres_tracks = genres.merge(top_tracks, on='gid')
print(genres_tracks.head())
```

	gid	name_x	tid	name_y	aid	mtid	composer	u_price
0	1	Rock	2260	Don't Stop M...	185	1	Mercury, Fre...	0.99
1	1	Rock	2933	Mysterious Ways	232	1	U2	0.99
2	1	Rock	2618	Speed Of Light	212	1	Billy Duffy/...	0.99
3	1	Rock	2998	When Love Co...	237	1	Bono/Clayton...	0.99
4	1	Rock	685	Who'll Stop ...	54	1	J. C. Fogerty	0.99

Step 2 - semi join

```
genres['gid'].isin(genres_tracks['gid'])
```



Step 2 - semi join

```
genres['gid'].isin(genres_tracks['gid'])
```

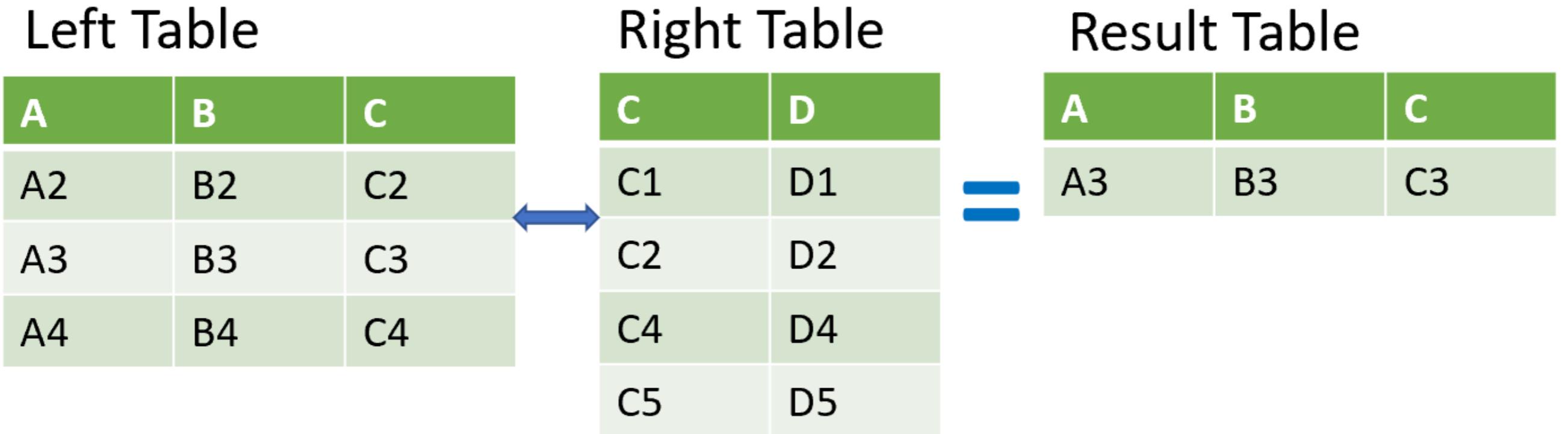
```
0    True
1    True
2    True
3    True
4   False
Name: gid, dtype: bool
```

Step 3 - semi join

```
genres_tracks = genres.merge(top_tracks, on='gid')
top_genres = genres[genres['gid'].isin(genres_tracks['gid'])]
print(top_genres.head())
```

```
   gid    name
0  1      Rock
1  2     Jazz
2  3    Metal
3  4  Alternative & Punk
4  6     Blues
```

What is an anti join?



Anti join:

- Returns the left table, excluding the intersection
- Returns only columns from the left table and *not* the right

Step 1 - anti join

```
genres_tracks = genres.merge(top_tracks, on='gid', how='left', indicator=True)  
print(genres_tracks.head())
```

	gid	name_x	tid	name_y	aid	mtid	composer	u_price	_merge
0	1	Rock	2260.0	Don't Stop M...	185.0	1.0	Mercury, Fre...	0.99	both
1	1	Rock	2933.0	Mysterious Ways	232.0	1.0	U2	0.99	both
2	1	Rock	2618.0	Speed Of Light	212.0	1.0	Billy Duffy/...	0.99	both
3	1	Rock	2998.0	When Love Co...	237.0	1.0	Bono/Clayton...	0.99	both
4	5	Rock And Roll	NaN	NaN	NaN	NaN	NaN	NaN	left_only

Step 2 - anti join

```
gid_list = genres_tracks.loc[genres_tracks['_merge'] == 'left_only', 'gid']
print(gid_list.head())
```

```
23      5
34      9
36     11
37     12
38     13
Name: gid, dtype: int64
```

Step 3 - anti join

```
genres_tracks = genres.merge(top_tracks, on='gid', how='left', indicator=True)
gid_list = genres_tracks.loc[genres_tracks['_merge'] == 'left_only', 'gid']
non_top_genres = genres[genres['gid'].isin(gid_list)]
print(non_top_genres.head())
```

```
   gid      name
0  5    Rock And Roll
1  9        Pop
2 11    Bossa Nova
3 12  Easy Listening
4 13    Heavy Metal
```

Let's practice!

JOINING DATA WITH PANDAS

Concatenate DataFrames together vertically

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

Concatenate two tables vertically

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3



A	B	C
A4	B4	C4
A5	B5	C5
A6	B6	C6

- `pandas .concat()` method can concatenate both vertical and horizontal.
 - `axis=0` , vertical

Basic concatenation

- 3 different tables
- Same column names
- Table variable names:
 - `inv_jan` (*top*)
 - `inv_feb` (*middle*)
 - `inv_mar` (*bottom*)

	iid	cid	invoice_date	total
0	1	2	2009-01-01	1.98
1	2	4	2009-01-02	3.96
2	3	8	2009-01-03	5.94

	iid	cid	invoice_date	total
0	7	38	2009-02-01	1.98
1	8	40	2009-02-01	1.98
2	9	42	2009-02-02	3.96

	iid	cid	invoice_date	total
0	14	17	2009-03-04	1.98
1	15	19	2009-03-04	1.98
2	16	21	2009-03-05	3.96

Basic concatenation

```
pd.concat([inv_jan, inv_feb, inv_mar])
```

	iid	cid	invoice_date	total
0	1	2	2009-01-01	1.98
1	2	4	2009-01-02	3.96
2	3	8	2009-01-03	5.94
0	7	38	2009-02-01	1.98
1	8	40	2009-02-01	1.98
2	9	42	2009-02-02	3.96
0	14	17	2009-03-04	1.98
1	15	19	2009-03-04	1.98
2	16	21	2009-03-05	3.96

Ignoring the index

```
pd.concat([inv_jan, inv_feb, inv_mar],  
          ignore_index=True)
```

	iid	cid	invoice_date	total
0	1	2	2009-01-01	1.98
1	2	4	2009-01-02	3.96
2	3	8	2009-01-03	5.94
3	7	38	2009-02-01	1.98
4	8	40	2009-02-01	1.98
5	9	42	2009-02-02	3.96
6	14	17	2009-03-04	1.98
7	15	19	2009-03-04	1.98
8	16	21	2009-03-05	3.96

Setting labels to original tables

```
pd.concat([inv_jan, inv_feb, inv_mar],  
          ignore_index=False,  
          keys=['jan','feb','mar'])
```

	iid	cid	invoice_date	total
jan	0	1	2009-01-01	1.98
	1	2	2009-01-02	3.96
	2	3	2009-01-03	5.94
feb	0	7	2009-02-01	1.98
	1	8	2009-02-01	1.98
	2	9	2009-02-02	3.96
mar	0	14	2009-03-04	1.98
	1	15	2009-03-04	1.98
	2	16	2009-03-05	3.96

Concatenate tables with different column names

Table: `inv_jan`

```
  iid  cid  invoice_date  total
0  1    2    2009-01-01   1.98
1  2    4    2009-01-02   3.96
2  3    8    2009-01-03   5.94
```

Table: `inv_feb`

```
  iid  cid  invoice_date  total  bill_ctry
0  7    38   2009-02-01   1.98   Germany
1  8    40   2009-02-01   1.98   France
2  9    42   2009-02-02   3.96   France
```

Concatenate tables with different column names

```
pd.concat([inv_jan, inv_feb],  
          sort=True)
```

	bill_ctry	cid	iid	invoice_date	total
0	NaN	2	1	2009-01-01	1.98
1	NaN	4	2	2009-01-02	3.96
2	NaN	8	3	2009-01-03	5.94
0	Germany	38	7	2009-02-01	1.98
1	France	40	8	2009-02-01	1.98
2	France	42	9	2009-02-02	3.96

Concatenate tables with different column names

```
pd.concat([inv_jan, inv_feb],  
          join='inner')
```

iid	cid	invoice_date	total
1	2	2009-01-01	1.98
2	4	2009-01-02	3.96
3	8	2009-01-03	5.94
7	38	2009-02-01	1.98
8	40	2009-02-01	1.98
9	42	2009-02-02	3.96

Using append method

.append()

- Simplified version of the `.concat()` method
- Supports: `ignore_index`, and `sort`
- Does Not Support: `keys` and `join`
 - Always `join = outer`

Append these tables

	iid	cid	invoice_date	total
0	1	2	2009-01-01	1.98
1	2	4	2009-01-02	3.96
2	3	8	2009-01-03	5.94

	iid	cid	invoice_date	total	bill_ctry
0	7	38	2009-02-01	1.98	Germany
1	8	40	2009-02-01	1.98	France
2	9	42	2009-02-02	3.96	France

	iid	cid	invoice_date	total
0	14	17	2009-03-04	1.98
1	15	19	2009-03-04	1.98
2	16	21	2009-03-05	3.96

Append the tables

```
inv_jan.append([inv_feb, inv_mar],  
               ignore_index=True,  
               sort=True)
```

	bill_ctry	cid	iid	invoice_date	total
0	NaN	2	1	2009-01-01	1.98
1	NaN	4	2	2009-01-02	3.96
2	NaN	8	3	2009-01-03	5.94
3	Germany	38	7	2009-02-01	1.98
4	France	40	8	2009-02-01	1.98
5	France	42	9	2009-02-02	3.96
6	NaN	17	14	2009-03-04	1.98
7	NaN	19	15	2009-03-04	1.98
8	NaN	21	16	2009-03-05	3.96

Let's practice!

JOINING DATA WITH PANDAS

Verifying integrity

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

Let's check our data

Possible merging issue:

A	B	C	C	D
A1	B1	C1	C1	D1
A2	B2	C2	C1	D2
A3	B3	C3	C1	D3

C2	D4
----	----

- Unintentional one-to-many relationship
- Unintentional many-to-many relationship

Possible concatenating issue:

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3

A	B	C
A3 (duplicate)	B3 (duplicate)	C3 (duplicate)
A4	B4	C4
A5	B5	C5

- Duplicate records possibly unintentionally introduced

Validating merges

`.merge(validate=None) :`

- Checks if merge is of specified type
- `'one_to_one'`
- `'one_to_many'`
- `'many_to_one'`
- `'many_to_many'`

Merge dataset for example

Table Name: tracks

	tid	name	aid	mtid	gid	u_price
0	2	Balls to the...	2	2	1	0.99
1	3	Fast As a Shark	3	2	1	0.99
2	4	Restless and...	3	2	1	0.99

Table Name: specs

	tid	milliseconds	bytes
0	2	342562	5510424
1	3	230619	3990994
2	2	252051	4331779

Merge validate: one_to_one

```
tracks.merge(specs, on='tid',  
            validate='one_to_one')
```

Traceback (most recent call last):

MergeError: Merge keys are not unique in right dataset; not a one-to-one merge

Merge validate: one_to_many

```
albums.merge(tracks, on='aid',  
            validate='one_to_many')
```

```
   aid  title          artid  tid  name          mtid  gid  u_price  
0  2  Balls to the...    2     2  Balls to the...    2     1  0.99  
1  3  Restless and...    2     3  Fast As a Shark    2     1  0.99  
2  3  Restless and...    2     4  Restless and...    2     1  0.99
```

Verifying concatenations

```
.concat	verify_integrity=False) :
```

- Check whether the new concatenated index contains duplicates
- Default value is False

Dataset for `.concat()` example

Table Name: `inv_feb`

	cid	invoice_date	total
iid			
7	38	2009-02-01	1.98
8	40	2009-02-01	1.98
9	42	2009-02-02	3.96

Table Name: `inv_mar`

	cid	invoice_date	total
iid			
9	17	2009-03-04	1.98
15	19	2009-03-04	1.98
16	21	2009-03-05	3.96

Verifying concatenation: example

```
pd.concat([inv_feb, inv_mar],  
          verify_integrity=True)
```

```
Traceback (most recent call last):  
ValueError: Indexes have overlapping  
values: Int64Index([9], dtype='int64',  
name='iid')
```

```
pd.concat([inv_feb, inv_mar],  
          verify_integrity=False)
```

iid	cid	invoice_date	total
7	38	2009-02-01	1.98
8	40	2009-02-01	1.98
9	42	2009-02-02	3.96
9	17	2009-03-04	1.98
15	19	2009-03-04	1.98
16	21	2009-03-05	3.96

Why verify integrity and what to do

Why:

- Real world data is often *NOT* clean

What to do:

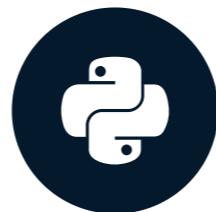
- Fix incorrect data
- Drop duplicate rows

Let's practice!

JOINING DATA WITH PANDAS

Using `merge_ordered()`

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

merge_ordered()

Left Table

A	B	C
A3	B3	C3
A2	B2	C2
A1	B1	C1

Right Table

C	D
C4	D4
C2	D2
C1	D1

Result Table

A	B	C	D
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	
		C4	D4

Method comparison

.merge() method:

- Column(s) to join on
 - `on`, `left_on`, and `right_on`
- Type of join
 - `how (left, right, inner, outer) {@}`
 - **default** inner
- Overlapping column names
 - `suffixes`
- Calling the method
 - `df1.merge(df2)`

merge_ordered() method:

- Column(s) to join on
 - `on`, `left_on`, and `right_on`
- Type of join
 - `how (left, right, inner, outer)`
 - **default** outer
- Overlapping column names
 - `suffixes`
- Calling the function
 - `pd.merge_ordered(df1, df2)`

Financial dataset



¹ Photo by Markus Spiske on Unsplash

Stock data

Table Name: appl

```
date      close
0 2007-02-01  12.087143
1 2007-03-01  13.272857
2 2007-04-01  14.257143
3 2007-05-01  17.312857
4 2007-06-01  17.434286
```

Table Name: mcd

```
date      close
0 2007-01-01  44.349998
1 2007-02-01  43.689999
2 2007-03-01  45.049999
3 2007-04-01  48.279999
4 2007-05-01  50.549999
```

Merging stock data

```
import pandas as pd  
pd.merge_ordered(appl, mcd, on='date', suffixes=('_aapl', '_mcd'))
```

	date	close_aapl	close_mcd
0	2007-01-01	NaN	44.349998
1	2007-02-01	12.087143	43.689999
2	2007-03-01	13.272857	45.049999
3	2007-04-01	14.257143	48.279999
4	2007-05-01	17.312857	50.549999
5	2007-06-01	17.434286	NaN

Forward fill

Before

A	B
A1	B1
A2	
A3	B3
A4	
A5	B5

After

A	B
A1	B1
A2	B1
A3	B3
A4	B3
A5	B5

 Fills missing
with
previous
value

Forward fill example

```
pd.merge_ordered(appl, mcd, on='date',  
                suffixes=('_aapl', '_mcd'),  
                fill_method='ffill')
```

	date	close_aapl	close_mcd
0	2007-01-01	NaN	44.349998
1	2007-02-01	12.087143	43.689999
2	2007-03-01	13.272857	45.049999
3	2007-04-01	14.257143	48.279999
4	2007-05-01	17.312857	50.549999
5	2007-06-01	17.434286	50.549999

```
pd.merge_ordered(appl, mcd, on='date',  
                suffixes=('_aapl', '_mcd'))
```

	date	close_AAPL	close_mcd
0	2007-01-01	NaN	44.349998
1	2007-02-01	12.087143	43.689999
2	2007-03-01	13.272857	45.049999
3	2007-04-01	14.257143	48.279999
4	2007-05-01	17.312857	50.549999
5	2007-06-01	17.434286	NaN

When to use merge_ordered()?

- Ordered data / time series
- Filling in missing values

Let's practice!

JOINING DATA WITH PANDAS

Using `merge_asof()`

JOINING DATA WITH PANDAS



Aaren Stubberfield

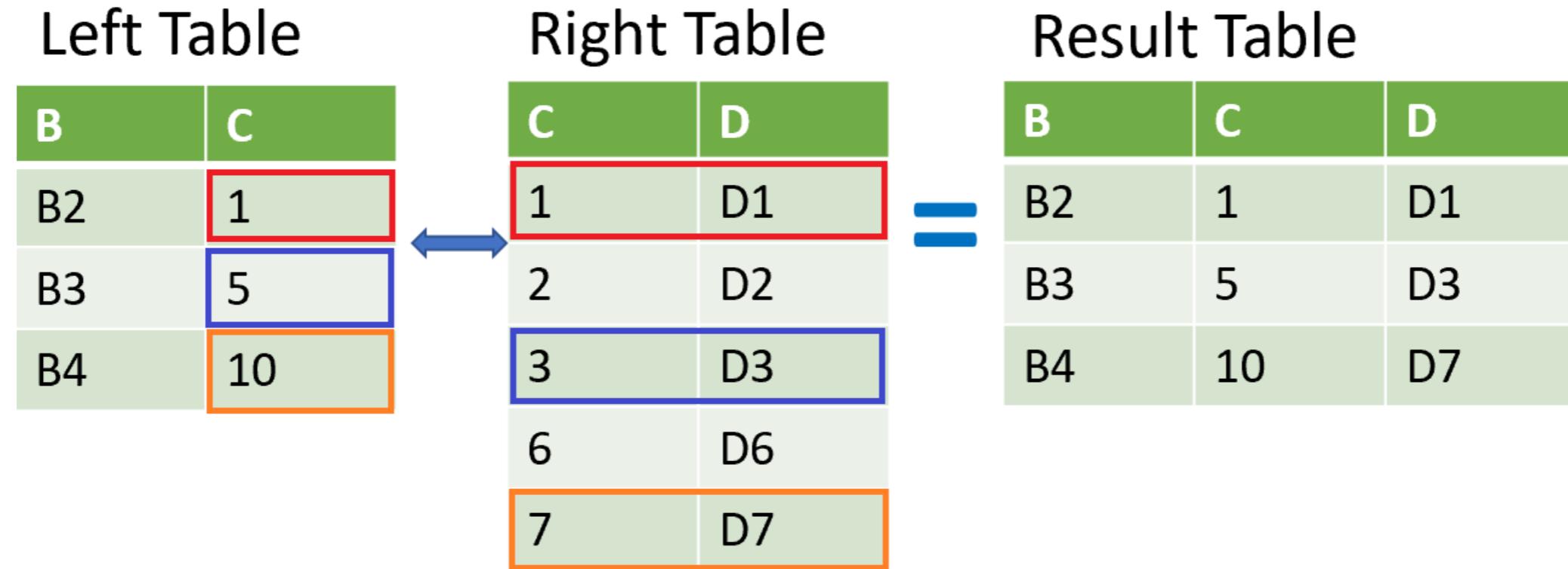
Instructor

Using merge_asof()

Left Table		Right Table		Result Table		
B	C	C	D	B	C	D
B2	1	1	D1	B2	1	D1
B3	5	2	D2	B3	5	D3
B4	10	3	D3	B4	10	D7
		6	D6			
		7	D7			

- Similar to a `merge_ordered()` left join
 - Similar features as `merge_ordered()`
- Match on the nearest key column and not exact matches.
 - Merged "on" columns must be sorted.

Using merge_asof()



- Similar to a `merge_ordered()` left join
 - Similar features as `merge_ordered()`
- Match on the nearest key column and not exact matches.
 - Merged "on" columns must be sorted.

Datasets

Table Name: visa

```
date_time           close
0 2017-11-17 16:00:00  110.32
1 2017-11-17 17:00:00  110.24
2 2017-11-17 18:00:00  110.065
3 2017-11-17 19:00:00  110.04
4 2017-11-17 20:00:00  110.0
5 2017-11-17 21:00:00  109.9966
6 2017-11-17 22:00:00  109.82
```

Table Name: ibm

```
date_time           close
0 2017-11-17 15:35:12  149.3
1 2017-11-17 15:40:34  149.13
2 2017-11-17 15:45:50  148.98
3 2017-11-17 15:50:20  148.99
4 2017-11-17 15:55:10  149.11
5 2017-11-17 16:00:03  149.25
6 2017-11-17 16:05:06  149.5175
7 2017-11-17 16:10:12  149.57
8 2017-11-17 16:15:30  149.59
9 2017-11-17 16:20:32  149.82
10 2017-11-17 16:25:47 149.96
```

merge_asof() example

```
pd.merge_asof(visa, ibm, on='date_time',  
              suffixes=('_visa','_ibm'))
```

	date_time	close_visa	close_ibm
0	2017-11-17 16:00:00	110.32	149.11
1	2017-11-17 17:00:00	110.24	149.83
2	2017-11-17 18:00:00	110.065	149.59
3	2017-11-17 19:00:00	110.04	149.505
4	2017-11-17 20:00:00	110.0	149.42
5	2017-11-17 21:00:00	109.9966	149.26
6	2017-11-17 22:00:00	109.82	148.97

Table Name: ibm

	date_time	close
0	2017-11-17 15:35:12	149.3
1	2017-11-17 15:40:34	149.13
2	2017-11-17 15:45:50	148.98
3	2017-11-17 15:50:20	148.99
4	2017-11-17 15:55:10	149.11
5	2017-11-17 16:00:03	149.25
6	2017-11-17 16:05:06	149.5175
7	2017-11-17 16:10:12	149.57
8	2017-11-17 16:15:30	149.59
9	2017-11-17 16:20:32	149.82
10	2017-11-17 16:25:47	149.96

merge_asof() example with direction

```
pd.merge_asof(visa, ibm, on=['date_time'],  
             suffixes=('_visa','_ibm'),  
             direction='forward')
```

	date_time	close_visa	close_ibm
0	2017-11-17 16:00:00	110.32	149.25
1	2017-11-17 17:00:00	110.24	149.6184
2	2017-11-17 18:00:00	110.065	149.59
3	2017-11-17 19:00:00	110.04	149.505
4	2017-11-17 20:00:00	110.0	149.42
5	2017-11-17 21:00:00	109.9966	149.26
6	2017-11-17 22:00:00	109.82	148.97

Table Name: ibm

	date_time	close
0	2017-11-17 15:35:12	149.3
1	2017-11-17 15:40:34	149.13
2	2017-11-17 15:45:50	148.98
3	2017-11-17 15:50:20	148.99
4	2017-11-17 15:55:10	149.11
5	2017-11-17 16:00:03	149.25
6	2017-11-17 16:05:06	149.5175
7	2017-11-17 16:10:12	149.57
8	2017-11-17 16:15:30	149.59
9	2017-11-17 16:20:32	149.82
10	2017-11-17 16:25:47	149.96

When to use merge_asof()

- Data sampled from a process
- Developing a training set (no data leakage)

Let's practice!

JOINING DATA WITH PANDAS

Selecting data with .query()

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

The `.query()` method

```
.query('SOME SELECTION STATEMENT')
```

- Accepts an input string
 - Input string used to determine what rows are returned
 - Input string similar to statement after **WHERE** clause in **SQL** statement
 - **Prior knowledge of SQL is not necessary**

Querying on a single condition

This table is `stocks`

	date	disney	nike
0	2019-07-01	143.009995	86.029999
1	2019-08-01	137.259995	84.5
2	2019-09-01	130.320007	93.919998
3	2019-10-01	129.919998	89.550003
4	2019-11-01	151.580002	93.489998
5	2019-12-01	144.630005	101.309998
6	2020-01-01	138.309998	96.300003
7	2020-02-01	117.650002	89.379997
8	2020-03-01	96.599998	82.739998
9	2020-04-01	99.580002	84.629997

```
stocks.query('nike >= 90')
```

	date	disney	nike
2	2019-09-01	130.320007	93.919998
4	2019-11-01	151.580002	93.489998
5	2019-12-01	144.630005	101.309998
6	2020-01-01	138.309998	96.300003

Querying on a multiple conditions, "and", "or"

This table is `stocks`

	date	disney	nike
0	2019-07-01	143.009995	86.029999
1	2019-08-01	137.259995	84.5
2	2019-09-01	130.320007	93.919998
3	2019-10-01	129.919998	89.550003
4	2019-11-01	151.580002	93.489998
5	2019-12-01	144.630005	101.309998
6	2020-01-01	138.309998	96.300003
7	2020-02-01	117.650002	89.379997
8	2020-03-01	96.599998	82.739998
9	2020-04-01	99.580002	84.629997

```
stocks.query('nike > 90 and disney < 140')
```

	date	disney	nike
2	2019-09-01	130.320007	93.919998
6	2020-01-01	138.309998	96.300003

```
stocks.query('nike > 96 or disney < 98')
```

	date	disney	nike
5	2019-12-01	144.630005	101.309998
6	2020-01-01	138.309998	96.300003
28	020-03-01	96.599998	82.739998

Updated dataset

This table is `stocks_long`

	date	stock	close
0	2019-07-01	disney	143.009995
1	2019-08-01	disney	137.259995
2	2019-09-01	disney	130.320007
3	2019-10-01	disney	129.919998
4	2019-11-01	disney	151.580002
5	2019-07-01	nike	86.029999
6	2019-08-01	nike	84.5
7	2019-09-01	nike	93.919998
8	2019-10-01	nike	89.550003
9	2019-11-01	nike	93.489998

Using .query() to select text

```
stocks_long.query('stock=="disney" or (stock=="nike" and close < 90)')
```

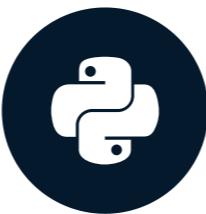
	date	stock	close
0	2019-07-01	disney	143.009995
1	2019-08-01	disney	137.259995
2	2019-09-01	disney	130.320007
3	2019-10-01	disney	129.919998
4	2019-11-01	disney	151.580002
5	2019-07-01	nike	86.029999
6	2019-08-01	nike	84.5
8	2019-10-01	nike	89.550003

Let's practice!

JOINING DATA WITH PANDAS

Reshaping data with .melt()

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

Wide versus long data

Wide Format

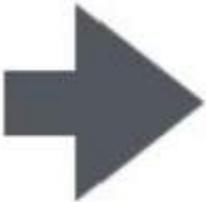
	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

Long Format

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

What does the `.melt()` method do?

- The melt method will allow us to unpivot our dataset



	first	last	height	weight	
0	John	Doe	5.5	130	
1	Mary	Bo	6.0	150	

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

Dataset in wide format

This table is called `social_fin`

financial	company	2019	2018	2017	2016
0 total_revenue	twitter	3459329	3042359	2443299	2529619
1 gross_profit	twitter	2322288	2077362	1582057	1597379
2 net_income	twitter	1465659	1205596	-108063	-456873
3 total_revenue	facebook	70697000	55838000	40653000	27638000
4 gross_profit	facebook	57927000	46483000	35199000	23849000
5 net_income	facebook	18485000	22112000	15934000	10217000

Example of .melt()

```
social_fin_tall = social_fin.melt(id_vars=['financial','company'])  
print(social_fin_tall.head(10))
```

	financial	company	variable	value
0	total_revenue	twitter	2019	3459329
1	gross_profit	twitter	2019	2322288
2	net_income	twitter	2019	1465659
3	total_revenue	facebook	2019	70697000
4	gross_profit	facebook	2019	57927000
5	net_income	facebook	2019	18485000
6	total_revenue	twitter	2018	3042359
7	gross_profit	twitter	2018	2077362
8	net_income	twitter	2018	1205596
9	total_revenue	facebook	2018	55838000

Melting with value_vars

```
social_fin_tall = social_fin.melt(id_vars=['financial','company'],  
                                  value_vars=['2018','2017'])  
print(social_fin_tall.head(9))
```

	financial	company	variable	value
0	total_revenue	twitter	2018	3042359
1	gross_profit	twitter	2018	2077362
2	net_income	twitter	2018	1205596
3	total_revenue	facebook	2018	55838000
4	gross_profit	facebook	2018	46483000
5	net_income	facebook	2018	22112000
6	total_revenue	twitter	2017	2443299
7	gross_profit	twitter	2017	1582057
8	net_income	twitter	2017	-108063

Melting with column names

```
social_fin_tall = social_fin.melt(id_vars=['financial','company'],  
                                   value_vars=['2018','2017'],  
                                   var_name=['year'], value_name='dollars')  
  
print(social_fin_tall.head(8))
```

	financial	company	year	dollars
0	total_revenue	twitter	2018	3042359
1	gross_profit	twitter	2018	2077362
2	net_income	twitter	2018	1205596
3	total_revenue	facebook	2018	55838000
4	gross_profit	facebook	2018	46483000
5	net_income	facebook	2018	22112000
6	total_revenue	twitter	2017	2443299
7	gross_profit	twitter	2017	1582057

Let's practice!

JOINING DATA WITH PANDAS

Course wrap-up

JOINING DATA WITH PANDAS



Aaren Stubberfield

Instructor

You're this high performance race car now



¹ Photo by jae park from Pexels

Data merging basics

- Inner join using `.merge()`
- One-to-one and one-to-many relationships
- Merging multiple tables

Merging tables with different join types

- Inner join using `.merge()`
- One-to-one and one-to-one relationships
- Merging multiple tables
- **Left, right, and outer joins**
- **Merging a table to itself and merging on indexes**

Advanced merging and concatenating

- Inner join using `.merge()`
- One-to-one and one-to-one relationships
- Merging multiple tables
- Left, right, and outer joins
- Merging a table to itself and merging on indexes
- **Filtering joins**
 - **semi and anti joins**
- **Combining data vertically with `.concat()`**
- **Verify data integrity**

Merging ordered and time-series data

- Inner join using `.merge()`
 - One-to-one and one-to-one relationships
 - Merging multiple tables
 - Left, right, and outer joins
 - Merging a table to itself and merging on indexes
 - Filtering joins
 - semi and anti joins
 - Combining data vertically with `.concat()`
 - Verify data integrity
- **Ordered data**
 - `merge_ordered()` and `merge_asof()`
 - **Manipulating data with `.melt()`**

Thank you!

JOINING DATA WITH PANDAS

Python For Data Science

python™ Basics Cheat Sheet

Learn Python Basics online at www.DataCamp.com

> Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

```
>>> x+2 #Sum of two variables
7
>>> x-2 #Subtraction of two variables
3
>>> x*2 #Multiplication of two variables
10
>>> x**2 #Exponentiation of a variable
25
>>> x%2 #Remainder of a variable
1
>>> x/float(2) #Division of a variable
2.5
```

Types and Type Conversion

```
str()
'5', '3.45', 'True' #Variables to strings

int()
5, 3, 1 #Variables to integers

float()
5.0, 1.0 #Variables to floats

bool()
True, True, True #Variables to booleans
```

> Libraries

pandas	NumPy	matplotlib	learn
Data analysis	Scientific computing	2D plotting	Machine learning

Import Libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

> Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

String Indexing

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

```
>>> my_string.upper() #String to uppercase
>>> my_string.lower() #String to lowercase
>>> my_string.count('w') #Count String elements
>>> my_string.replace('e', 'i') #Replace String elements
>>> my_string.strip() #Strip whitespaces
```

> NumPy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

```
Subset
>>> my_array[1] #Select item at index 1
2

Slice
>>> my_array[0:2] #Select items at index 0 and 1
array([1, 2])

Subset 2D Numpy arrays
>>> my_2darray[:,0] #my_2darray[rows, columns]
array([1, 4])
```

Numpy Array Operations

```
>>> my_array > 3
array([False, False, False, True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Functions

```
>>> my_array.shape #Get the dimensions of the array
>>> np.append(other_array) #Append items to an array
>>> np.insert(my_array, 1, 5) #Insert items in an array
>>> np.delete(my_array,[1]) #Delete items in an array
>>> np.mean(my_array) #Mean of the array
>>> np.median(my_array) #Median of the array
>>> my_array.correlcoef() #Correlation coefficient
>>> np.std(my_array) #Standard deviation
```

> Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset

```
>>> my_list[1] #Select item at index 1
>>> my_list[-3] #Select 3rd last item
```

Slice

```
>>> my_list[1:3] #Select items at index 1 and 2
>>> my_list[1:] #Select items after index 0
>>> my_list[:3] #Select items before index 3
>>> my_list[:] #Copy my_list
```

Subset Lists of Lists

```
>>> my_list2[1][0] #my_list[list][itemOfList]
>>> my_list2[1][:2]
```

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

```
>>> my_list.index(a) #Get the index of an item
>>> my_list.count(a) #Count an item
>>> my_list.append('!) #Append an item at a time
>>> my_list.remove('!) #Remove an item
>>> del(my_list[0:1]) #Remove an item
>>> my_list.reverse() #Reverse the list
>>> my_list.extend('!) #Append an item
>>> my_list.pop(-1) #Remove an item
>>> my_list.insert(0,'!) #Insert an item
>>> my_list.sort() #Sort the list
```

> Python IDEs (Integrated Development Environment)

ANACONDA.

Leading open data science platform powered by Python

SPYDER

Free IDE that is included with Anaconda

jupyter

Create and share documents with live code

> Asking For Help

```
>>> help(str)
```

Learn Data Skills Online at
www.DataCamp.com