

Python Basics

Python Cheat Sheet for Beginners

Learn Python online at www.DataCamp.com

> How to use this cheat sheet

Python is the most popular programming language in data science. It is easy to learn and comes with a wide array of powerful libraries for data analysis. This cheat sheet provides beginners and intermediate users a guide to starting using python. Use it to jump-start your journey with python. If you want more detailed Python cheat sheets, check out the following cheat sheets below:



Importing data in python



Data wrangling in pandas

> Accessing help and getting object types

```
1 + 1 # Everything after the hash symbol is ignored by Python
help(max) # Display the documentation for the max function
type('a') # Get the type of an object - this returns str
```

> Importing packages

Python packages are a collection of useful tools developed by the open-source community. They extend the capabilities of the python language. To install a new package (for example, pandas), you can go to your command prompt and type in pip install pandas. Once a package is installed, you can import it as follows.

```
import pandas # Import a package without an alias
import pandas as pd # Import a package with an alias
from pandas import DataFrame # Import an object from a package
```

> The working directory

The working directory is the default file path that python reads or saves files into. An example of the working directory is "C://file/path". The os library is needed to set and get the working directory.

```
import os # Import the operating system package
os.getcwd() # Get the current directory
os.chdir("new/working/directory") # Set the working directory to a new file path
```

> Operators

Arithmetic operators

```
102 + 37 # Add two numbers with +
102 - 37 # Subtract a number with -
4 * 6 # Multiply two numbers with *
22 / 7 # Divide a number by another with /
```

```
22 // 7 # Integer divide a number with //
3 ** 4 # Raise to the power with **
22 % 7 # Returns 1 # Get the remainder after division with %
```

Assignment operators

```
a = 5 # Assign a value to a
x[0] = 1 # Change the value of an item in a list
```

Numeric comparison operators

```
3 == 3 # Test for equality with ==
3 != 3 # Test for inequality with !=
3 > 1 # Test greater than with >
```

```
3 >= 3 # Test greater than or equal to with >=
3 < 4 # Test less than with <
3 <= 4 # Test less than or equal to with <=
```

Logical operators

```
~(2 == 2) # Logical NOT with ~
(1 != 1) & (1 < 1) # Logical AND with &
```

```
(1 >= 1) | (1 < 1) # Logical OR with |
(1 != 1) ^ (1 < 1) # Logical XOR with ^
```

> Getting started with lists

A list is an ordered and changeable sequence of elements. It can hold integers, characters, floats, strings, and even objects.

Creating lists

```
# Create lists with [], elements separated by commas
x = [1, 3, 2]
```

List functions and methods

```
x.sorted(x) # Return a sorted copy of the list e.g., [1,2,3]
x.sort() # Sorts the list in-place (replaces x)
reversed(x) # Reverse the order of elements in x e.g., [2,3,1]
x.reversed() # Reverse the list in-place
x.count(2) # Count the number of element 2 in the list
```

Selecting list elements

Python lists are zero-indexed (the first element has index 0). For ranges, the first element is included but the last is not.

```
# Define the list
x = ['a', 'b', 'c', 'd', 'e']           x[1:3] # Select 1st (inclusive) to 3rd (exclusive)
x[0] # Select the 0th element in the list x[2:] # Select the 2nd to the end
x[-1] # Select the last element in the list x[:3] # Select 0th to 3rd (exclusive)
```

Concatenating lists

```
# Define the x and y lists
x = [1, 3, 6]           x + y # Returns [1, 3, 6, 10, 15, 21]
y = [10, 15, 21]         3 * x # Returns [1, 3, 6, 1, 3, 6, 1, 3, 6]
```

> Getting started with dictionaries

A dictionary stores data values in key-value pairs. That is, unlike lists which are indexed by position, dictionaries are indexed by their keys, the names of which must be unique.

Creating dictionaries

```
# Create a dictionary with {}
{'a': 1, 'b': 4, 'c': 9}
```

Dictionary functions and methods

```
x = {'a': 1, 'b': 2, 'c': 3} # Define the x dictionary
x.keys() # Get the keys of a dictionary, returns dict_keys(['a', 'b', 'c'])
x.values() # Get the values of a dictionary, returns dict_values([1, 2, 3])
```

Selecting dictionary elements

```
x['a'] # 1 # Get a value from a dictionary by specifying the key
```

> NumPy arrays

NumPy is a python package for scientific computing. It provides multidimensional array objects and efficient operations on them. To import NumPy, you can run this Python code import numpy as np

Creating arrays

```
# Convert a python list to a NumPy array
np.array([1, 2, 3]) # Returns array([1, 2, 3])
# Return a sequence from start (inclusive) to end (exclusive)
np.arange(1,5) # Returns array([1, 2, 3, 4])
# Return a stepped sequence from start (inclusive) to end (exclusive)
np.arange(1,5,2) # Returns array([1, 3])
# Repeat values n times
np.repeat([1, 3, 6], 3) # Returns array([1, 1, 1, 3, 3, 3, 6, 6, 6])
# Repeat values n times
np.tile([1, 3, 6], 3) # Returns array([1, 3, 6, 1, 3, 6, 1, 3, 6])
```

> Math functions and methods

All functions take an array as the input.

```
np.log(x) # Calculate logarithm
np.exp(x) # Calculate exponential
np.max(x) # Get maximum value
np.min(x) # Get minimum value
np.sum(x) # Calculate sum
np.mean(x) # Calculate mean
np.quantile(x, q) # Calculate q-th quantile
np.round(x, n) # Round to n decimal places
np.var(x) # Calculate variance
np.std(x) # Calculate standard deviation
```

> Getting started with characters and strings

```
# Create a string with double or single quotes
"DataCamp"
```

```
# Embed a quote in string with the escape character \
"He said, \"DataCamp\""
```

```
# Create multi-line strings with triple quotes
"""
A Frame of Data
Tidy, Mine, Analyze It
Now You Have Meaning
Citation: https://mdsr-book.github.io/haikus.html
"""
```

```
str[0] # Get the character at a specific position
str[0:2] # Get a substring from starting to ending index (exclusive)
```

Combining and splitting strings

```
"Data" + "Framed" # Concatenate strings with +, this returns 'DataFramed'
3 * "data" # Repeat strings with *, this returns 'data data data'
"beekeepers".split("e") # Split a string on a delimiter, returns ['b', ' ', 'k', ' ', 'p', 'rs']
```

Mutate strings

```
str = "Jack and Jill" # Define str
str.upper() # Convert a string to uppercase, returns 'JACK AND JILL'
str.lower() # Convert a string to lowercase, returns 'jack and jill'
str.title() # Convert a string to title case, returns 'Jack And Jill'
str.replace("J", "P") # Replaces matches of a substring with another, returns 'Pack and Pill'
```

> Getting started with DataFrames

Pandas is a fast and powerful package for data analysis and manipulation in python. To import the package, you can use import pandas as pd. A pandas DataFrame is a structure that contains two-dimensional data stored as rows and columns. A pandas series is a structure that contains one-dimensional data.

Creating DataFrames

```
# Create a dataframe from a dictionary
pd.DataFrame({
  'a': [1, 2, 3],
  'b': np.array([4, 4, 6]),
  'c': ['x', 'x', 'y']
})
```

```
# Create a dataframe from a list of dictionaries
pd.DataFrame([
  {'a': 1, 'b': 4, 'c': 'x'},
  {'a': 1, 'b': 4, 'c': 'x'},
  {'a': 3, 'b': 6, 'c': 'y'}
])
```

Selecting DataFrame Elements

Select a row, column or element from a dataframe. Remember: all positions are counted from zero, not one.

```
# Select the 3rd row
df.iloc[3]
# Select one column by name
df['col1']
# Select multiple columns by names
df[['col1', 'col2']]
# Select 2nd column
df.iloc[:, 2]
# Select the element in the 3rd row, 2nd column
df.iloc[3, 2]
```

Manipulating DataFrames

```
# Concatenate DataFrames vertically
pd.concat([df, df])
# Concatenate DataFrames horizontally
pd.concat([df, df], axis="columns")
# Get rows matching a condition
df.query("logical_condition")
# Drop columns by name
df.drop(columns=['col_name'])
# Rename columns
df.rename(columns={"oldname": "newname"})
# Add a new column
df.assign(temp_f=9 / 5 * df['temp_c'] + 32)
# Calculate the mean of each column
df.mean()
# Get summary statistics by column
df.agg(aggregation_function)
# Get unique rows
df.drop_duplicates()
# Sort by values in a column
df.sort_values(by='col_name')
# Get rows with largest values in a column
df.nlargest(n, 'col_name')
```

Python For Data Science

NumPy Cheat Sheet

Learn NumPy online at www.DataCamp.com

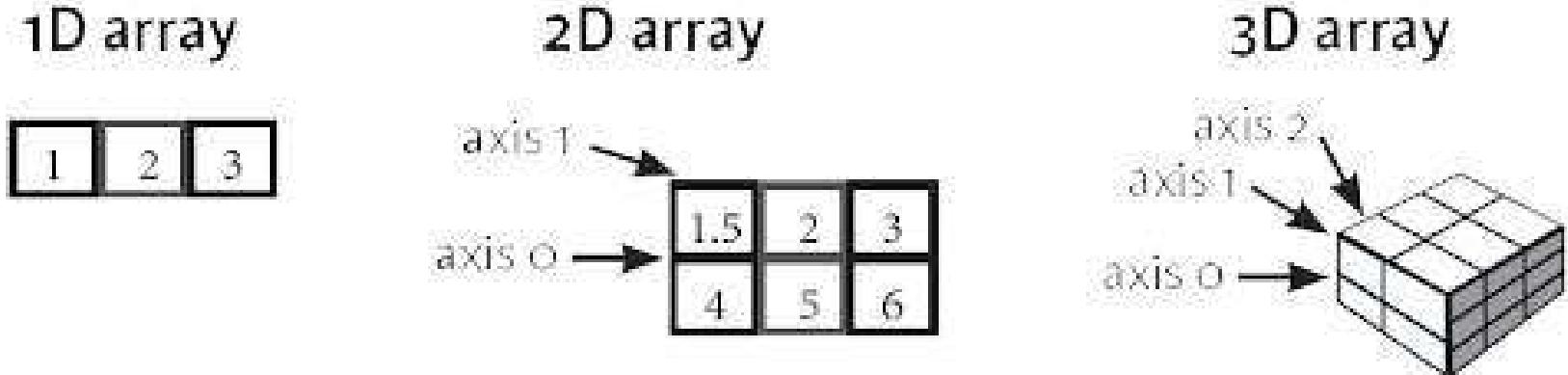
Numpy

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([(1.5,2,3), (4,5,6)],[(3,2,1), (4,5,6)]), dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4)) #Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16) #Create an array of ones
>>> d = np.arange(10,25,5) #Create an array of evenly spaced values (step value)
>>> np.linspace(0,2,9) #Create an array of evenly spaced values (number of samples)
>>> e = np.full((2,2),7) #Create a constant array
>>> f = np.eye(2) #Create a 2x2 identity matrix
>>> np.random.random((2,2)) #Create an array with random values
>>> np.empty((3,2)) #Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Inspecting Your Array

```
>>> a.shape #Array dimensions
>>> len(a) #Length of array
>>> b.ndim #Number of array dimensions
>>> e.size #Number of array elements
>>> b.dtype #Data type of array elements
>>> b.dtype.name #Name of data type
>>> b.astype(int) #Convert an array to a different type
```

Data Types

```
>>> np.int64 #Signed 64-bit integer types
>>> np.float32 #Standard double-precision floating point
>>> np.complex #Complex numbers represented by 128 floats
>>> np.bool #Boolean type storing TRUE and FALSE values
>>> np.object #Python object type
>>> np.string_ #Fixed-length string type
>>> np_unicode_ #Fixed-length unicode type
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b #Subtraction
array([-0.5, 0. , 0. ],
      [-3. , -3. , -3. ])
>>> np.subtract(a,b) #Subtraction
>>> b + a #Addition
array([ 2.5, 4. , 6. ],
      [ 5. , 7. , 9. ])
>>> np.add(b,a) Addition
>>> a / b #Division
array([[ 0.66666667, 1. , 1. ],
      [ 0.25 , 0.4 , 0.5 ]])
>>> np.divide(a,b) #Division
>>> a * b #Multiplication
array([[ 1.5, 4. , 9. ],
      [ 4. , 10. , 18. ]])
>>> np.multiply(a,b) #Multiplication
>>> np.exp(b) #Exponentiation
>>> np.sqrt(b) #Square root
>>> np.sin(a) #Print sines of an array
>>> np.cos(b) #Element-wise cosine
>>> np.log(a) #Element-wise natural logarithm
>>> e.dot(f) #Dot product
array([[ 7., 7.],
      [ 7., 7.]])
```

Comparison

```
>>> a == b #Element-wise comparison
array([[False, True, True],
      [False, False, False]], dtype=bool)
>>> a < b #Element-wise comparison
array([True, False, False], dtype=bool)
>>> np.array_equal(a, b) #Array-wise comparison
```

Aggregate Functions

```
>>> a.sum() #Array-wise sum
>>> a.min() #Array-wise minimum value
>>> b.max(axis=0) #Maximum value of an array row
>>> b.cumsum(axis=1) #Cumulative sum of the elements
>>> a.mean() #Mean
>>> np.median(b) #Median
>>> np.correlcoef(a) #Correlation coefficient
>>> np.std(b) #Standard deviation
```

Copying Arrays

```
>>> h = a.view() #Create a view of the array with the same data
>>> np.copy(a) #Create a copy of the array
>>> h = a.copy() #Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort() #Sort an array
>>> c.sort(axis=0) #Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Subsetting

```
>>> a[2] #Select the element at the 2nd index
3
>>> b[1,2] #Select the element at row 1 column 2 (equivalent to b[1][2])
6.0
```

1	2	3
1.5	2	3
4	5	6

Slicing

```
>>> a[0:2] #Select items at index 0 and 1
array([1, 2])
>>> b[0:2,1] #Select items at rows 0 and 1 in column 1
array([ 2., 2.])
>>> b[:,1] #Select all items at row 0 (equivalent to b[0:, 1])
array([[1.5, 2., 3.]])
>>> c[1,...] #Same as [1,:,:]
array([[ 3., 2., 1.],
       [ 4., 5., 6.]])
>>> a[ : :-1] #Reversed array a array([3, 2, 1])
```

1	2	3
1.5	2	3
4	5	6

Boolean Indexing

```
>>> a[a<2] #Select elements from a less than 2
array([1])
```

1	2	3
---	---	---

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]] #Select elements (1,0),(0,1),(1,2) and (0,0)
array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:, [0,1,2,0]] #Select a subset of the matrix's rows and columns
array([[ 4. , 5. , 6. , 4. ],
       [ 1.5, 2. , 3. , 1.5],
       [ 4. , 5. , 6. , 4. ],
       [ 1.5, 2. , 3. , 1.5]])
```

1	2	3
---	---	---

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b) #Permute array dimensions
>>> i.T #Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel() #Flatten the array
>>> g.reshape(3,-2) #Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6)) #Return a new array with shape (2,6)
>>> np.append(h,g) #Append items to an array
>>> np.insert(a, 1, 5) #Insert items in an array
>>> np.delete(a,[1]) #Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0) #Concatenate arrays
array([[ 1., 2., 3., 10.],
       [ 2., 15.],
       [ 3., 20.]])
>>> np.vstack((a,b)) #Stack arrays vertically (row-wise)
array([[ 1., 2., 3.],
       [ 1.5, 2., 3.],
       [ 4., 5., 6.]]))
>>> np.r_[e,f] #Stack arrays vertically (row-wise)
>>> np.hstack((e,f)) #Stack arrays horizontally (column-wise)
array([[ 7., 7., 1., 0.],
       [ 7., 7., 0., 1.]])
>>> np.column_stack((a,d)) #Create stacked column-wise arrays
array([[ 1, 10],
       [ 2, 15],
       [ 3, 20]])
>>> np.c_[a,d] #Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3) #Split the array horizontally at the 3rd index
[array([1]),array([2]),array([3])]
>>> np.vsplit(c,2) #Split the array vertically at the 2nd index
[array([[ 1.5, 2. , 1. ]]),
 [ 4. , 5. , 6. ]])
array([[[ 3., 2., 3.]],
 [ 4., 5., 6. ]])
```



Python For Data Science

Pandas Basics Cheat Sheet

Learn Pandas Basics online at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

Series

A **one-dimensional** labeled array capable of holding any data type

a	3
b	-5
c	7
d	4

Index →

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

Dataframe

A **two-dimensional** labeled data structure with columns of potentially different types

	Country	Capital	Population
Index →	Belgium	Brussels	11190846
0	India	New Delhi	1303171035
1	Brazil	Brasilia	207847528

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
   'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
   'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
   columns=['Country', 'Capital', 'Population'])
```

Dropping

```
>>> s.drop(['a', 'c']) #Drop values from rows (axis=0)
>>> df.drop('Country', axis=1) #Drop values from columns(axis=1)
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Sort & Rank

```
>>> df.sort_index() #Sort by labels along an axis
>>> df.sort_values(by='Country') #Sort by the values along an axis
>>> df.rank() #Assign ranks to entries
```

> I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> df.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')

Read multiple sheets from the same file
>>> xlsx = pd.ExcelFile('file.xlsx')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)

read_sql() is a convenience wrapper around read_sql_table() and read_sql_query()
>>> df.to_sql('myDF', engine)
```

> Selection

Also see NumPy Arrays

Getting

```
>>> s['b'] #Get one element
-5
>>> df[1:] #Get subset of a DataFrame
   Country Capital Population
1 India New Delhi 1303171035
2 Brazil Brasilia 207847528
```

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]] #Select single value by row & column
'Belgium'
>>> df.iat[[0],[0]]
'Belgium'
```

By Label

```
>>> df.loc[[0], ['Country']] #Select single value by row & column labels
'Belgium'
>>> df.at[[0], ['Country']]
'Belgium'
```

By Label/Position

```
>>> df.ix[2] #Select single row of subset of rows
Country Brazil
Capital Brasilia
Population 207847528
>>> df.ix[:, 'Capital'] #Select a single column of subset of columns
0 Brussels
1 New Delhi
2 Brasilia
>>> df.ix[1, 'Capital'] #Select rows and columns
'New Delhi'
```

Boolean Indexing

```
>>> s[~(s > 1)] #Series s where value is not >1
>>> s[(s < -1) | (s > 2)] #s where value is <-1 or >2
>>> df[df['Population']>1200000000] #Use filter to adjust DataFrame
```

Setting

```
>>> s['a'] = 6 #Set index a of Series s to 6
```

> Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape #(rows,columns)
>>> df.index #Describe index
>>> df.columns #Describe DataFrame columns
>>> df.info() #Info on DataFrame
>>> df.count() #Number of non-NA values
```

Summary

```
>>> df.sum() #Sum of values
>>> df.cumsum() #Cumulative sum of values
>>> df.min()/df.max() #Minimum/maximum values
>>> df.idxmin()/df.idxmax() #Minimum/Maximum index value
>>> df.describe() #Summary statistics
>>> df.mean() #Mean of values
>>> df.median() #Median of values
```

> Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f) #Apply function
>>> df.applymap(f) #Apply function element-wise
```

> Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a 10.0
b NaN
c 5.0
d 7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
a 10.0
b -5.0
c 5.0
d 7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

Learn Data Skills Online at
www.DataCamp.com

Working with text data in Python

Learn Python online at www.DataCamp.com

> Example data used throughout this cheat sheet

Throughout this cheat sheet, we'll be using two pandas series named suits and rock_paper_scissors.

```
import pandas as pd

suits = pd.Series(["clubs", "Diamonds", "hearts", "Spades"])
rock_paper_scissors = pd.Series(["rock", "paper", "scissors"])
```

> String lengths and substrings

```
# Get the number of characters with .str.len()
suits.str.len() # Returns 5 8 6 6

# Get substrings by position with .str[]
suits.str[2:5] # Returns "ubs" "amo" "art" "ade"

# Get substrings by negative position with .str[]
suits.str[:-3] # "cl" "Diamo" "hea" "Spa"

# Remove whitespace from the start/end with .str.strip()
rock_paper_scissors.str.strip() # "rock" "paper" "scissors"

# Pad strings to a given length with .str.pad()
suits.str.pad(8, fillchar="_") # "__clubs" "Diamonds" "__hearts" "__Spades"
```

> Changing case

```
# Convert to lowercase with .str.lower()
suits.str.lower() # "clubs" "diamonds" "hearts" "spades"

# Convert to uppercase with .str.upper()
suits.str.upper() # "CLUBS" "DIAMONDS" "HEARTS" "SPADES"

# Convert to title case with .str.title()
pd.Series("hello, world!").str.title() # "Hello, World!"

# Convert to sentence case with .str.capitalize()
pd.Series("hello, world!").str.capitalize() # "Hello, world!"
```

> Formatting settings

```
# Generate an example DataFrame named df
df = pd.DataFrame({"x": [0.123, 4.567, 8.901]})
#   x
# 0 0.123
# 1 4.567
# 2 8.901
```

```
# Visualize and format table output
df.style.format(precision = 1)
```

-	x
0	0.1
1	4.5
2	8.9

The output of style.format is an HTML table

> Splitting strings

```
# Split strings into list of characters with .str.split(pat="")
suits.str.split(pat="")
```

```
# [, "c" "l" "u" "b" "s", ]
# [, "D" "i" "a" "m" "o" "n" "d" "s", ]
# [, "h" "e" "a" "r" "t" "s", ]
# [, "S" "p" "a" "d" "e" "s", ]
```

```
# Split strings by a separator with .str.split()
suits.str.split(pat = "a")
```

```
# ["clubs"]
# ["Di", "monds"]
# ["he", "rts"]
# ["Sp", "des"]
```

```
# Split strings and return DataFrame with .str.split(expand=True)
suits.str.split(pat = "a", expand=True)
```

```
#      0      1
# 0  clubs  None
# 1  Di     monds
# 2  he     rts
# 3  Sp     des
```

> Joining or concatenating strings

```
# Combine two strings with +
suits + "5" # "clubs5" "Diamonds5" "hearts5" "spades5"
```

```
# Collapse character vector to string with .str.cat()
suits.str.cat(sep=", ") # "clubs, Diamonds, hearts, Spades"
```

```
# Duplicate and concatenate strings with *
suits * 2 # "clubsclubs" "DiamondsDiamonds" "heartshearts" "spadesSpades"
```

> Detecting Matches

```
# Detect if a regex pattern is present in strings with .str.contains()
suits.str.contains("[ae]") # False True True True
```

```
# Count the number of matches with .str.count()
suits.str.count("[ae]") # 0 1 2 2
```

```
# Locate the position of substrings with str.find()
suits.str.find("e") # -1 -1 1 4
```

> Extracting matches

```
# Extract matches from strings with str.findall()
suits.str.findall(".[ae]") # [] ["ia"] ["he"["pa", "de"]]
```

```
# Extract capture groups with .str.extractall()
suits.str.extractall("[ae](.)")
```

#	0 1
# 1 0	a m
# 2 0	e a
# 3 0	a d
# 1	e s

```
# Get subset of strings that match with x[x.str.contains()]
suits[suits.str.contains("d")] # "Diamonds" "Spades"
```

> Replacing matches

```
# Replace a regex match with another string with .str.replace()
suits.str.replace("a", "4") # "clubs" "Di4monds" "he4rts" "Sp4des"
```

```
# Remove a suffix with .str.removesuffix()
suits.str.removesuffix # "club" "Diamond" "heart" "Spade"
```

```
# Replace a substring with .str.slice_replace()
rhymes = pd.Series(["vein", "gain", "deign"])
rhymes.str.slice_replace(0, 1, "r") # "rein" "rain" "reign"
```

Learn Python Online at
www.DataCamp.com

Python For Data Science

Matplotlib Cheat Sheet

Learn Matplotlib online at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

> Prepare The Data

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 + np.random.random((10, 10))
>>> data2 = 3 + np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

> Create Plot

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

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

Axes

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) #row-col-num
>>> ax2 = fig.add_subplot(222)
>>> fig3, axes = plt.subplots(nrows=2, ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

> Save Plot

```
>>> plt.savefig('foo.jpg') #Save figures
>>> plt.savefig('foo.png', transparent=True) #Save transparent figures
```

> Show Plot

```
>>> plt.show()
```

> Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y) #Draw points with lines or markers connecting them
>>> ax.scatter(x,y) #Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5]) #Plot vertical rectangles (constant width)
>>> axes[0,0].barh([0.5,1,2.5],[0,1,2]) #Plot horizontal rectangles (constant height)
>>> axes[1,1].axhline(0.65) #Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65) #Draw a vertical line across axes
>>> ax.fill(x,y,color='blue') #Draw filled polygons
>>> ax.fill_between(x,y,color='yellow') #Fill between y-values and 0
```

2D Data

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img, #Color mapped or RGB arrays
                  cmap='gist_earth',
                  interpolation='nearest',
                  vmin=-2,
                  vmax=2)
>>> axes2[0].pcolor(data2) #Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data) #Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U) #Plot contours
>>> axes2[1].contourf(data1) #Plot filled contours
>>> axes2[2]= ax.clabel(CS) #Label a contour plot
```

Vector Fields

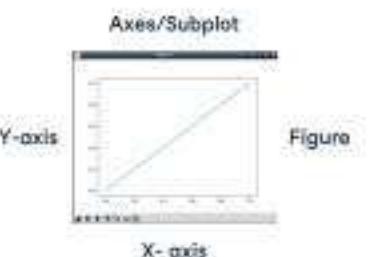
```
>>> axes[0,1].arrow(0,0,0.5,0.5) #Add an arrow to the axes
>>> axes[1,1].quiver(y,z) #Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V) #Plot a 2D field of arrows
```

Data Distributions

```
>>> ax1.hist(y) #Plot a histogram
>>> ax3.boxplot(y) #Make a box and whisker plot
>>> ax3.violinplot(z) #Make a violin plot
```

> Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

- 1 Prepare Data
- 2 Create Plot
- 3 Plot
- 4 Customized Plot
- 5 Save Plot
- 6 Show Plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4] #Step 1
>>> y = [10,28,25,38]
>>> fig = plt.figure() #Step 2
>>> ax = fig.add_subplot(111) #Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3) #Step 3, 4
>>> ax.scatter([2,4,6],
              [5,15,25],
              color='darkgreen',
              marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png') #Step 5
>>> plt.show() #Step 6
```

> Close and Clear

```
>>> plt.clf() #Clear an axis
>>> plt.cla() #Clear the entire figure
>>> plt.close() #Close a window
```

> Plotting Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x*x2, x, x*x3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                  cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

LineStyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x*x2,y*x2,'-')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
           -2.1,
           'Example Graph',
           style='italic')
>>> ax.annotate("Sine",
               xy=(0, 0),
               xycoords='data',
               xytext=(10.5, 0),
               textcoords='data',
               arrowprops=dict(facecolor="black",
                               connectionstyle="arc3"),
               color='red')
```

MathText

```
>>> plt.title(r'$\sin(x)=15$', fontsize=20)
```

Limits, Legends and Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1) #Add padding to a plot
>>> ax.axis('equal') #Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5]) #Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5) #Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes', #Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best') #No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5), #Manually set x-ticks
                  ticklabels=[3,10B,-12,"Foo"])
>>> ax.tick_params(axis='y', #Make y-ticks longer and go in and out
                  direction='inout',
                  length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5, #Adjust the spacing between subplots
                           hspace=0.5,
                           left=0.125,
                           right=0.9,
                           top=0.9,
                           bottom=0.1)
>>> fig.tight_layout() #Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False) #Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position([0,'outward',10]) #Move the bottom axis line outward
```

Data Visualization with Plotly Express in Python

Learn Plotly online at www.DataCamp.com

> What is plotly?

Plotly Express is a high-level data visualization package that allows you to create interactive plots with very little code. It is built on top of Plotly Graph Objects, which provides a lower-level interface for developing custom visualizations.

> Interactive controls in Plotly



Plotly plots have interactive controls shown in the top-right of the plot. The controls allow you to do the following:

- Download plot as a png:** Save your interactive plot as a static PNG.
- Zoom:** Zoom in on a region of interest in the plot.
- Pan:** Move around in the plot.
- Box Select:** Select a rectangular region of the plot to be highlighted.
- Lasso Select:** Draw a region of the plot to be highlighted.
- Autoscale:** Zoom to a "best" scale.
- Reset axes:** Return the plot to its original state.
- Toggle Spike Lines:** Show or hide lines to the axes whenever you hover over data.
- Show closest data on hover:** Show details for the nearest data point to the cursor.
- Compare data on hover:** Show the nearest data point to the x-coordinate of the cursor.

> Plotly Express code pattern

The code pattern for creating plots is to call the plotting function, passing a data frame as the first argument. The x argument is a string naming the column to be used on the x-axis. The y argument can either be a string or a list of strings naming column(s) to be used on the y-axis.

```
px.plotting_fn(dataframe, # Dataframe being visualized
               x=[“column-for-x-axis”], # Accepts a string or a list of strings
               y=[“columns-for-y-axis”], # Accepts a string or a list of strings
               title=“Overall plot title”, # Accepts a string
               xaxis_title=“X-axis title”, # Accepts a string
               yaxis_title=“Y-axis title”, # Accepts a string
               width=width_in_pixels, # Accepts an integer
               height=height_in_pixels) # Accepts an integer
```

> Common plot types

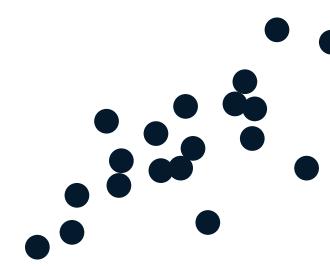
Import plotly

```
# import plotly express as px
import plotly.express as px
```

Scatter plots

```
# Create a scatterplot on a DataFrame named clinical_data
px.scatter(clinical_data, x=“experiment_1”, y=“experiment_2”)
```

Set the size argument to the name of a numeric column to control the size of the points and create a bubble plot.



Line plots

```
# Create a lineplot on a DataFrame named stock_data
px.line(stock_data, x=“date”, y=[“FB”, “AMZN”])
```



Bar plots

```
# Create a barplot on a DataFrame named commodity_data
px.bar(commodity_data, x=“nation”, y=[“gold”, “silver”, “bronze”],
       color_discrete_map={“gold”: “yellow”,
                           “silver”: “grey”,
                           “bronze”: “brown”})
```



Swap the x and y arguments to draw horizontal bars.

Histograms

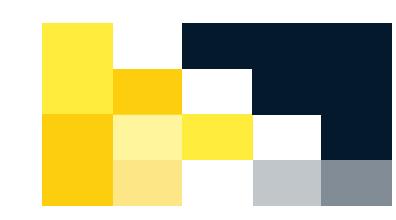
```
# Create a histogram on a DataFrame named bill_data
px.histogram(bill_data, x=“total_bill”)
```



Set the nbins argument to control the number of bins shown in the histogram.

Heatmaps

```
# Create a heatmap on a DataFrame named iris_data
px.imshow(iris_data.corr(numeric_only=True),
          zmin=-1, zmax=1, color_continuous_scale='rdbu')
```



Set the text_auto argument to True to display text values for each cell.

> Customizing plots in plotly

The code pattern for customizing a plot is to save the figure object returned from the plotting function, call its .update_traces() method, then call its .show() method to display it.

```
# Create a plot with plotly (can be of any type)
fig = px.some_plotting_function()
# Customize and show it with .update_traces() and .show()
fig.update_traces()
fig.show()
```

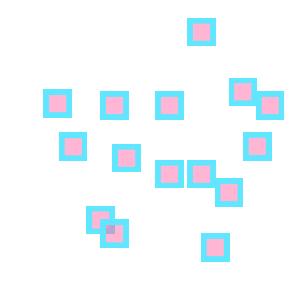
Customizing markers in Plotly

When working with visualizations like scatter plots, lineplots, and more, you can customize markers according to certain properties. These include:

- size: set the marker size
- color: set the marker color
- opacity: set the marker transparency
- line: set the width and color of a border
- symbol: set the shape of the marker

```
# In this example, we’re updating a scatter plot named fig_sct
```

```
fig_sct.update_traces(marker={"size": 24,
                               "color": “magenta”,
                               “opacity”: 0.5,
                               “line”: {"width": 2, “color”: “cyan”},
                               “symbol”: “square”})
fig_sct.show()
```



Customizing lines in Plotly

When working with visualizations that contain lines, you can customize them according to certain properties. These include:

- color: set the line color
- dash: set the dash style (“solid”, “dot”, “dash”, “longdash”, “dashdot”, “longdashdot”)
- width: set the line width
- shape: set how values are connected (“linear”, “spline”, “hv”, “vh”, “hvvh”, “vhv”)

```
# In this example, we’re updating a scatter plot named fig_ln
```

```
fig_ln.update_traces(patch={"line": {"dash": “dot”,
                                      “shape”: “spline”,
                                      “width”: 6}})
fig_ln.show()
```



Customizing bars in Plotly

When working with barplots and histograms, you can update the bars themselves according to the following properties:

- size: set the marker size
- color: set the marker color
- opacity: set the marker transparency
- line: set the width and color of a border
- symbol: set the shape of the marker

```
# In this example, we’re updating a scatter plot named fig_bar
```

```
fig_bar.update_traces(marker={"color": “magenta”,
                               “opacity”: 0.5,
                               “line”: {"width": 2, “color”: “cyan”}})
fig_bar.show()
```



```
# In this example, we’re updating a histogram named fig_hst
```

```
fig_hst.update_traces(marker={"color": “magenta”,
                               “opacity”: 0.5,
                               “line”: {"width": 2, “color”: “cyan”}})
fig_hst.show()
```



Learn Data Skills Online at
www.DataCamp.com

Working with Dates and Times in Python

Learn Python Basics online at www.DataCamp.com

> Key definitions

When working with dates and times, you will encounter technical terms and jargon such as the following:

- **Date:** Handles dates without time.
- **POSIXct:** Handles date & time in calendar time.
- **POSIXlt:** Handles date & time in local time.
- **Hms:** Parses periods with hour, minute, and second
- **Timestamp:** Represents a single pandas date & time
- **Interval:** Defines an open or closed range between dates and times
- **Time delta:** Computes time difference between different datetimes

> The ISO8601 datetime format

The **ISO 8601 datetime format** specifies datetimes from the largest to the smallest unit of time (**YYYY-MM-DD HH:MM:SS TZ**). Some of the advantages of ISO 8601 are:

- It avoids ambiguities between MM/DD/YYYY and DD/MM/YYYY formats.
- The 4-digit year representation mitigates overflow problems after the year 2099.
- Using numeric month values (08 not AUG) makes it language independent, so dates make sense throughout the world.
- Python is optimized for this format since it makes comparison and sorting easier.

> Packages used in this cheat sheet

Load the packages and dataset used in this cheatsheet.

```
import datetime as dt
import time as tm
import pytz
import pandas as pd
```

In this cheat sheet, we will be using 3 pandas series — `iso`, `us`, `non_us`, and 1 pandas DataFrame `parts`

iso
1969-07-20 20:17:40
1969-11-19 06:54:35
1971-02-05 09:18:11

us
07/20/1969 20:17:40
11/19/1969 06:54:35
02/05/1971 09:18:11

non_us
20/07/1969 20:17:40
19/11/1969 06:54:35
05/02/1971 09:18:11

parts		
year	month	day
1969	7	20
1969	11	19
1971	2	5

> Getting the current date and time

```
# Get the current date
dt.date.today()
```

```
# Get the current date and time
dt.datetime.now()
```

> Reading date, datetime, and time columns in a CSV file

```
# Specify datetime column
pd.read_csv("filename.csv", parse_dates = ["col1", "col2"])
```

```
# Specify datetime column
pd.read_csv("filename.csv", parse_dates = {"col1": ["year", "month", "day"]})
```

> Parsing dates, datetimes, and times

```
# Parse dates in ISO format
pd.to_datetime(iso)
```

```
# Parse dates in US format
pd.to_datetime(us, dayfirst=False)
```

```
# Parse dates in NON US format
pd.to_datetime(non_us, dayfirst=True)
```

```
# Parse dates, guessing a single format
pd.to_datetime(iso, infer_datetime_format=True)
```

```
# Parse dates in single, specified format
pd.to_datetime(iso, format="%Y-%m-%d %H:%M:%S")
```

```
# Parse dates in single, specified format
pd.to_datetime(us, format="%m/%d/%Y %H:%M:%S")
```

```
# Make dates from components
pd.to_datetime(parts)
```

> Extracting components

```
# Parse strings to datetimes
dttm = pd.to_datetime(iso)
```

```
# Get year from datetime pandas series
dttm.dt.year
```

```
# Get day of the year from datetime pandas series
dttm.dt.day_of_year
```

```
# Get month name from datetime pandas series
dttm.dt.month_name()
```

```
# Get day name from datetime pandas series
dttm.dt.day_name()
```

```
# Get datetime.datetime format from datetime pandas series
dttm.dt.to_pydatetime()
```

> Rounding dates

```
# Rounding dates to nearest time unit
dttm.dt.round('1min')
```

```
# Flooring dates to nearest time unit
dttm.dt.floor('1min')
```

```
# Ceiling dates to nearest time unit
dttm.dt.ceil('1min')
```

> Arithmetic

```
# Create two datetimes
now = dt.datetime.now()
then = pd.Timestamp('2021-09-15 10:03:30')
```

```
# Get time elapsed as timedelta object
now - then
```

```
# Get time elapsed in seconds
(now - then).total_seconds()
```

```
# Adding a day to a datetime
dt.datetime(2022,8,5,11,13,50) + dt.timedelta(days=1)
```

> Time Zones

```
# Get current time zone
tm.localtime().tm_zone
```

```
# Get a list of all time zones
pytz.all_timezones
```

```
# Parse strings to datetimes
dttm = pd.to_datetime(iso)
```

```
# Get datetime with timezone using location
dttm.dt.tz_localize('CET', ambiguous='infer')
```

```
# Get datetime with timezone using UTC offset
dttm.dt.tz_localize('+0100')
```

```
# Convert datetime from one timezone to another
dttm.dt.tz_localize('+0100').tz_convert('US/Central')
```

> Time Intervals

```
# Create interval datetimes
start_1 = pd.Timestamp('2021-10-21 03:02:10')
finish_1 = pd.Timestamp('2022-09-15 10:03:30')
start_2 = pd.Timestamp('2022-08-21 03:02:10')
finish_2 = pd.Timestamp('2022-12-15 10:03:30')
```

```
# Specify the interval between two datetimes
pd.Interval(start_1, finish_1, closed='right')
```

```
# Get the length of an interval
pd.Interval(start_1, finish_1, closed='right').length
```

```
# Determine if two intervals are intersecting
pd.Interval(start_1, finish_1, closed='right').overlaps(pd.Interval(start_2, finish_2, closed='right'))
```

> Time Deltas

```
# Define a timedelta in days
pd.Timedelta(7, "d")
```

```
# Convert timedelta to seconds
pd.Timedelta(7, "d").total_seconds()
```

Learn Data Skills Online at
www.DataCamp.com

Python For Data Science

SciPy Cheat Sheet

Learn SciPy online at www.DataCamp.com

SciPy



The SciPy library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

> Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j),2j,3j], [(4j,5j,6j)])
>>> c = np.array([(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)])
```

Index Tricks

```
>>> np.mgrid[0:5,0:5] #Create a dense meshgrid
>>> np.ogrid[0:2,0:2] #Create an open meshgrid
>>> np.r_[[3,[0]*5,-1:1:10j]] #Stack arrays vertically (row-wise)
>>> np.c_[b,c] #Create stacked column-wise arrays
```

Shape Manipulation

```
>>> np.transpose(b) #Permute array dimensions
>>> b.flatten() #Flatten the array
>>> np.hstack((b,c)) #Stack arrays horizontally (column-wise)
>>> np.vstack((a,b)) #Stack arrays vertically (row-wise)
>>> np.hsplit(c,2) #Split the array horizontally at the 2nd index
>>> np.vsplit(d,2) #Split the array vertically at the 2nd index
```

Polynomials

```
>>> from numpy import poly1d
>>> p = poly1d([3,4,5]) #Create a polynomial object
```

Vectorizing Functions

```
>>> def myfunc(a):
...     if a < 0:
...         return a*2
...     else:
...         return a/2
>>> np.vectorize(myfunc) #Vectorize functions
```

Type Handling

```
>>> np.real(c) #Return the real part of the array elements
>>> np.imag(c) #Return the imaginary part of the array elements
>>> np.real_if_close(c,tol=1000) #Return a real array if complex parts close to 0
>>> np.cast['f'](np.pi) #Cast object to a data type
```

Other Useful Functions

```
>>> np.angle(b,deg=True) #Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5) #Create an array of evenly spaced values(number of samples)
>>> g [3:] += np.pi
>>> np.unwrap(g) #Unwrap
>>> np.logspace(0,10,3) #Create an array of evenly spaced values (log scale)
>>> np.select([c<4],[c*2]) #Return values from a list of arrays depending on conditions
>>> misc.factorial(a) #Factorial
>>> misc.comb(10,3,exact=True) #Combine N things taken at k time
>>> misc.central_diff_weights(3) #Weights for N-point central derivative
>>> misc.derivative(myfunc,1.0) #Find the n-th derivative of a function at a point
```

> Linear Algebra

You'll use the linalg and sparse modules.

Note that scipy.linalg contains and expands on numpy.linalg.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

```
>>> A.I #Inverse
>>> linalg.inv(A) #Inverse
>>> A.T #Transpose matrix
>>> A.H #Conjugate transposition
>>> np.trace(A) #Trace
```

Norm

```
>>> linalg.norm(A) #Frobenius norm
>>> linalg.norm(A,1) #L1 norm (max column sum)
>>> linalg.norm(A,np.inf) #L inf norm (max row sum)
```

Rank

```
>>> np.linalg.matrix_rank(C) #Matrix rank
```

Determinant

```
>>> linalg.det(A) #Determinant
```

Solving linear problems

```
>>> linalg.solve(A,b) #Solver for dense matrices
>>> E = np.mat(a).T #Solver for dense matrices
>>> linalg.lstsq(D,E) #Least-squares solution to linear matrix equation
```

Generalized inverse

```
>>> linalg.pinv(C) #Compute the pseudo-inverse of a matrix (least-squares solver)
```

```
>>> linalg.pinv2(C) #Compute the pseudo-inverse of a matrix (SVD)
```

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1) #Create a 2X2 identity matrix
>>> G = np.mat(np.identity(2)) #Create a 2x2 identity matrix
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C) #Compressed Sparse Row matrix
>>> I = sparse.csc_matrix(D) #Compressed Sparse Column matrix
>>> J = sparse.dok_matrix(A) #Dictionary Of Keys matrix
>>> E.todense() #Sparse matrix to full matrix
>>> sparse.isspmatrix_csc(A) #Identify sparse matrix
```

Sparse Matrix Routines

Inverse

```
>>> sparse.linalg.inv(I) #Inverse
```

Norm

```
>>> sparse.linalg.norm(I) #Norm
```

Solving linear problems

```
>>> sparse.linalg.spsolve(H,I) #Solver for sparse matrices
```

Sparse Matrix Functions

```
>>> sparse.linalg.expm(I) #Sparse matrix exponential
```

Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1) #Eigenvalues and eigenvectors
>>> sparse.linalg.svds(H, 2) #SVD
```

Matrix Functions

Addition

```
>>> np.add(A,D) #Addition
```

Subtraction

```
>>> np.subtract(A,D) #Subtraction
```

Division

```
>>> np.divide(A,D) #Division
```

Multiplication

```
>>> np.multiply(D,A) #Multiplication
>>> np.dot(A,D) #Dot product
>>> np.vdot(A,D) #Vector dot product
>>> np.inner(A,D) #Inner product
>>> np.outer(A,D) #Outer product
>>> np.tensordot(A,D) #Tensor dot product
>>> np.kron(A,D) #Kronecker product
```

Exponential Functions

```
>>> linalg.expm(A) #Matrix exponential
>>> linalg.expm2(A) #Matrix exponential (Taylor Series)
>>> linalg.expm3(D) #Matrix exponential (eigenvalue decomposition)
```

Logarithm Function

```
>>> linalg.logm(A) #Matrix logarithm
```

Trigonometric Functions

```
>>> linalg.sinm(D) #Matrix sine
>>> linalg.cosm(D) #Matrix cosine
>>> linalg.tanm(A) #Matrix tangent
```

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D) #Hyperbolic matrix sine
>>> linalg.coshm(D) #Hyperbolic matrix cosine
>>> linalg.tanhm(A) #Hyperbolic matrix tangent
```

Matrix Sign Function

```
>>> np.sign(A) #Matrix sign function
```

Matrix Square Root

```
>>> linalg.sqrtm(A) #Matrix square root
```

Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x) #Evaluate matrix function
```

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A) #Solve ordinary or generalized eigenvalue problem for square matrix
>>> l1, l2 = la #Unpack eigenvalues
>>> v[:,0] #First eigenvector
>>> v[:,1] #Second eigenvector
>>> linalg.eigvals(A) #Unpack eigenvalues
```

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B) #Singular Value Decomposition (SVD)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N) #Construct sigma matrix in SVD
```

LU Decomposition

```
>>> P,L,U = linalg.lu(C) #LU Decomposition
```

> Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

Learn Data Skills Online at
www.DataCamp.com

Python For Data Science

PySpark RDD Cheat Sheet

Learn PySpark RDD online at www.DataCamp.com

Spark



PySpark is the Spark Python API that exposes the Spark programming model to Python.

> Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

```
>>> sc.version #Retrieve SparkContext version
>>> sc.pythonVer #Retrieve Python version
>>> sc.master #Master URL to connect to
>>> str(sc.sparkHome) #Path where Spark is installed on worker nodes
>>> str(sc.sparkUser()) #Retrieve name of the Spark User running SparkContext
>>> sc.appName #Return application name
>>> sc.applicationId #Retrieve application ID
>>> sc.defaultParallelism #Return default level of parallelism
>>> sc.defaultMinPartitions #Default minimum number of partitions for RDDs
```

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
...     .setMaster("local")
...     .setAppName("My app")
...     .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

> Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([('a',[ "x", "y", "z"]),
...                         ('b',[ "p", "r"])]))
```

External Data

Read either one text file from HDFS, a local file system or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

> Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions() #List the number of partitions
>>> rdd.count() #Count RDD instances 3
>>> rdd.countByKey() #Count RDD instances by key
defaultdict(<type 'int'>,{'a':2,'b':1})
>>> rdd.countByValue() #Count RDD instances by value
defaultdict(<type 'int'>,{'b',2}:1,{'a',2}:1,{'a',7}:1)
>>> rdd.collectAsMap() #Return (key,value) pairs as a dictionary
{'a': 2, 'b': 2}
>>> rdd3.sum() #Sum of RDD elements 4950
>>> sc.parallelize([]).isEmpty() #Check whether RDD is empty
True
```

Summary

```
>>> rdd3.max() #Maximum value of RDD elements
99
>>> rdd3.min() #Minimum value of RDD elements
0
>>> rdd3.mean() #Mean value of RDD elements
49.5
>>> rdd3.stdev() #Standard deviation of RDD elements
28.86607004772218
>>> rdd3.variance() #Compute variance of RDD elements
833.25
>>> rdd3.histogram(3) #Compute histogram by bins
([0,33,66,99],[33,33,34])
>>> rdd3.stats() #Summary statistics (count, mean, stdev, max & min)
```

> Applying Functions

```
#Apply a function to each RDD element
>>> rdd.map(lambda x: x+(x[1],x[0])).collect()
[('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]
#Apply a function to each RDD element and flatten the result
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))
>>> rdd5.collect()
['a',7,7,'a','a',2,2,'a','b',2,2,'b']
#Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys
>>> rdd4.flatMapValues(lambda x: x).collect()
[('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

> Selecting Data

Getting

```
>>> rdd.collect() #Return a list with all RDD elements
[('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2) #Take first 2 RDD elements
[('a', 7), ('a', 2)]
>>> rdd.first() #Take first RDD element
('a', 7)
>>> rdd.top(2) #Take top 2 RDD elements
[('b', 2), ('a', 7)]
```

Sampling

```
>>> rdd3.sample(False, 0.15, 81).collect() #Return sampled subset of rdd3
[3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

Filtering

```
>>> rdd.filter(lambda x: "a" in x).collect() #Filter the RDD
[('a',7),('a',2)]
>>> rdd5.distinct().collect() #Return distinct RDD values
[('a',2),('b',7)]
>>> rdd.keys().collect() #Return (key,value) RDD's keys
[('a', 'a', 'b')]
```

> Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g) #Apply a function to all RDD elements
('a', 7)
('b', 2)
('a', 2)
```

> Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y).collect() #Merge the rdd values for each key
[('a',9),('b',2)]
>>> rdd.reduce(lambda a, b: a + b) #Merge the rdd values
('a',7,'a',2,'b',2)
```

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2) #Return RDD of grouped values
.mapValues(list)
.collect()
>>> rdd.groupByKey() #Group rdd by key
.mapValues(list)
.collect()
[('a',[7,2]),('b',[2])]
```

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
#Aggregate RDD elements of each partition and then the results
>>> rdd3.aggregate((0,0),seqOp,combOp)
(4950,100)
#Aggregate values of each RDD key
>>> rdd.aggregateByKey((0,0),seqOp,combOp).collect()
[('a',(9,2)), ('b',(2,1))]
#Aggregate the elements of each partition, and then the results
>>> rdd3.fold(0,add)
4950
#Merge the values for each key
>>> rdd.foldByKey(0, add).collect()
[('a',9),('b',2)]
#Create tuples of RDD elements by applying a function
>>> rdd3.keyBy(lambda x: x*x).collect()
```

> Mathematical Operations

```
>>> rdd.subtract(rdd2).collect() #Return each rdd value not contained in rdd2
[('b',2),('a',7)]
#Return each (key,value) pair of rdd2 with no matching key in rdd
>>> rdd2.subtractByKey(rdd).collect()
[('d', 1)]
>>> rdd.cartesian(rdd2).collect() #Return the Cartesian product of rdd and rdd2
```

> Sort

```
>>> rdd2.sortBy(lambda x: x[1]).collect() #Sort RDD by given function
[('d',1),('b',1),('a',2)]
>>> rdd2.sortByKey().collect() #Sort (key, value) RDD by key
[('a',2),('b',1),('d',1)]
```

> Repartitioning

```
>>> rdd.repartition(4) #New RDD with 4 partitions
>>> rdd.coalesce(1) #Decrease the number of partitions in the RDD to 1
```

> Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
...                         'org.apache.hadoop.mapred.TextOutputFormat')
```

> Stopping SparkContext

```
>>> sc.stop()
```

> Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

Hello Python!

INTRODUCTION TO PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

How you will learn

The screenshot shows a learning interface for an "Introduction to Python" course. At the top, there's a navigation bar with a logo, the path "Learn / Courses / Introduction to Python", and a "Course Outline" button. Below the navigation is a toolbar with a "Light Mode" switch.

Exercise: Python as a calculator

Python is perfectly suited to do basic calculations. It can do addition, subtraction, multiplication and division.

The code in the script gives some examples.

Now it's your turn to practice!

Instructions: 500 XP

- Print the sum of 5 + 5.
- Print the result of subtracting 5 from 10.
- Multiply 3 by 5.
- Divide 10 by 2.

Take Hint (30 XP)

script.py

```
1 # Addition
2 print(5 + 5)
3
4 # Subtraction
5 print(10 - 5)
6
7 # Multiplication
8 print(3 * 5)
9
10 # Division
11 print(10 / 2)
```

Run Code **Submit Answer**

Python Shell

```
In [1]:
```

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 Python exercise interface. On the left, there's a sidebar with navigation links: 'Learn / Courses / Introduction to Python' and a 'Course Outline' button. Below that is a section titled 'Exercise' with the sub-section 'Python as a calculator'. It contains text about Python being suited for calculations and some examples. A 'Instructions' section with a '100 XP' badge lists four tasks: 'Print the sum of 5 + 5.', 'Print the result of subtracting 5 from 5.', 'Multiply 3 by 5.', and 'Divide 10 by 2.'. A 'Take Hint (-30 XP)' button is also present. The main area is a code editor titled 'script.py' containing the following code:

```
1 # Addition
2
3 # Subtraction
4
5 # Multiplication
6
7 # Division
```

Below the code editor are three buttons: 'Run Code', 'Submit Answer', and a refresh icon. At the bottom is an 'IPython Shell' window with the prompt 'In: [1]:'.

IPython Shell

Execute Python commands

The screenshot shows a Python exercise interface. On the left, there's a sidebar with navigation links: 'Learn / Courses / Introduction to Python' and a 'Course Outline' button. Below that is a section titled 'Exercise' with a sub-section 'Python as a calculator'. It contains text about Python being suited for calculations and some sample code. To the right is a code editor window titled 'script.py' containing the following code:

```
script.py
1 # Addition
2
3 # Subtraction
4
5 # Multiplication
6
7 # Division
8
9
10
11
```

Below the code editor are three buttons: 'Run Code', 'Submit Answer', and a question mark icon. To the left of the code editor is a box labeled 'Instructions' with '100 XP' and a list of four tasks:

- Print the sum of 5 + 5.
- Print the result of subtracting 5 from 5.
- Multiply 5 by 5.
- Divide 10 by 2.

At the bottom left is a button 'Take Hint (-30 XP)'. At the very bottom is an 'IPython Shell' window with the prompt 'In [1]:'.

IPython Shell

The screenshot shows a web-based Python exercise interface. At the top, there's a navigation bar with a logo, 'Learn / Courses / Introduction to Python', and a 'Course Outline' button. Below the navigation is a toolbar with a green circle icon, a 'Light Mode' toggle, and other icons. On the left, a sidebar titled 'Exercise' contains the title 'Python as a calculator'. It explains that Python is suited for basic calculations like addition, subtraction, multiplication, and division. It also mentions that the script provides examples. A 'Instructions' section lists four tasks: 'Print the sum of 5 + 5', 'Print the result of subtracting 5 from 5', 'Multiply 5 by 5', and 'Divide 10 by 2'. There's a 'Take Hint (-50 XP)' button. The main area is a dark-themed IPython Shell window titled 'script.py'. It has a code editor with a single digit '1' at the top, a 'Run Code' button, and a 'Submit Answer' button which is highlighted in green. Below the editor is an 'IPython Shell' tab and the text 'In [1]:'. The bottom right corner of the shell window has a small circular icon.

Python Script

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

The screenshot shows a Python script editor interface. The main window displays a code editor with a dark theme containing the following Python code:

```
script.py
1 # Addition
2
3 # Subtraction
4
5 # Multiplication
6
7 # Division
8
9
10
11
```

The code editor has a green border. Below it is an IPython Shell window showing the prompt `In [1]:`.

On the left side of the interface, there is a sidebar with the following sections:

- Exercise**:
 - Python as a calculator**: A brief text explaining Python's suitability for calculations.
 - Instructions**:
 - Print the sum of 5 + 5.
 - Print the result of subtracting 5 from 5.
 - Multiply 3 by 5.
 - Divide 10 by 2.
 - Take Hint (-30 XP)**

At the top of the interface, there are navigation buttons for "Course Outline" and "Light Mode".

Python Script

The screenshot shows a Python script editor interface. At the top, there's a navigation bar with a logo, 'Learn / Courses / Introduction to Python', and buttons for 'Course Outline' and 'Light Mode'. Below the navigation is a sidebar on the left containing the title 'Python as a calculator', a brief description about Python being suited for calculations, and some introductory text. A 'Instructions' section with a yellow '100 XP' badge lists four tasks: 'Print the sum of 4 + 5.', 'Print the result of subtracting 5 from 5.', 'Multiply 3 by 5.', and 'Divide 10 by 2.'. A 'Take Hint (-30 XP)' button is also present. The main workspace is a dark-themed code editor titled 'script.py' with a single line of code '1'. At the bottom right of the editor are 'Run Code' and 'Submit Answer' buttons. Below the editor is an 'IPython Shell' window showing the prompt 'In [1]:'.

Python Script

The screenshot shows a Python script exercise interface. At the top, there's a navigation bar with 'Learn' and 'Courses' buttons, followed by 'Introduction to Python'. Below that is a 'Course Outline' section with left and right arrows. On the far right of the header is a green circular icon with a white dot and a small triangle, and a 'Light Mode' button.

The main area is titled 'Exercise' with a sub-section 'Python as a calculator'. It contains text explaining that Python is good for calculations and provides examples. Below this is an 'Instructions' section with a '100 XP' badge. It lists four tasks:

- Print the sum of 4 + 5.
- Print the result of subtracting 5 from 5.
- Multiply 3 by 5.
- Divide 10 by 2.

At the bottom of the exercise area is a 'Take Hint (-30 XP)' button.

The central part of the interface features a code editor window titled 'script.py' containing the number '1'. Below it is an 'IPython Shell' window showing the prompt 'In [1]:'. At the bottom of the shell window are three buttons: a blue 'Run Code' button, a red 'Submit Answer' button, and a grey 'Clear' button.

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

DataCamp Interface

The screenshot shows a DataCamp exercise interface for "Introduction to Python".

Header: Learn / Courses / Introduction to Python

Exercise Tab: Exercise

Section Title: Python as a calculator

Description: Python is perfectly suited to do basic calculations. It can do addition, subtraction, multiplication and division.

Text: The code in the script gives some examples.

Text: Now it's your turn to practice!

Instructions: 100 XP

Code Editor: script.py

```
1 # Addition
2
3 # Subtraction
4
5 # Multiplication
6
7 # Division
8
9
10
11
```

Buttons: Run Code, Submit Answer

IPython Shell: In [1]:

Buttons: Take Hint (-30 XP)

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

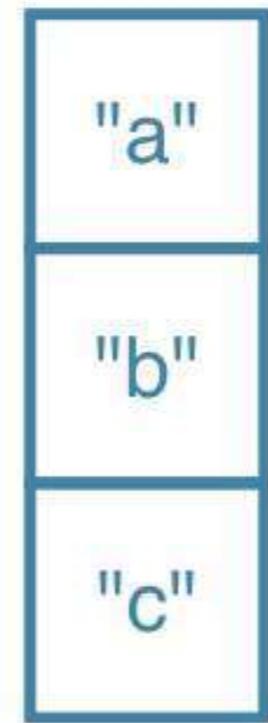
```
del(fam[2])
```

```
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
```

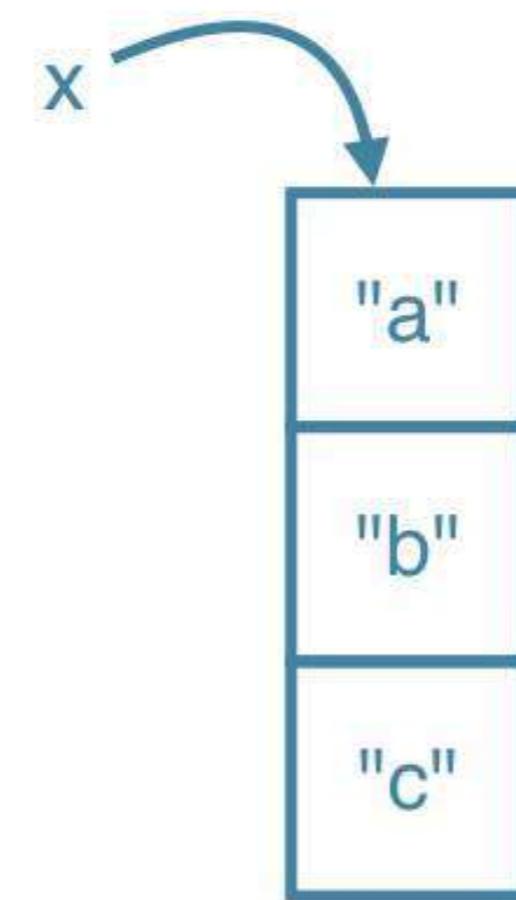
```
y[1] = "z"
```

```
y
```

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

```
x
```

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



Behind the scenes (1)

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

```
y = x
```

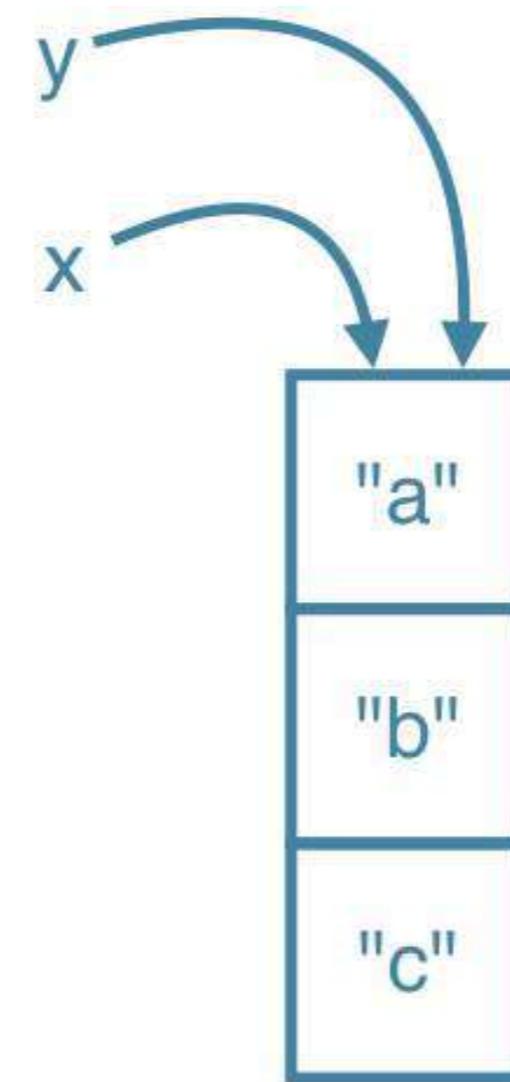
```
y[1] = "z"
```

```
y
```

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

```
x
```

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



Behind the scenes (1)

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

```
y = x
```

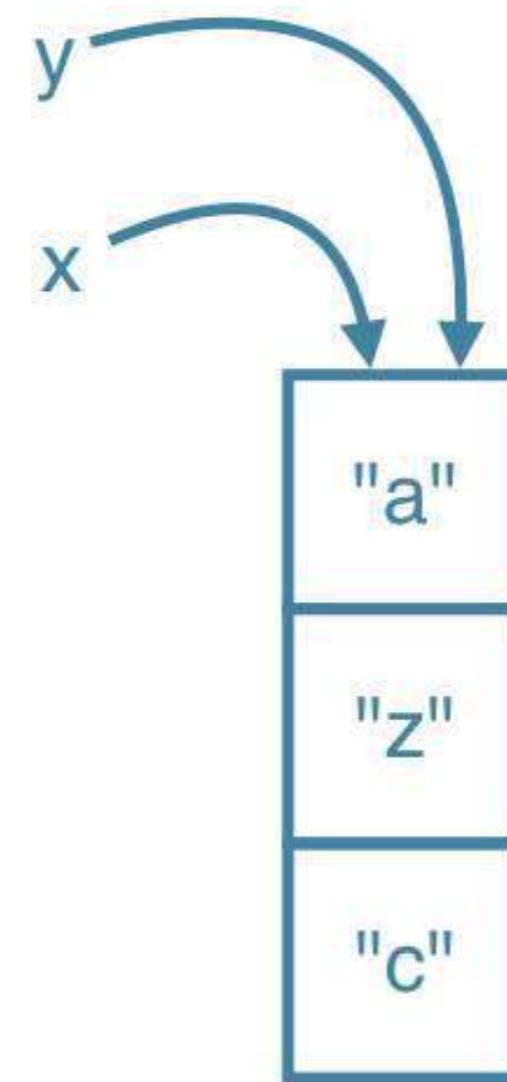
```
y[1] = "z"
```

```
y
```

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

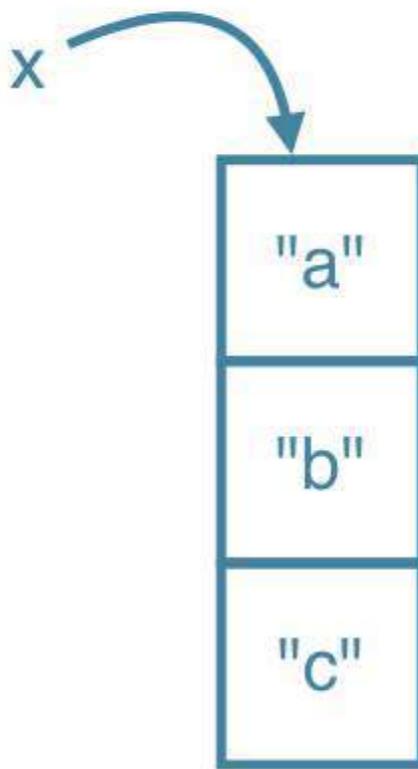
```
x
```

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



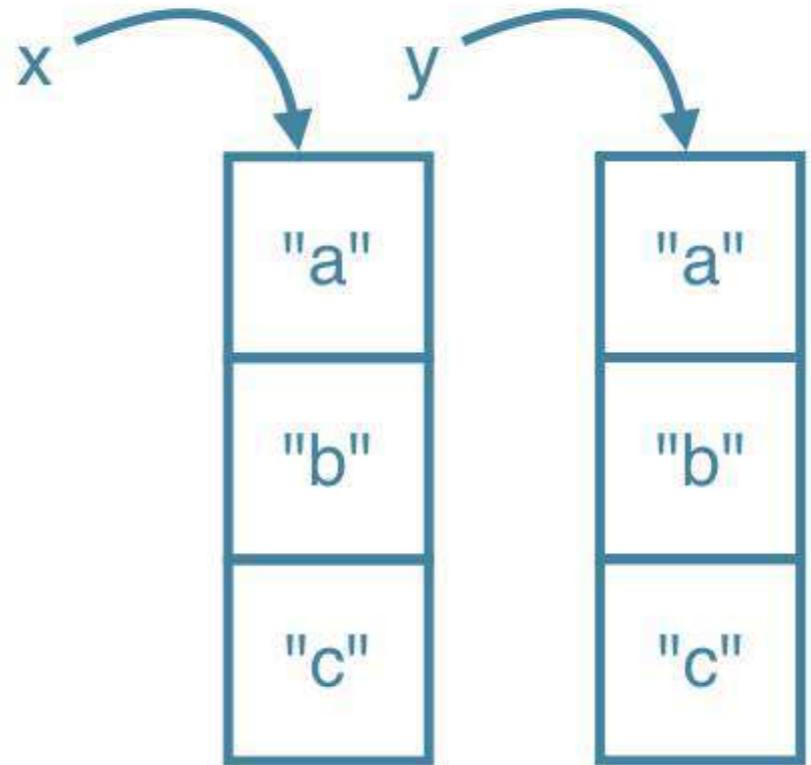
Behind the scenes (2)

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



Behind the scenes (2)

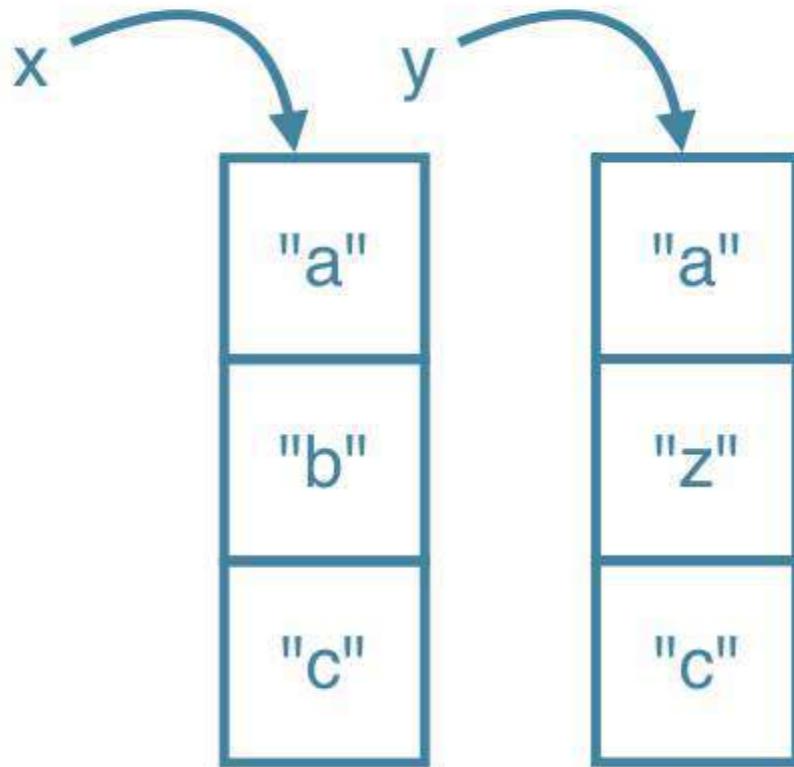
```
x = ["a", "b", "c"]  
y = list(x)  
y = x[:]
```



Behind the scenes (2)

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

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

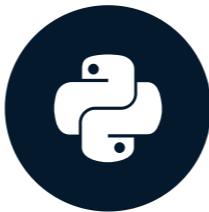


Let's practice!

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] →



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() → 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:

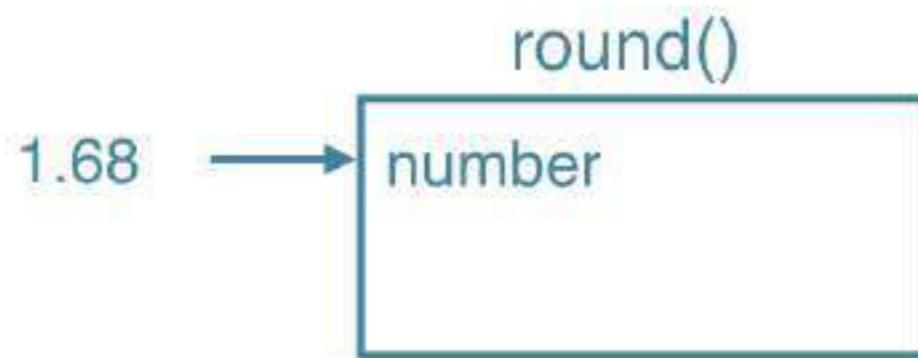
```
round(number, ndigits=None)
```

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:

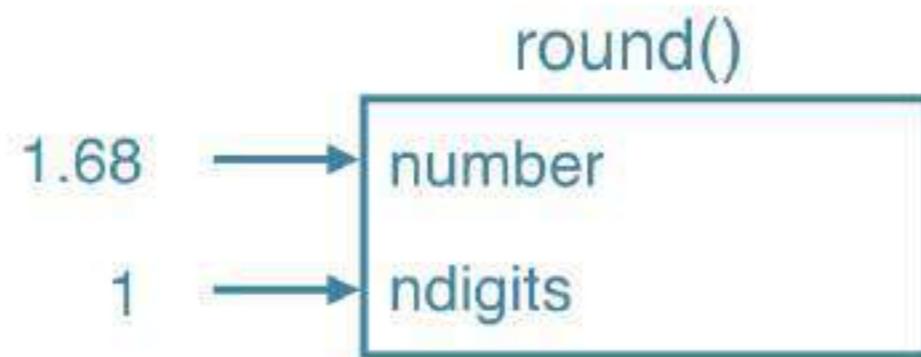
```
round(number, ndigits=None)
```

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:

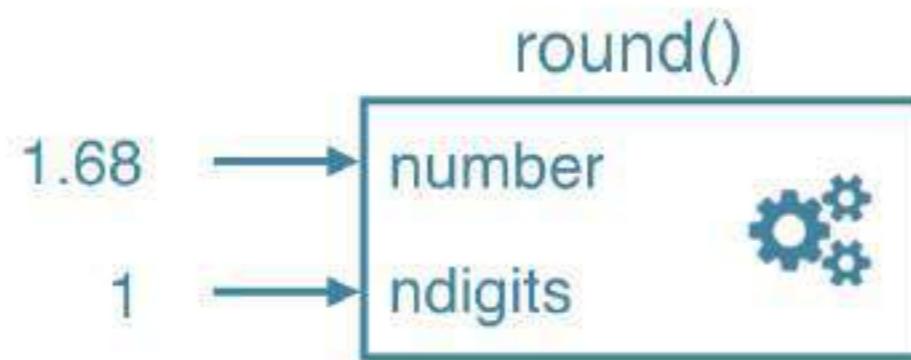
```
round(number, ndigits=None)
```

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:

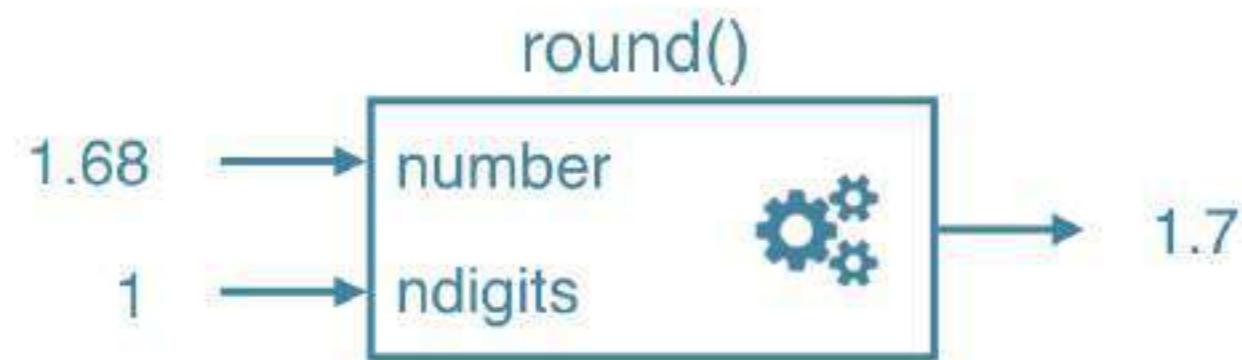
```
round(number, ndigits=None)
```

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:

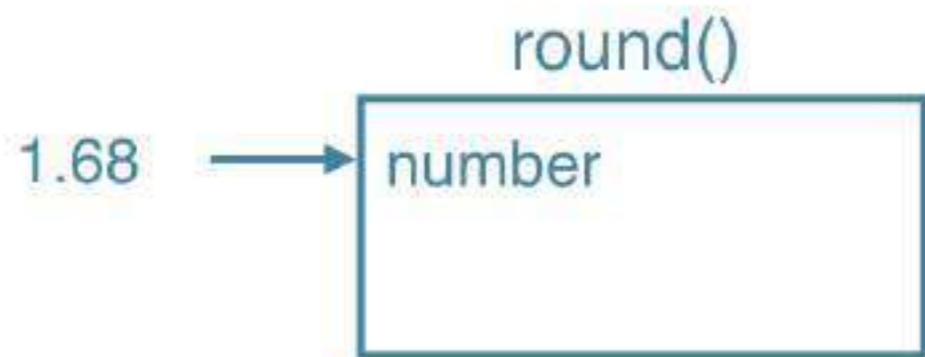
```
round(number, ndigits=None)
```

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:

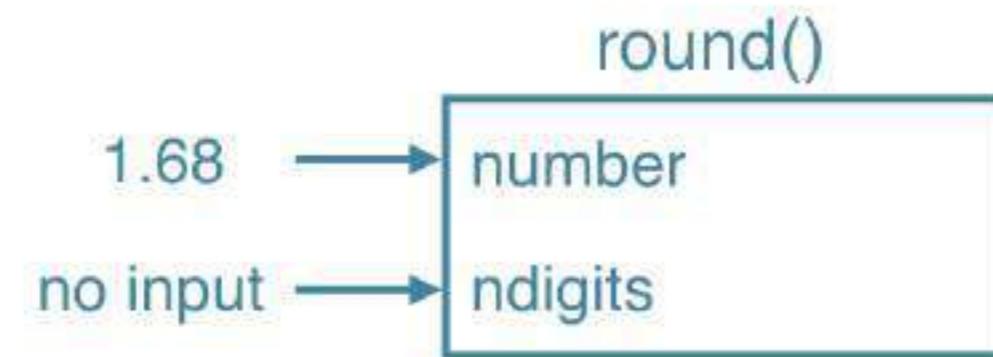
```
round(number, ndigits=None)
```

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:

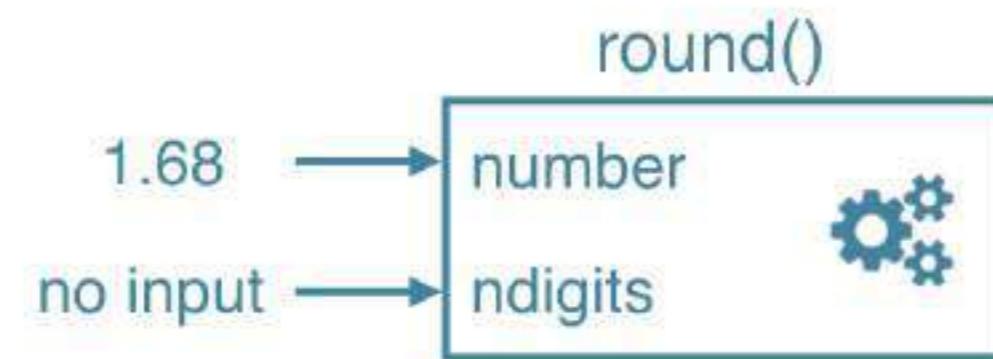
```
round(number, ndigits=None)
```

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:

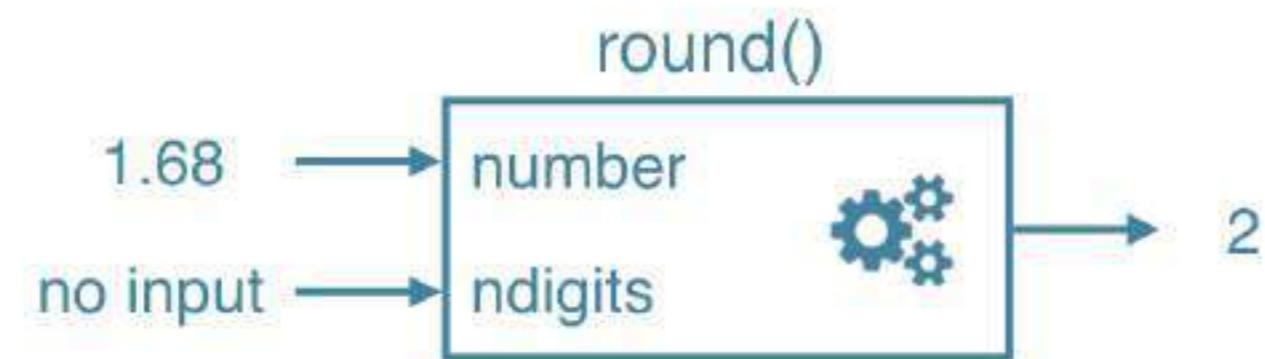
```
round(number, ndigits=None)
```

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

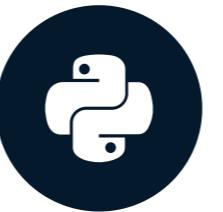
- How to know?
- Standard task -> probably function exists!
- The internet is your friend

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")
```

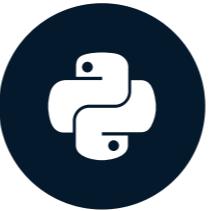
```
6
```

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")

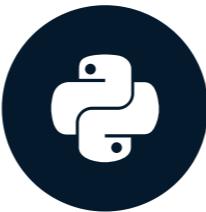
...
np_fam = np.array(fam_ext) # Clearly using NumPy
```

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

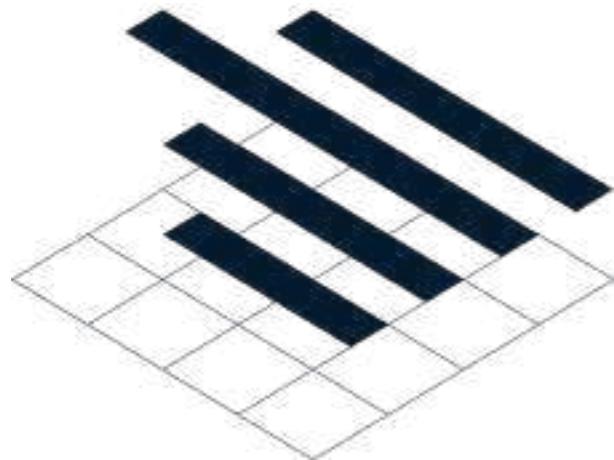
INTERMEDIATE PYTHON



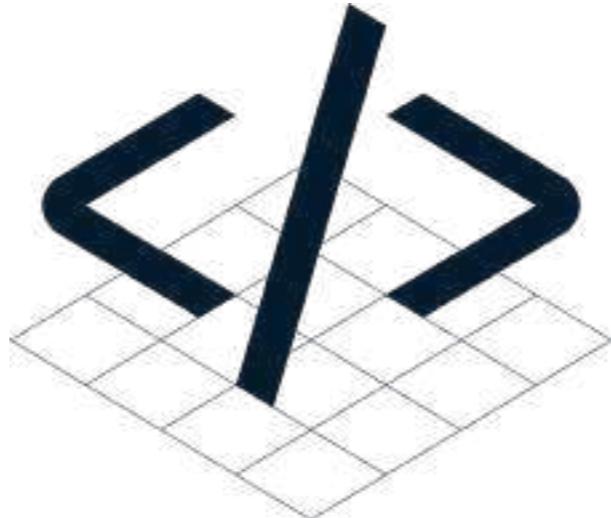
Hugo Bowne-Anderson
Data Scientist at DataCamp

Basic plots with Matplotlib

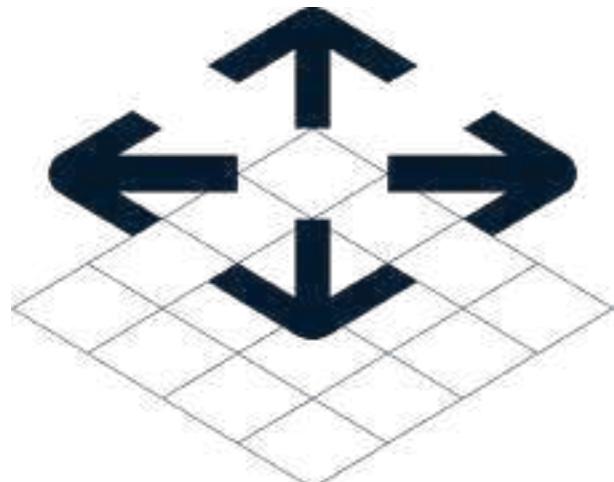
- Visualization



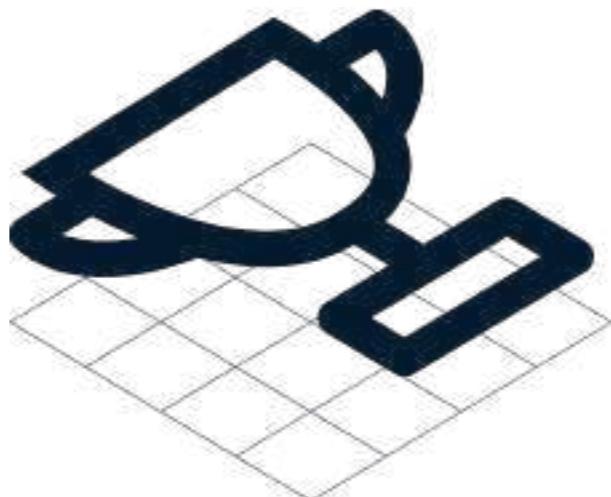
- Data Structure



- Control Structures

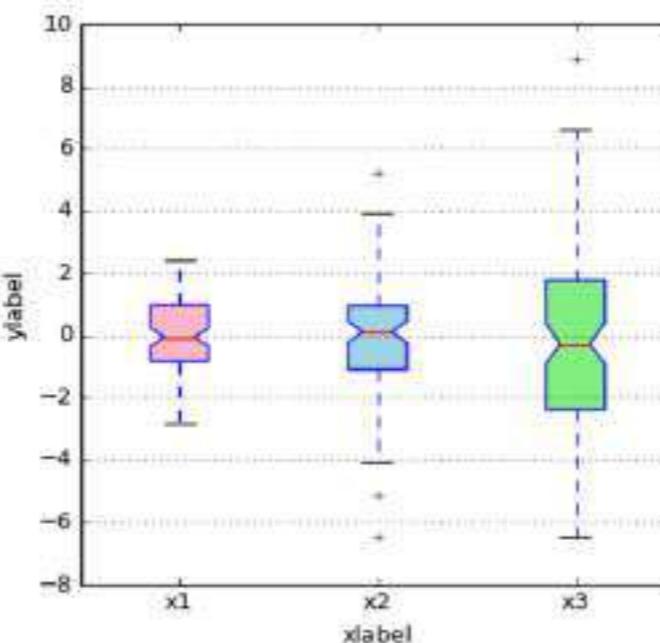
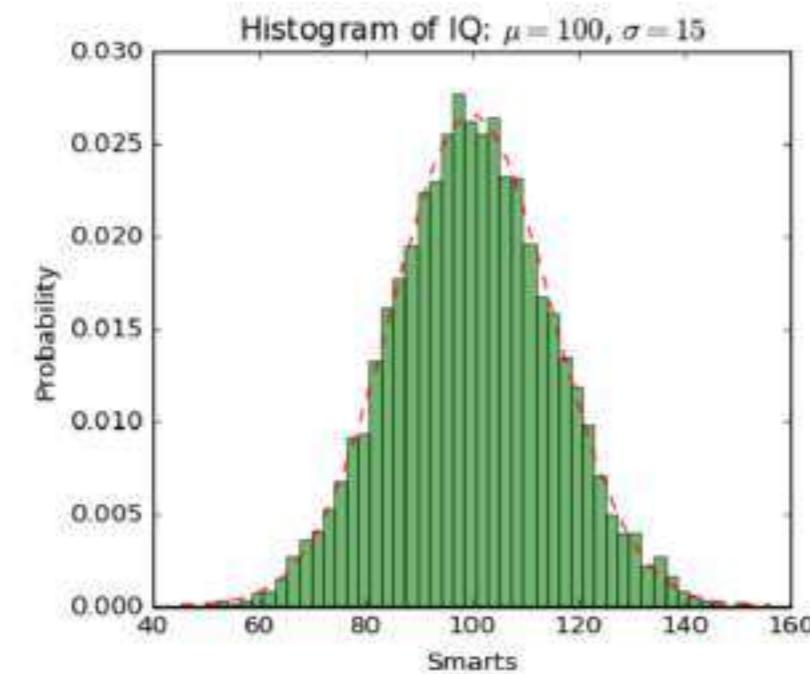


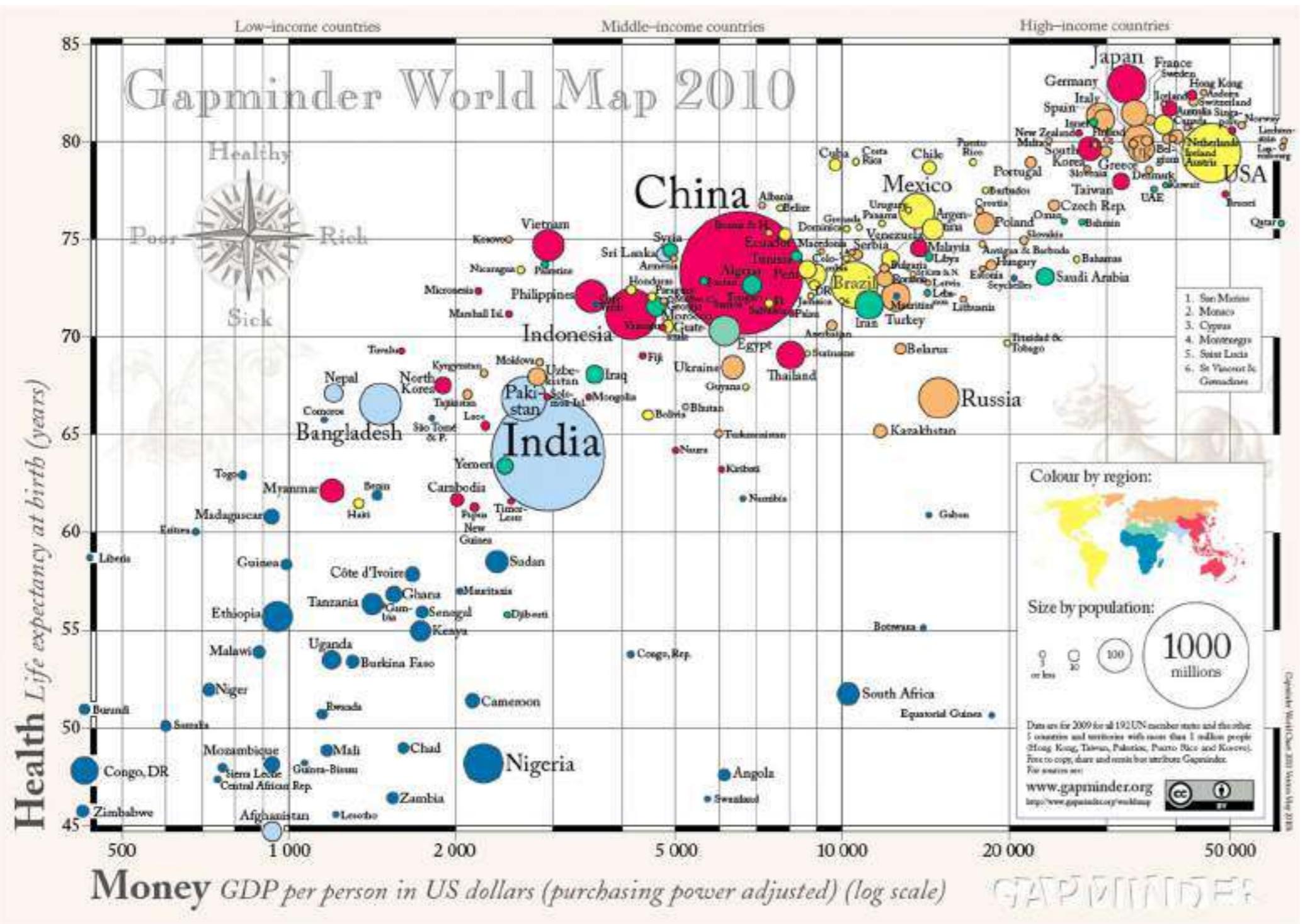
- Case Study



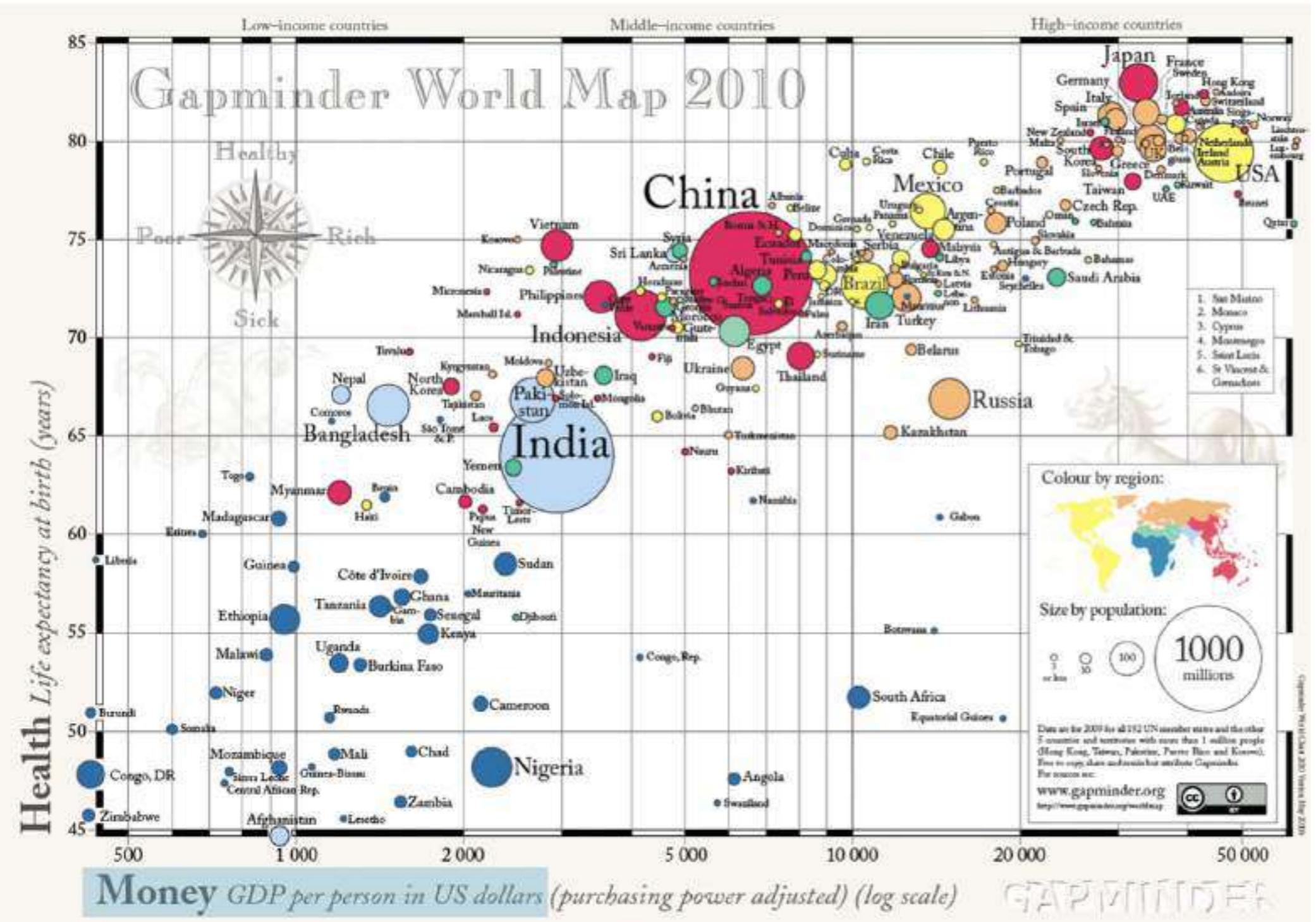
Data visualization

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

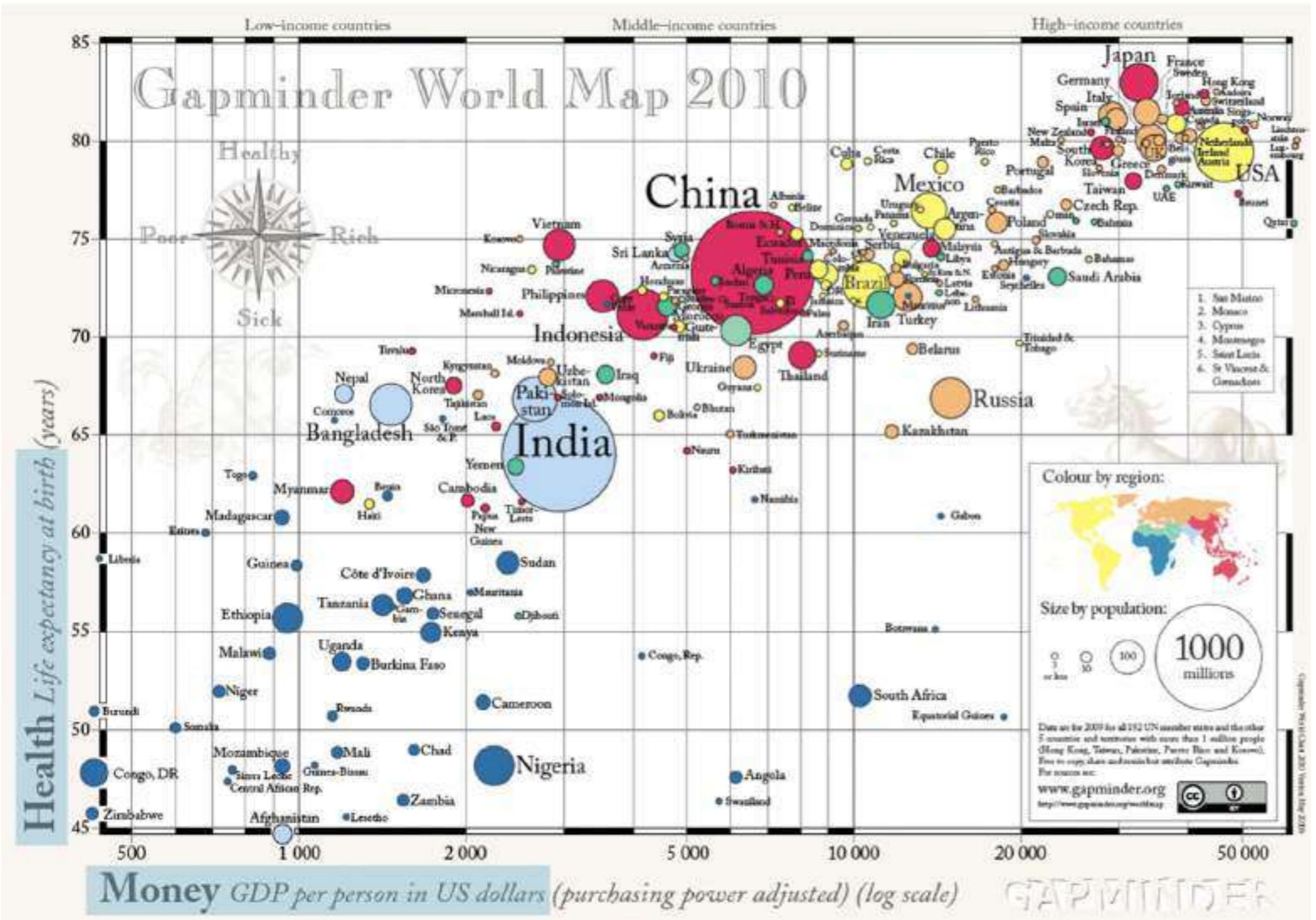




¹ Source: GapMinder, Wealth and Health of Nations



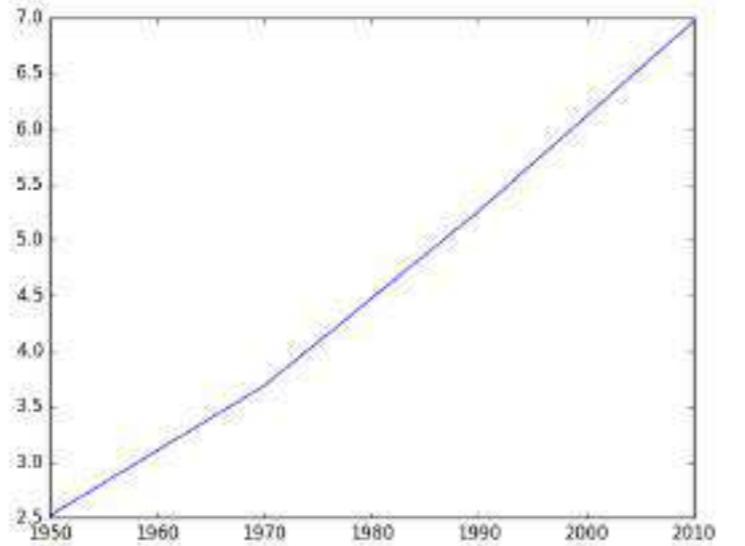
¹ Source: GapMinder, Wealth and Health of Nations



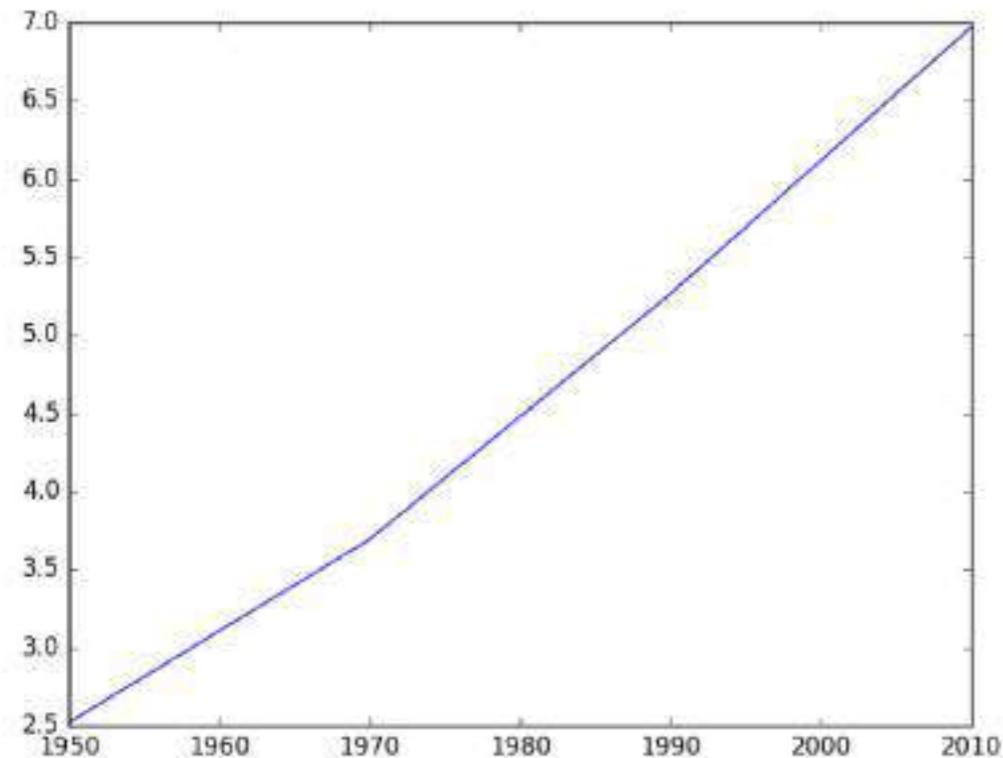
¹ Source: GapMinder, Wealth and Health of Nations

Matplotlib

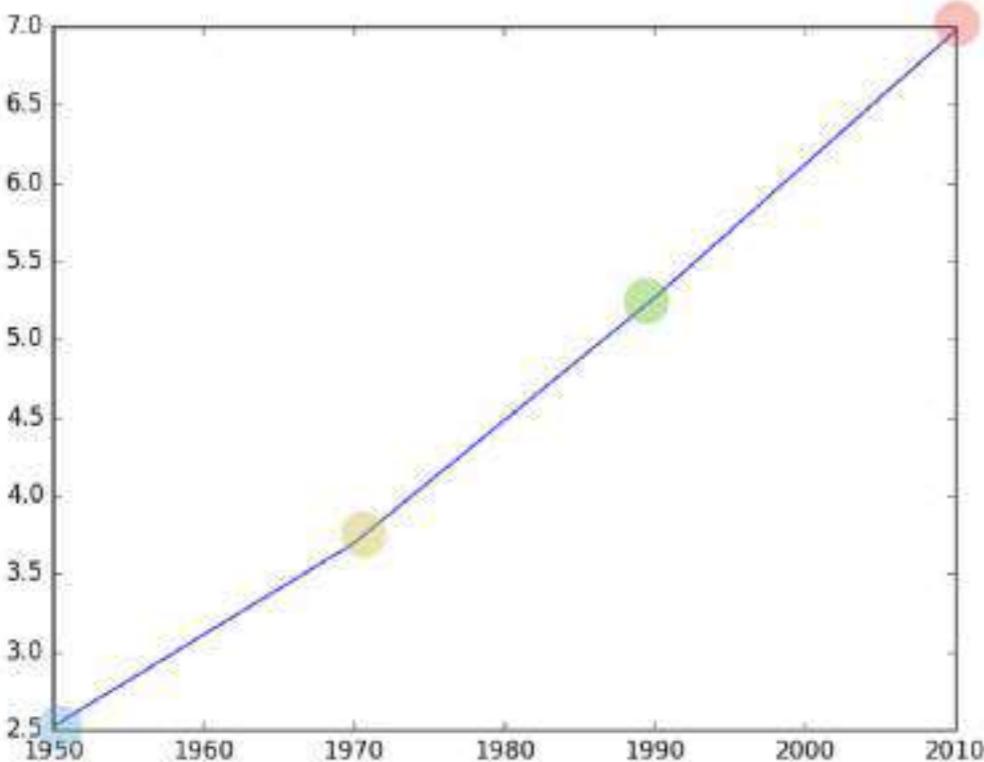
```
import matplotlib.pyplot as plt  
year = [1950, 1970, 1990, 2010]  
pop = [2.519, 3.692, 5.263, 6.972]  
plt.plot(year, pop)  
plt.show()
```



Matplotlib



Matplotlib



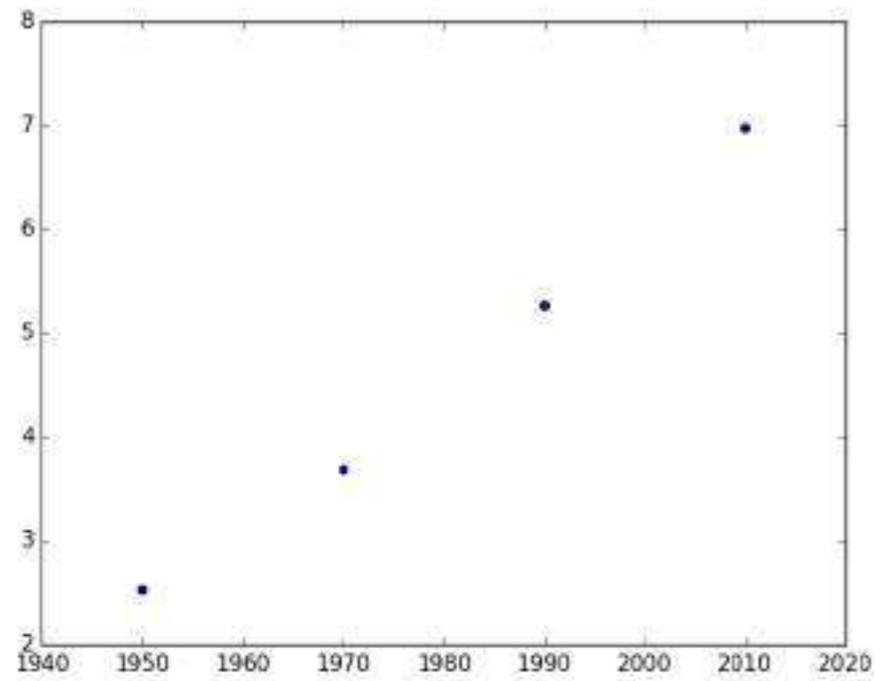
```
year = [1950 , 1970 , 1990 , 2010]
pop  = [2.519, 3.692, 5.263, 6.972]
```

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



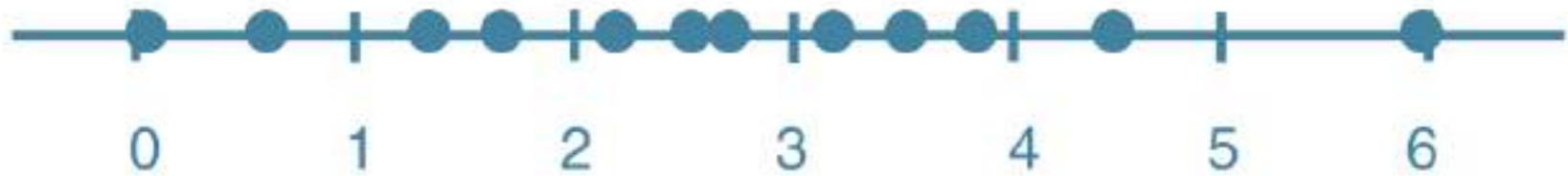
Hugo Bowne-Anderson
Data Scientist at DataCamp

Histogram

- Explore dataset
- Get idea about distribution

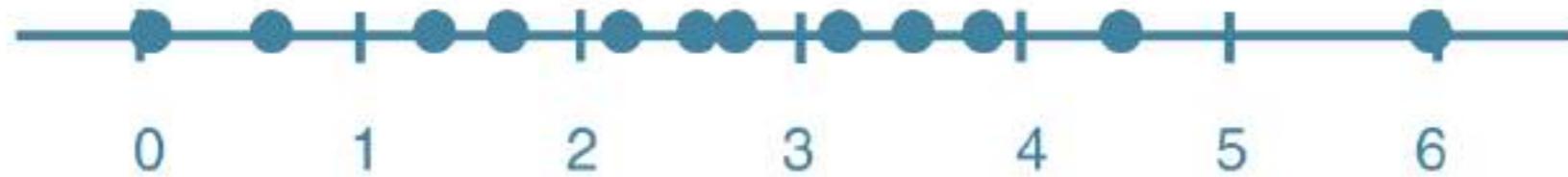
Histogram

- Explore dataset
- Get idea about distribution



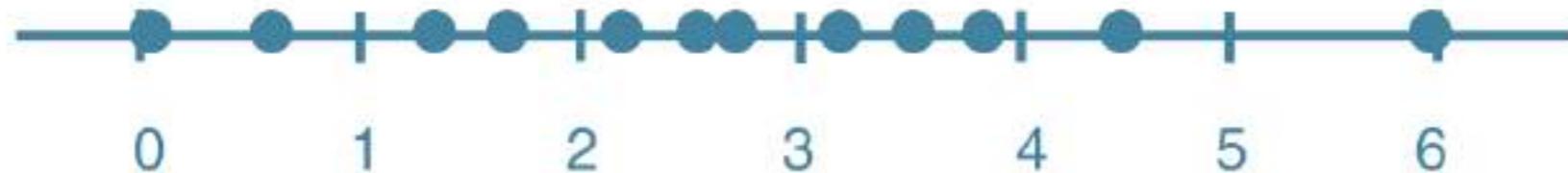
Histogram

- Explore dataset
- Get idea about distribution



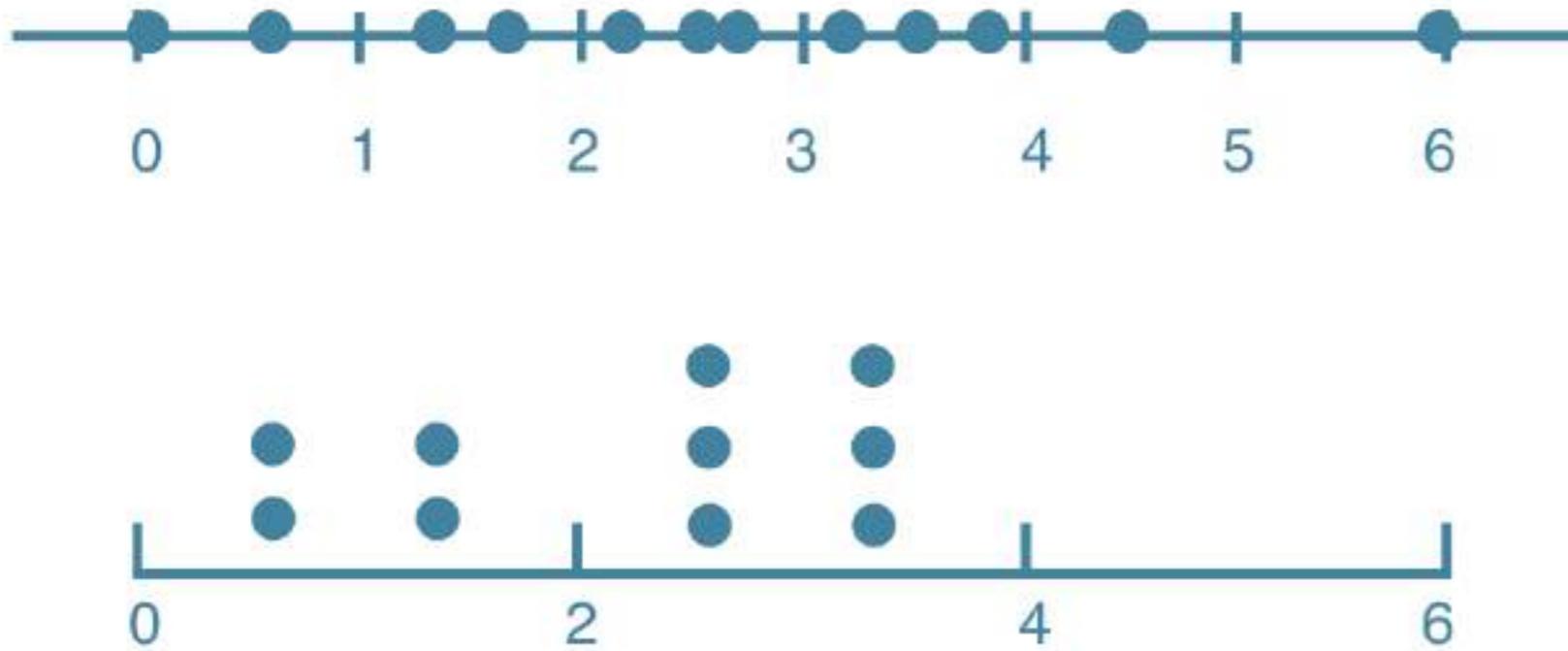
Histogram

- Explore dataset
- Get idea about distribution



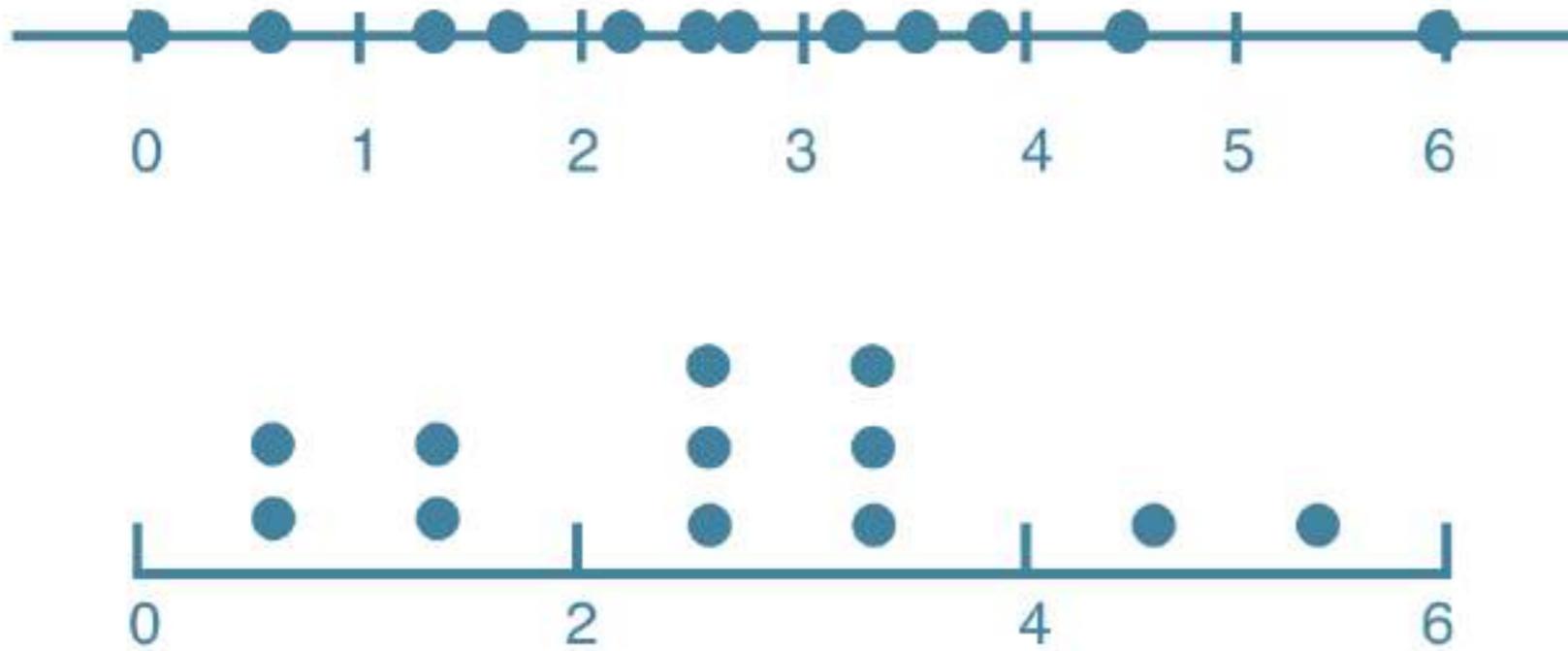
Histogram

- Explore dataset
- Get idea about distribution



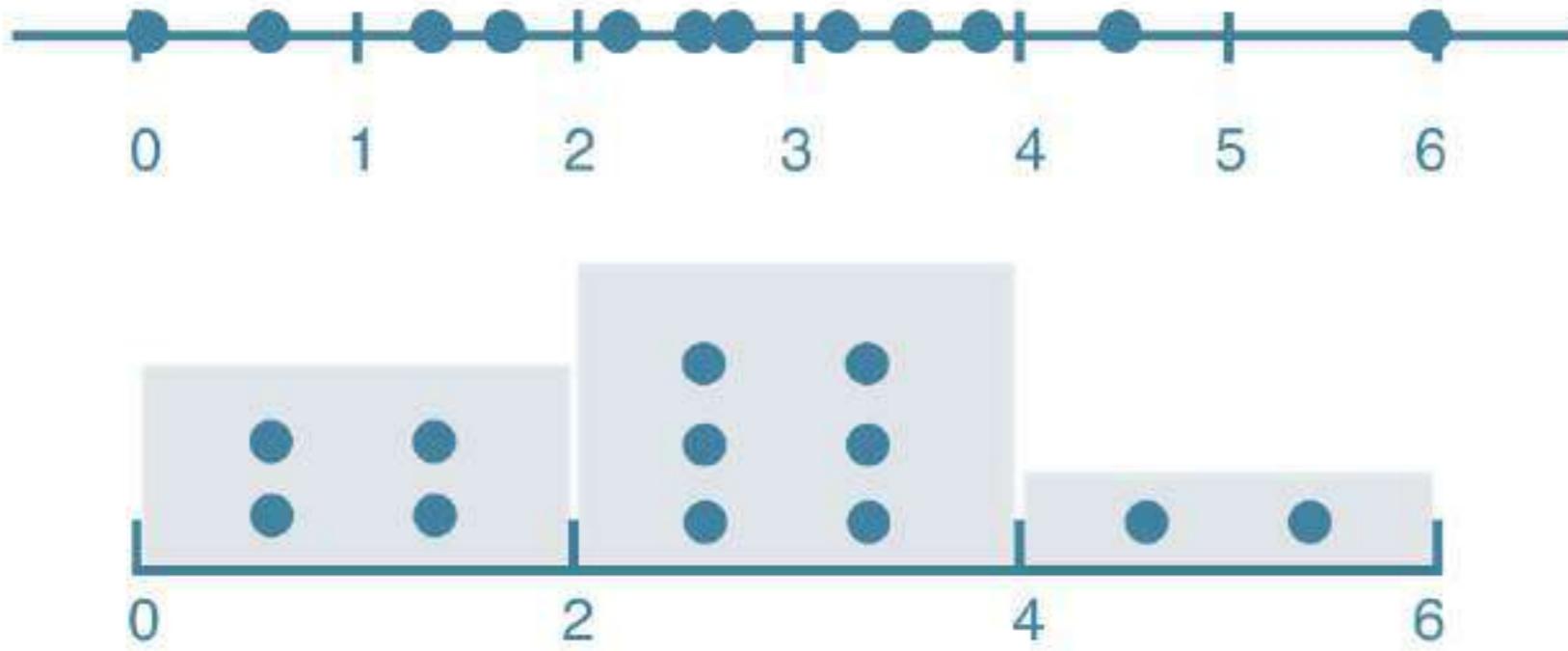
Histogram

- 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:

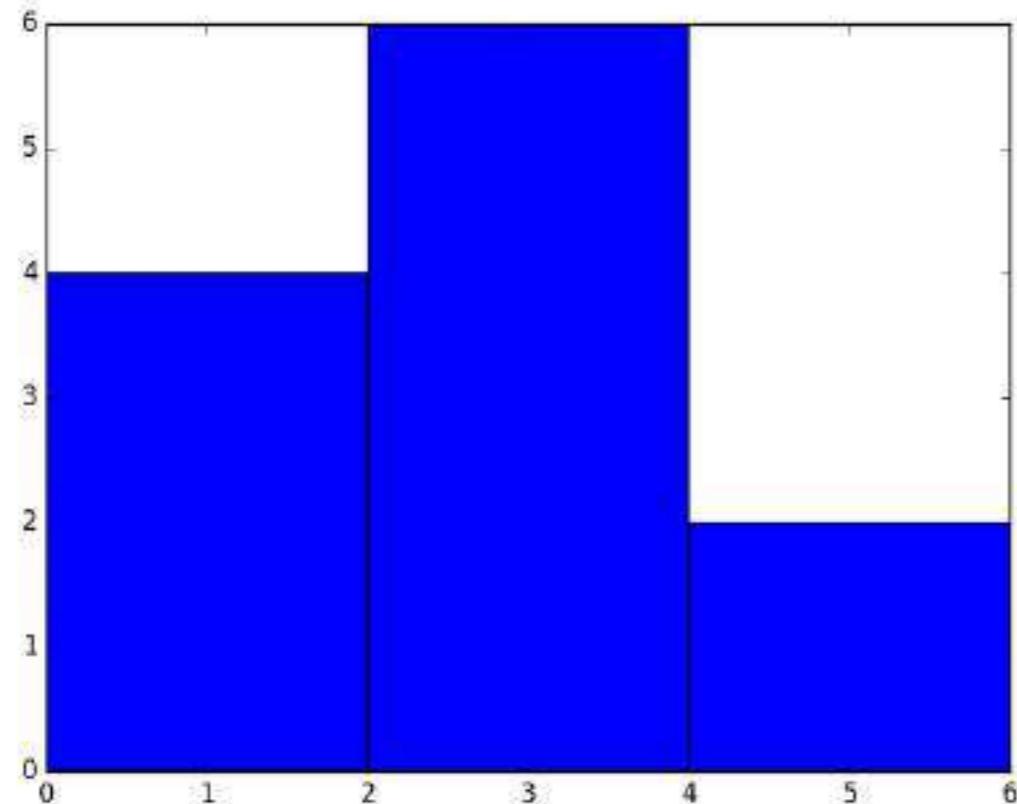
```
hist(x, bins=None, range=None, density=False, weights=None,  
cumulative=False, bottom=None, histtype='bar', align='mid',  
orientation='vertical', rwidth=None, log=False, color=None,  
label=None, stacked=False, *, data=None, **kwargs)
```

Plot a histogram.

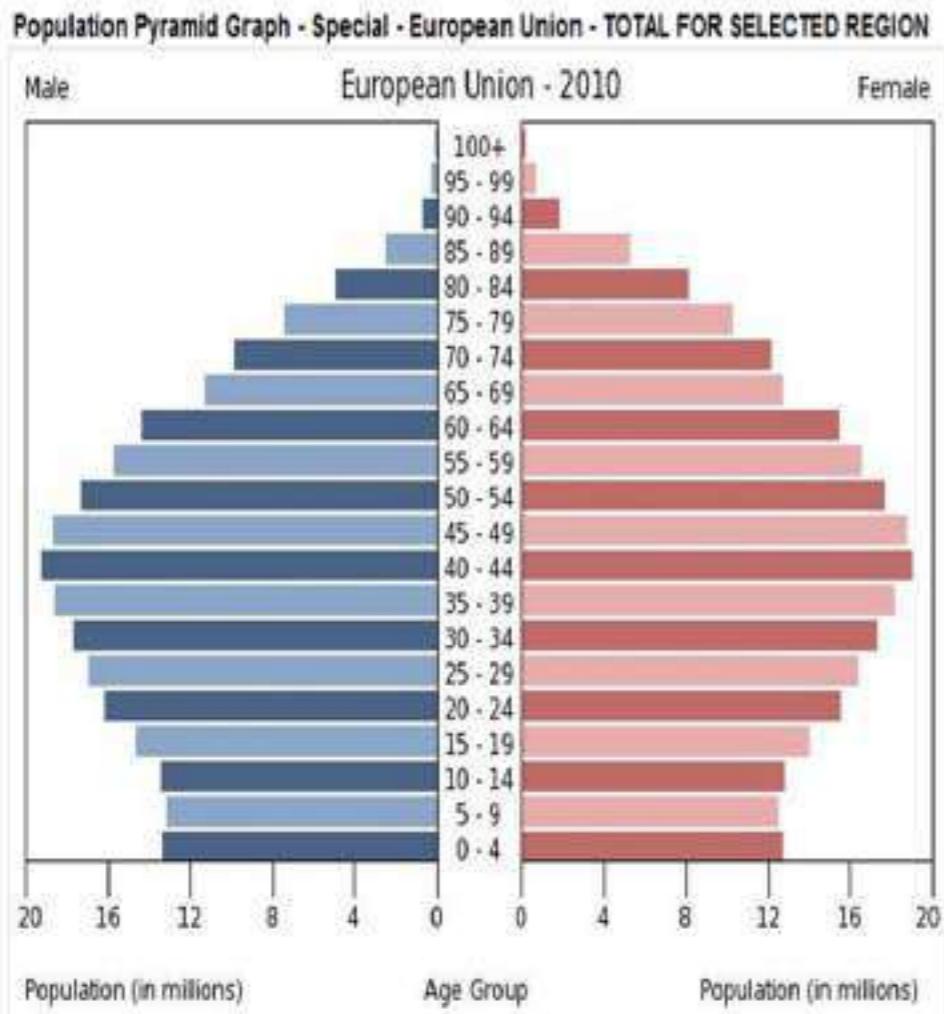
Compute and draw the histogram of `*x*`. The return value is a tuple `(*n*, *bins*, *patches*)` or `([*n0*, *n1*, ...], *bins*, [*patches0*, *patches1*, ...])` if the input contains multiple data.

Matplotlib example

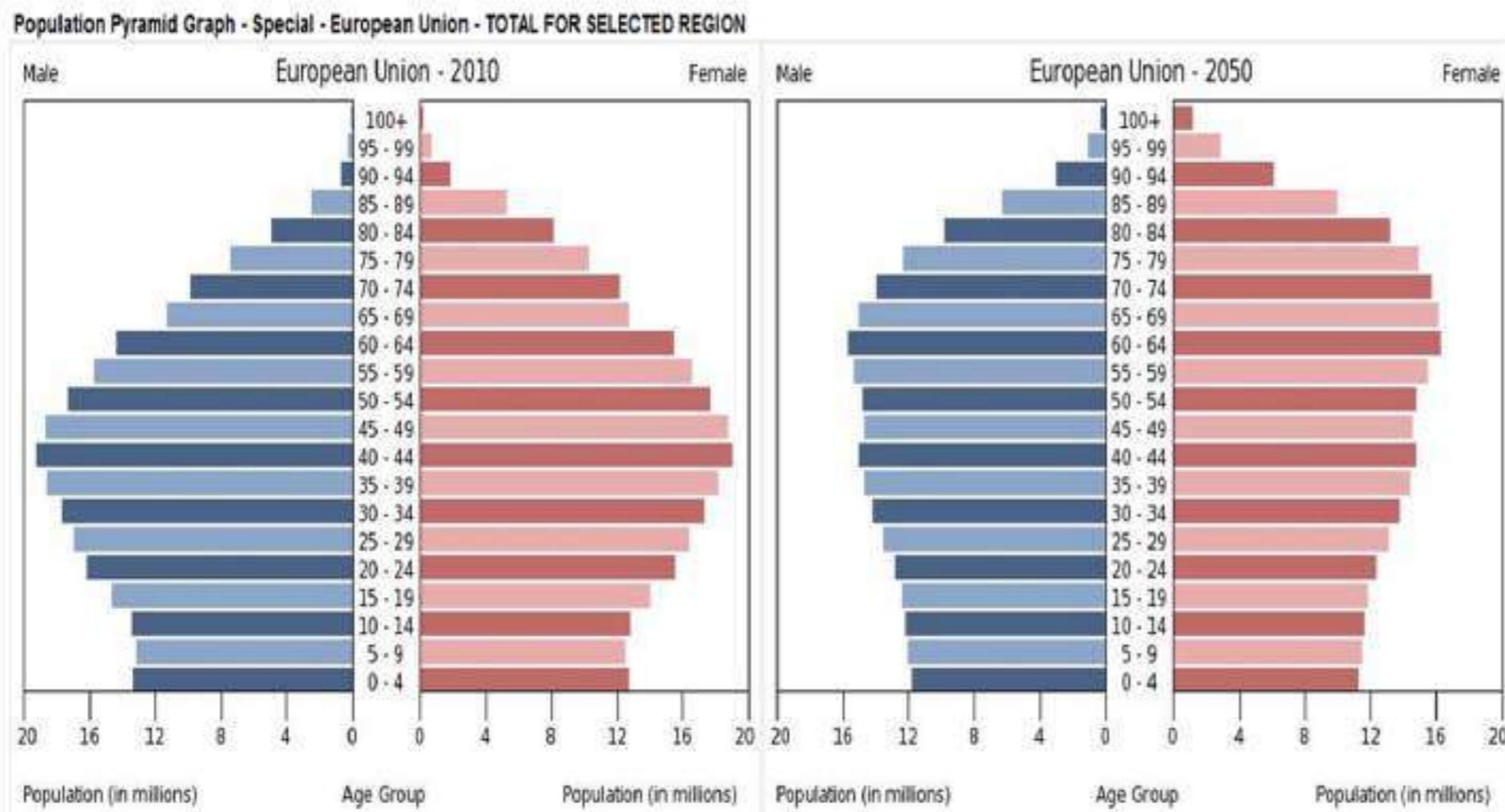
```
values = [0,0.6,1.4,1.6,2.2,2.5,2.6,3.2,3.5,3.9,4.2,6]
plt.hist(values, bins=3)
plt.show()
```



Population pyramid



Population pyramid



Let's practice!

INTERMEDIATE PYTHON

Customization

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

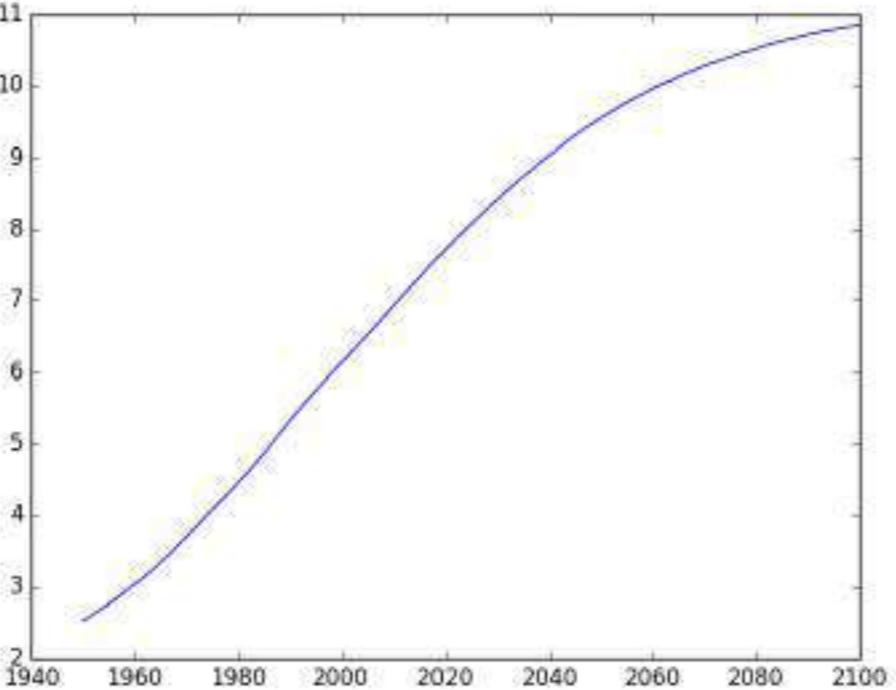
Data visualization

- 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

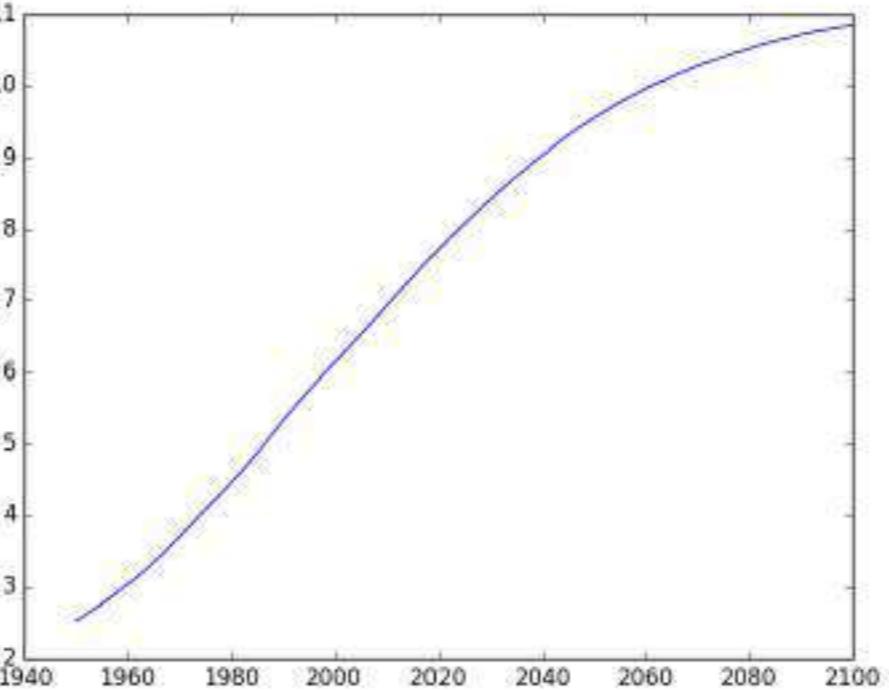
population.py

```
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

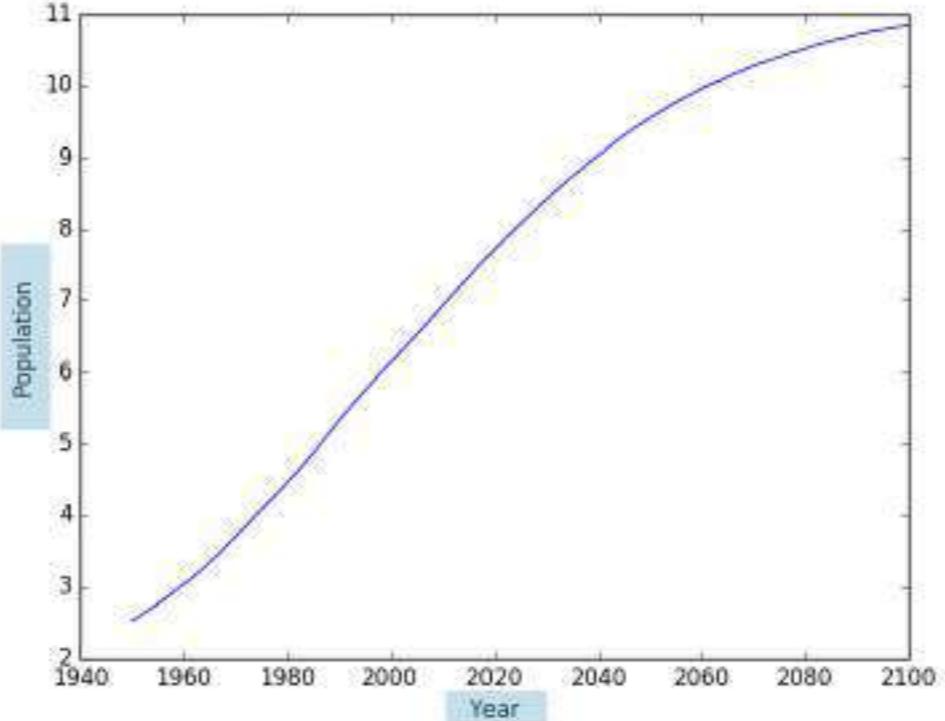
population.py

```
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')

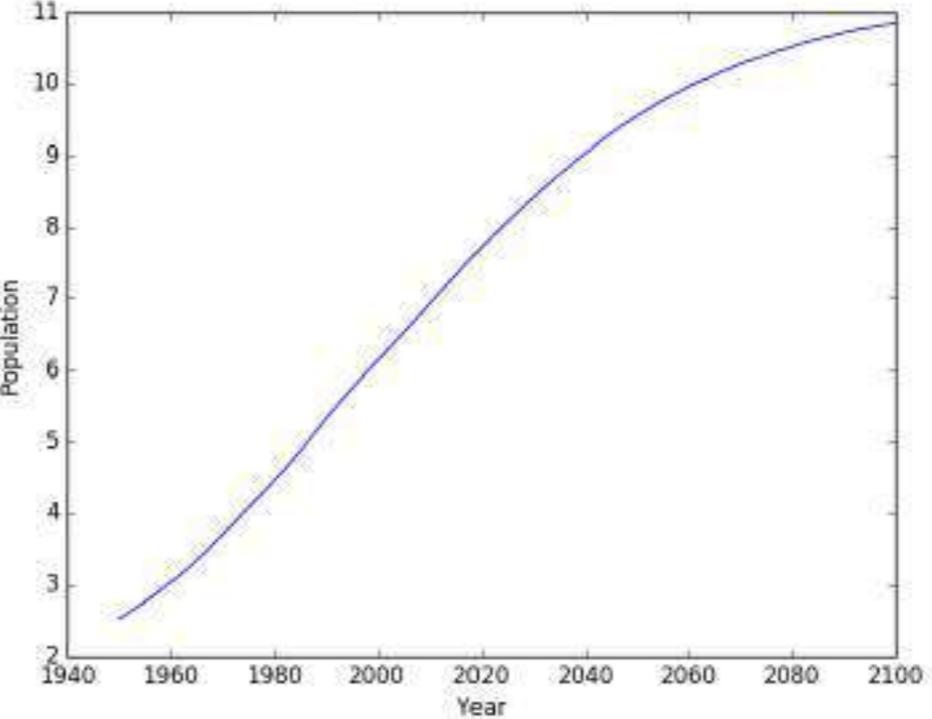
plt.show()
```



Title

population.py

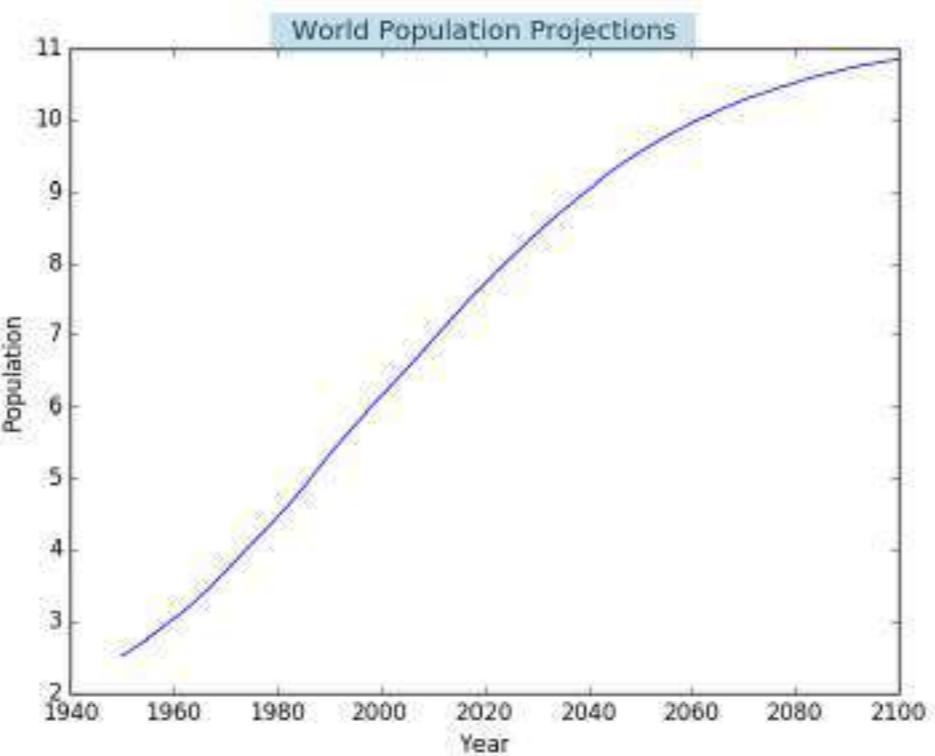
```
import matplotlib.pyplot as plt  
year = [1950, 1951, 1952, ..., 2100]  
pop = [2.538, 2.57, 2.62, ..., 10.85]  
  
plt.plot(year, pop)  
  
plt.xlabel('Year')  
plt.ylabel('Population')  
plt.title('World Population Projections')  
  
plt.show()
```



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

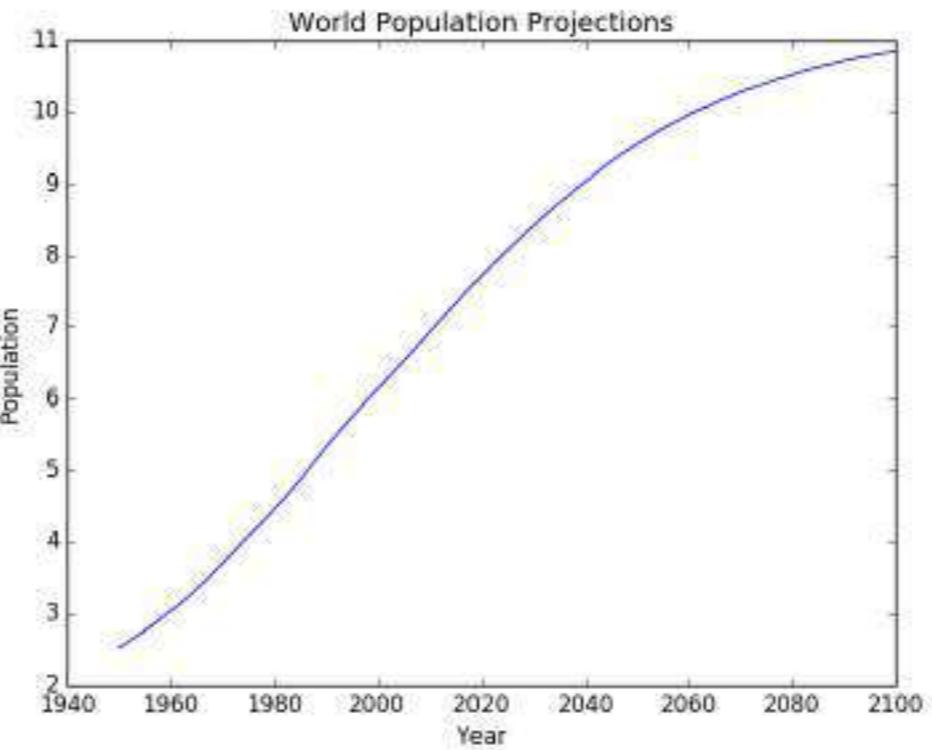
population.py

```
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

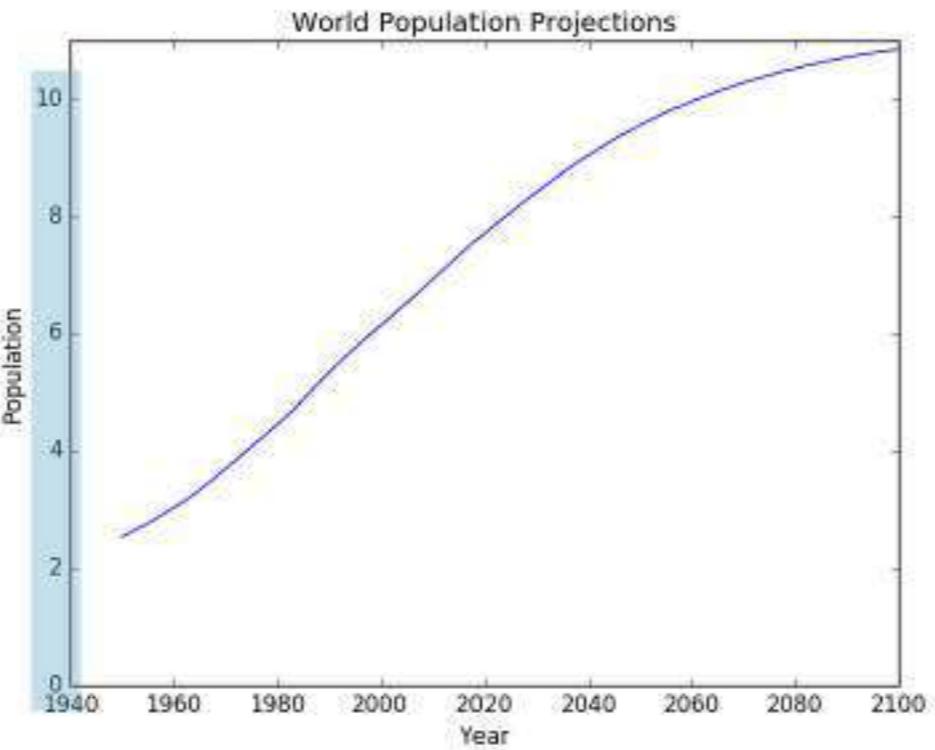
population.py

```
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)

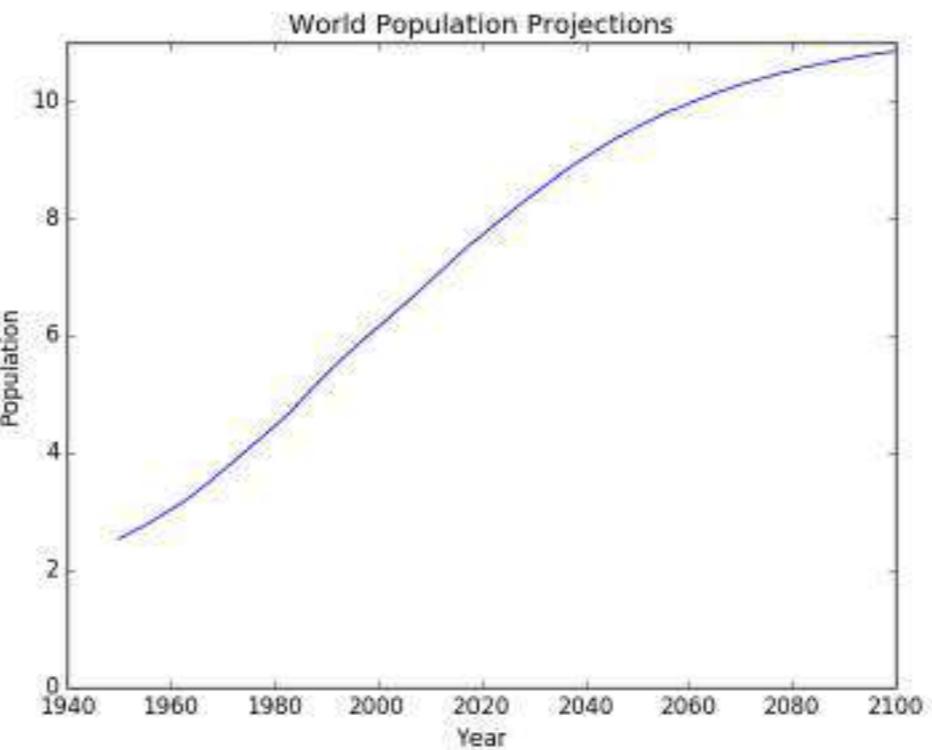
population.py

```
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)

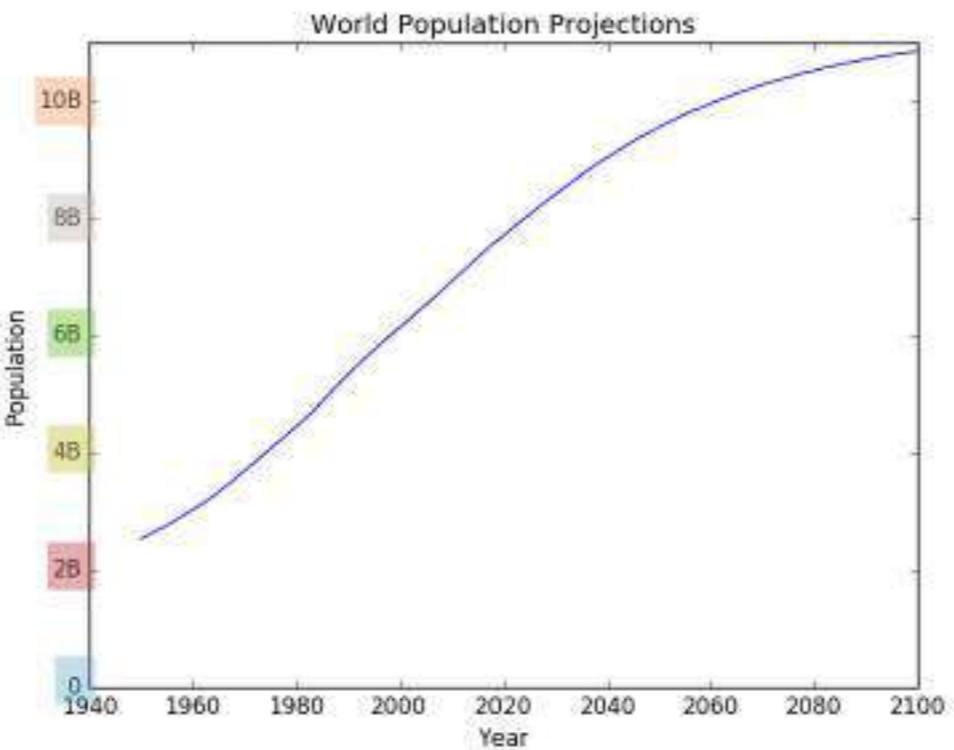
population.py

```
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

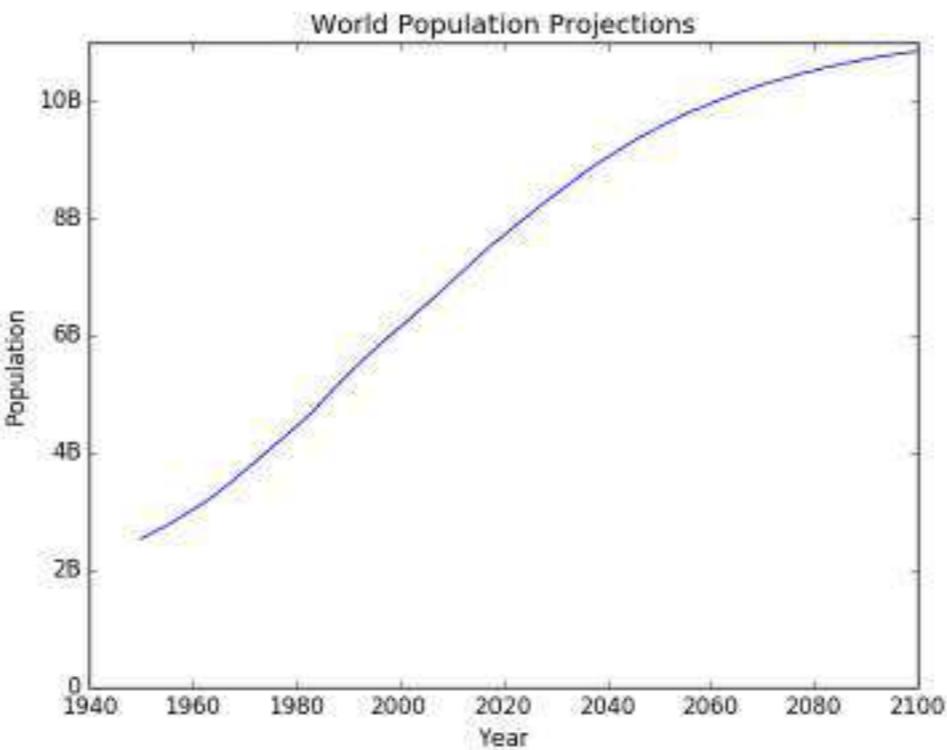
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

# 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

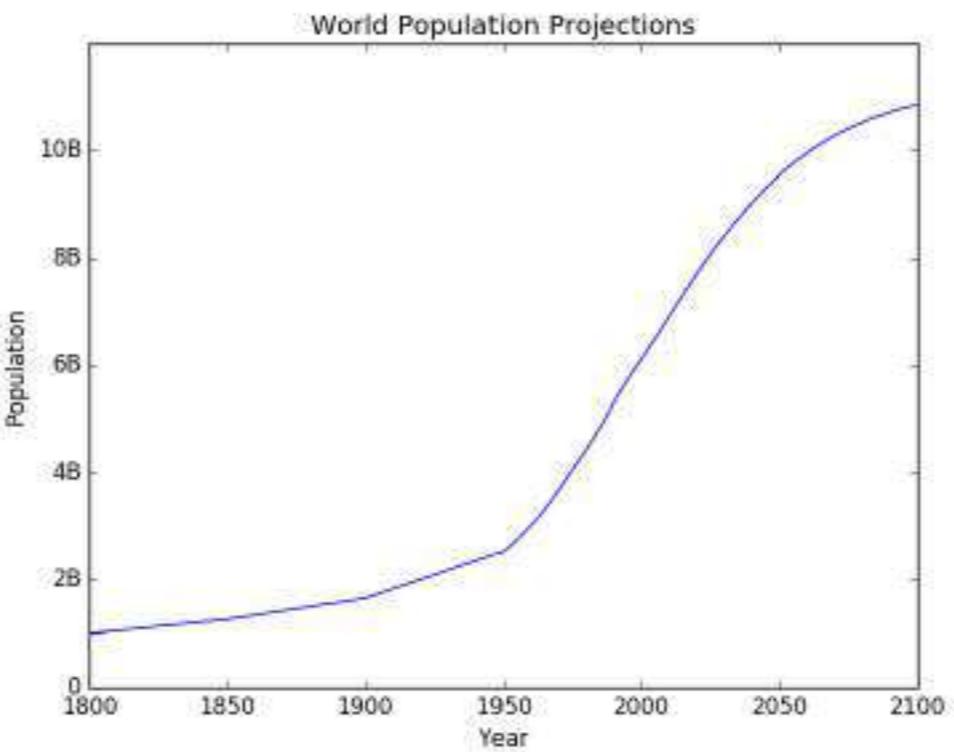
```
import matplotlib.pyplot as plt
year = [1950, 1951, 1952, ..., 2100]
pop = [2.538, 2.57, 2.62, ..., 10.85]

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

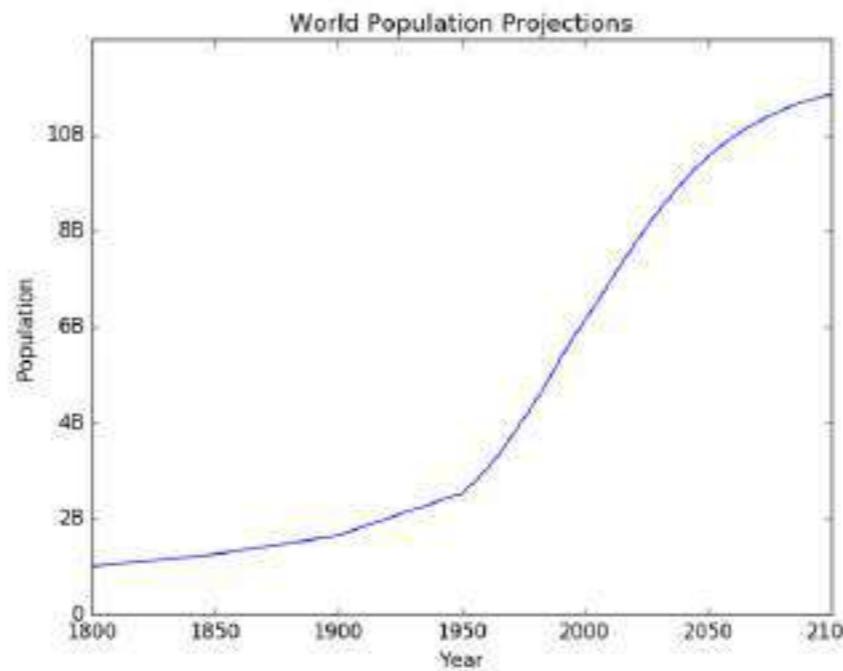
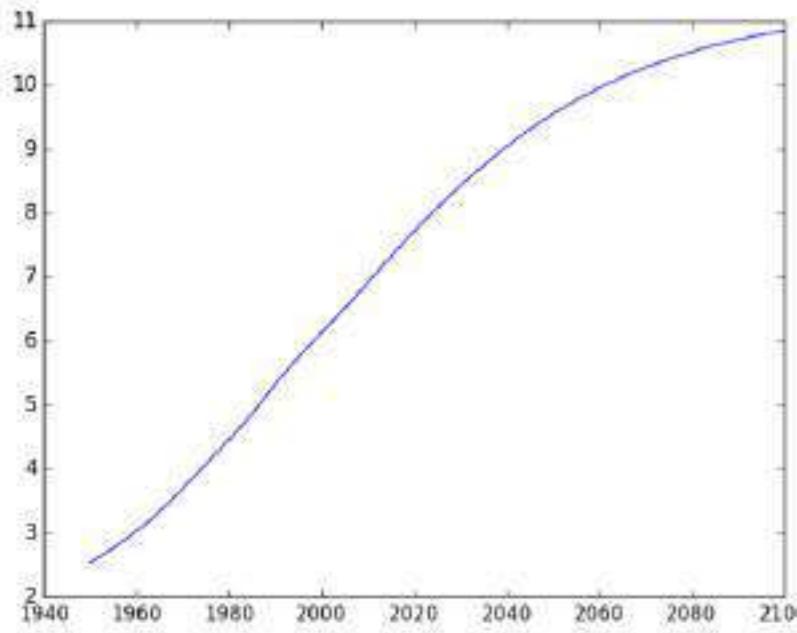
plt.plot(year, pop)

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

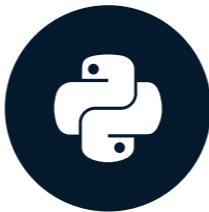


Let's practice!

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

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)
```

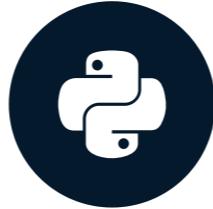
	country	capital	area	population	name_length
BR	Brazil	Brasilia	8.516	200.40	6
RU	Russia	Moscow	17.100	143.50	6
IN	India	New Delhi	3.286	1252.00	5
CH	China	Beijing	9.597	1357.00	5
SA	South Africa	Pretoria	1.221	52.98	12

Let's practice!

INTERMEDIATE PYTHON

Random Numbers

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

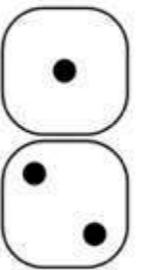


100 x





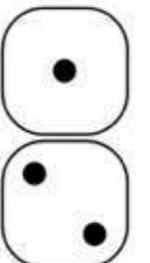
100 x



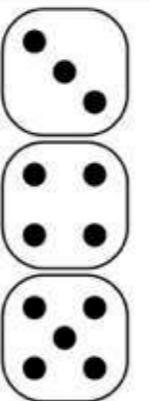
-1



100 x



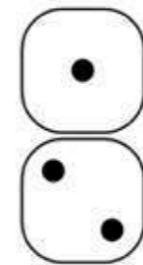
-1



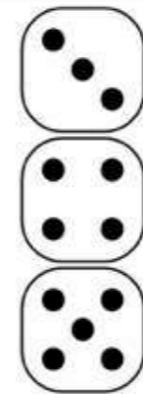
+1



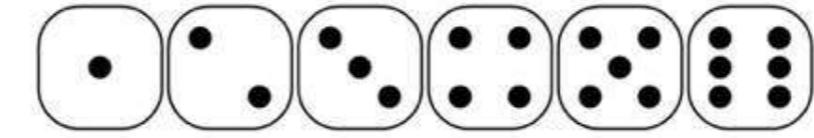
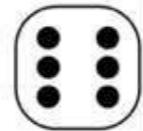
100 x



-1

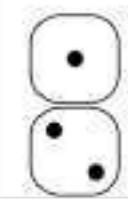


+1

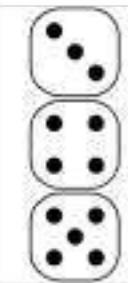


+1 +2 +3 +4 +5 +6

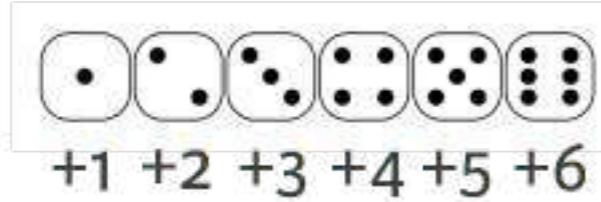
100 x



-1



+1



- 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

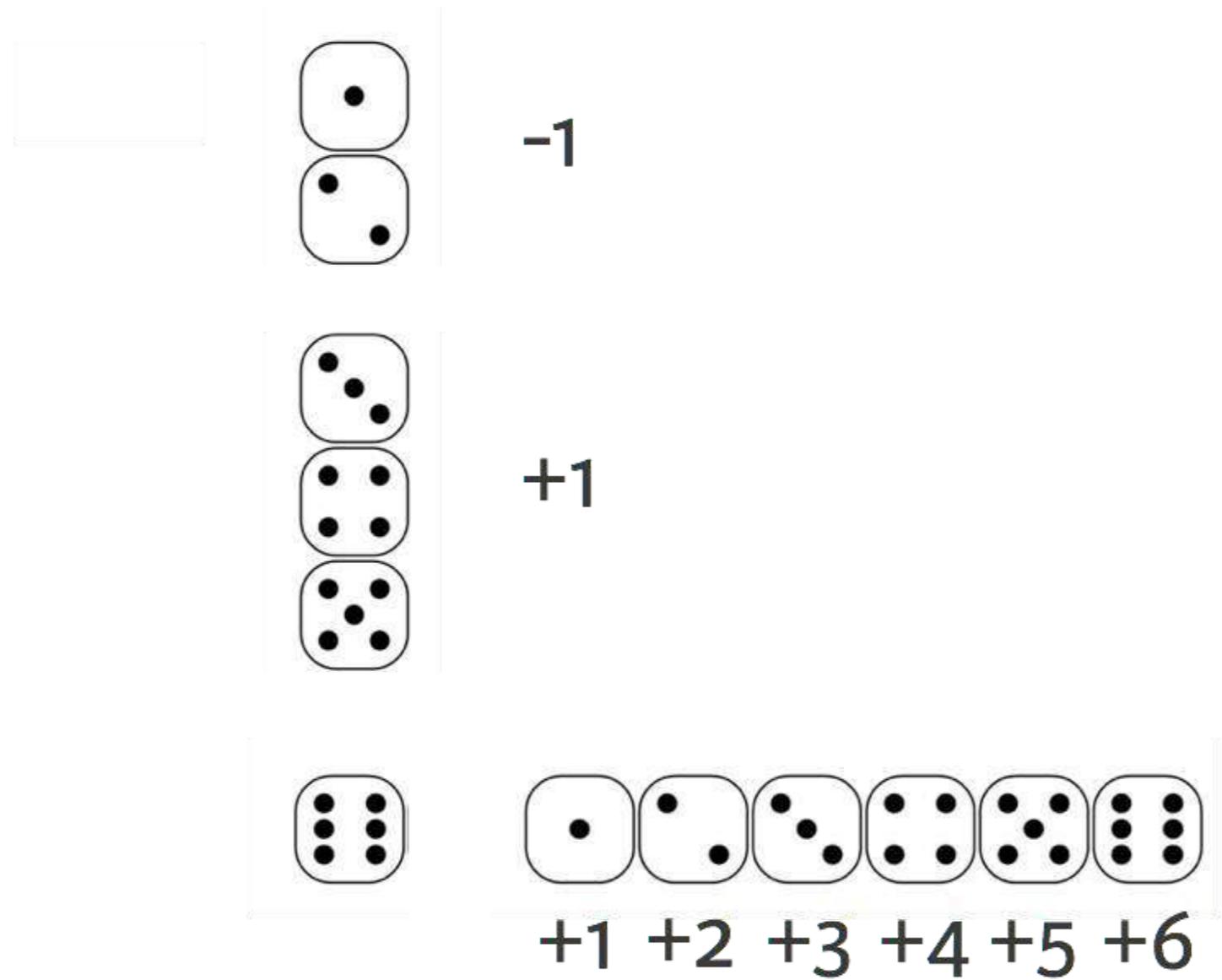
Random Walk

INTERMEDIATE PYTHON

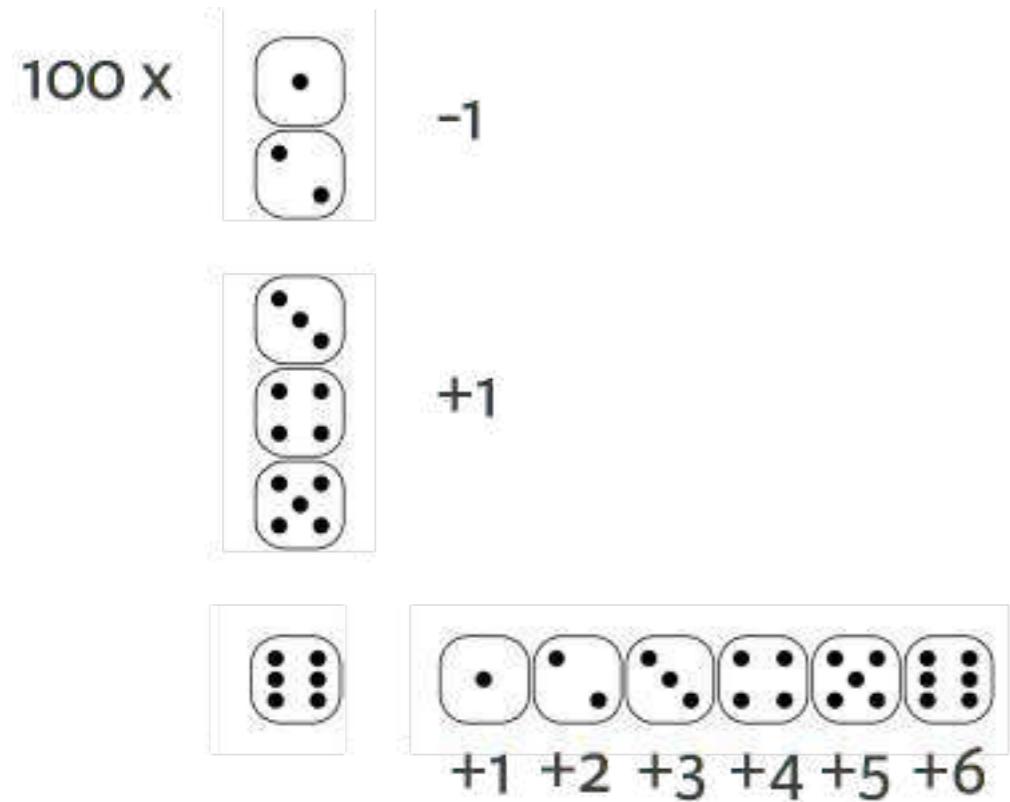


Hugo Bowne-Anderson
Data Scientist at DataCamp

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

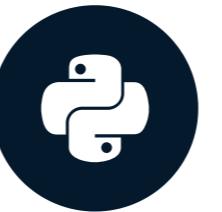
```
[0, 0, 1, 1, 1, 1, 1, 1, 2, 3, 3]
```

Let's practice!

INTERMEDIATE PYTHON

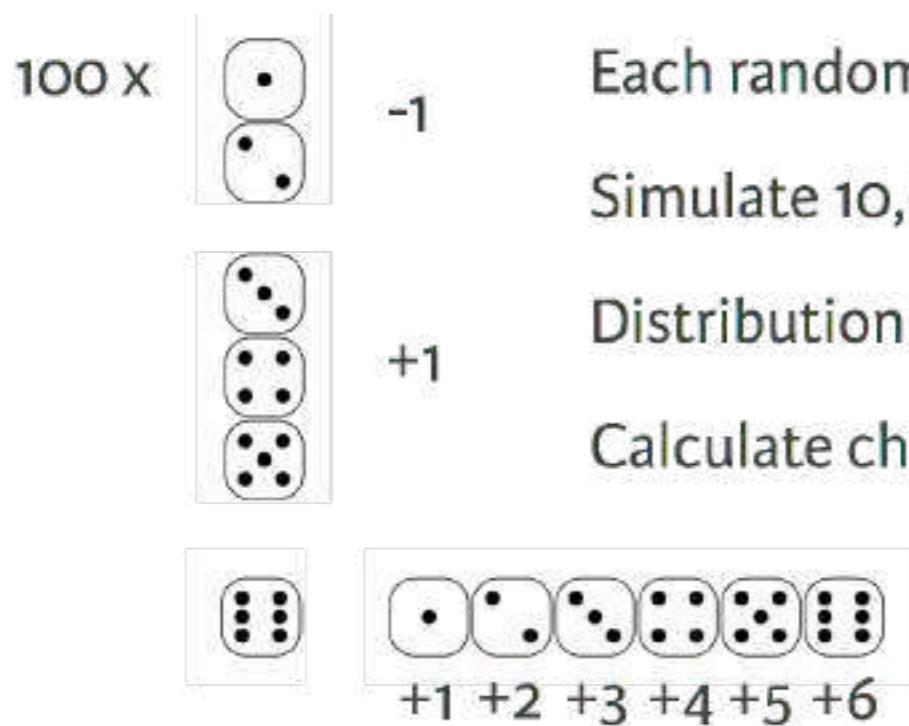
Distribution

INTERMEDIATE PYTHON



Hugo Bowne-Anderson
Data Scientist at DataCamp

Distribution



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)
```

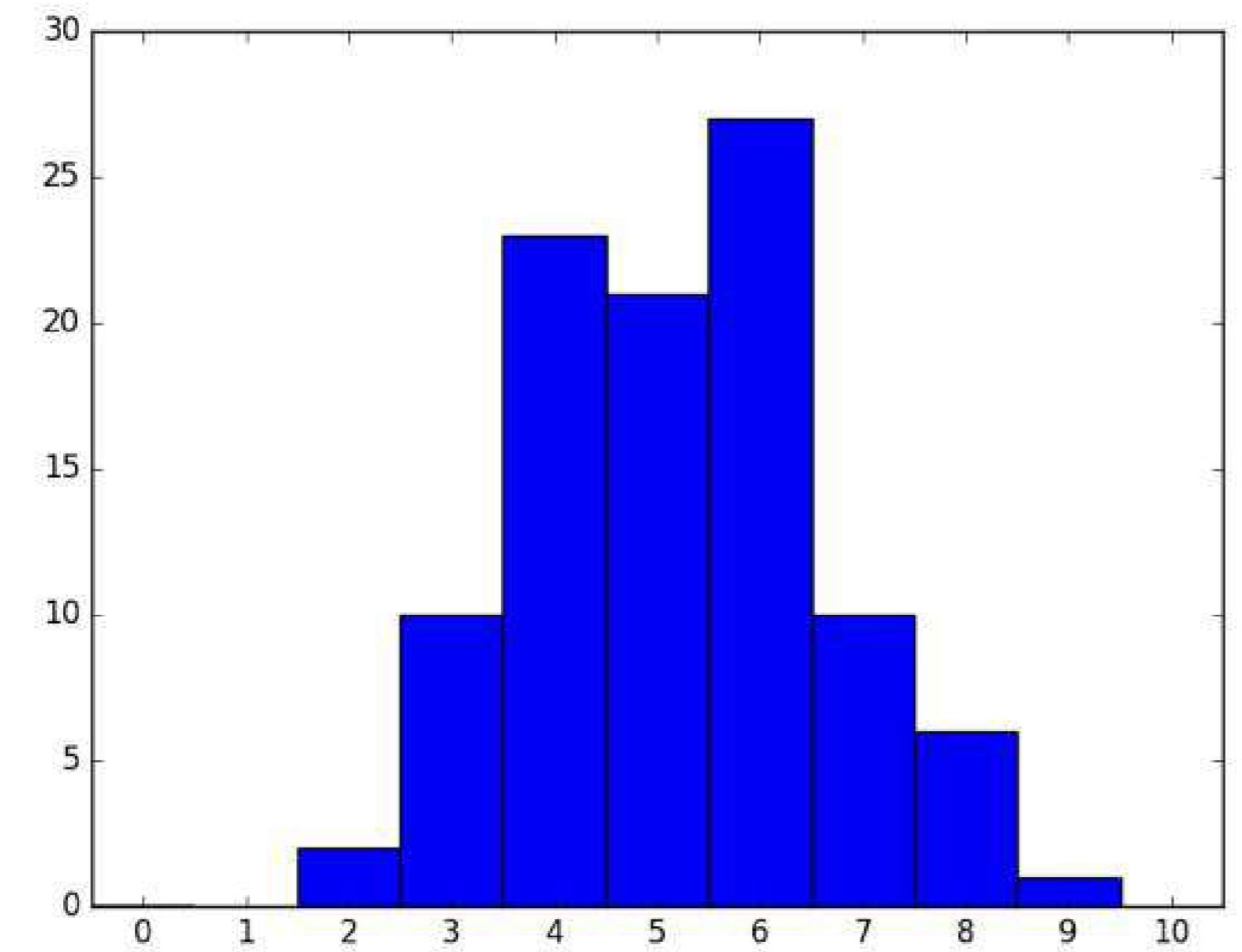
```
[3, 6, 4, 5, 4, 5, 3, 5, 4, 6, 6, 8, 6, 4, 7, 5, 7, 4, 3, 3, ..., 4]
```

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

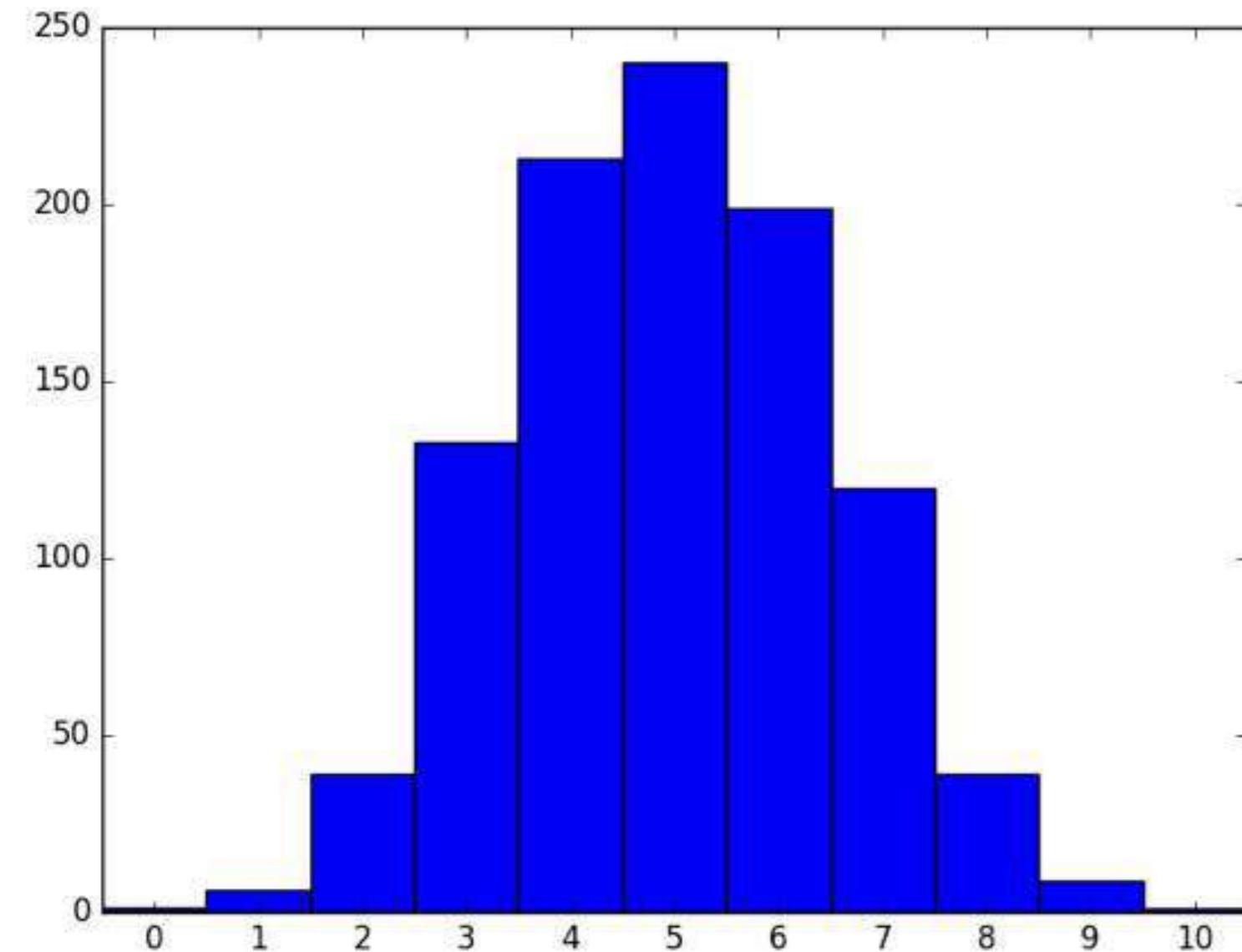


Histogram, 1,000 runs

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

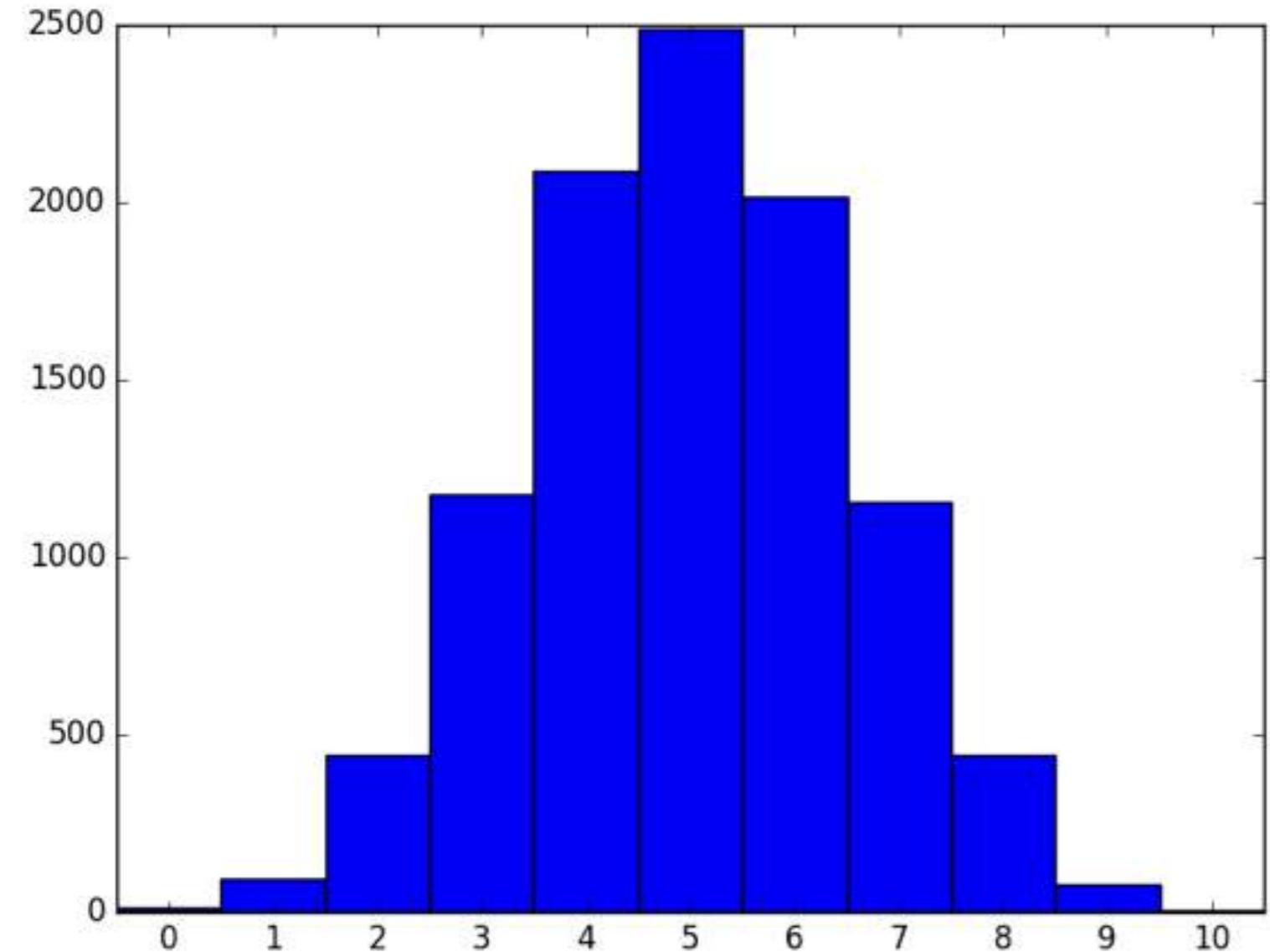


Histogram, 10,000 runs

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

User-defined functions

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson
Instructor

You'll learn:

- Define functions without parameters
- Define functions with one parameter
- Define functions that return a value
- Later: multiple arguments, multiple return values

Built-in functions

- str()

```
x = str(5)
```

```
print(x)
```

```
'5'
```

```
print(type(x))
```

```
<class 'str'>
```

Defining a function

```
def square():      # <- Function header  
    new_value = 4 ** 2      # <- Function body  
    print(new_value)  
square()
```

16

Function parameters

```
def square(value):  
    new_value = value ** 2  
    print(new_value)
```

```
square(4)
```

16

```
square(5)
```

25

Return values from functions

- Return a value from a function using return

```
def square(value):  
    new_value = value ** 2  
    return new_value  
  
num = square(4)  
  
print(num)
```

16

Docstrings

- Docstrings describe what your function does
- Serve as documentation for your function
- Placed in the immediate line after the function header
- In between triple double quotes """

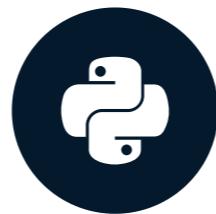
```
def square(value):  
    """Return the square of a value."""  
    new_value = value ** 2  
    return new_value
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Multiple Parameters and Return Values

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson
Instructor

Multiple function parameters

- Accept more than 1 parameter:

```
def raise_to_power(value1, value2):  
    """Raise value1 to the power of value2."""  
    new_value = value1 ** value2  
    return new_value
```

- Call function: # of arguments = # of parameters

```
result = raise_to_power(2, 3)  
  
print(result)
```

A quick jump into tuples

- Make functions return multiple values: Tuples!
- Tuples:
 - Like a list - can contain multiple values
 - Immutable - can't modify values!
 - Constructed using parentheses ()

```
even_nums = (2, 4, 6)
```

```
print(type(even_nums))
```

```
<class 'tuple'>
```

Unpacking tuples

- Unpack a tuple into several variables:

```
even_nums = (2, 4, 6)
```

```
a, b, c = even_nums
```

```
print(a)
```

```
2
```

```
print(b)
```

```
4
```

```
print(c)
```

```
6
```

Accessing tuple elements

- Access tuple elements like you do with lists:

```
even_nums = (2, 4, 6)
```

```
print(even_nums[1])
```

4

```
second_num = even_nums[1]
```

```
print(second_num)
```

4

- Uses zero-indexing

Returning multiple values

```
def raise_both(value1, value2):  
    """Raise value1 to the power of value2  
    and vice versa."""  
  
    new_value1 = value1 ** value2  
    new_value2 = value2 ** value1  
  
    new_tuple = (new_value1, new_value2)  
  
    return new_tuple
```

```
result = raise_both(2, 3)  
  
print(result)
```

```
(8, 9)
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Bringing it all together

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson
Instructor

You've learned:

- How to write functions
 - Accept multiple parameters
 - Return multiple values
- Up next: Functions for analyzing Twitter data

Basic ingredients of a function

- Function Header

```
def raise_both(value1, value2):
```

- Function body

```
    """Raise value1 to the power of value2  
and vice versa."""
```

```
    new_value1 = value1 ** value2
```

```
    new_value2 = value2 ** value1
```

```
    new_tuple = (new_value1, new_value2)
```

```
    return new_tuple
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Congratulations!

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson

Instructor

Next chapters:

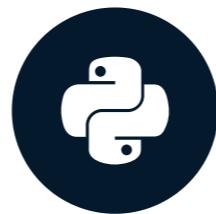
- Functions with default arguments
- Functions that accept an arbitrary number of parameters
- Nested functions
- Error-handling within functions
- More function use in data science!

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Scope and user-defined functions

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson
Instructor

Crash course on scope in functions

- Not all objects are accessible everywhere in a script
- Scope - part of the program where an object or name may be accessible
 - Global scope - defined in the main body of a script
 - Local scope - defined inside a function
 - Built-in scope - names in the pre-defined built-ins module

Global vs. local scope (1)

```
def square(value):
    """Returns the square of a number."""
    new_val = value ** 2
    return new_val

square(3)
```

9

new_val

```
<hr />-----
NameError                                Traceback (most recent call last)
<ipython-input-3-3cc6c6de5c5c> in <module>()
<hr />-> 1 new_value
NameError: name 'new_val' is not defined
```

Global vs. local scope (2)

```
new_val = 10

def square(value):
    """Returns the square of a number."""
    new_val = value ** 2
    return new_val

square(3)
```

9

new_val

10

Global vs. local scope (3)

```
new_val = 10

def square(value):
    """Returns the square of a number."""
    new_value2 = new_val ** 2
    return new_value2

square(3)
```

100

```
new_val = 20

square(3)
```

400

Global vs. local scope (4)

```
new_val = 10

def square(value):
    """Returns the square of a number."""
    global new_val
    new_val = new_val ** 2
    return new_val

square(3)
```

100

new_val

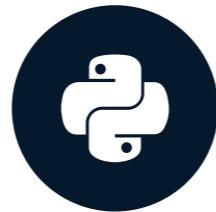
100

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Nested functions

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson

Instructor

Nested functions (1)

```
def outer( ... ):  
    """ ... """  
  
    x = ...  
  
  
    def inner( ... ):  
        """ ... """  
  
        y = x ** 2  
  
    return ...
```

Nested functions (2)

```
def mod2plus5(x1, x2, x3):  
    """Returns the remainder plus 5 of three values."  
  
    new_x1 = x1 % 2 + 5  
    new_x2 = x2 % 2 + 5  
    new_x3 = x3 % 2 + 5  
  
    return (new_x1, new_x2, new_x3)
```

Nested functions (3)

```
def mod2plus5(x1, x2, x3):
    """Returns the remainder plus 5 of three values."""

    def inner(x):
        """Returns the remainder plus 5 of a value."""
        return x % 2 + 5

    return (inner(x1), inner(x2), inner(x3))
```

```
print(mod2plus5(1, 2, 3))
```

```
(6, 5, 6)
```

Returning functions

```
def raise_val(n):
    """Return the inner function."""

    def inner(x):
        """Raise x to the power of n."""
        raised = x ** n
        return raised

    return inner
```

```
square = raise_val(2)
cube = raise_val(3)
print(square(2), cube(4))
```

4 64

Using nonlocal

```
def outer():
    """Prints the value of n."""
    n = 1

    def inner():
        nonlocal n
        n = 2
        print(n)

    inner()
    print(n)
```

```
outer()
```

```
2  
2
```

Scopes searched

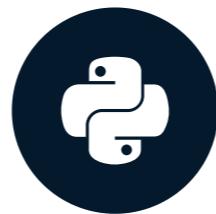
- Local scope
- Enclosing functions
- Global
- Built-in

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Default and flexible arguments

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson
Instructor

You'll learn:

- Writing functions with default arguments
- Using flexible arguments
 - Pass any number of arguments to a functions

Add a default argument

```
def power(number, pow=1):  
    """Raise number to the power of pow."""  
    new_value = number ** pow  
    return new_value  
  
power(9, 2)
```

81

power(9, 1)

9

power(9)

9

Flexible arguments: *args (1)

```
def add_all(*args):
    """Sum all values in *args together."""

    # Initialize sum
    sum_all = 0

    # Accumulate the sum
    for num in args:
        sum_all += num

    return sum_all
```

Flexible arguments: *args (2)

```
add_all(1)
```

```
1
```

```
add_all(1, 2)
```

```
3
```

```
add_all(5, 10, 15, 20)
```

```
50
```

Flexible arguments: **kwargs

```
print_all(name="Hugo Bowne-Anderson", employer="DataCamp")
```

```
name: Hugo Bowne-Anderson  
employer: DataCamp
```

Flexible arguments: **kwargs

```
def print_all(**kwargs):
    """Print out key-value pairs in **kwargs."""

    # Print out the key-value pairs
    for key, value in kwargs.items():
        print(key + ": " + value)
```

```
print_all(name="dumbledore", job="headmaster")
```

```
job: headmaster
name: dumbledore
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Bringing it all together

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson
Instructor

Next exercises:

- Generalized functions:
 - Count occurrences for any column
 - Count occurrences for an arbitrary number of columns

Add a default argument

```
def power(number, pow=1):  
    """Raise number to the power of pow."""  
    new_value = number ** pow  
    return new_value
```

```
power(9, 2)
```

```
81
```

```
power(9)
```

```
9
```

Flexible arguments: *args (1)

```
def add_all(*args):
    """Sum all values in *args together."""

    # Initialize sum
    sum_all = 0

    # Accumulate the sum
    for num in args:
        sum_all = sum_all + num

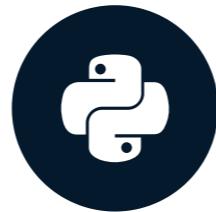
    return sum_all
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Lambda functions

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson

Instructor

Lambda functions

```
raise_to_power = Lambda x, y: x ** y
```

```
raise_to_power(2, 3)
```

8

Anonymous functions

- Function map takes two arguments: `map(func, seq)`
- `map()` applies the function to ALL elements in the sequence

```
nums = [48, 6, 9, 21, 1]

square_all = map(lambda num: num ** 2, nums)

print(square_all)
```

```
<map object at 0x103e065c0>
```

```
print(list(square_all))
```

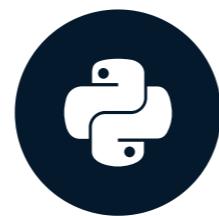
```
[2304, 36, 81, 441, 1]
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

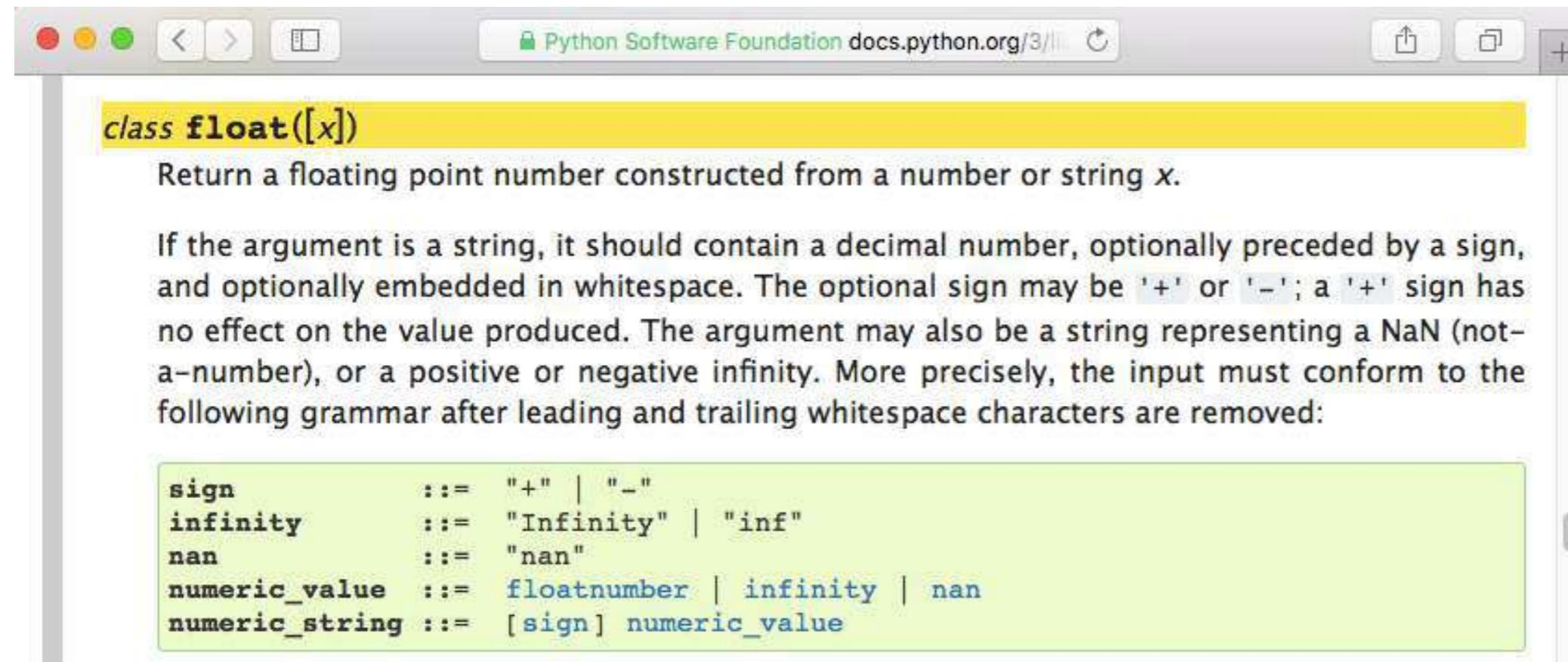
Introduction to error handling

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson
Instructor

The float() function



The screenshot shows a web browser window displaying the Python documentation for the `float()` function. The title bar reads "Python Software Foundation docs.python.org/3/". The main content area has a yellow header bar containing the function name `class float([x])`. Below this, a gray text block describes the function's purpose: "Return a floating point number constructed from a number or string `x`". A larger text block explains the grammar for the input string, stating: "If the argument is a string, it should contain a decimal number, optionally preceded by a sign, and optionally embedded in whitespace. The optional sign may be '+' or '-'; a '+' sign has no effect on the value produced. The argument may also be a string representing a NaN (not-a-number), or a positive or negative infinity. More precisely, the input must conform to the following grammar after leading and trailing whitespace characters are removed:". Below this explanation is a green code block showing the grammar rules:

```
sign      ::= "+" | "-"
infinity  ::= "Infinity" | "inf"
nan       ::= "nan"
numeric_value ::= floatnumber | infinity | nan
numeric_string ::= [sign] numeric_value
```

Passing an incorrect argument

```
float(2)
```

```
2.0
```

```
float('2.3')
```

```
2.3
```

```
float('hello')
```

```
<hr />-----
ValueError                                Traceback (most recent call last)
<ipython-input-3-d0ce8bcc8b2> in <module>()
<hr />-> 1 float('hi')
ValueError: could not convert string to float: 'hello'
```

Passing valid arguments

```
def sqrt(x):  
    """Returns the square root of a number."""  
    return x ** (0.5)  
sqrt(4)
```

2.0

sqrt(10)

3.1622776601683795

Passing invalid arguments

```
sqrt('hello')
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-4-cfb99c64761f> in <module>()
----> 1 sqrt('hello')
<ipython-input-1-939b1a60b413> in sqrt(x)
      1 def sqrt(x):
----> 2     return x**(0.5)
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'float'
```

Errors and exceptions

- Exceptions - caught during execution
- Catch exceptions with try-except clause
 - Runs the code following try
 - If there's an exception, run the code following except

Errors and exceptions

```
def sqrt(x):  
    """Returns the square root of a number."""  
    try:  
        return x ** 0.5  
    except:  
        print('x must be an int or float')  
  
sqrt(4)
```

2.0

sqrt(10.0)

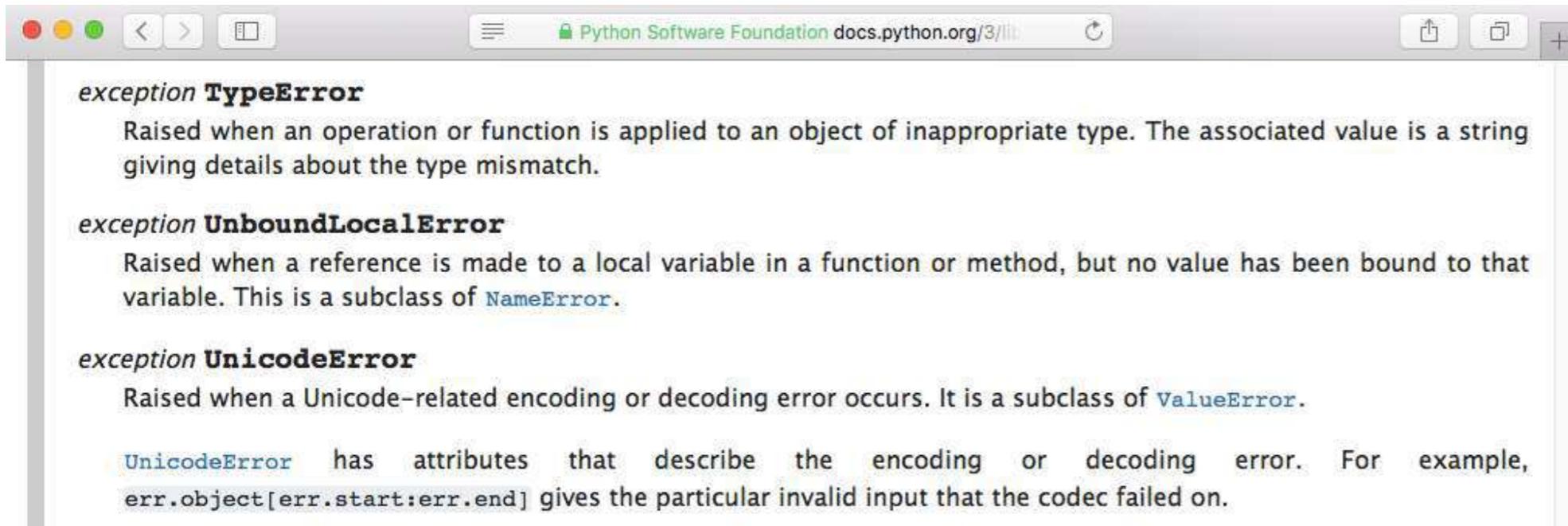
3.1622776601683795

sqrt('hi')

x must be an int or float

Errors and exceptions

```
def sqrt(x):
    """Returns the square root of a number."""
    try:
        return x ** 0.5
    except TypeError:
        print('x must be an int or float')
```



The screenshot shows a web browser window displaying the Python Software Foundation documentation at docs.python.org/3/. The page content is as follows:

- exception `TypeError`**
Raised when an operation or function is applied to an object of inappropriate type. The associated value is a string giving details about the type mismatch.
- exception `UnboundLocalError`**
Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable. This is a subclass of `NameError`.
- exception `UnicodeError`**
Raised when a Unicode-related encoding or decoding error occurs. It is a subclass of `ValueError`.
`UnicodeError` has attributes that describe the encoding or decoding error. For example, `err.object[err.start:err.end]` gives the particular invalid input that the codec failed on.

Errors and exceptions

```
sqrt(-9)
```

```
(1.8369701987210297e-16+3j)
```

```
def sqrt(x):
    """Returns the square root of a number."""
    if x < 0:
        raise ValueError('x must be non-negative')
    try:
        return x ** 0.5
    except TypeError:
        print('x must be an int or float')
```

Errors and exceptions

```
sqrt(-2)
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-2-4cf32322fa95> in <module>()
----> 1 sqrt(-2)
<ipython-input-1-a7b8126942e3> in sqrt(x)
      1 def sqrt(x):
      2     if x < 0:
----> 3         raise ValueError('x must be non-negative')
      4     try:
      5         return x**(0.5)
ValueError: x must be non-negative
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Bringing it all together

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson
Instructor

Errors and exceptions

```
def sqrt(x):  
    try:  
        return x ** 0.5  
    except:  
        print('x must be an int or float')
```

```
sqrt(4)
```

```
2.0
```

```
sqrt('hi')
```

```
x must be an int or float
```

Errors and exceptions

```
def sqrt(x):
    if x < 0:
        raise ValueError('x must be non-negative')
    try:
        return x ** 0.5
    except TypeError:
        print('x must be an int or float')
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Congratulations!

PYTHON DATA SCIENCE TOOLBOX (PART 1)



Hugo Bowne-Anderson

Instructor

What you've learned:

- Write functions that accept single and multiple arguments
- Write functions that return one or many values
- Use default, flexible, and keyword arguments
- Global and local scope in functions
- Write lambda functions
- Handle errors

There's more to learn!

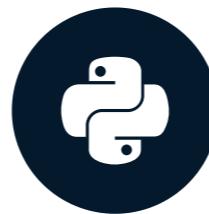
- Create lists with list comprehensions
- Iterators - you've seen them before!
- Case studies to apply these techniques to Data Science

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 1)

Introduction to iterators

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

Iterating with a for loop

- We can iterate over a list using a for loop

```
employees = ['Nick', 'Lore', 'Hugo']
for employee in employees:
    print(employee)
```

```
Nick
Lore
Hugo
```

Iterating with a for loop

- We can iterate over a string using a for loop

```
for letter in 'DataCamp':  
    print(letter)
```

```
D  
a  
t  
a  
c  
a  
m  
p
```

Iterating with a for loop

- We can iterate over a range object using a for loop

```
for i in range(4):  
    print(i)
```

```
0  
1  
2  
3
```

Iterators vs. iterables

- Iterable
 - Examples: lists, strings, dictionaries, file connections
 - An object with an associated `iter()` method
 - Applying `iter()` to an iterable creates an iterator
- Iterator
 - Produces next value with `next()`

Iterating over iterables: next()

```
word = 'Da'  
it = iter(word)  
next(it)
```

```
'D'
```

```
next(it)
```

```
'a'
```

```
next(it)
```

```
StopIteration          Traceback (most recent call last)  
<ipython-input-11-2cdb14c0d4d6> in <module>()  
-> 1 next(it)  
StopIteration:
```

Iterating at once with *

```
word = 'Data'  
it = iter(word)  
print(*it)
```

```
Data
```

```
print(*it)
```

- No more values to go through!

Iterating over dictionaries

```
pythonistas = {'hugo': 'bowne-anderson', 'francis': 'castro'}  
for key, value in pythonistas.items():  
    print(key, value)
```

```
francis castro  
hugo bowne-anderson
```

Iterating over file connections

```
file = open('file.txt')  
it = iter(file)  
print(next(it))
```

This is the first line.

```
print(next(it))
```

This is the second line.

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Playing with iterators

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

Using enumerate()

```
avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']
e = enumerate(avengers)
print(type(e))
```

```
<class 'enumerate'>
```

```
e_list = list(e)
print(e_list)
```

```
[(0, 'hawkeye'), (1, 'iron man'), (2, 'thor'), (3, 'quicksilver')]
```

enumerate() and unpack

```
avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']
for index, value in enumerate(avengers):
    print(index, value)
```

```
0 hawkeye
1 iron man
2 thor
3 quicksilver
```

```
for index, value in enumerate(avengers, start=10):
    print(index, value)
```

```
10 hawkeye
11 iron man
12 thor
13 quicksilver
```

Using zip()

```
avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']
names = ['barton', 'stark', 'odinson', 'maximoff']
z = zip(avengers, names)
print(type(z))
```

```
<class 'zip'>
```

```
z_list = list(z)
print(z_list)
```

```
[('hawkeye', 'barton'), ('iron man', 'stark'),
 ('thor', 'odinson'), ('quicksilver', 'maximoff')]
```

zip() and unpack

```
avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']
names = ['barton', 'stark', 'odinson', 'maximoff']
for z1, z2 in zip(avengers, names):
    print(z1, z2)
```

```
hawkeye barton
iron man stark
thor odinson
quicksilver maximoff
```

Print zip with *

```
avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']
names = ['barton', 'stark', 'odinson', 'maximoff']
z = zip(avengers, names)
print(*z)
```

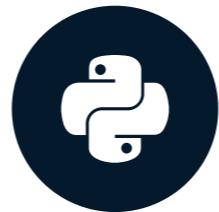
```
('hawkeye', 'barton') ('iron man', 'stark')
('thor', 'odinson') ('quicksilver', 'maximoff')
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Using iterators to load large files into memory

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

Loading data in chunks

- There can be too much data to hold in memory
- Solution: load data in chunks!
- Pandas function: `read_csv()`
 - Specify the chunk: `chunk_size`

Iterating over data

```
import pandas as pd  
  
result = []  
  
for chunk in pd.read_csv('data.csv', chunksize=1000):  
    result.append(sum(chunk['x']))  
  
total = sum(result)  
  
print(total)
```

4252532

Iterating over data

```
import pandas as pd  
  
total = 0  
  
for chunk in pd.read_csv('data.csv', chunksize=1000):  
    total += sum(chunk['x'])  
  
print(total)
```

4252532

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Congratulations!

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

What's next?

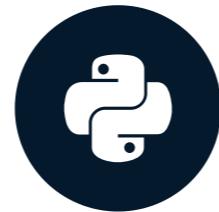
- List comprehensions and generators
- List comprehensions:
 - Create lists from other lists, DataFrame columns, etc.
 - Single line of code
 - More efficient than using a for loop

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

List comprehensions

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

Populate a list with a for loop

```
nums = [12, 8, 21, 3, 16]
new_nums = []
for num in nums:
    new_nums.append(num + 1)
print(new_nums)
```

```
[13, 9, 22, 4, 17]
```

A list comprehension

```
nums = [12, 8, 21, 3, 16]  
new_nums = [num + 1 for num in nums]  
print(new_nums)
```

```
[13, 9, 22, 4, 17]
```

For loop and list comprehension syntax

```
new_nums = [num + 1 for num in nums]
```

```
for num in nums:  
    new_nums.append(num + 1)  
print(new_nums)
```

```
[13, 9, 22, 4, 17]
```

List comprehension with range()

```
result = [num for num in range(11)]  
print(result)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

List comprehensions

- Collapse for loops for building lists into a single line
- Components
 - Iterable
 - Iterator variable (represent members of iterable)
 - Output expression

Nested loops (1)

```
pairs_1 = []
for num1 in range(0, 2):
    for num2 in range(6, 8):
        pairs_1.append(num1, num2)
print(pairs_1)
```

```
[(0, 6), (0, 7), (1, 6), (1, 7)]
```

- How to do this with a list comprehension?

Nested loops (2)

```
pairs_2 = [(num1, num2) for num1 in range(0, 2) for num2 in range(6, 8)]  
print(pairs_2)
```

```
[(0, 6), (0, 7), (1, 6), (1, 7)]
```

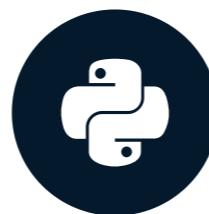
- Tradeoff: readability

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Advanced comprehensions

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

Conditionals in comprehensions

- Conditionals on the iterable

```
[num ** 2 for num in range(10) if num % 2 == 0]
```

```
[0, 4, 16, 36, 64]
```

- Python documentation on the `%` operator: The `%` (modulo) operator yields the remainder from the division of the first argument by the second.

```
5 % 2
```

```
1
```

```
6 % 2
```

```
0
```

Conditionals in comprehensions

- Conditionals on the output expression

```
[num ** 2 if num % 2 == 0 else 0 for num in range(10)]
```

```
[0, 0, 4, 0, 16, 0, 36, 0, 64, 0]
```

Dict comprehensions

- Create dictionaries
- Use curly braces {} instead of brackets []

```
pos_neg = {num: -num for num in range(9)}  
print(pos_neg)
```

```
{0: 0, 1: -1, 2: -2, 3: -3, 4: -4, 5: -5, 6: -6, 7: -7, 8: -8}
```

```
print(type(pos_neg))
```

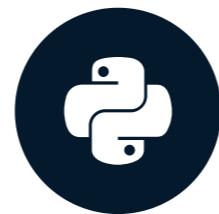
```
<class 'dict'>
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Introduction to generator expressions

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

Generator expressions

- Recall list comprehension

```
[2 * num for num in range(10)]
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

- Use `()` instead of `[]`

```
(2 * num for num in range(10))
```

```
<generator object <genexpr> at 0x1046bf888>
```

List comprehensions vs. generators

- List comprehension - returns a list
- Generators - returns a generator object
- Both can be iterated over

Printing values from generators (1)

```
result = (num for num in range(6))
for num in result:
    print(num)
```

```
0
1
2
3
4
5
```

```
result = (num for num in range(6))
print(list(result))
```

```
[0, 1, 2, 3, 4, 5]
```

Printing values from generators (2)

```
result = (num for num in range(6))
```

- Lazy evaluation

```
print(next(result))
```

```
0
```

```
print(next(result))
```

```
1
```

```
print(next(result))
```

```
2
```

```
print(next(result))
```

```
3
```

```
print(next(result))
```

```
4
```

Generators vs list comprehensions

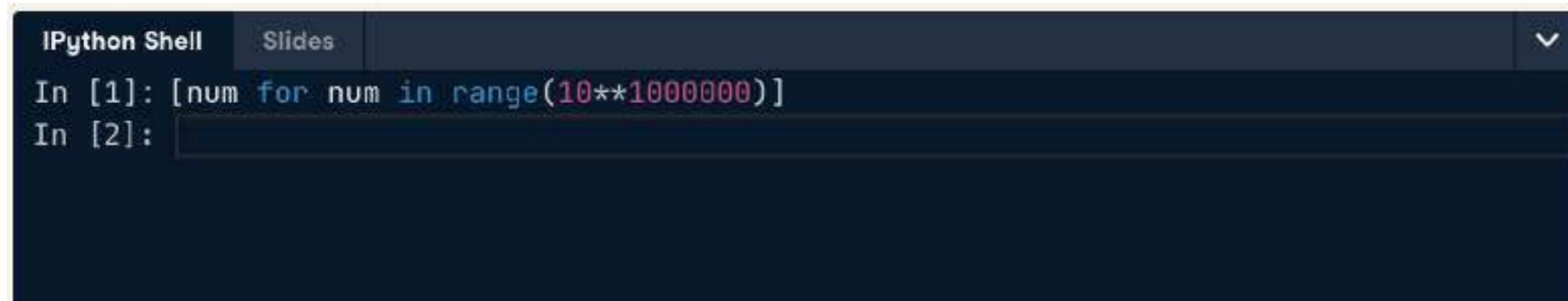


The screenshot shows an IPython Shell interface. The top navigation bar has tabs for "IPython Shell" and "Slides". The main area displays two code cells:

```
In [1]: [num for num in range(10**1000000)]
```

The second cell, In [2], is currently active and empty, indicated by a cursor in the text input field.

Generators vs list comprehensions



The screenshot shows an IPython Shell interface. The tabs at the top are "IPython Shell" and "Slides". In the IPython Shell tab, there are two code cells labeled "In [1]" and "In [2]". The first cell contains the Python code: `[num for num in range(10**1000000)]`. The second cell is currently active and has a small selection box around its beginning.



Generators vs list comprehensions



The screenshot shows an IPython Shell interface. The top navigation bar has tabs for "IPython Shell" and "Slides". The main area displays two code cells:

```
In [1]: (num for num in range(10**1000000))
Out[1]: <generator object <genexpr> at 0x7f8c2447a7d8>
```

In [2]: |

The code in In [1] creates a generator object that iterates over a range from 0 to 1,000,000. The output is a generator object at memory address 0x7f8c2447a7d8.

Conditionals in generator expressions

```
even_nums = (num for num in range(10) if num % 2 == 0)
print(list(even_nums))
```

```
[0, 2, 4, 6, 8]
```

Generator functions

- Produces generator objects when called
- Defined like a regular function - `def`
- Yields a sequence of values instead of returning a single value
- Generates a value with `yield` keyword

Build a generator function

- sequence.py

```
def num_sequence(n):
    """Generate values from 0 to n."""
    i = 0
    while i < n:
        yield i
        i += 1
```

Use a generator function

```
result = num_sequence(5)  
print(type(result))
```

```
<class 'generator'>
```

```
for item in result:  
    print(item)
```

```
0  
1  
2  
3  
4
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Wrapping up comprehensions and generators.

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

Re-cap: list comprehensions

- Basic

[output expression for iterator variable in iterable]

- Advanced

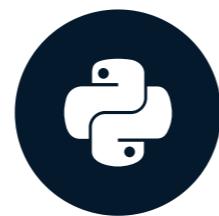
[output expression +
conditional on output for iterator variable in iterable +
conditional on iterable]

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Welcome to the case study!

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

World bank data

- Data on world economies for over half a century
- Indicators
 - Population
 - Electricity consumption
 - CO2 emissions
 - Literacy rates
 - Unemployment
 - Mortality rates

Using zip()

```
avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']
names = ['barton', 'stark', 'odinson', 'maximoff']
z = zip(avengers, names)
print(type(z))
```

```
<class 'zip'>
```

```
print(list(z))
```

```
[('hawkeye', 'barton'), ('iron man', 'stark'),
 ('thor', 'odinson'), ('quicksilver', 'maximoff')]
```

Defining a function

- raise.py

```
def raise_both(value1, value2):  
    """Raise value1 to the power of value2  
    and vice versa."""  
    new_value1 = value1 ** value2  
    new_value2 = value2 ** value1  
    new_tuple = (new_value1, new_value2)  
    return new_tuple
```

Re-cap: list comprehensions

Basic

```
[output expression for iterator variable in iterable]
```

Advanced

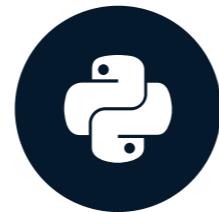
```
[output expression +  
conditional on output for iterator variable in iterable +  
conditional on iterable]
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Using Python generators for streaming data

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

Generators for the large data limit

- Use a generator to load a file line by line
- Works on streaming data!
- Read and process the file until all lines are exhausted

Build a generator function

- sequence.py

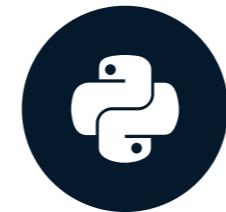
```
def num_sequence(n):
    """Generate values from 0 to n."""
    i = 0
    while i < n:
        yield i
        i += 1
```

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Using pandas' read_csv iterator for streaming data

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

Reading files in chunks

- Up next:
 - `read_csv()` function and `chunk_size` argument
 - Look at specific indicators in specific countries
 - Write a function to generalize tasks

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Final thoughts

PYTHON DATA SCIENCE TOOLBOX (PART 2)



Hugo Bowne-Anderson
Data Scientist at DataCamp

You've applied your skills in:

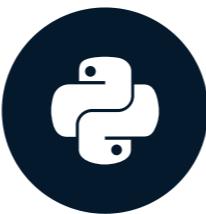
- User-defined functions
- Iterators
- List comprehensions
- Generators

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Introducing DataFrames

DATA MANIPULATION WITH PANDAS



Richie Cotton

Data Evangelist 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

Data Evangelist 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

Data Evangelist 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"]]
```

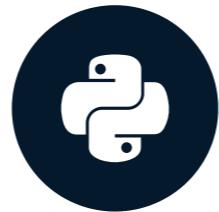
```
   name  height_cm      bmi
4    Max        59  83.309394
0    Bella       56  76.530612
3   Cooper       49  70.803832
5   Stella       18  61.728395
```

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)
```

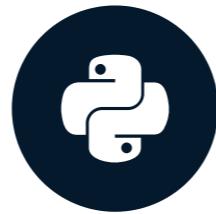
```
Labrador          0.250
Chow Chow         0.250
Schnauzer        0.125
St. Bernard      0.125
Poodle           0.125
Chihuahua        0.125
Name: breed, dtype: float64
```

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

Data Evangelist 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

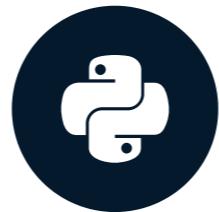
	date	city	country	avg_temp_c
0	2000-01-01	Abidjan	Côte D'Ivoire	27.293
1	2000-02-01	Abidjan	Côte D'Ivoire	27.685
2	2000-03-01	Abidjan	Côte D'Ivoire	29.061
3	2000-04-01	Abidjan	Côte D'Ivoire	28.162
4	2000-05-01	Abidjan	Côte D'Ivoire	27.547

Let's practice!

DATA MANIPULATION WITH PANDAS

Slicing and subsetting with .loc and .iloc

DATA MANIPULATION WITH PANDAS



Richie Cotton

Data Evangelist 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

Data Evangelist 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")
```

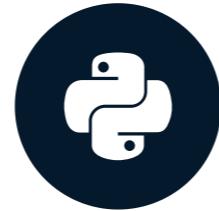
```
breed
Beagle      36.362667
Boxer       59.358667
Chihuahua   19.561250
Chow Chow    52.413333
Dachshund   20.236667
Labrador     55.875000
Poodle       51.637750
St. Bernard  66.654300
dtype: float64
```

Let's practice!

DATA MANIPULATION WITH PANDAS

Visualizing your data

DATA MANIPULATION WITH PANDAS



Maggie Matsui

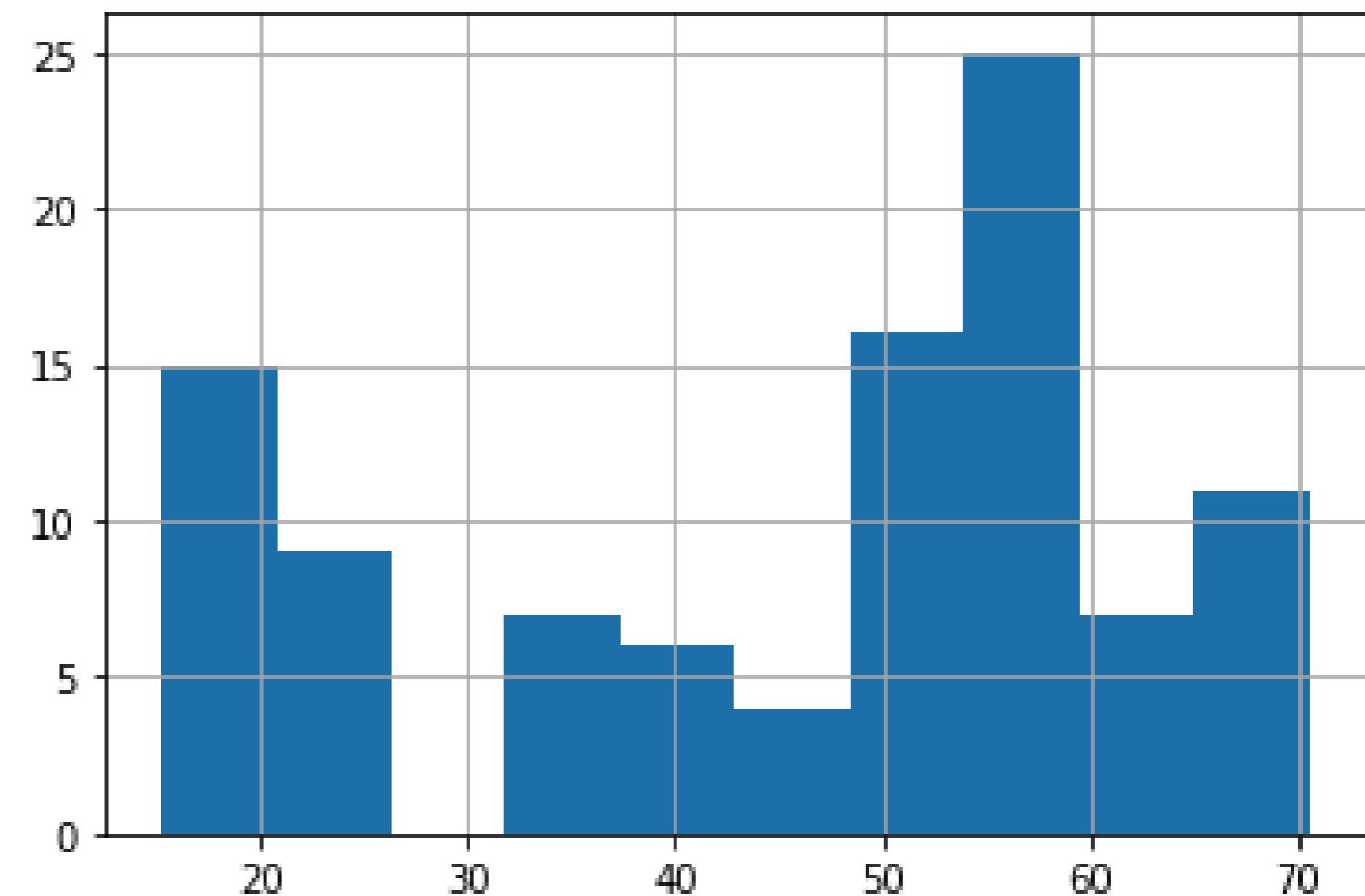
Senior Content Developer at DataCamp

Histograms

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

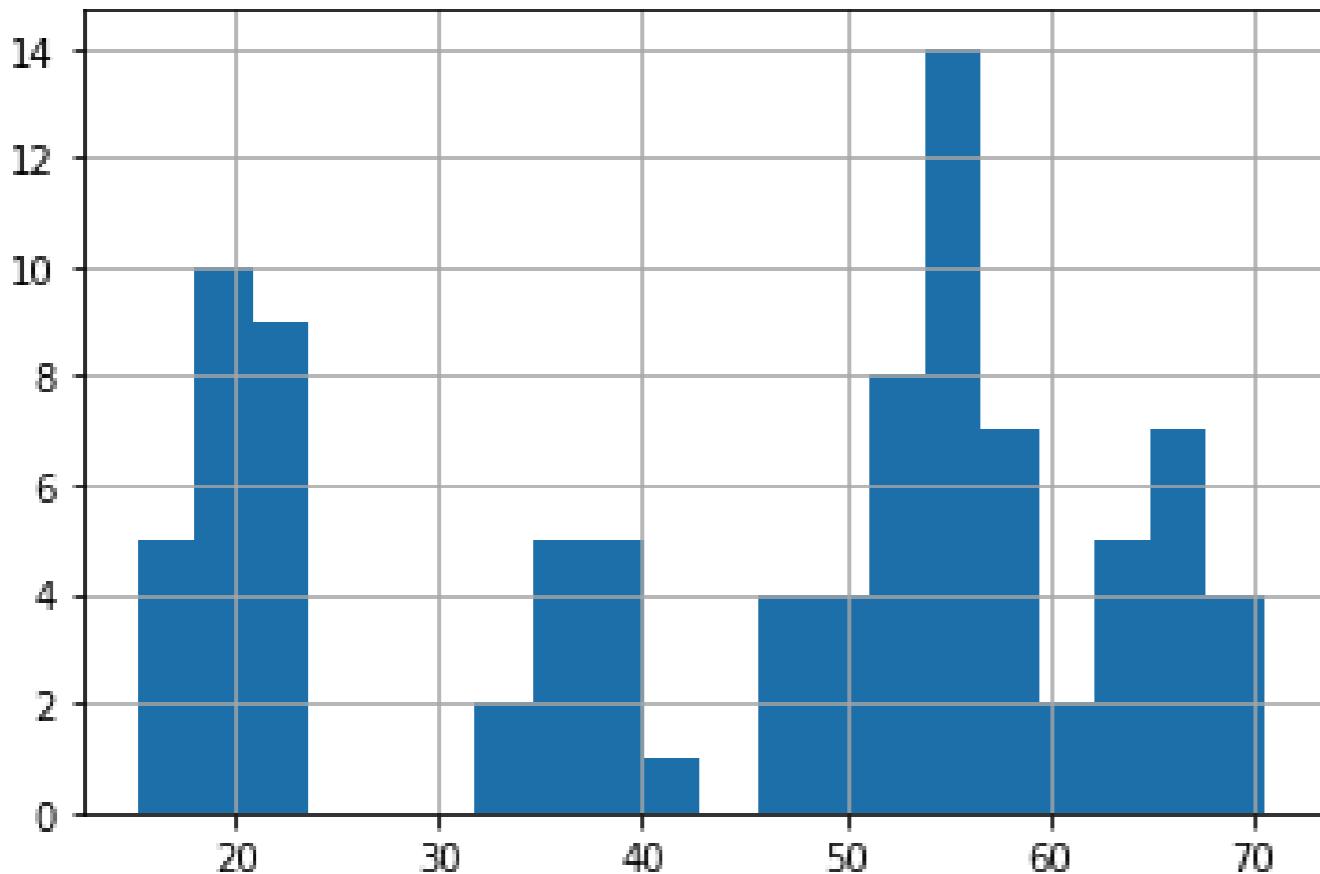
```
dog_pack["height_cm"].hist()
```

```
plt.show()
```

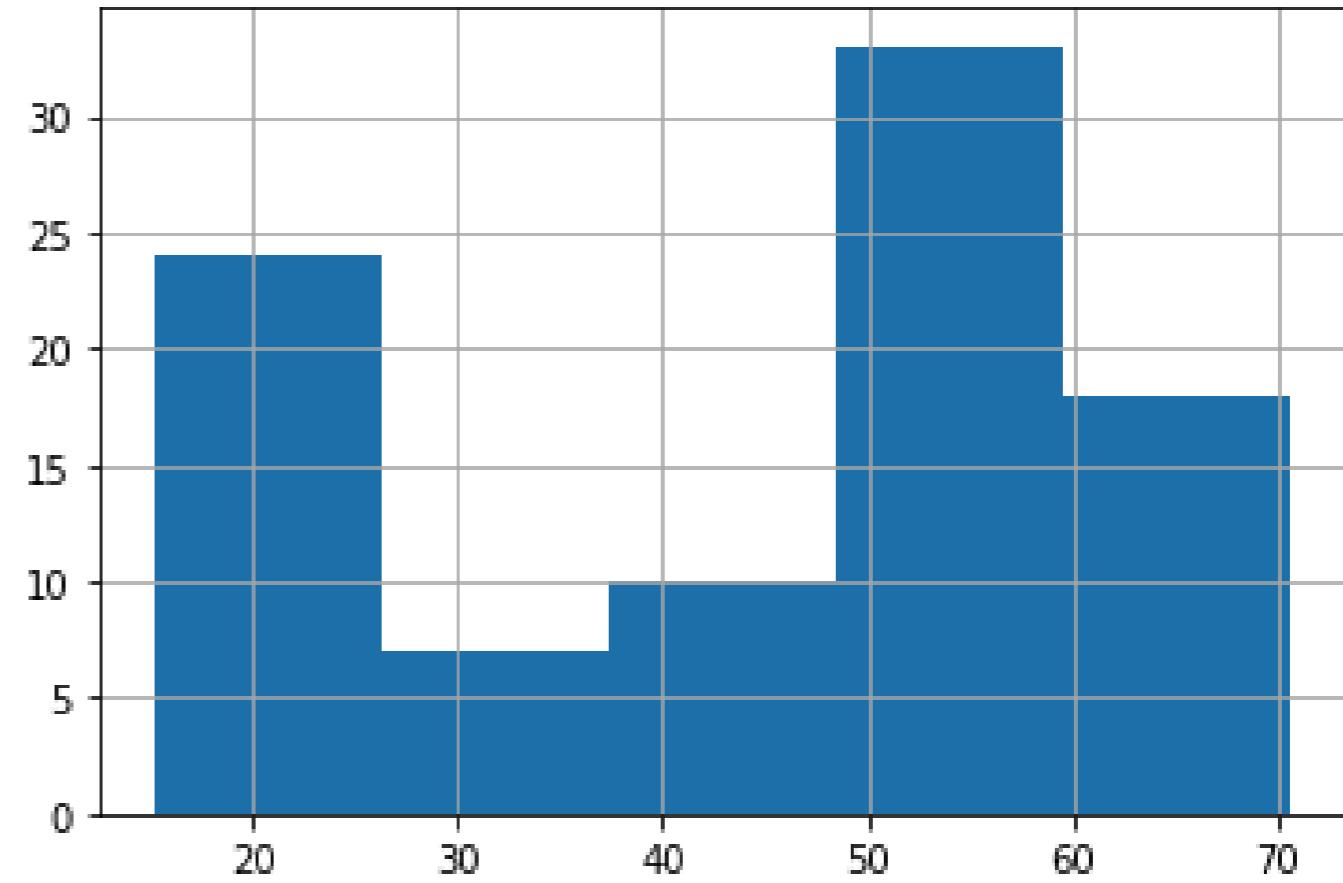


Histograms

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



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



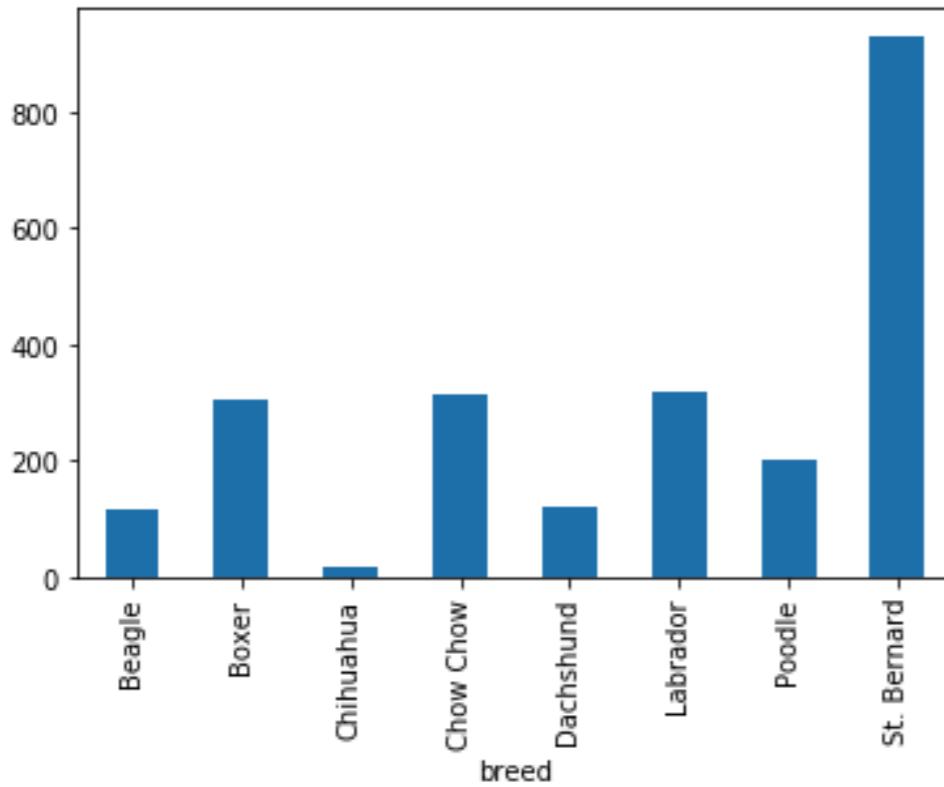
Bar plots

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

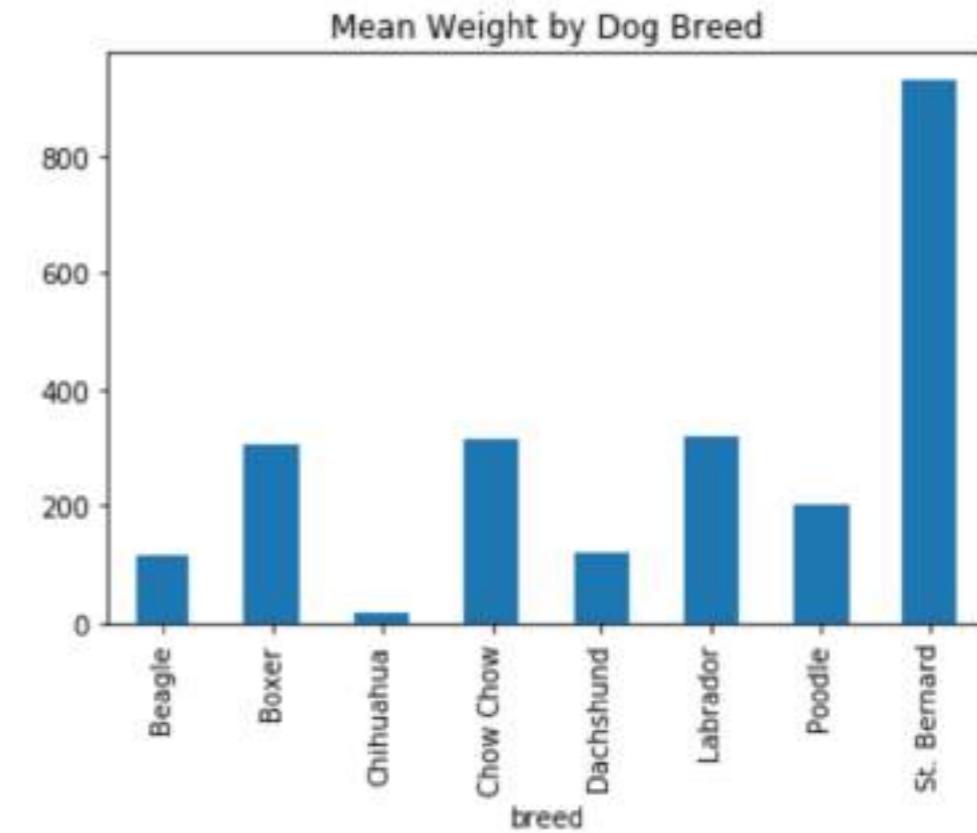
```
breed  
Beagle      10.636364  
Boxer       30.620000  
Chihuahua   1.491667  
Chow Chow    22.535714  
Dachshund   9.975000  
Labrador    31.850000  
Poodle      20.400000  
St. Bernard  71.576923  
Name: weight_kg, dtype: float64
```

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()
```

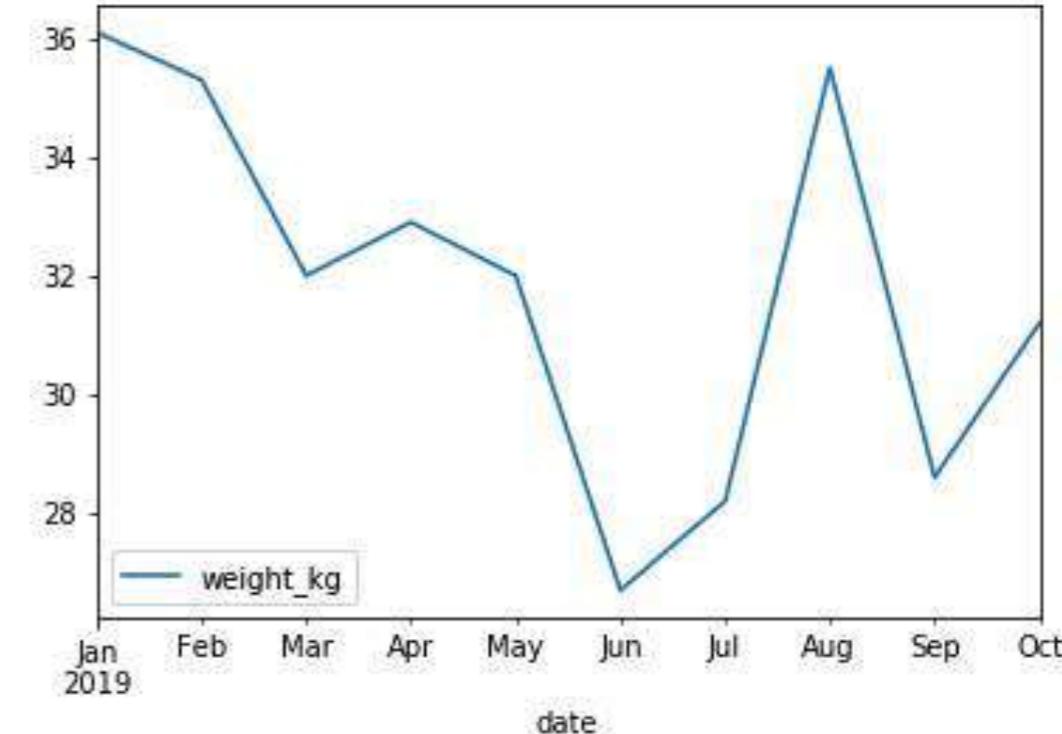


Line plots

```
sully.head()
```

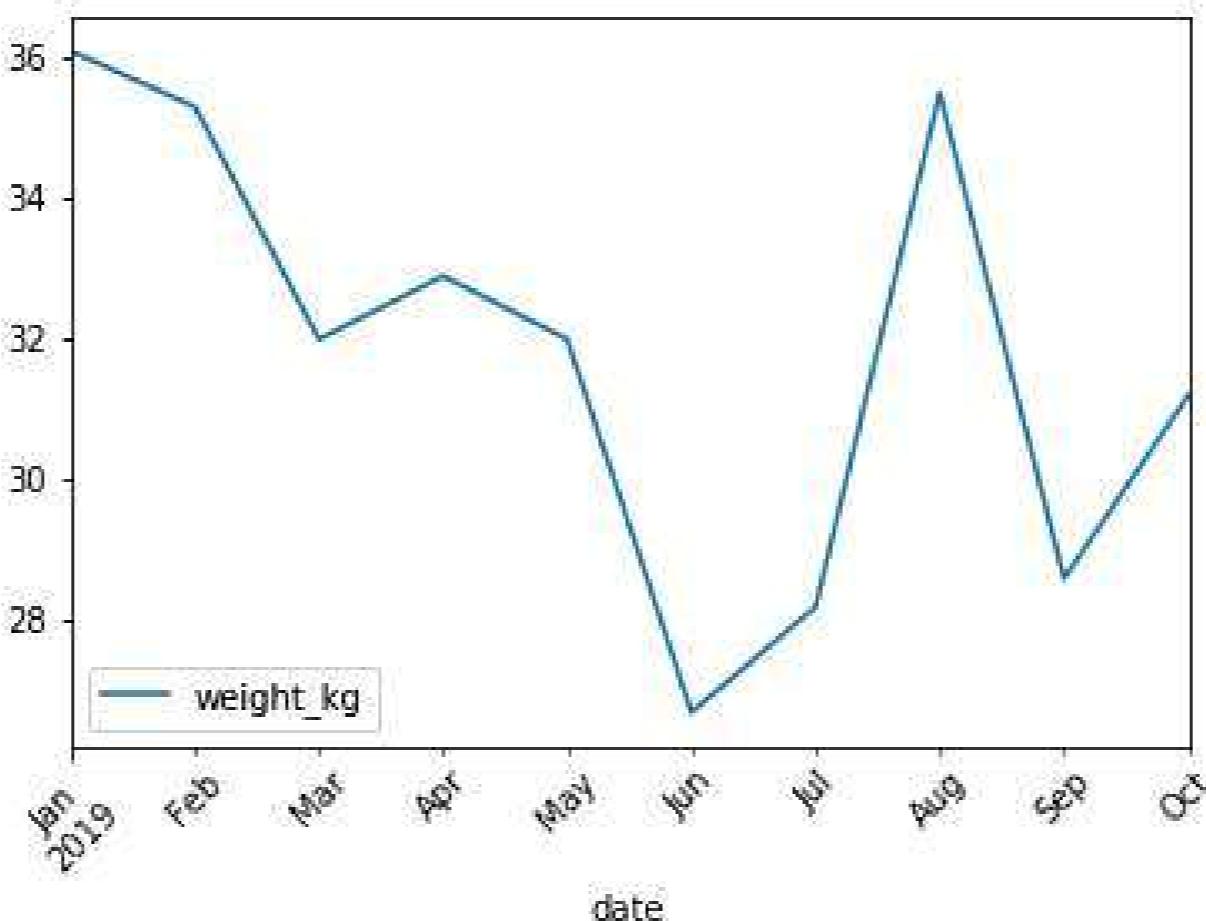
```
      date    weight_kg
0 2019-01-31        36.1
1 2019-02-28        35.3
2 2019-03-31        32.0
3 2019-04-30        32.9
4 2019-05-31        32.0
```

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



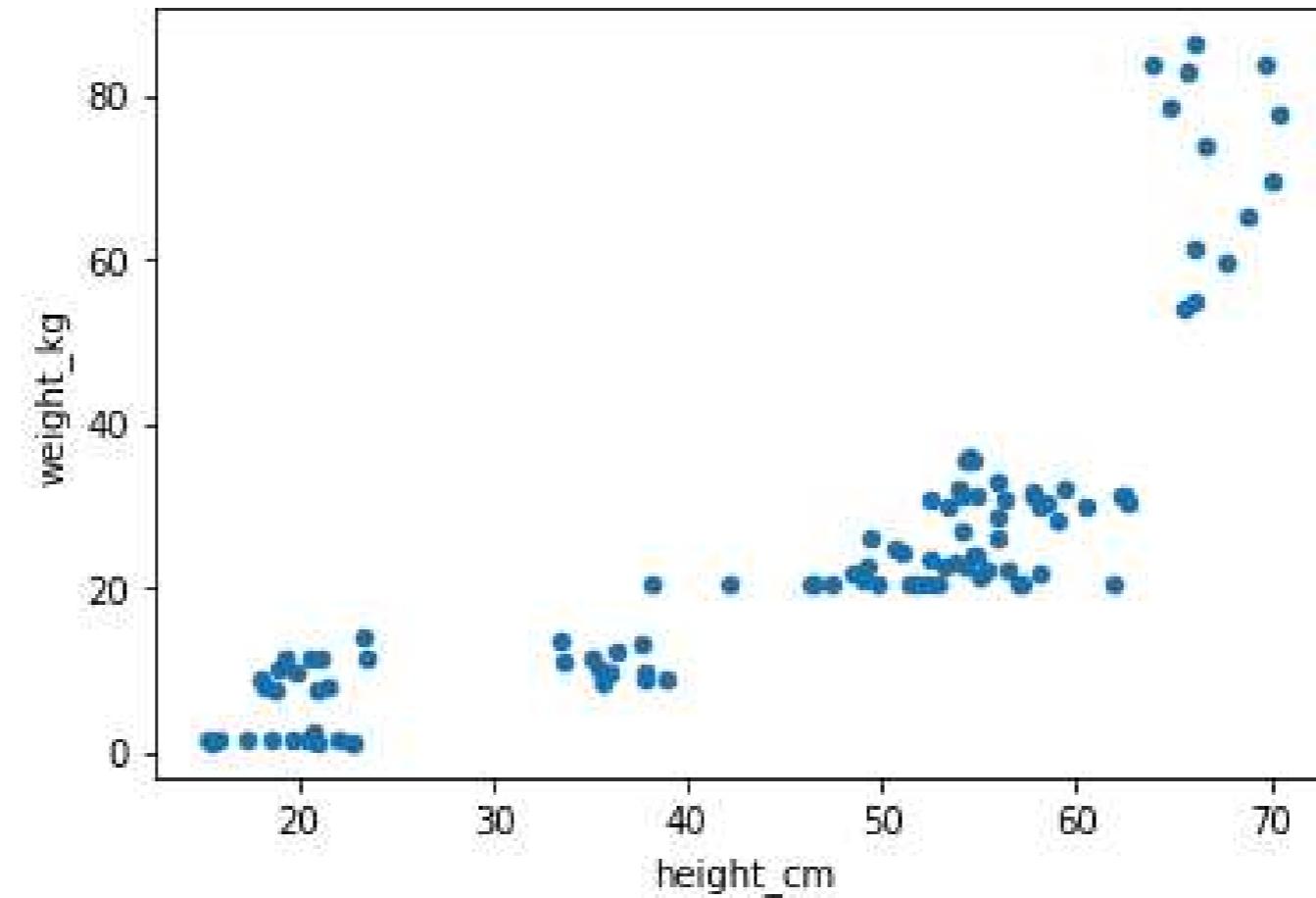
Rotating axis labels

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



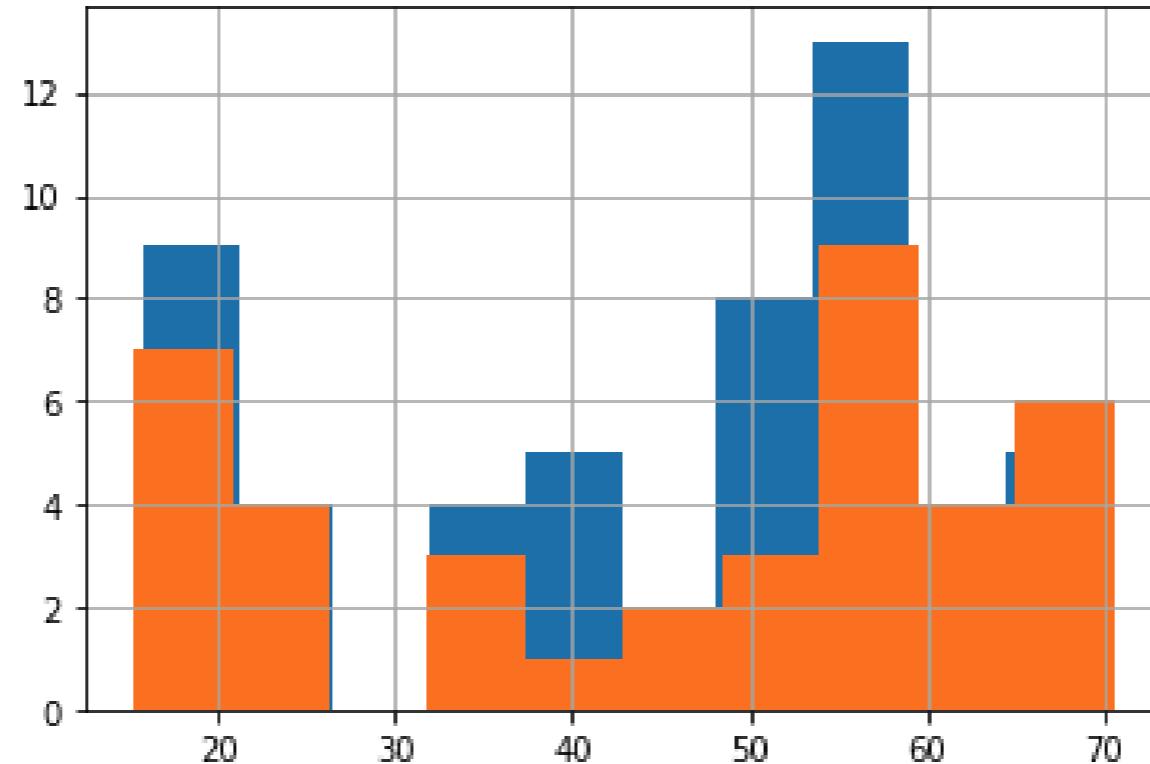
Scatter plots

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



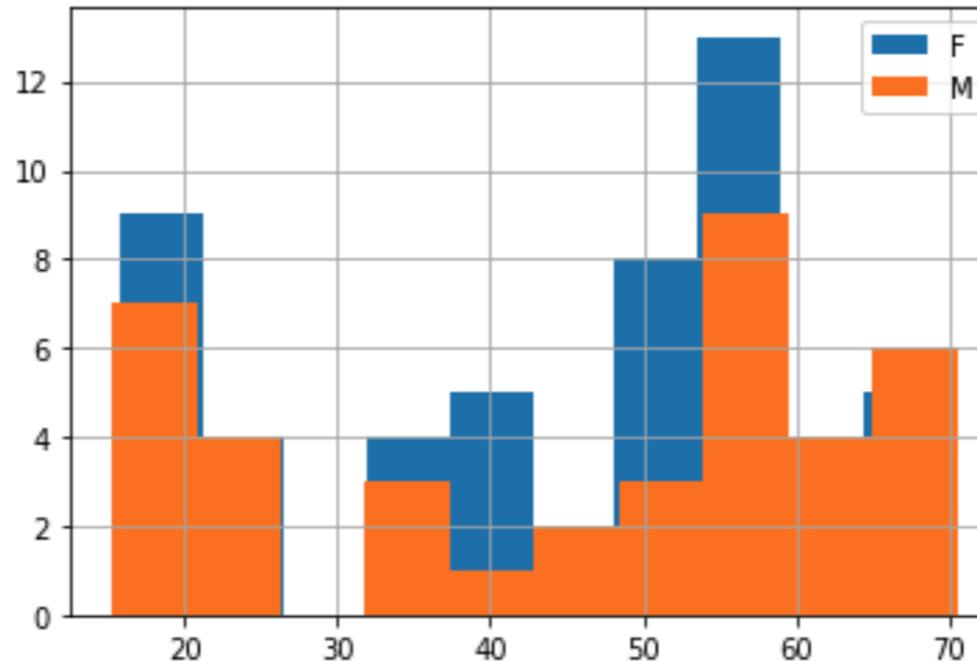
Layering plots

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



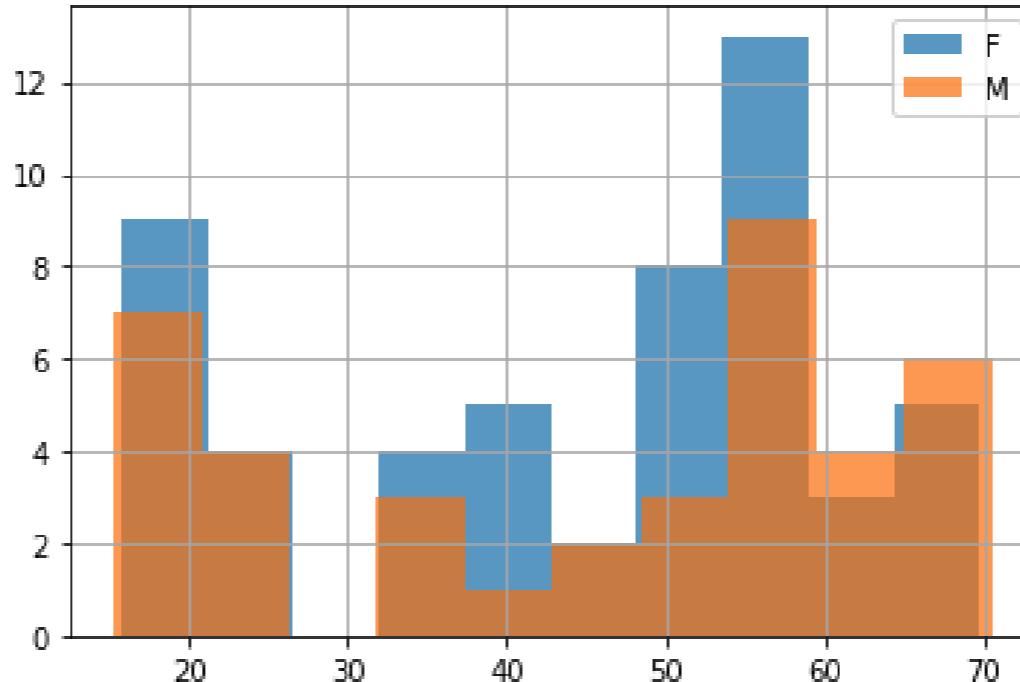
Add a legend

```
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

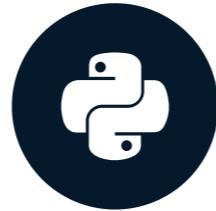
[1014 rows x 6 columns]

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
```

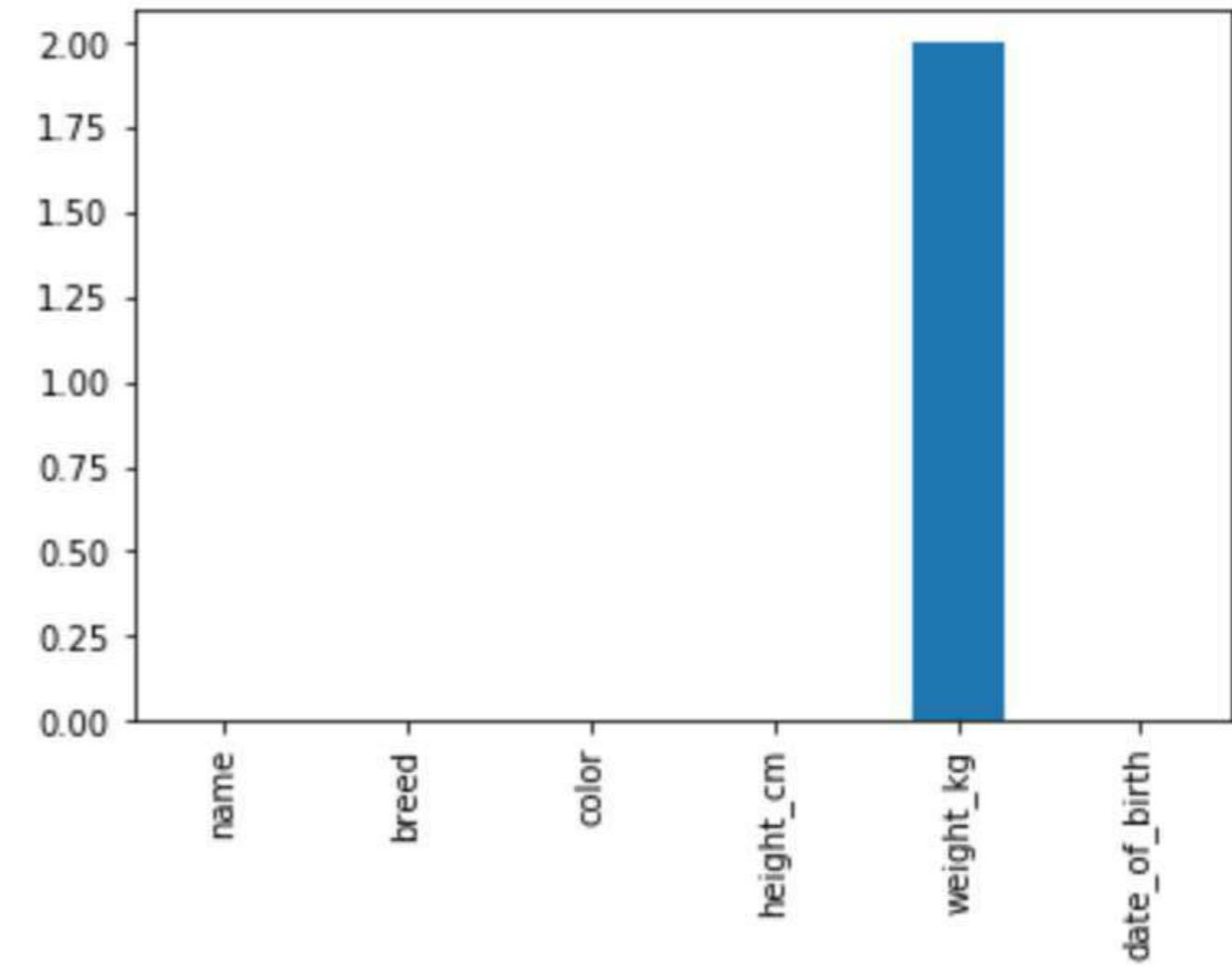
Counting missing values

```
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)
```

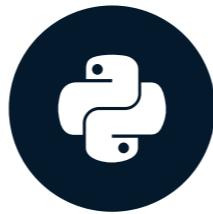
	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	0.0	2013-07-01
1	Charlie	Poodle	Black	43	24.0	2016-09-16
2	Lucy	Chow Chow	Brown	46	24.0	2014-08-25
3	Cooper	Schnauzer	Gray	49	0.0	2011-12-11
4	Max	Labrador	Black	59	29.0	2017-01-20
5	Stella	Chihuahua	Tan	18	2.0	2015-04-20
6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

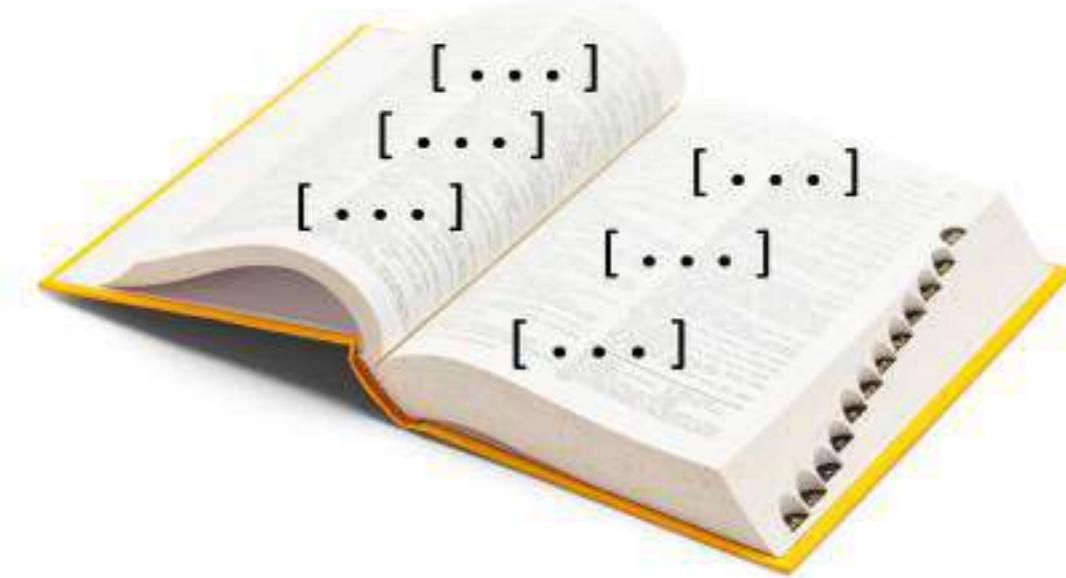
From a list of dictionaries

- 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

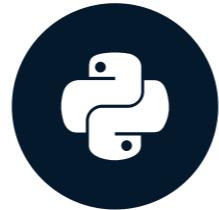
```
name,breed,height_cm,weight_kg,d_o_b,bmi  
Ginger,Dachshund,22,10,2019-03-14,206.611570  
Scout,Dalmatian,59,25,2019-05-09,71.818443
```

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

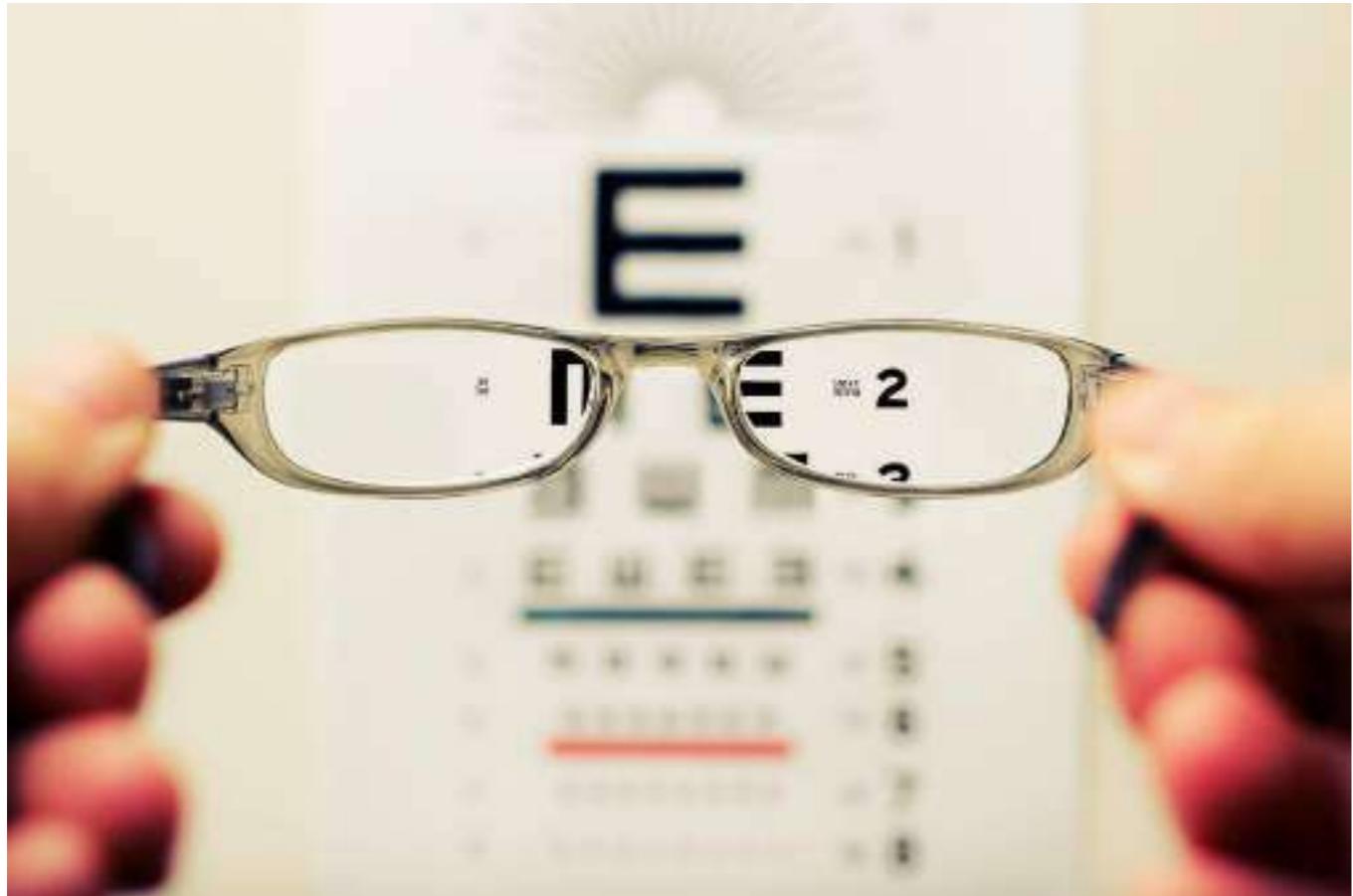
JOINING DATA WITH PANDAS



Aaren Stubberfield

Instructor

For clarity



Tables = DataFrames

Merging = Joining

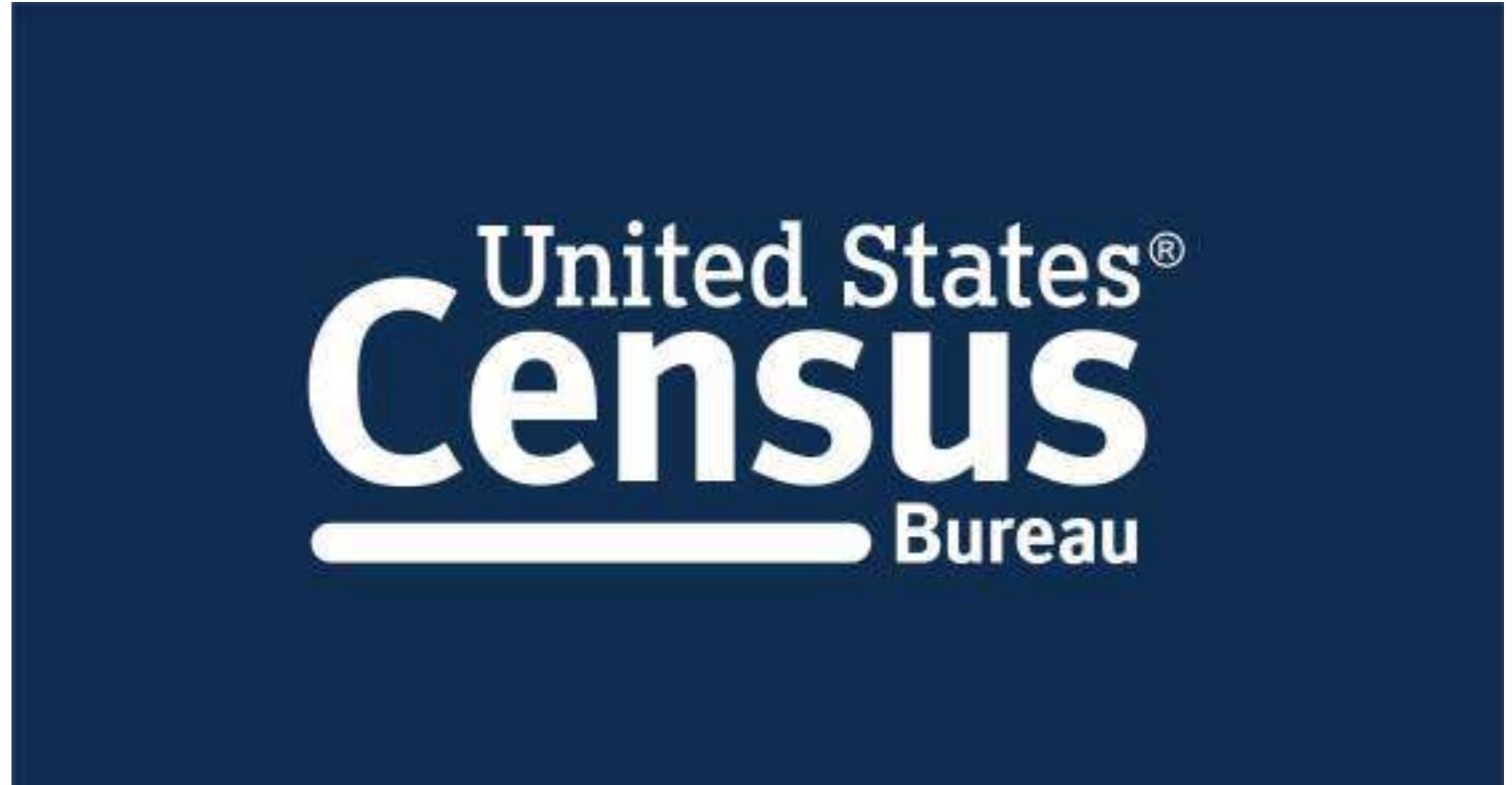
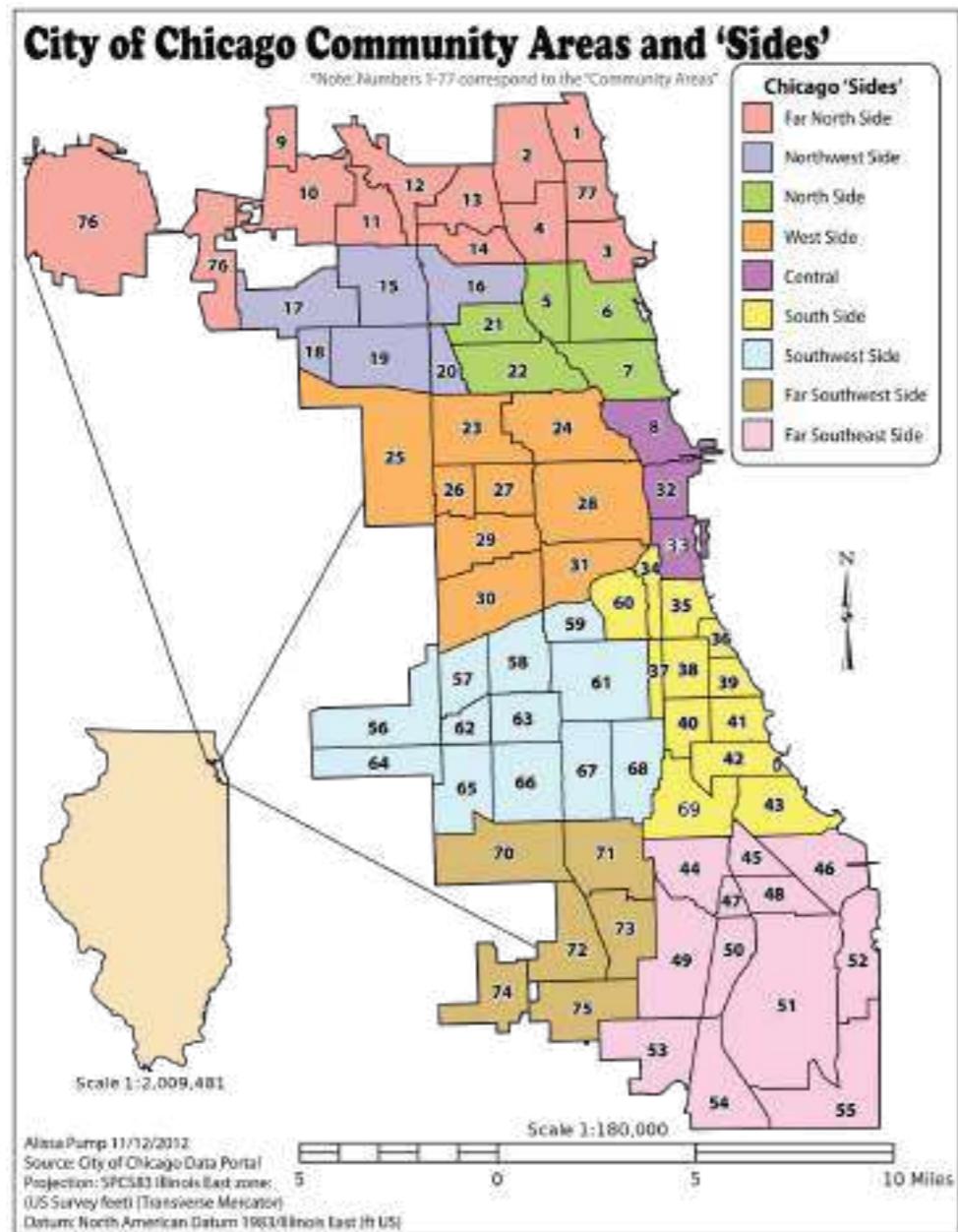
¹ Photo by David Travis on Unsplash

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))
```

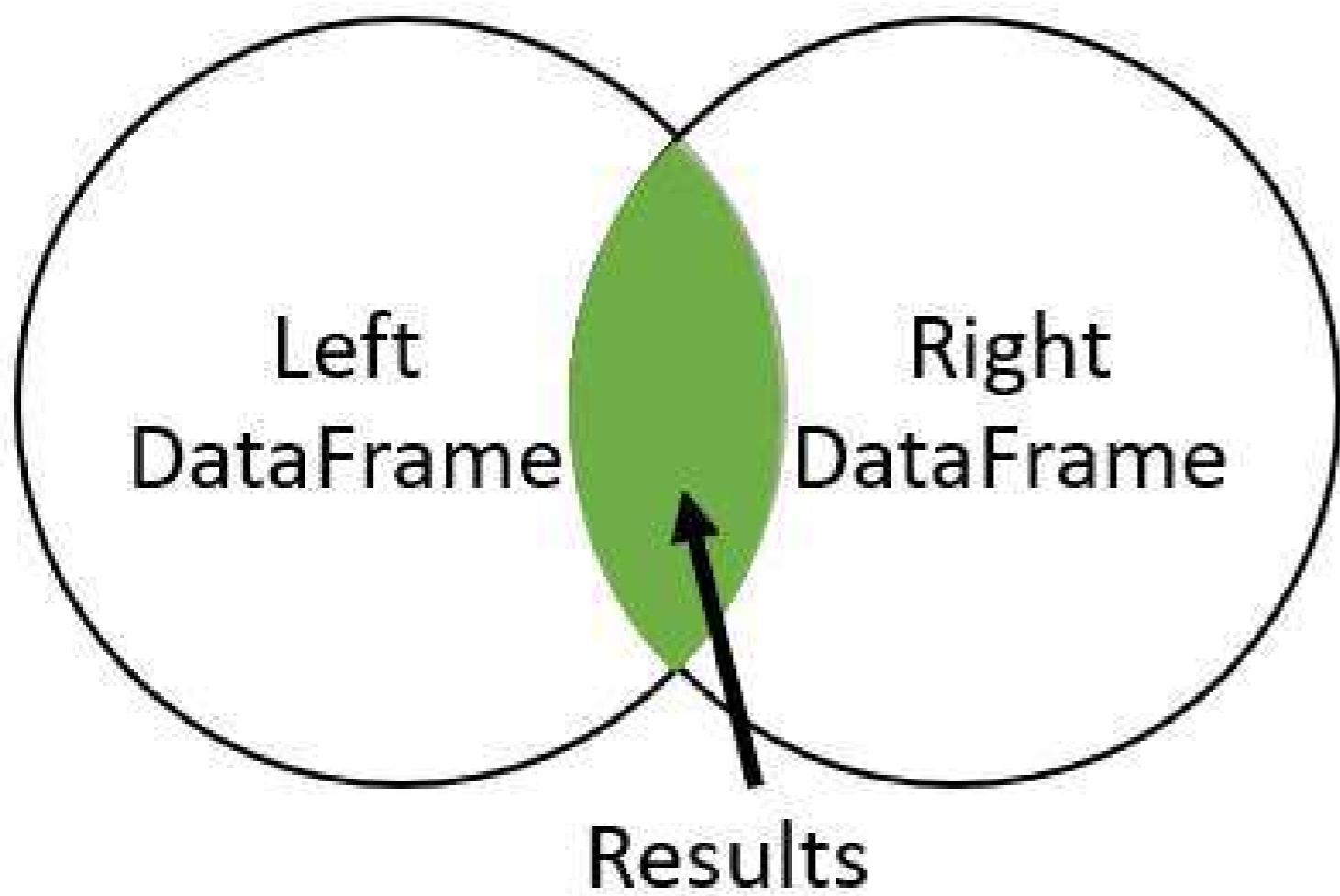
	ward	alderman	address_x	zip_x	pop_2000	pop_2010	change	address_y	zip_y
0	1	Proco "Joe" ...	2058 NORTH W...	60647	52951	56149	6%	2765 WEST SA...	60647
1	2	Brian Hopkins	1400 NORTH ...	60622	54361	55805	3%	WM WASTE MAN...	60622
2	3	Pat Dowell	5046 SOUTH S...	60609	40385	53039	31%	17 EAST 38TH...	60653
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

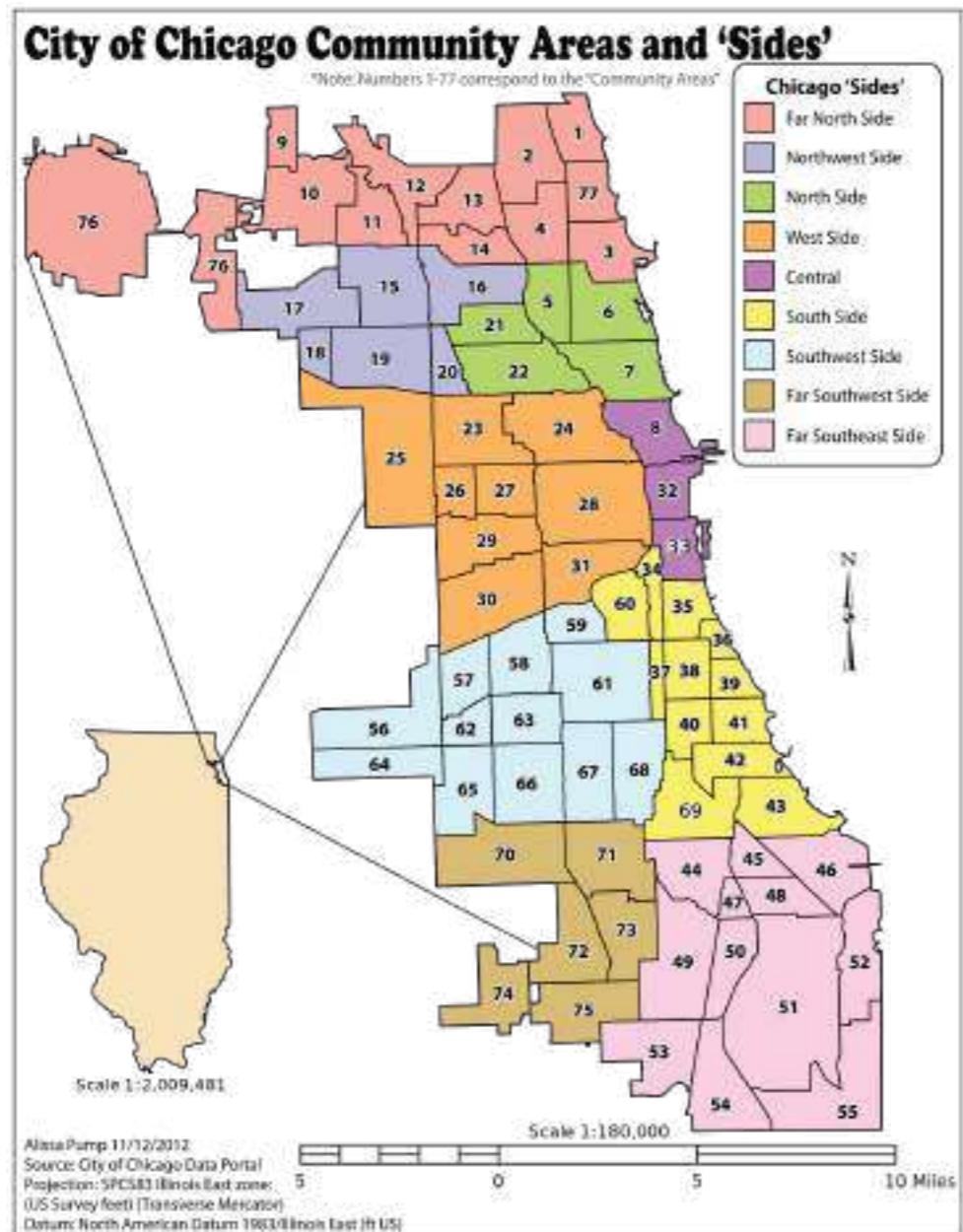
	ward	pop_2000	pop_2010	change	address	zip
0	1	52951	56149	6%	2765 WEST SA...	60647
1	2	54361	55805	3%	WM WASTE MAN...	60622
2	3	40385	53039	31%	17 EAST 38TH...	60653
3	4	51953	54589	5%	31ST ST HARB...	60653
4	5	55302	51455	-7%	JACKSON PARK...	60637

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)
```

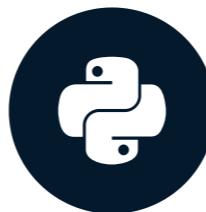
```
(10000, 9)
```

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...

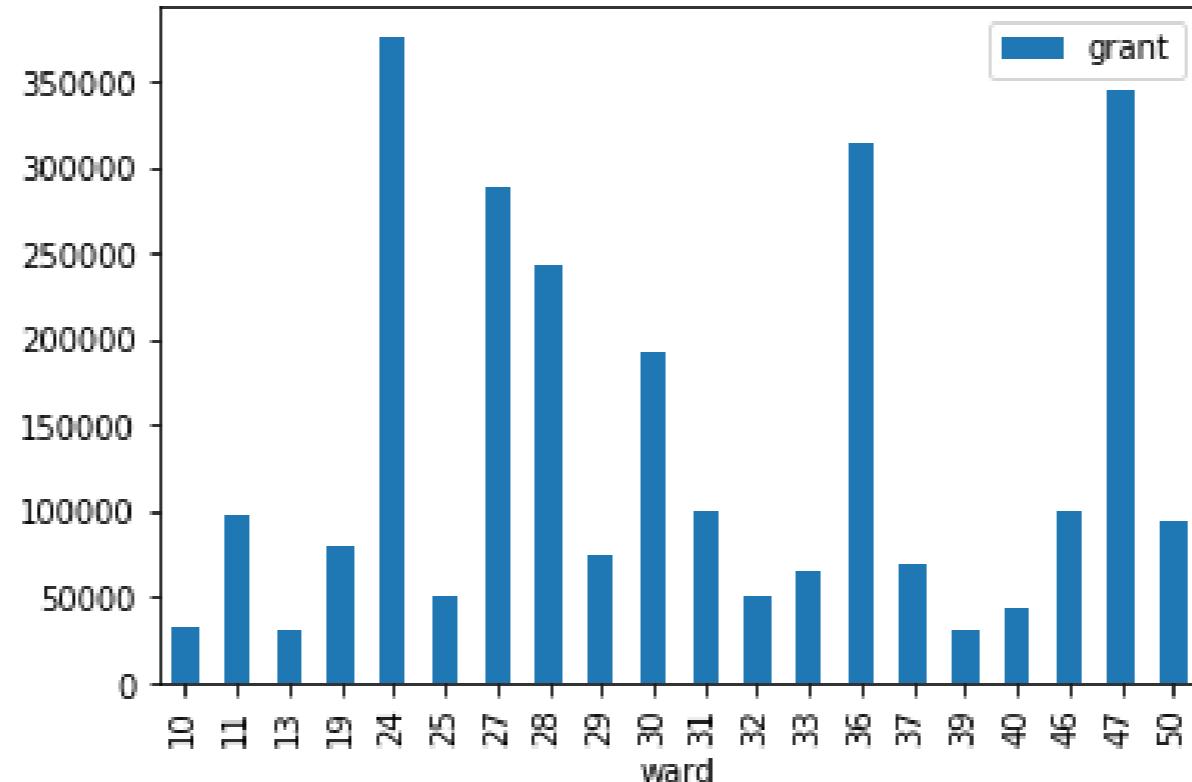
Merging multiple tables

```
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

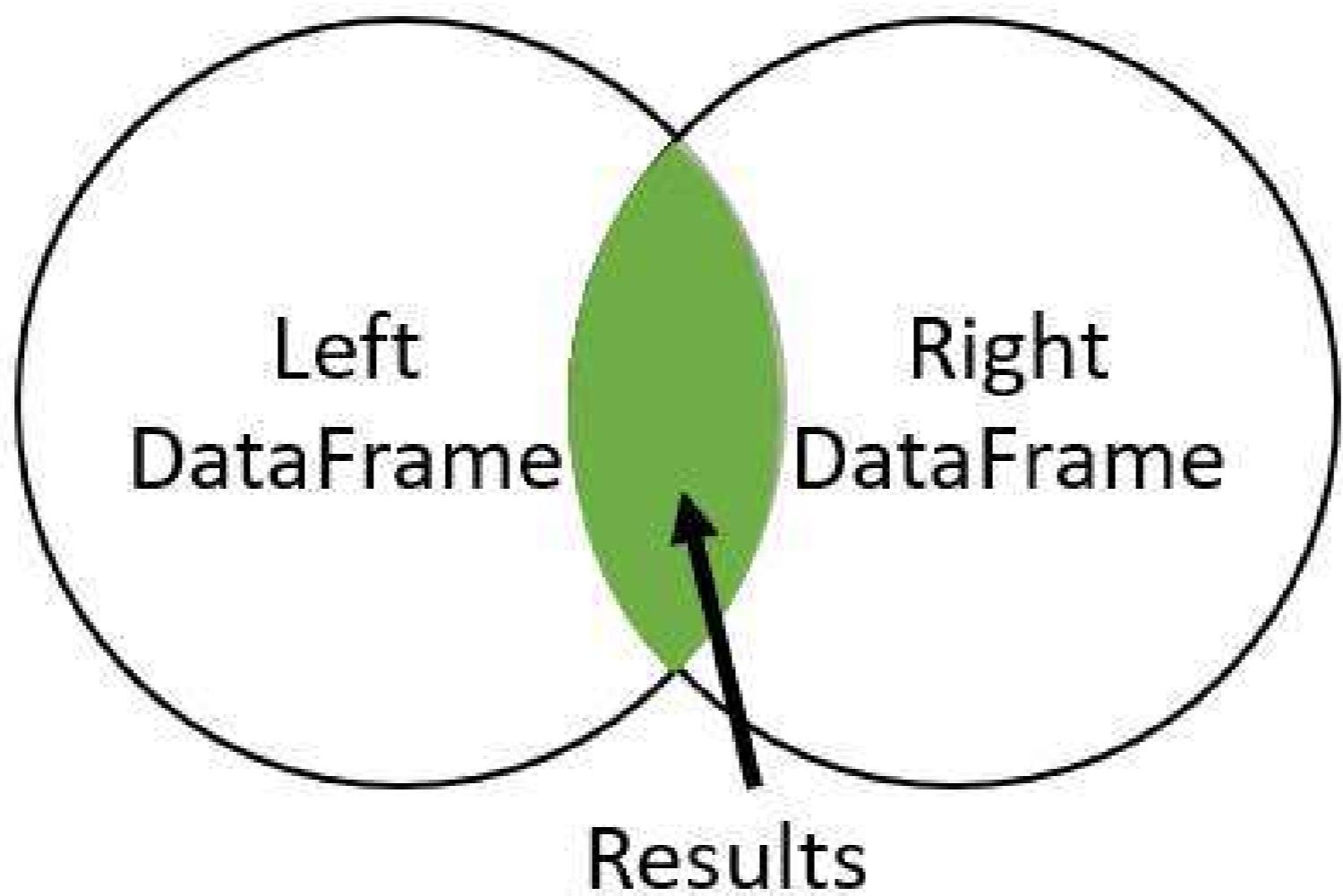


Aaren Stubberfield

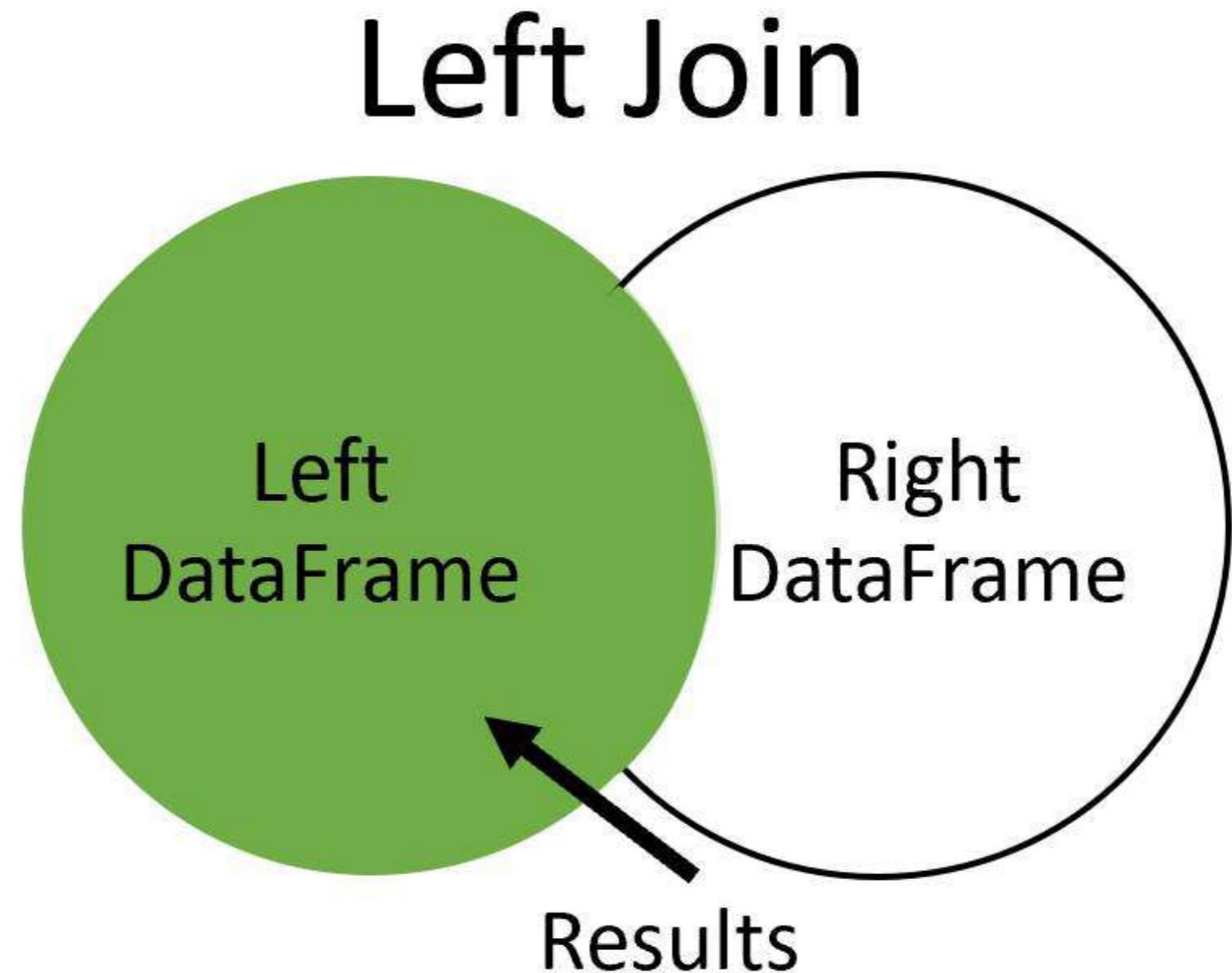
Instructor

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...      2006-11-16
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

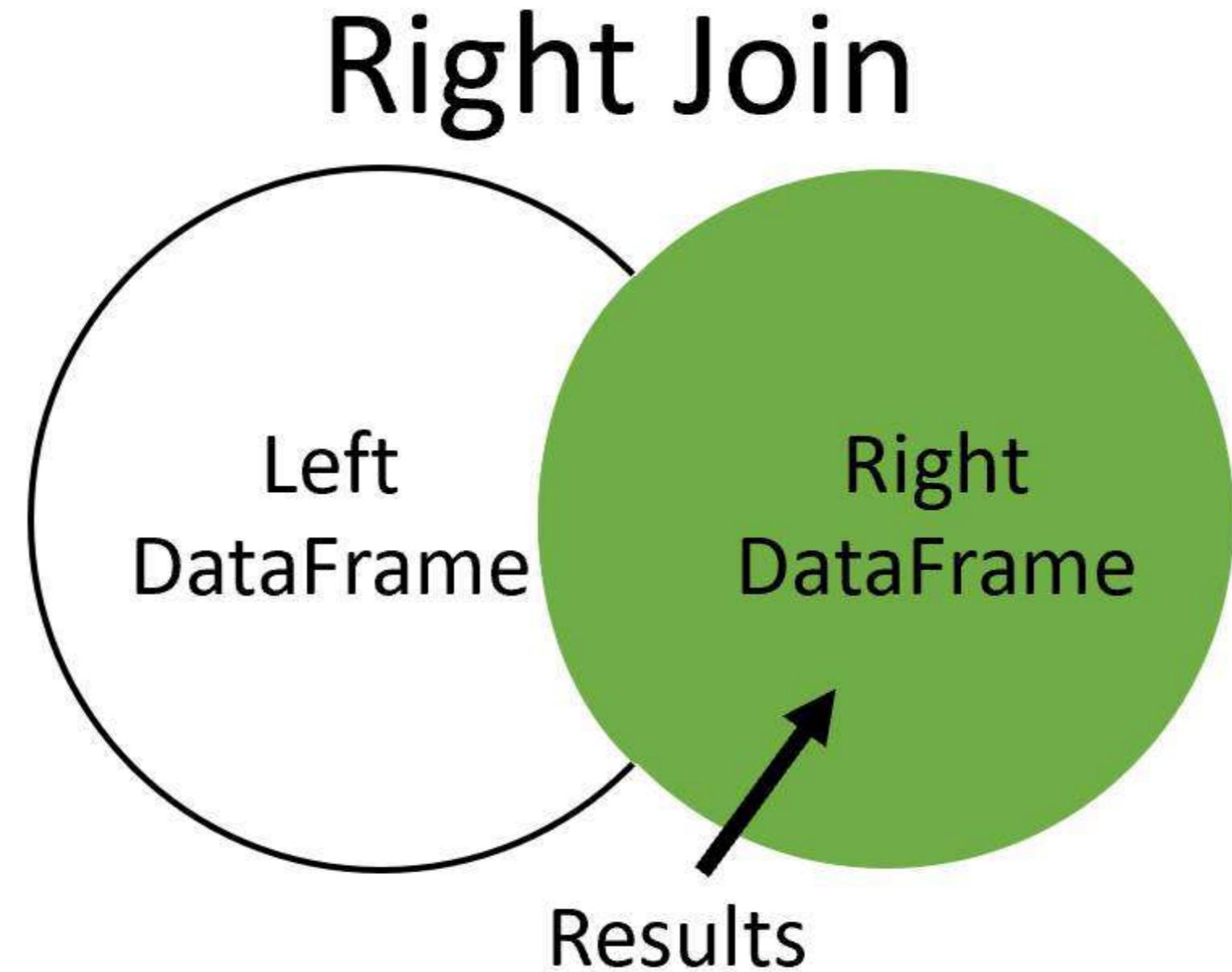
JOINING DATA WITH PANDAS



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

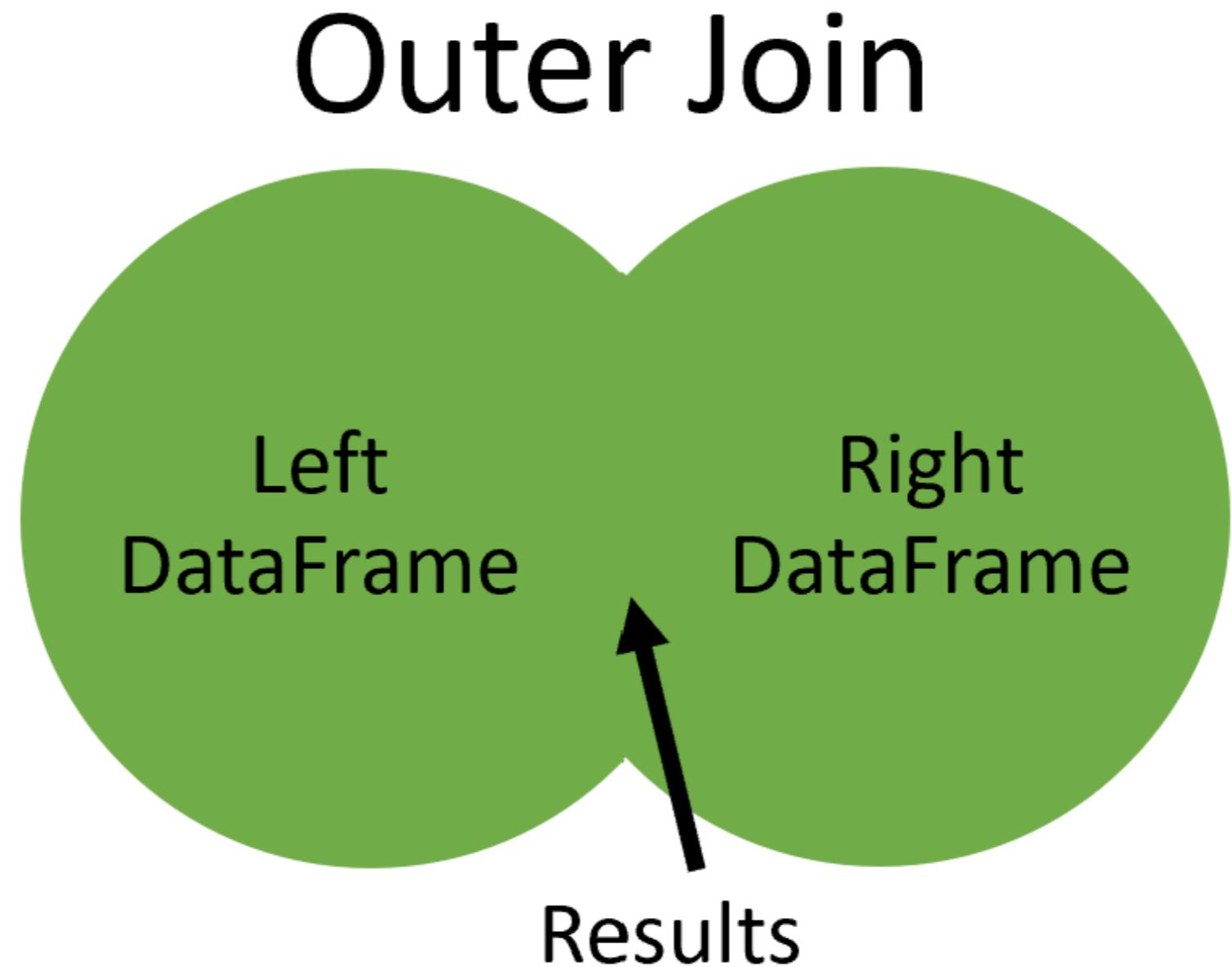
	movie_id	genre
4998	10947	TV Movie
5994	13187	TV Movie
7443	22488	TV Movie
10061	78814	TV Movie
10790	153397	TV Movie

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



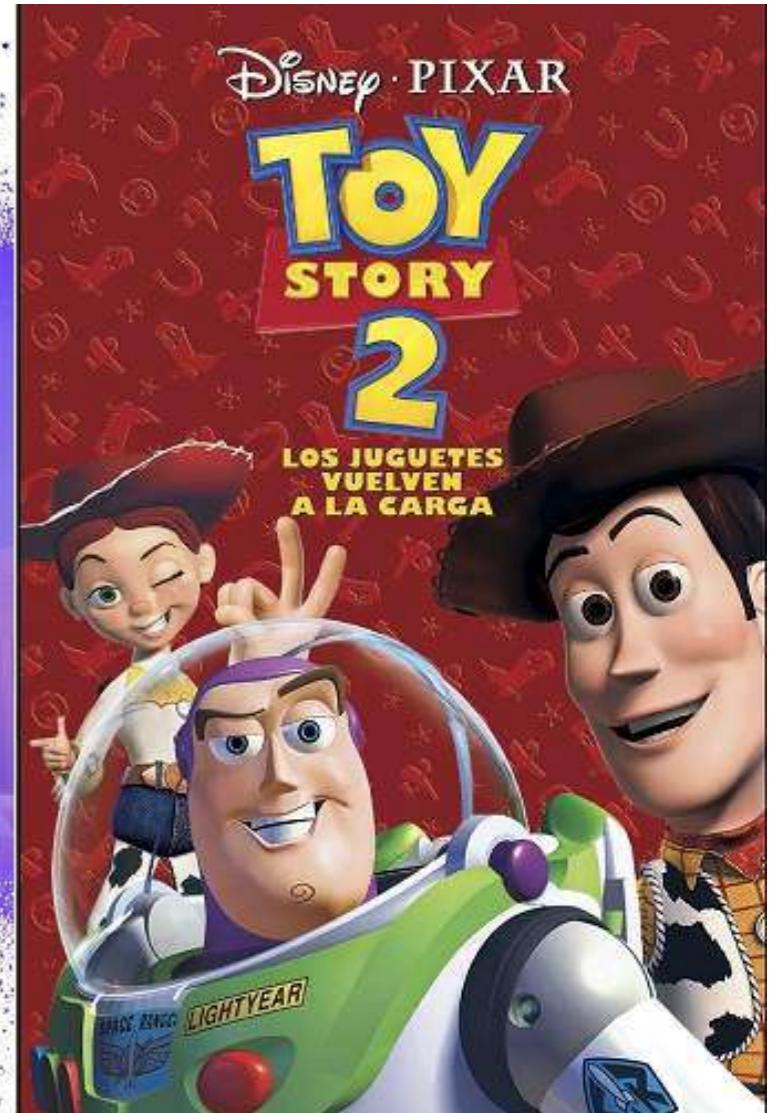
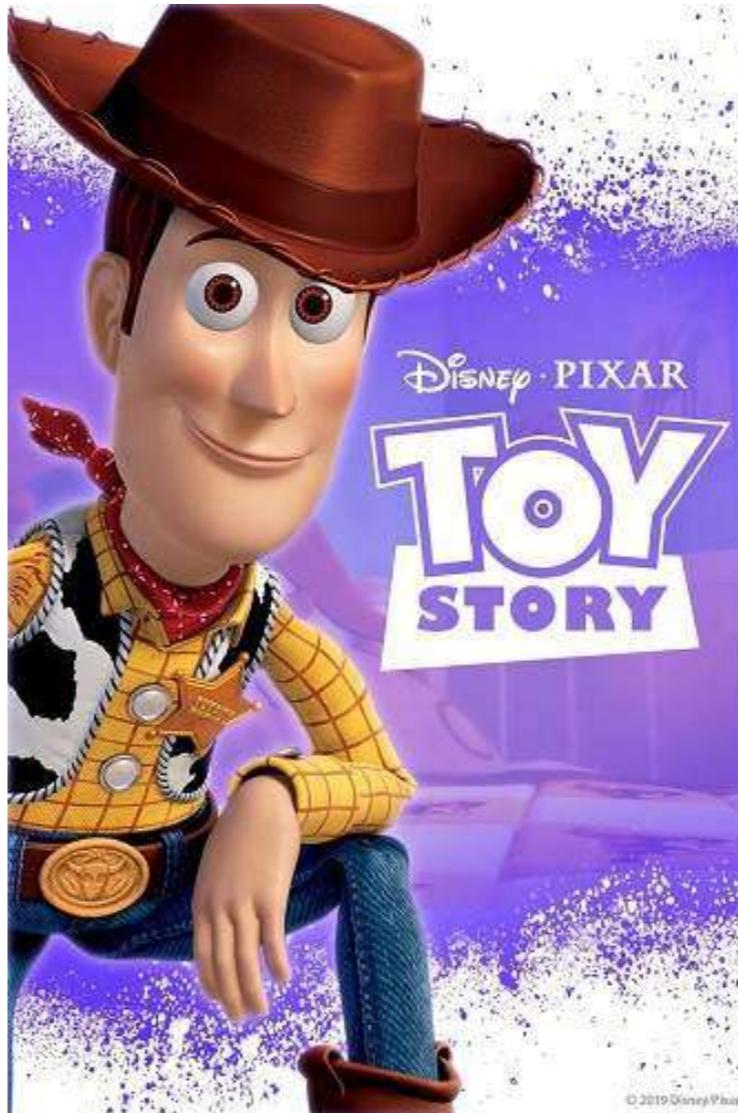
Aaren Stubberfield

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
```

```
      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
```

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

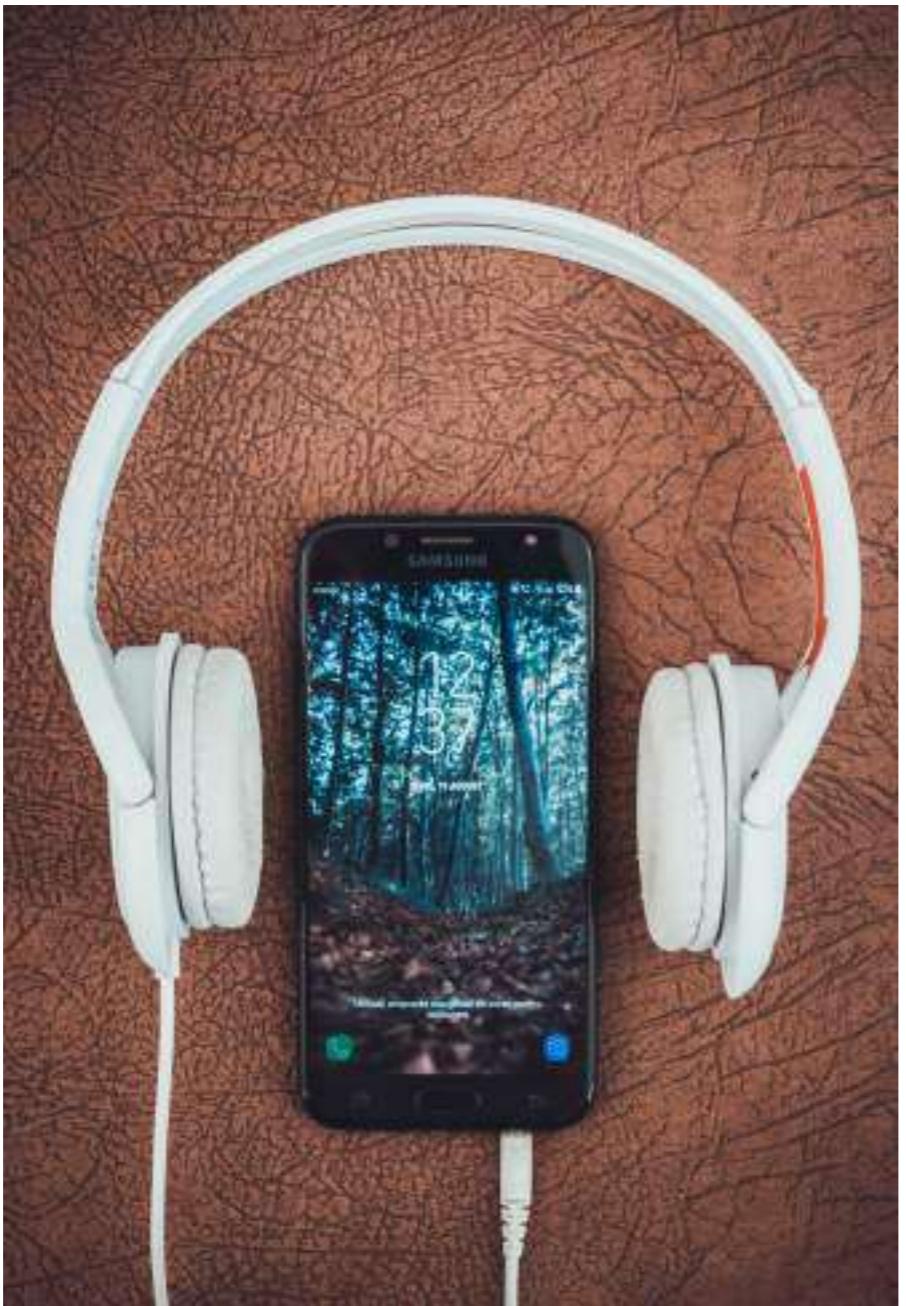
Result Table

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

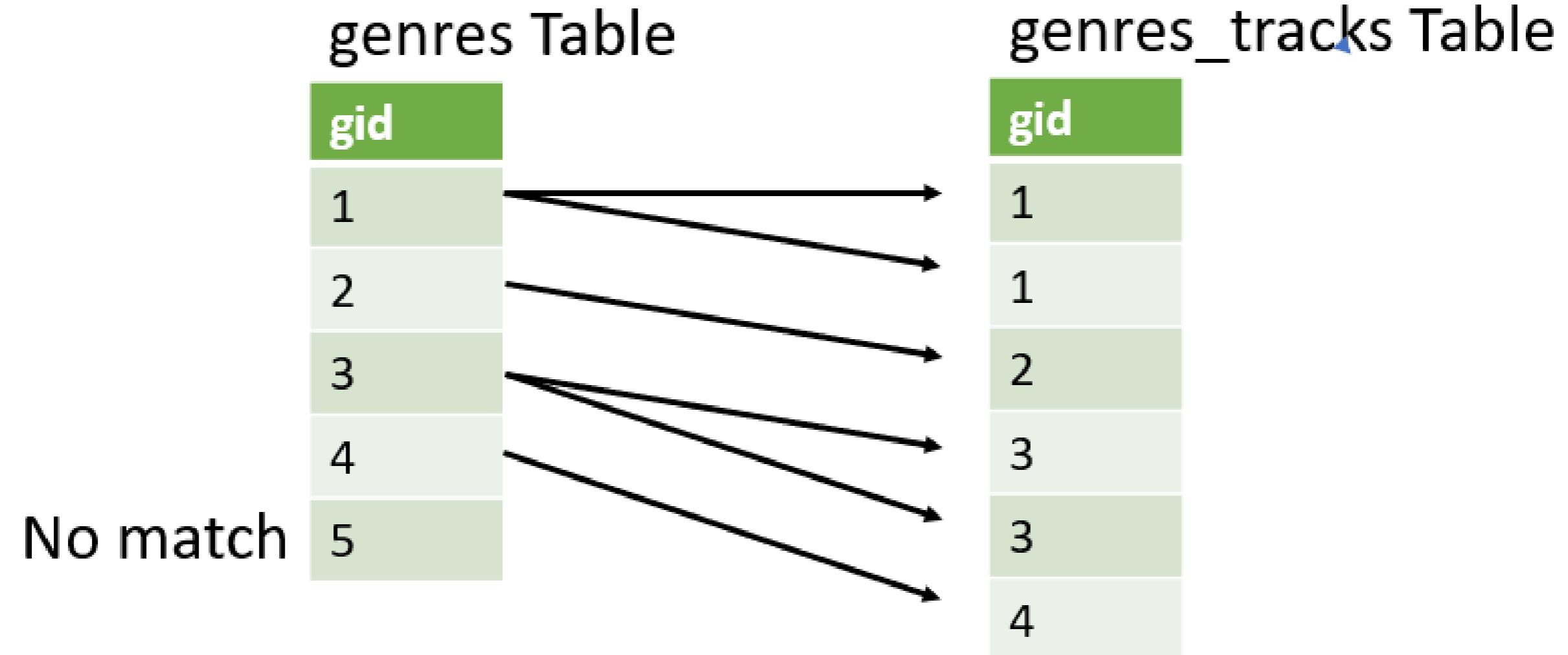
Step 1 - semi join

```
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'])
```

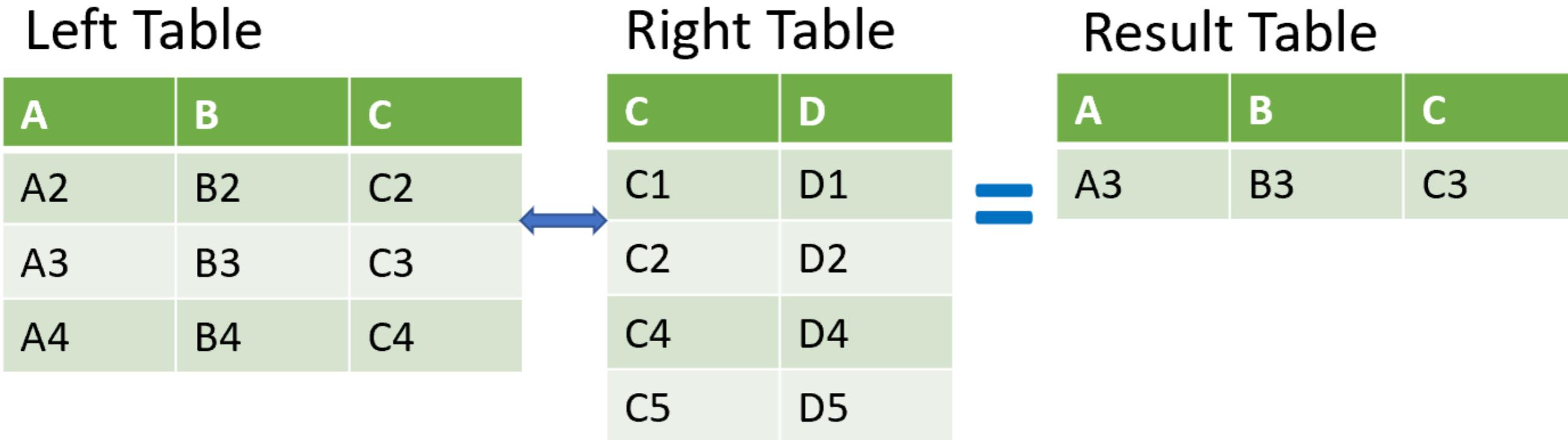
```
0    True
1    True
2    True
3    True
4   False
Name: gid, dtype: bool
```

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

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:

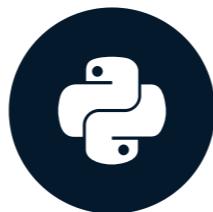
- Fix incorrect data
- Drop duplicate rows

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()?

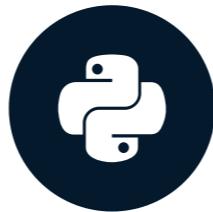
- Ordered data / time series
- Filling in missing values

Let's practice!

JOINING DATA WITH PANDAS

Using `merge_asof()`

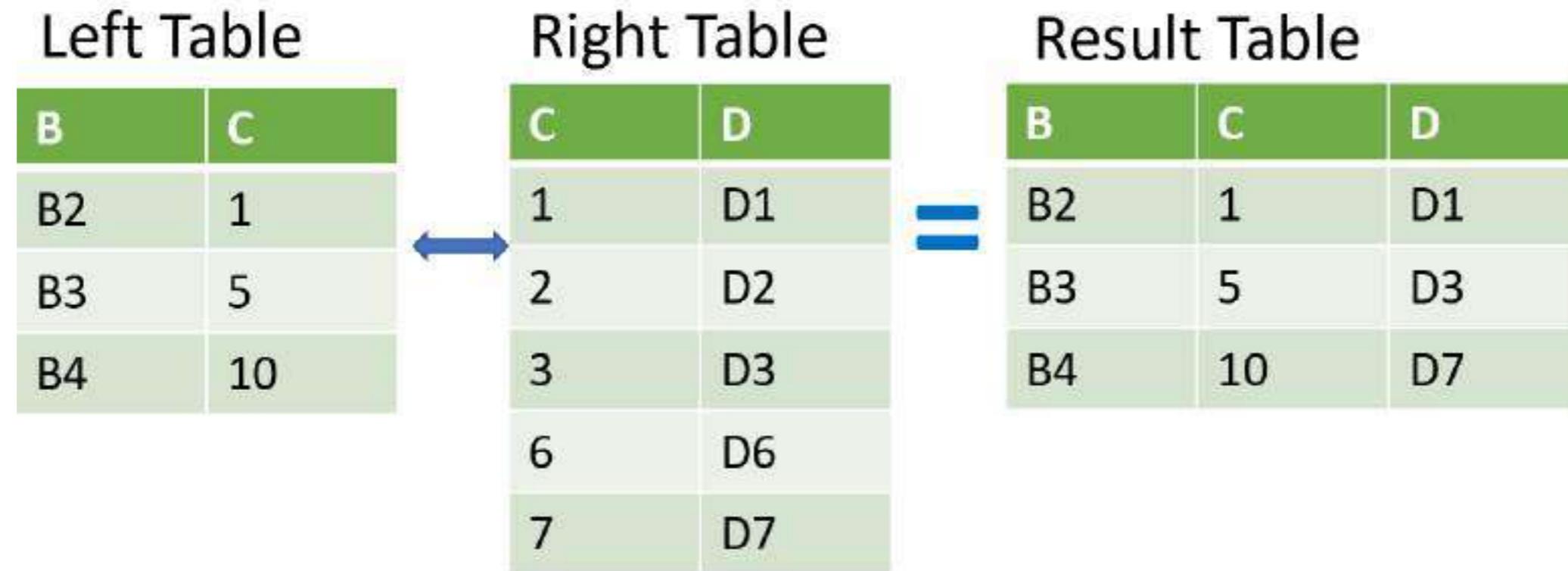
JOINING DATA WITH PANDAS



Aaren Stubberfield

Instructor

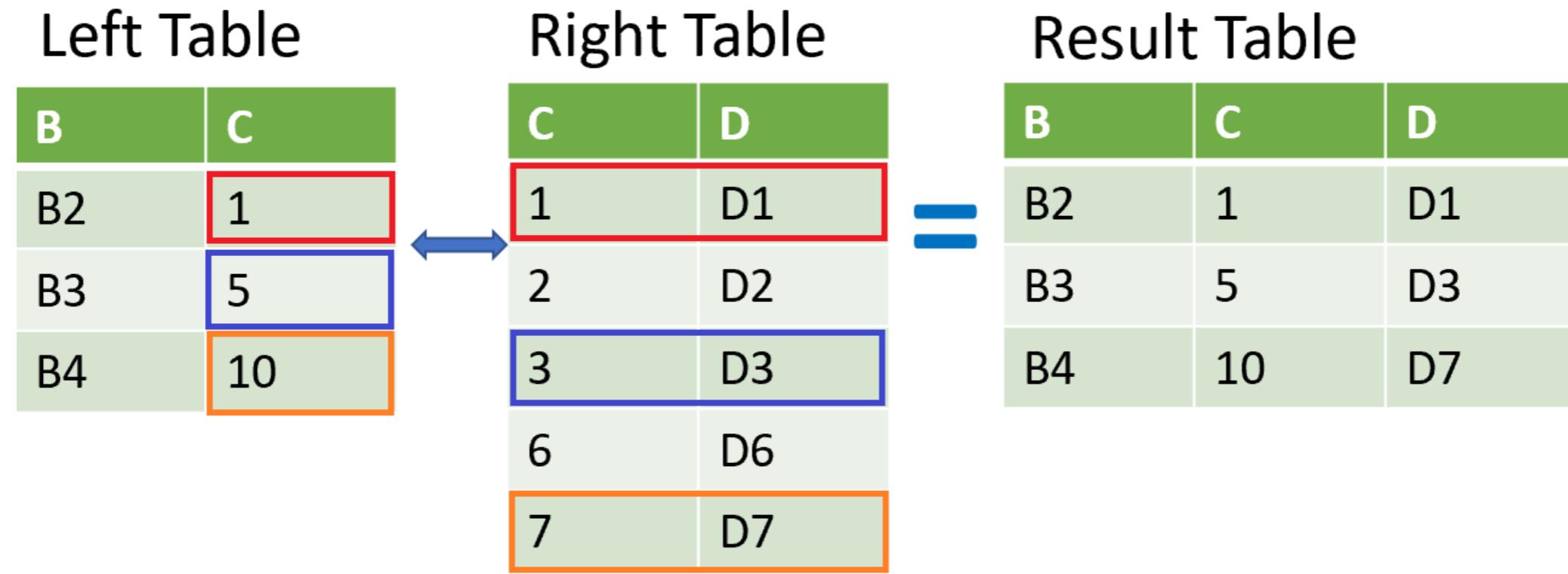
Using merge_asof()



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)')
```

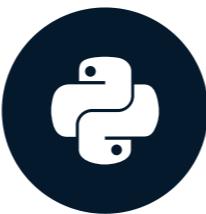
	date	stock	close
0	2019-07-01	disney	143.009995
1	2019-08-01	disney	137.259995
2	2019-09-01	disney	130.320007
3	2019-10-01	disney	129.919998
4	2019-11-01	disney	151.580002
5	2019-07-01	nike	86.029999
6	2019-08-01	nike	84.5
8	2019-10-01	nike	89.550003

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

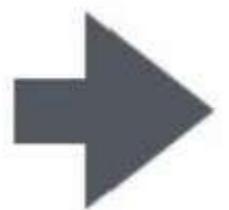
Long Format

	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

Data type constraints

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @ DataCamp

Course outline



Diagnose dirty
data

Course outline



Diagnose dirty
data



Side effects of
dirty data

Course outline



Diagnose dirty
data



Side effects of
dirty data



Clean data

Course outline



Diagnose dirty
data



Side effects of
dirty data



Clean data

Chapter 1 - Common data problems

Why do we need to clean data?



Why do we need to clean data?



Why do we need to clean data?



Data type constraints

Datatype	Example
Text data	First name, last name, address ...
Integers	# Subscribers, # products sold ...
Decimals	Temperature, \$ exchange rates ...
Binary	Is married, new customer, yes/no, ...
Dates	Order dates, ship dates ...
Categories	Marriage status, gender ...

Python data type
str
int
float
bool
datetime
category

Strings to integers

```
# Import CSV file and output header  
sales = pd.read_csv('sales.csv')  
sales.head(2)
```

```
SalesOrderID      Revenue      Quantity  
0              43659     23153$          12  
1              43660     1457$           2
```

```
# Get data types of columns  
sales.dtypes
```

```
SalesOrderID      int64  
Revenue          object  
Quantity         int64  
dtype: object
```

String to integers

```
# Get DataFrame information  
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 31465 entries, 0 to 31464  
Data columns (total 3 columns):  
SalesOrderID      31465 non-null int64  
Revenue          31465 non-null object  
Quantity         31465 non-null int64  
dtypes: int64(2), object(1)  
memory usage: 737.5+ KB
```

String to integers

```
# Print sum of all Revenue column  
sales['Revenue'].sum()
```

```
'23153$1457$36865$32474$472$27510$16158$5694$6876$40487$807$6893$9153$6895$4216..
```

```
# Remove $ from Revenue column  
sales['Revenue'] = sales['Revenue'].str.strip('$')  
sales['Revenue'] = sales['Revenue'].astype('int')
```

```
# Verify that Revenue is now an integer  
assert sales['Revenue'].dtype == 'int'
```

The assert statement

```
# This will pass  
assert 1+1 == 2
```

```
# This will not pass  
assert 1+1 == 3
```

```
AssertionError
```

```
    assert 1+1 == 3
```

```
AssertionError:
```

```
Traceback (most recent call last)
```

Numeric or categorical?

```
...    marriage_status    ...  
...          3    ...  
...          1    ...  
...          2    ...
```

0 = Never married

1 = Married

2 = Separated

3 = Divorced

```
df['marriage_status'].describe()
```

```
marriage_status  
...  
mean           1.4  
std            0.20  
min            0.00  
50%           1.8 ...
```

Numeric or categorical?

```
# Convert to categorical  
df["marriage_status"] = df["marriage_status"].astype('category')  
df.describe()
```

```
marriage_status  
count      241  
unique       4  
top         1  
freq      120
```

Let's practice!

CLEANING DATA IN PYTHON

Data range constraints

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @ DataCamp

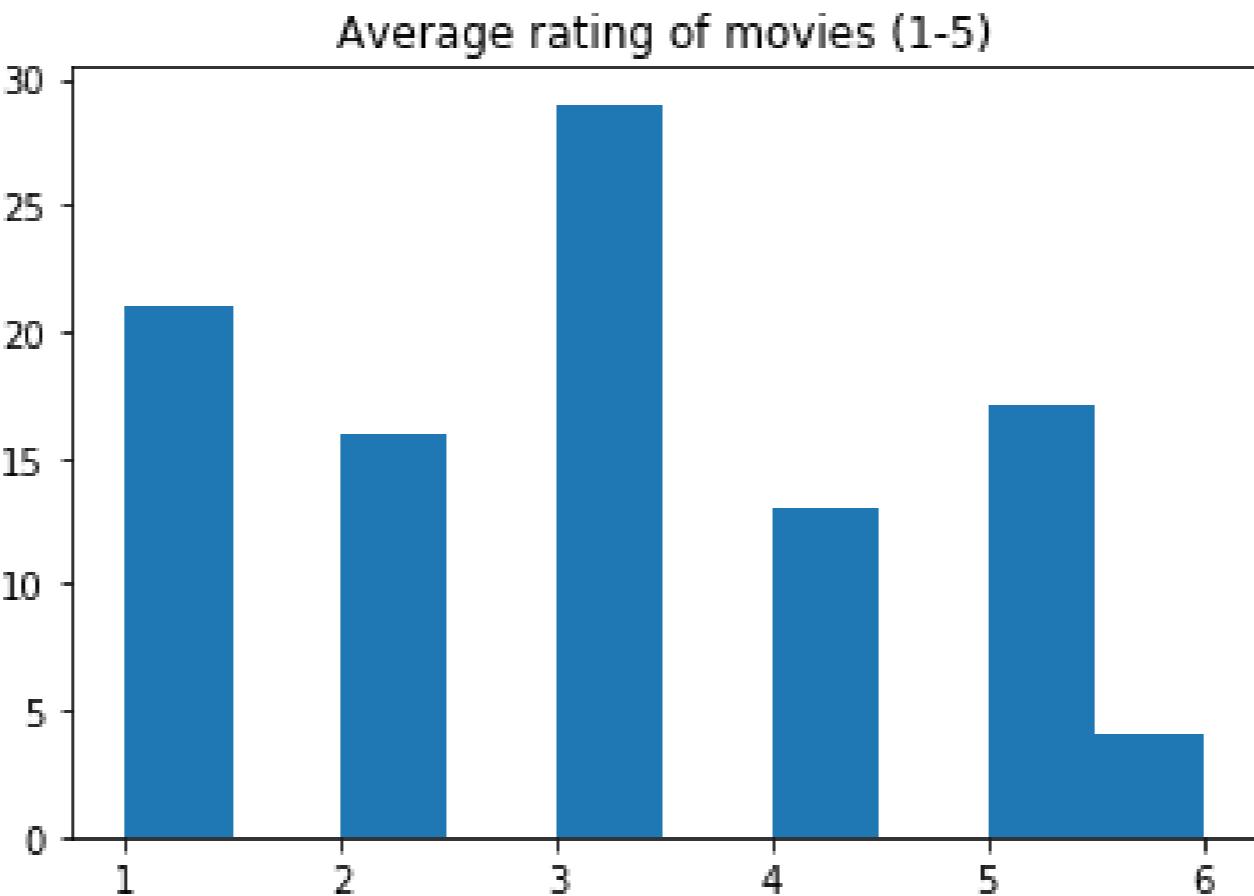
Motivation

```
movies.head()
```

```
      movie_name      avg_rating  
0   The Godfather        5  
1   Frozen 2            3  
2    Shrek              4  
...  
...
```

Motivation

```
import matplotlib.pyplot as plt  
plt.hist(movies['avg_rating'])  
plt.title('Average rating of movies (1-5)')
```



Motivation

Can future sign-ups exist?

```
# Import date time
import datetime as dt
today_date = dt.date.today()
user_signups[user_signups['subscription_date'] > dt.date.today()]
```

	subscription_date	user_name	...	Country
0	01/05/2021	Marah	...	Nauru
1	09/08/2020	Joshua	...	Austria
2	04/01/2020	Heidi	...	Guinea
3	11/10/2020	Rina	...	Turkmenistan
4	11/07/2020	Christine	...	Marshall Islands
5	07/07/2020	Ayanna	...	Gabon

How to deal with out of range data?

- Dropping data
- Setting custom minimums and maximums
- Treat as missing and impute
- Setting custom value depending on business assumptions

Movie example

```
import pandas as pd  
# Output Movies with rating > 5  
movies[movies['avg_rating'] > 5]
```

	movie_name	avg_rating
23	A Beautiful Mind	6
65	La Vita e Bella	6
77	Amelie	6

```
# Drop values using filtering  
movies = movies[movies['avg_rating'] <= 5]  
# Drop values using .drop()  
movies.drop(movies[movies['avg_rating'] > 5].index, inplace = True)  
# Assert results  
assert movies['avg_rating'].max() <= 5
```

Movie example

```
# Convert avg_rating > 5 to 5  
movies.loc[movies['avg_rating'] > 5, 'avg_rating'] = 5
```

```
# Assert statement  
assert movies['avg_rating'].max() <= 5
```

Remember, no output means it passed

Date range example

```
import datetime as dt
import pandas as pd
# Output data types
user_signups.dtypes
```

```
subscription_date    object
user_name            object
Country              object
dtype: object
```

```
# Convert to date
user_signups['subscription_date'] = pd.to_datetime(user_signups['subscription_date']).dt.date
```

Date range example

```
today_date = dt.date.today()
```

Drop the data

```
# Drop values using filtering
user_signups = user_signups[user_signups['subscription_date'] < today_date]
# Drop values using .drop()
user_signups.drop(user_signups[user_signups['subscription_date'] > today_date].index, inplace = True)
```

Hardcode dates with upper limit

```
# Drop values using filtering
user_signups.loc[user_signups['subscription_date'] > today_date, 'subscription_date'] = today_date
# Assert is true
assert user_signups.subscription_date.max().date() <= today_date
```

Let's practice!

CLEANING DATA IN PYTHON

Uniqueness constraints

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @ DataCamp

What are duplicate values?

All columns have the same values

first_name	last_name	address	height	weight
Justin	Saddlemeyer	Boulevard du Jardin Botanique 3, Bruxelles	193 cm	87 kg
Justin	Saddlemeyer	Boulevard du Jardin Botanique 3, Bruxelles	193 cm	87 kg

What are duplicate values?

Most columns have the same values

first_name	last_name	address	height	weight
Justin	Saddlemeyer	Boulevard du Jardin Botanique 3, Bruxelles	193 cm	87 kg
Justin	Saddlemeyer	Boulevard du Jardin Botanique 3, Bruxelles	194 cm	87 kg

Why do they happen?



**Data Entry &
Human Error**

Why do they happen?



**Data Entry &
Human Error**

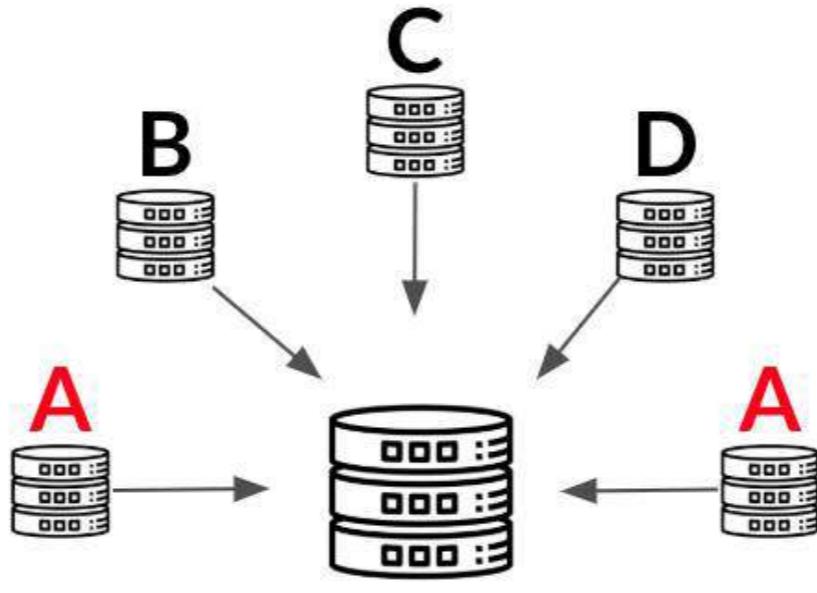


**Bugs and design
errors**

Why do they happen?



Data Entry &
Human Error



Join or merge
Errors



Bugs and design
errors

How to find duplicate values?

```
# Print the header  
height_weight.head()
```

	first_name	last_name	address	height	weight
0	Lane	Reese	534-1559 Nam St.	181	64
1	Ivor	Pierce	102-3364 Non Road	168	66
2	Roary	Gibson	P.O. Box 344, 7785 Nisi Ave	191	99
3	Shannon	Little	691-2550 Consectetuer Street	185	65
4	Abdul	Fry	4565 Risus St.	169	65

How to find duplicate values?

```
# Get duplicates across all columns  
duplicates = height_weight.duplicated()  
print(duplicates)
```

```
1      False  
...     ...  
22     True  
23     False  
...     ...
```

How to find duplicate values?

```
# Get duplicate rows  
duplicates = height_weight.duplicated()  
height_weight[duplicates]
```

	first_name	last_name	address	height	weight
100	Mary	Colon	4674 Ut Rd.	179	75
101	Ivor	Pierce	102-3364 Non Road	168	88
102	Cole	Palmer	8366 At, Street	178	91
103	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.	196	83

How to find duplicate rows?

The `.duplicated()` method

`subset` : List of column names to check for duplication.

`keep` : Whether to keep `first` ('first'), `last` ('last') or `all` (`False`) duplicate values.

```
# Column names to check for duplication
column_names = ['first_name', 'last_name', 'address']
duplicates = height_weight.duplicated(subset = column_names, keep = False)
```

How to find duplicate rows?

```
# Output duplicate values  
height_weight[duplicates]
```

	first_name	last_name		address	height	weight
1	Ivor	Pierce		102-3364 Non Road	168	66
22	Cole	Palmer		8366 At, Street	178	91
28	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.		195	83
37	Mary	Colon		4674 Ut Rd.	179	75
100	Mary	Colon		4674 Ut Rd.	179	75
101	Ivor	Pierce		102-3364 Non Road	168	88
102	Cole	Palmer		8366 At, Street	178	91
103	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.		196	83

How to find duplicate rows?

```
# Output duplicate values  
height_weight[duplicates].sort_values(by = 'first_name')
```

	first_name	last_name		address	height	weight
22	Cole	Palmer		8366 At, Street	178	91
102	Cole	Palmer		8366 At, Street	178	91
28	Desirae	Shannon	P.O. Box 643, 5251	Consectetuer, Rd.	195	83
103	Desirae	Shannon	P.O. Box 643, 5251	Consectetuer, Rd.	196	83
1	Ivor	Pierce		102-3364 Non Road	168	66
101	Ivor	Pierce		102-3364 Non Road	168	88
37	Mary	Colon		4674 Ut Rd.	179	75
100	Mary	Colon		4674 Ut Rd.	179	75

How to find duplicate rows?

```
# Output duplicate values  
height_weight[duplicates].sort_values(by = 'first_name')
```

	first_name	last_name		address	height	weight
22	Cole	Palmer		8366 At, Street	178	91
102	Cole	Palmer		8366 At, Street	178	91
28	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.		195	83
103	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.		196	83
1	Ivor	Pierce		102-3364 Non Road	168	66
101	Ivor	Pierce		102-3364 Non Road	168	88
37	Mary	Colon		4674 Ut Rd.	179	75
100	Mary	Colon		4674 Ut Rd.	179	75

How to find duplicate rows?

```
# Output duplicate values  
height_weight[duplicates].sort_values(by = 'first_name')
```

	first_name	last_name			address	height	weight
22	Cole	Palmer			8366 At, Street	178	91
102	Cole	Palmer			8366 At, Street	178	91
28	Desirae	Shannon	P.O. Box 643, 5251	Consectetuer, Rd.		195	83
103	Desirae	Shannon	P.O. Box 643, 5251	Consectetuer, Rd.		196	83
1	Ivor	Pierce		102-3364 Non Road		168	66
101	Ivor	Pierce		102-3364 Non Road		168	88
37	Mary	Colon		4674 Ut Rd.		179	75
100	Mary	Colon		4674 Ut Rd.		179	75

How to treat duplicate values?

```
# Output duplicate values  
height_weight[duplicates].sort_values(by = 'first_name')
```

	first_name	last_name		address	height	weight
22	Cole	Palmer		8366 At, Street	178	91
102	Cole	Palmer		8366 At, Street	178	91
28	Desirae	Shannon	P.O. Box 643, 5251	Consectetuer, Rd.	195	83
103	Desirae	Shannon	P.O. Box 643, 5251	Consectetuer, Rd.	196	83
1	Ivor	Pierce		102-3364 Non Road	168	66
101	Ivor	Pierce		102-3364 Non Road	168	88
37	Mary	Colon		4674 Ut Rd.	179	75
100	Mary	Colon		4674 Ut Rd.	179	75

How to treat duplicate values?

The `.drop_duplicates()` method

`subset` : List of column names to check for duplication.

`keep` : Whether to keep `first` ('`first`'), `last` ('`last`') or `all` (`False`) duplicate values.

`inplace` : Drop duplicated rows directly inside DataFrame without creating new object (`True`).

```
# Drop duplicates  
height_weight.drop_duplicates(inplace = True)
```

How to treat duplicate values?

```
# Output duplicate values  
column_names = ['first_name', 'last_name', 'address']  
duplicates = height_weight.duplicated(subset = column_names, keep = False)  
height_weight[duplicates].sort_values(by = 'first_name')
```

	first_name	last_name		address	height	weight
28	Desirae	Shannon	P.O. Box 643, 5251	Consectetuer, Rd.	195	83
103	Desirae	Shannon	P.O. Box 643, 5251	Consectetuer, Rd.	196	83
1	Ivor	Pierce		102-3364 Non Road	168	66
101	Ivor	Pierce		102-3364 Non Road	168	88

How to treat duplicate values?

```
# Output duplicate values  
column_names = ['first_name', 'last_name', 'address']  
duplicates = height_weight.duplicated(subset = column_names, keep = False)  
height_weight[duplicates].sort_values(by = 'first_name')
```

	first_name	last_name	address	height	weight
28	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.	195	83
103	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.	196	83
1	Ivor	Pierce	102-3364 Non Road	168	66
101	Ivor	Pierce	102-3364 Non Road	168	88

How to treat duplicate values?

The `.groupby()` and `.agg()` methods

```
# Group by column names and produce statistical summaries
column_names = ['first_name', 'last_name', 'address']
summaries = {'height': 'max', 'weight': 'mean'}
height_weight = height_weight.groupby(by = column_names).agg(summaries).reset_index()
# Make sure aggregation is done
duplicates = height_weight.duplicated(subset = column_names, keep = False)
height_weight[duplicates].sort_values(by = 'first_name')
```

first_name	last_name	address	height	weight
------------	-----------	---------	--------	--------

Let's practice!

CLEANING DATA IN PYTHON

Membership constraints

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @DataCamp

Chapter 2 - Text and categorical data problems

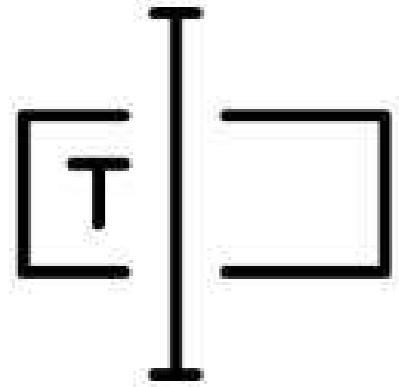
Categories and membership constraints

Predefined finite set of categories

Type of data	Example values	Numeric representation
Marriage Status	unmarried , married	0 , 1
Household Income Category	0-20K , 20-40K , ...	0 , 1 , ..
Loan Status	default , payed , no_loan	0 , 1 , 2

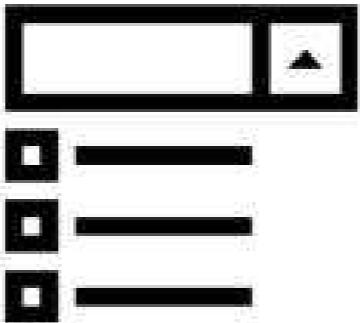
*Marriage status can **only** be unmarried _or_ married*

Why could we have these problems?

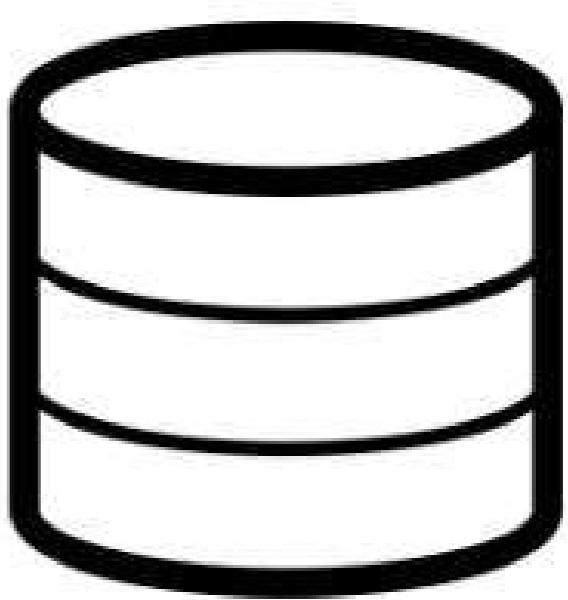


Free text

Or



Dropdowns



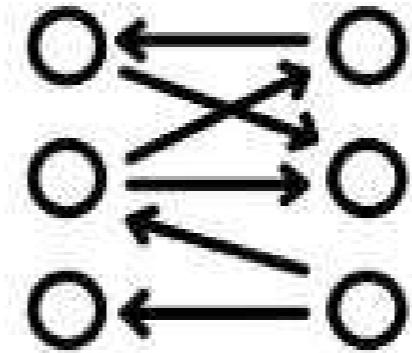
Data Entry Errors

Parsing Errors

How do we treat these problems?



*Dropping
Data*



*Remapping
Categories*



*Inferring
Categories*

An example

```
# Read study data and print it  
study_data = pd.read_csv('study.csv')  
study_data
```

```
name    birthday blood_type  
1      Beth    2019-10-20      B-  
2 Ignatius 2020-07-08      A-  
3      Paul    2019-08-12      O+  
4      Helen   2019-03-17      O-  
5 Jennifer 2019-12-17      Z+  
6 Kennedy  2020-04-27      A+  
7      Keith   2019-04-19     AB+
```

```
# Correct possible blood types  
categories
```

	blood_type
1	O-
2	O+
3	A-
4	A+
5	B+
6	B-
7	AB+
8	AB-

An example

```
# Read study data and print it  
study_data = pd.read_csv('study.csv')  
study_data
```

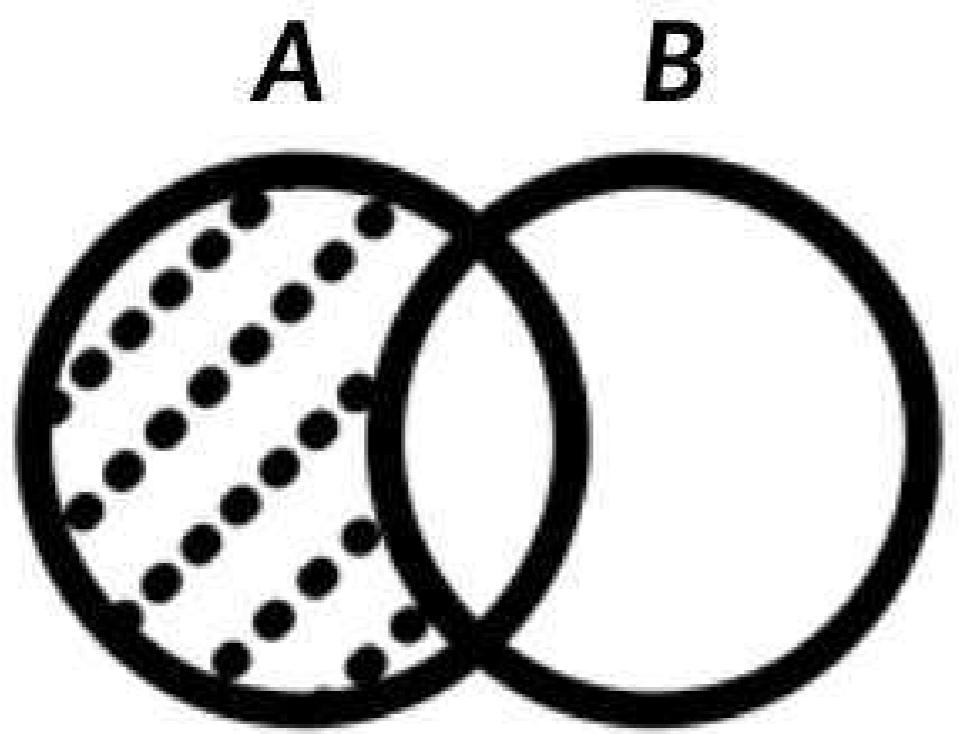
	name	birthday	blood_type
1	Beth	2019-10-20	B-
2	Ignatius	2020-07-08	A-
3	Paul	2019-08-12	O+
4	Helen	2019-03-17	O-
5	Jennifer	2019-12-17	Z+ <--
6	Kennedy	2020-04-27	A+
7	Keith	2019-04-19	AB+

```
# Correct possible blood types  
categories
```

	blood_type
1	O-
2	O+
3	A-
4	A+
5	B+
6	B-
7	AB+
8	AB-

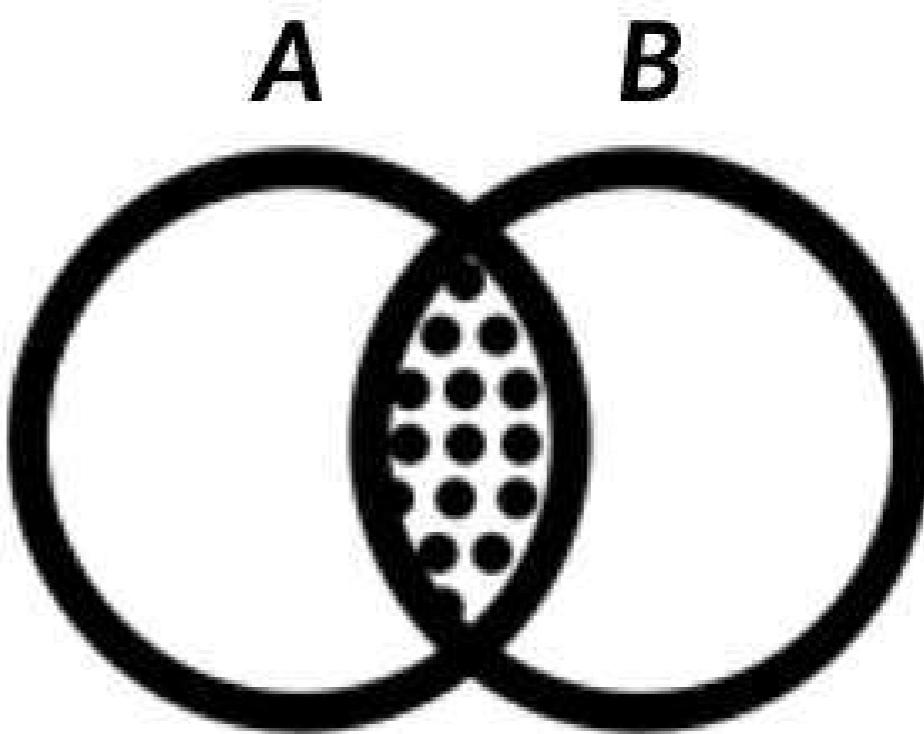
A note on joins

Anti Joins



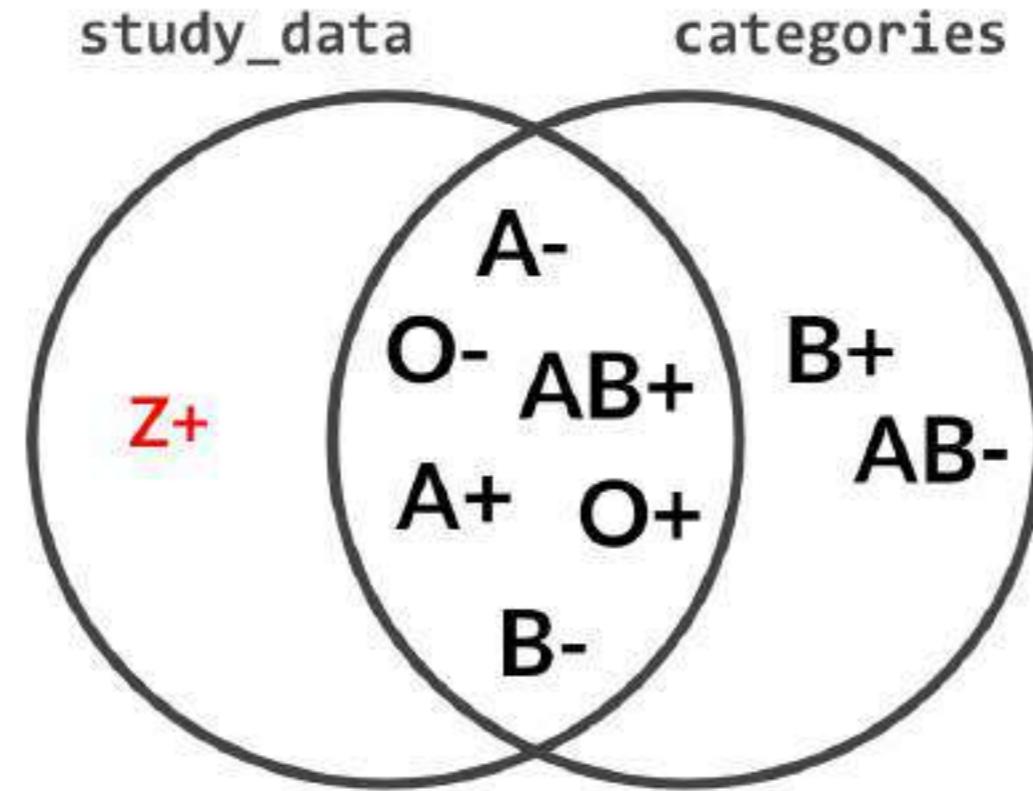
What is in A and not in B

Inner Joins



What is in both A and B

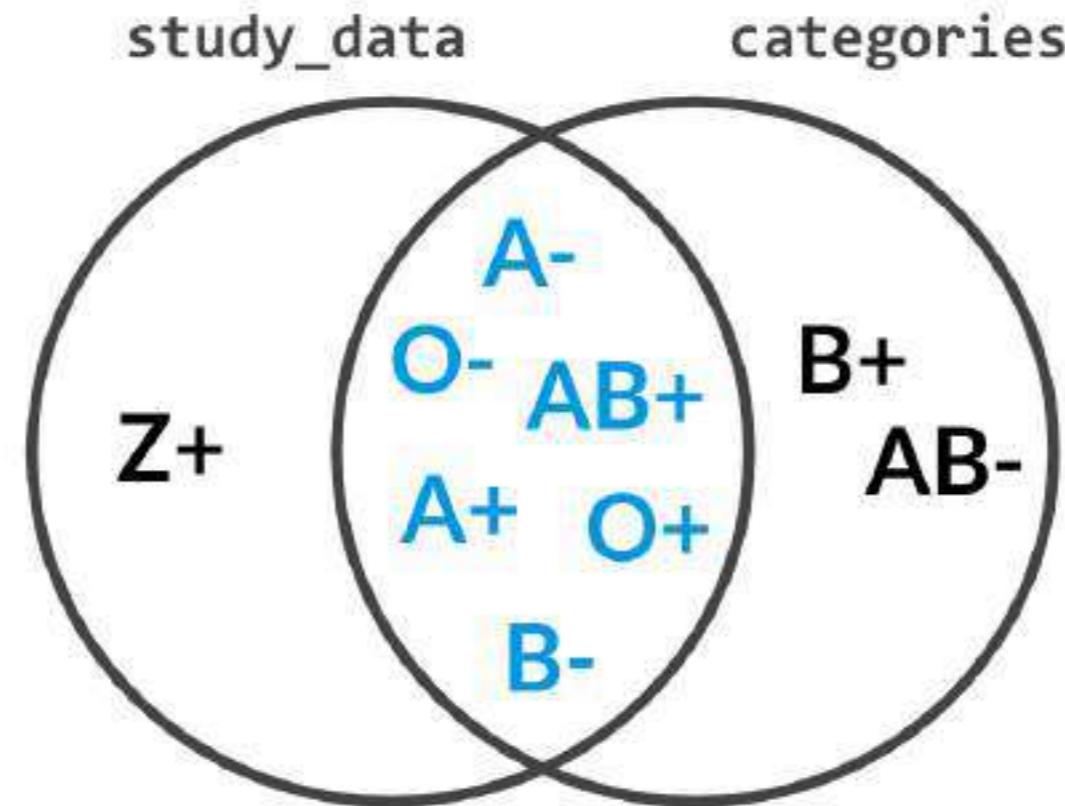
A left anti join on blood types



What is in study_data only

*Returns only rows
containing Z+*

An inner join on blood types



What is in study_data and categories only

*Returns all the rows except those
containing Z+, B+ and AB-*

Finding inconsistent categories

```
inconsistent_categories = set(study_data['blood_type']).difference(categories['blood_type'])  
print(inconsistent_categories)
```

```
{'Z+'}
```

```
# Get and print rows with inconsistent categories  
inconsistent_rows = study_data['blood_type'].isin(inconsistent_categories)  
study_data[inconsistent_rows]
```

```
   name    birthday blood_type  
5 Jennifer 2019-12-17          Z+
```

Dropping inconsistent categories

```
inconsistent_categories = set(study_data['blood_type']).difference(categories['blood_type'])
inconsistent_rows = study_data['blood_type'].isin(inconsistent_categories)
inconsistent_data = study_data[inconsistent_rows]
# Drop inconsistent categories and get consistent data only
consistent_data = study_data[~inconsistent_rows]
```

		name	birthday	blood_type
1		Beth	2019-10-20	B-
2		Ignatius	2020-07-08	A-
3		Paul	2019-08-12	O+
4		Helen	2019-03-17	O-
...	

Let's practice!

CLEANING DATA IN PYTHON

Categorical variables

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @DataCamp

What type of errors could we have?

I) Value inconsistency

- *Inconsistent fields:* 'married' , 'Maried' , 'UNMARRIED' , 'not married' ..
- *_Trailing white spaces:* _ 'married ' , ' married ' ..

II) Collapsing too many categories to few

- *Creating new groups:* 0-20K , 20-40K categories ... from continuous household income data
- *Mapping groups to new ones:* Mapping household income categories to 2 'rich' , 'poor'

III) Making sure data is of type category (seen in Chapter 1)

Value consistency

Capitalization: 'married' , 'Married' , 'UNMARRIED' , 'unmarried' ..

```
# Get marriage status column  
marriage_status = demographics['marriage_status']  
marriage_status.value_counts()
```

```
unmarried      352  
married        268  
MARRIED       204  
UNMARRIED     176  
dtype: int64
```

Value consistency

```
# Get value counts on DataFrame  
marriage_status.groupby('marriage_status').count()
```

	household_income	gender
marriage_status		
MARRIED	204	204
UNMARRIED	176	176
married	268	268
unmarried	352	352

Value consistency

```
# Capitalize  
marriage_status['marriage_status'] = marriage_status['marriage_status'].str.upper()  
marriage_status['marriage_status'].value_counts()
```

```
UNMARRIED      528  
MARRIED        472
```

```
# Lowercase  
marriage_status['marriage_status'] = marriage_status['marriage_status'].str.lower()  
marriage_status['marriage_status'].value_counts()
```

```
unmarried      528  
married        472
```

Value consistency

Trailing spaces: 'married ' , 'married' , 'unmarried' , ' unmarried' ..

```
# Get marriage status column  
marriage_status = demographics['marriage_status']  
marriage_status.value_counts()
```

```
unmarried    352  
unmarried    268  
married      204  
married      176  
dtype: int64
```

Value consistency

```
# Strip all spaces  
demographics = demographics['marriage_status'].str.strip()  
demographics['marriage_status'].value_counts()
```

```
unmarried      528  
married        472
```

Collapsing data into categories

Create categories out of data: `income_group` column from `income` column.

```
# Using qcut()
import pandas as pd
group_names = ['0-200K', '200K-500K', '500K+']
demographics['income_group'] = pd.qcut(demographics['household_income'], q = 3,
                                         labels = group_names)

# Print income_group column
demographics[['income_group', 'household_income']]
```

```
category    household_income
0      200K-500K     189243
1        500K+     778533
..
```

Collapsing data into categories

Create categories out of data: `income_group` column from `income` column.

```
# Using cut() - create category ranges and names
ranges = [0,200000,500000,np.inf]
group_names = ['0-200K', '200K-500K', '500K+']
# Create income group column
demographics['income_group'] = pd.cut(demographics['household_income'], bins=ranges,
                                         labels=group_names)
demographics[['income_group', 'household_income']]
```

	category	Income
0	0-200K	189243
1	500K+	778533

Collapsing data into categories

Map categories to fewer ones: reducing categories in categorical column.

operating_system column is: 'Microsoft', 'MacOS', 'IOS', 'Android', 'Linux'

operating_system column should become: 'DesktopOS', 'MobileOS'

```
# Create mapping dictionary and replace
mapping = {'Microsoft':'DesktopOS', 'MacOS':'DesktopOS', 'Linux':'DesktopOS',
           'IOS':'MobileOS', 'Android':'MobileOS'}
devices['operating_system'] = devices['operating_system'].replace(mapping)
devices['operating_system'].unique()
```

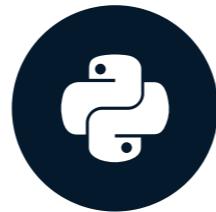
```
array(['DesktopOS', 'MobileOS'], dtype=object)
```

Let's practice!

CLEANING DATA IN PYTHON

Cleaning text data

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @ DataCamp

What is text data?

Type of data	Example values
Names	Alex , Sara ...
Phone numbers	+96171679912 ...
Emails	`adel@datacamp.com`..
Passwords	...

Common text data problems

1) *Data inconsistency:*

+96171679912 or 0096171679912 or ..?

2) *Fixed length violations:*

Passwords needs to be at least 8 characters

3) *Typos:*

+961.71.679912

Example

```
phones = pd.read_csv('phones.csv')  
print(phones)
```

	Full name	Phone number
0	Noelani A. Gray	001-702-397-5143
1	Myles Z. Gomez	001-329-485-0540
2	Gil B. Silva	001-195-492-2338
3	Prescott D. Hardin	+1-297-996-4904
4	Benedict G. Valdez	001-969-820-3536
5	Reece M. Andrews	4138
6	Hayfa E. Keith	001-536-175-8444
7	Hedley I. Logan	001-681-552-1823
8	Jack W. Carrillo	001-910-323-5265
9	Lionel M. Davis	001-143-119-9210

Example

```
phones = pd.read_csv('phones.csv')  
print(phones)
```

	Full name	Phone number	
0	Noelani A. Gray	001-702-397-5143	
1	Myles Z. Gomez	001-329-485-0540	
2	Gil B. Silva	001-195-492-2338	
3	Prescott D. Hardin	+1-297-996-4904	<-- Inconsistent data format
4	Benedict G. Valdez	001-969-820-3536	
5	Reece M. Andrews	4138	<-- Length violation
6	Hayfa E. Keith	001-536-175-8444	
7	Hedley I. Logan	001-681-552-1823	
8	Jack W. Carrillo	001-910-323-5265	
9	Lionel M. Davis	001-143-119-9210	

Example

```
phones = pd.read_csv('phones.csv')  
print(phones)
```

	Full name	Phone number
0	Noelani A. Gray	0017023975143
1	Myles Z. Gomez	0013294850540
2	Gil B. Silva	0011954922338
3	Prescott D. Hardin	0012979964904
4	Benedict G. Valdez	0019698203536
5	Reece M. Andrews	NaN
6	Hayfa E. Keith	0015361758444
7	Hedley I. Logan	0016815521823
8	Jack W. Carrillo	0019103235265
9	Lionel M. Davis	0011431199210

Fixing the phone number column

```
# Replace "+" with "00"
phones["Phone number"] = phones["Phone number"].str.replace("+", "00")
phones
```

	Full name	Phone number
0	Noelani A. Gray	001-702-397-5143
1	Myles Z. Gomez	001-329-485-0540
2	Gil B. Silva	001-195-492-2338
3	Prescott D. Hardin	001-297-996-4904
4	Benedict G. Valdez	001-969-820-3536
5	Reece M. Andrews	4138
6	Hayfa E. Keith	001-536-175-8444
7	Hedley I. Logan	001-681-552-1823
8	Jack W. Carrillo	001-910-323-5265
9	Lionel M. Davis	001-143-119-9210

Fixing the phone number column

```
# Replace "--" with nothing
phones["Phone number"] = phones["Phone number"].str.replace("--", "")
phones
```

	Full name	Phone number
0	Noelani A. Gray	0017023975143
1	Myles Z. Gomez	0013294850540
2	Gil B. Silva	0011954922338
3	Prescott D. Hardin	0012979964904
4	Benedict G. Valdez	0019698203536
5	Reece M. Andrews	4138
6	Hayfa E. Keith	0015361758444
7	Hedley I. Logan	0016815521823
8	Jack W. Carrillo	0019103235265
9	Lionel M. Davis	0011431199210

Fixing the phone number column

```
# Replace phone numbers with lower than 10 digits to NaN  
digits = phones['Phone number'].str.len()  
phones.loc[digits < 10, "Phone number"] = np.nan  
phones
```

	Full name	Phone number
0	Noelani A. Gray	0017023975143
1	Myles Z. Gomez	0013294850540
2	Gil B. Silva	0011954922338
3	Prescott D. Hardin	0012979964904
4	Benedict G. Valdez	0019698203536
5	Reece M. Andrews	NaN
6	Hayfa E. Keith	0015361758444
7	Hedley I. Logan	0016815521823
8	Jack W. Carrillo	0019103235265
9	Lionel M. Davis	0011431199210

Fixing the phone number column

```
# Find length of each row in Phone number column  
sanity_check = phone['Phone number'].str.len()
```

```
# Assert minimum phone number length is 10  
assert sanity_check.min() >= 10
```

```
# Assert all numbers do not have "+" or "-"  
assert phone['Phone number'].str.contains("+|-").any() == False
```

Remember, assert returns nothing if the condition passes

But what about more complicated examples?

```
phones.head()
```

	Full name	Phone number
0	Olga Robinson	+ (01706) - 25891
1	Justina Kim	+ 0500 - 571437
2	Tamekah Henson	+ 0800 - 1111
3	Miranda Solis	+ 07058 - 879063
4	Caldwell Gilliam	+ (016977) - 8424

Supercharged control + F

Regular expressions in action

```
# Replace letters with nothing
phones['Phone number'] = phones['Phone number'].str.replace(r'\D+', '')
phones.head()
```

	Full name	Phone number
0	Olga Robinson	0170625891
1	Justina Kim	0500571437
2	Tamekah Henson	08001111
3	Miranda Solis	07058879063
4	Caldwell Gilliam	0169778424

Let's practice!

CLEANING DATA IN PYTHON

Uniformity

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @ DataCamp

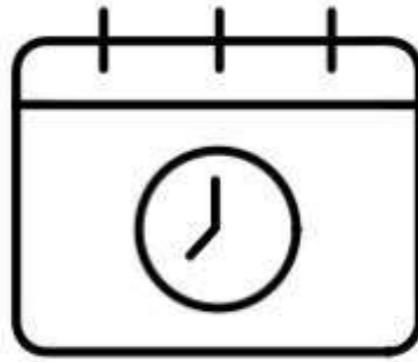
In this chapter

Chapter 3 - Advanced data problems

Data range constraints



Out of range movie ratings



Subscription dates in the future

Uniformity

Column	Unit
Temperature	32°C is also 89.6°F
Weight	70 Kg is also 11 st.
Date	26-11-2019 is also 26, November, 2019
Money	100\$ is also 10763.90¥

An example

```
temperatures = pd.read_csv('temperature.csv')  
temperatures.head()
```

	Date	Temperature
0	03.03.19	14.0
1	04.03.19	15.0
2	05.03.19	18.0
3	06.03.19	16.0
4	07.03.19	62.6

An example

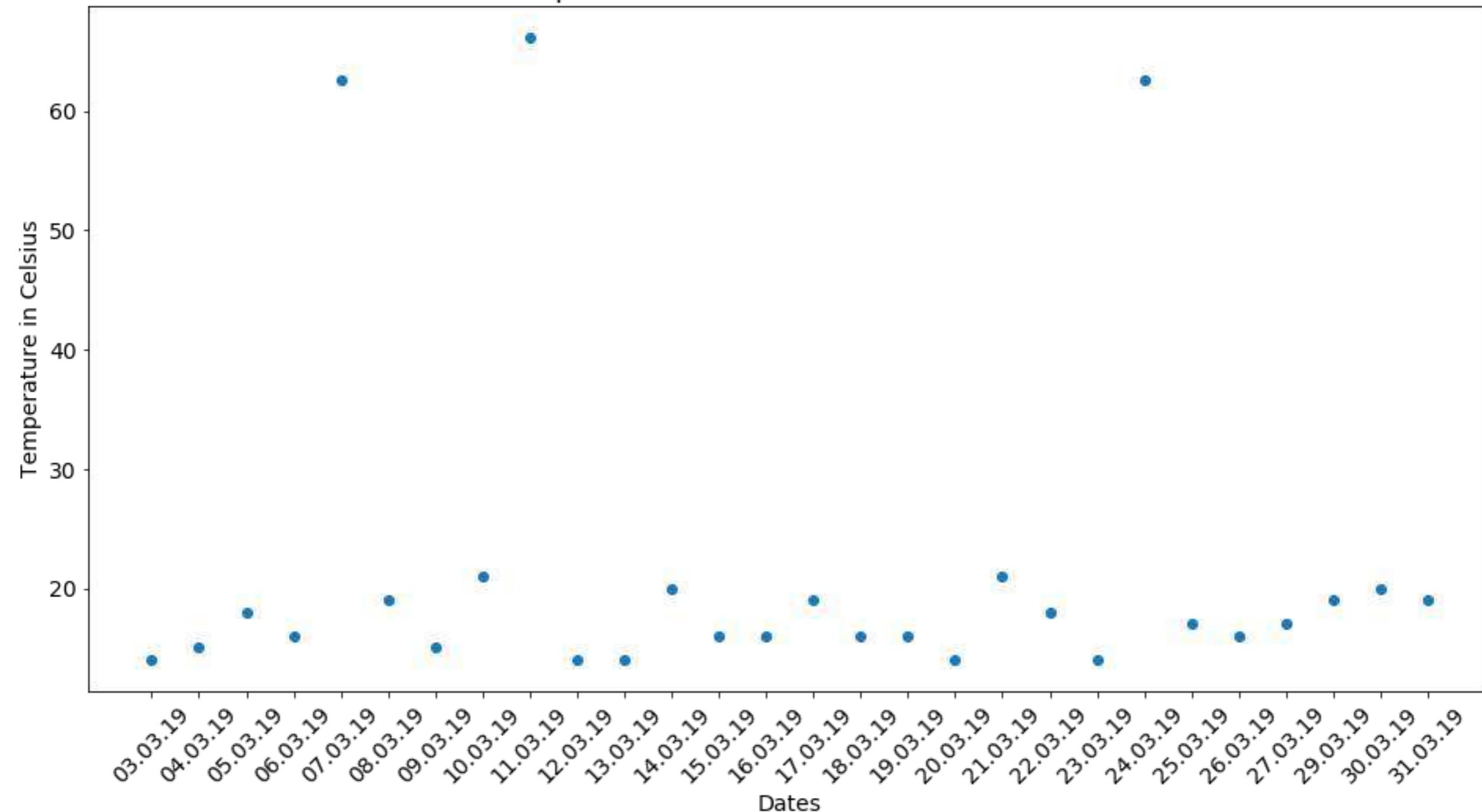
```
temperatures = pd.read_csv('temperature.csv')  
temperatures.head()
```

```
      Date  Temperature  
0  03.03.19          14.0  
1  04.03.19          15.0  
2  05.03.19          18.0  
3  06.03.19          16.0  
4  07.03.19         62.6  <--
```

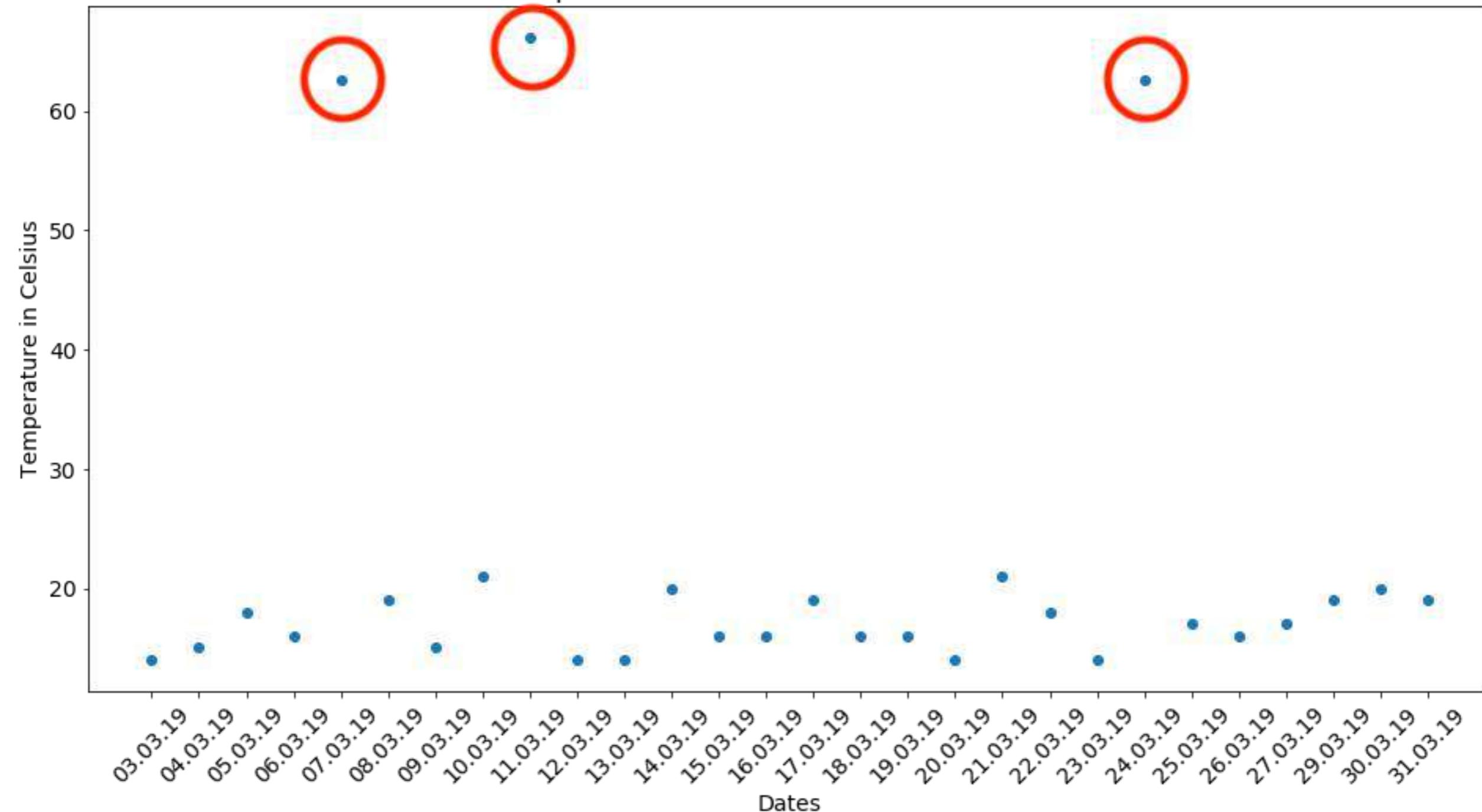
An example

```
# Import matplotlib
import matplotlib.pyplot as plt
# Create scatter plot
plt.scatter(x = 'Date', y = 'Temperature', data = temperatures)
# Create title, xlabel and ylabel
plt.title('Temperature in Celsius March 2019 - NYC')
plt.xlabel('Dates')
plt.ylabel('Temperature in Celsius')
# Show plot
plt.show()
```

Temperature in Celsius in March 2019 - NYC



Temperature in Celsius in March 2019 - NYC



Treating temperature data

$$C = (F - 32) \times \frac{5}{9}$$

```
temp_fah = temperatures.loc[temperatures['Temperature'] > 40, 'Temperature']
temp_cels = (temp_fah - 32) * (5/9)
temperatures.loc[temperatures['Temperature'] > 40, 'Temperature'] = temp_cels
```

```
# Assert conversion is correct
assert temperatures['Temperature'].max() < 40
```

Treating date data

```
birthdays.head()
```

```
Birthday First name Last name
0      27/27/19      Rowan     Nunez
1      03-29-19      Brynn      Yang
2  March 3rd, 2019    Sophia    Reilly
3      24-03-19      Deacon    Prince
4      06-03-19    Griffith     Neal
```

Treating date data

```
birthdays.head()
```

	Birthday	First name	Last name	
0	27/27/19	Rowan	Nunez	??
1	03-29-19	Brynn	Yang	MM-DD-YY
2	March 3rd, 2019	Sophia	Reilly	Month D, YYYY
3	24-03-19	Deacon	Prince	
4	06-03-19	Griffith	Neal	

Datetime formatting

`datetime` is useful for representing dates

`pandas.to_datetime()`

Date	datetime format
25-12-2019	%d-%m-%Y
December 25th 2019	%c
12-25-2019	%m-%d-%Y
...	...

- Can recognize most formats automatically
- Sometimes fails with erroneous or unrecognizable formats

Treating date data

```
# Converts to datetime - but won't work!
birthdays['Birthday'] = pd.to_datetime(birthdays['Birthday'])
```

```
ValueError: month must be in 1..12
```

```
# Will work!
birthdays['Birthday'] = pd.to_datetime(birthdays['Birthday'],
                                       # Attempt to infer format of each date
                                       infer_datetime_format=True,
                                       # Return NA for rows where conversion failed
                                       errors = 'coerce')
```

Treating date data

```
birthdays.head()
```

```
Birthday First name Last name
0          NaT      Rowan     Nunez
1 2019-03-29      Brynn      Yang
2 2019-03-03    Sophia    Reilly
3 2019-03-24   Deacon    Prince
4 2019-06-03  Griffith     Neal
```

Treating date data

```
birthdays['Birthday'] = birthdays['Birthday'].dt.strftime("%d-%m-%Y")  
birthdays.head()
```

	Birthday	First name	Last name
0	NaT	Rowan	Nunez
1	29-03-2019	Brynn	Yang
2	03-03-2019	Sophia	Reilly
3	24-03-2019	Deacon	Prince
4	03-06-2019	Griffith	Neal

Treating ambiguous date data

Is 2019-03-08 in August or March?

- Convert to NA and treat accordingly
- Infer format by understanding data source
- Infer format by understanding previous and subsequent data in DataFrame

Let's practice!

CLEANING DATA IN PYTHON

Cross field validation

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @ DataCamp

Motivation

```
import pandas as pd\n\nflights = pd.read_csv('flights.csv')\nflights.head()
```

	flight_number	economy_class	business_class	first_class	total_passengers
0	DL140	100	60	40	200
1	BA248	130	100	70	300
2	MEA124	100	50	50	200
3	AFR939	140	70	90	300
4	TKA101	130	100	20	250

Cross field validation

The use of **multiple** fields in a dataset to sanity check data integrity

	flight_number	economy_class	business_class	first_class	total_passengers
0	DL140	100	+ 60	+ 40	= 200
1	BA248	130	+ 100	+ 70	= 300
2	MEA124	100	+ 50	+ 50	= 200
3	AFR939	140	+ 70	+ 90	= 300
4	TKA101	130	+ 100	+ 20	= 250

```
sum_classes = flights[['economy_class', 'business_class', 'first_class']].sum(axis = 1)
passenger_equ = sum_classes == flights['total_passengers']
# Find and filter out rows with inconsistent passenger totals
inconsistent_pass = flights[~passenger_equ]
consistent_pass = flights[passenger_equ]
```

Cross field validation

```
users.head()
```

```
user_id    Age   Birthday
0      32985    22  1998-03-02
1      94387    27  1993-12-04
2      34236    42  1978-11-24
3      12551    31  1989-01-03
4      55212    18  2002-07-02
```

Cross field validation

```
import pandas as pd
import datetime as dt

# Convert to datetime and get today's date
users['Birthday'] = pd.to_datetime(users['Birthday'])
today = dt.date.today()
# For each row in the Birthday column, calculate year difference
age_manual = today.year - users['Birthday'].dt.year
# Find instances where ages match
age_equ = age_manual == users['Age']
# Find and filter out rows with inconsistent age
inconsistent_age = users[~age_equ]
consistent_age = users[age_equ]
```

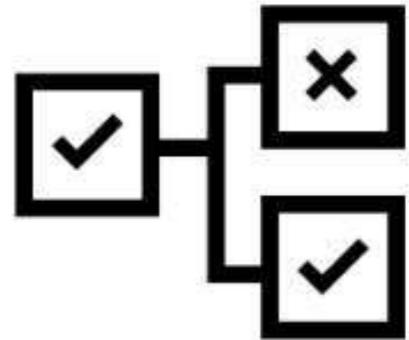
What to do when we catch inconsistencies?



*Dropping
Data*



*Set to missing
and impute*



*Apply rules from
domain knowledge*

Let's practice!

CLEANING DATA IN PYTHON

Completeness

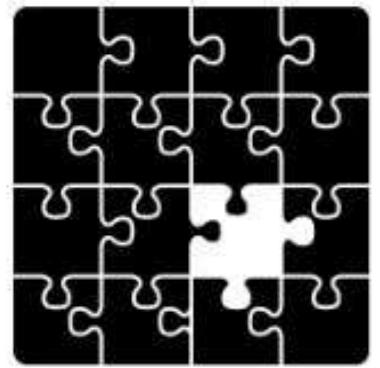
CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @ DataCamp

What is missing data?



Occurs when no data value is stored for a variable in an observation

Can be represented as NA , nan , 0 ,

Technical error

Human error

Airquality example

```
import pandas as pd  
airquality = pd.read_csv('airquality.csv')  
print(airquality)
```

	Date	Temperature	CO2
987	20/04/2004	16.8	0.0
2119	07/06/2004	18.7	0.8
2451	20/06/2004	-40.0	NaN
1984	01/06/2004	19.6	1.8
8299	19/02/2005	11.2	1.2
...

Airquality example

```
import pandas as pd  
airquality = pd.read_csv('airquality.csv')  
print(airquality)
```

	Date	Temperature	CO2	
987	20/04/2004	16.8	0.0	
2119	07/06/2004	18.7	0.8	
2451	20/06/2004	-40.0	NaN	<--
1984	01/06/2004	19.6	1.8	
8299	19/02/2005	11.2	1.2	
...	

Airquality example

```
# Return missing values  
airquality.isna()
```

	Date	Temperature	CO2
987	False	False	False
2119	False	False	False
2451	False	False	True
1984	False	False	False
8299	False	False	False

Airquality example

```
# Get summary of missingness  
airquality.isna().sum()
```

```
Date          0  
Temperature   0  
CO2         366  
dtype: int64
```

Missingno

Useful package for visualizing and understanding missing data

```
import missingno as msno
import matplotlib.pyplot as plt
# Visualize missingness
msno.matrix(airquality)
plt.show()
```

1

Date

9357

Temperature

CO₂

2

3

Airquality example

```
# Isolate missing and complete values aside  
missing = airquality[airquality['CO2'].isna()]  
complete = airquality[~airquality['CO2'].isna()]
```

Airquality example

```
# Describe complete DataFramee  
complete.describe()
```

```
Temperature          CO2  
count    8991.000000  8991.000000  
mean      18.317829   1.739584  
std       8.832116   1.537580  
min     -1.900000   0.000000  
...  
max      44.600000  11.900000
```

```
# Describe missing DataFramee  
missing.describe()
```

```
Temperature          CO2  
count    366.000000   0.0  
mean     -39.655738   NaN  
std       5.988716   NaN  
min     -49.000000   NaN  
...  
max     -30.000000   NaN
```

Airquality example

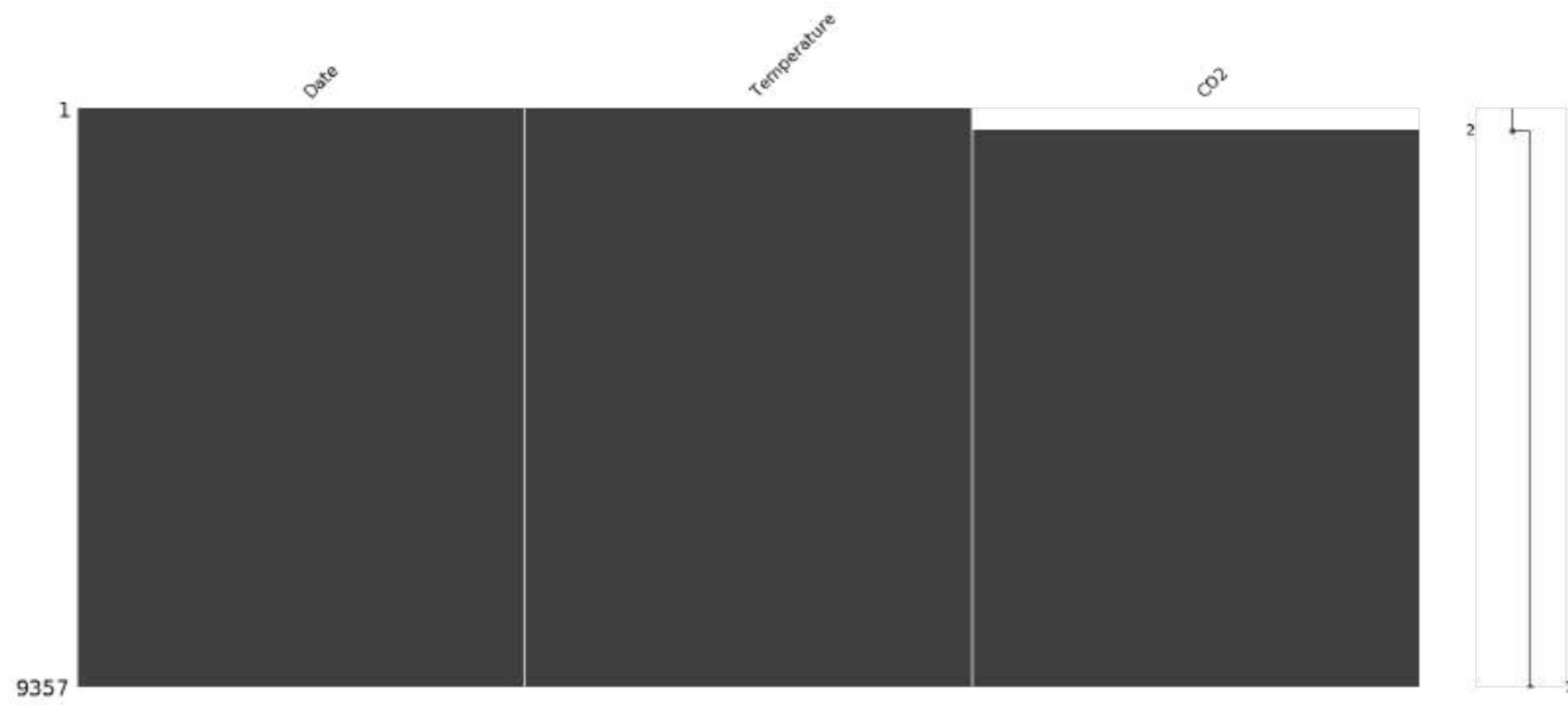
```
# Describe complete DataFramee  
complete.describe()
```

```
Temperature          CO2  
count    8991.000000  8991.000000  
mean      18.317829   1.739584  
std       8.832116   1.537580  
min     -1.900000   0.000000  
...  
max      44.600000  11.900000
```

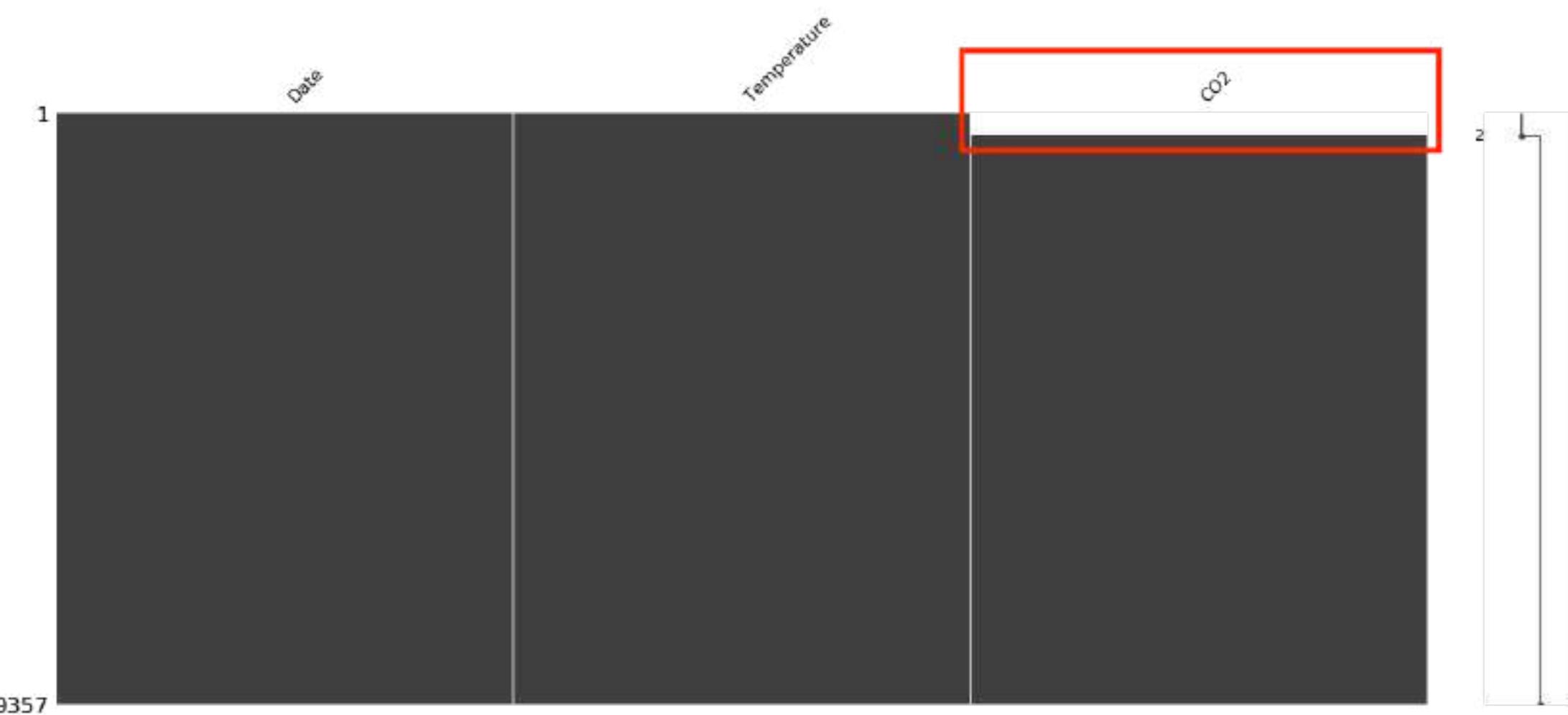
```
# Describe missing DataFramee  
missing.describe()
```

```
Temperature          CO2  
count    366.000000   0.0  
mean     -39.655738   NaN    <--  
std       5.988716   NaN  
min     -49.000000   NaN    <--  
...  
max     -30.000000   NaN    <--
```

```
sorted_airquality = airquality.sort_values(by = 'Temperature')
sns.matrix(sorted_airquality)
plt.show()
```



```
sorted_airquality = airquality.sort_values(by = 'Temperature')
msno.matrix(sorted_airquality)
plt.show()
```



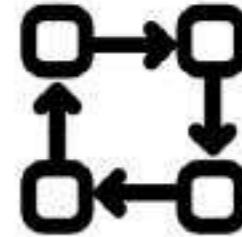
Missingness types



*Missing Completely
at Random*
(MCAR)



*Missing at
Random*
(MAR)



*Missing Not at
Random*
(MNAR)

Missingness types



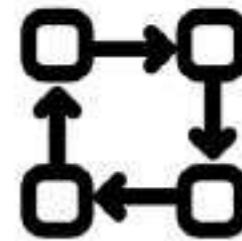
**Missing Completely
at Random**
(MCAR)

*No systematic relationship
between missing data and
other values*

*Data entry errors when
inputting data*



**Missing at
Random**
(MAR)



**Missing Not at
Random**
(MNAR)

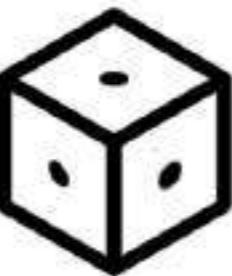
Missingness types



**Missing Completely
at Random**
(MCAR)

*No systematic relationship
between missing data and
other values*

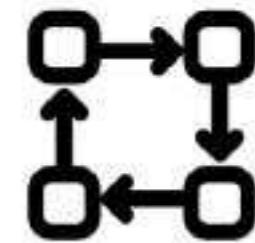
*Data entry errors when
inputting data*



**Missing at
Random**
(MAR)

*Systematic relationship
between missing data and
other observed values*

*Missing ozone data for high
temperatures*



**Missing Not at
Random**
(MNAR)

Missingness types



**Missing Completely
at Random**
(MCAR)

*No systematic relationship
between missing data and
other values*

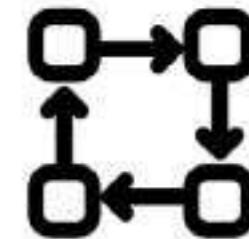
*Data entry errors when
inputting data*



**Missing at
Random**
(MAR)

*Systematic relationship
between missing data and
other observed values*

*Missing ozone data for high
temperatures*



**Missing Not at
Random**
(MNAR)

*Systematic relationship
between missing data and
unobserved values*

*Missing temperature values for
high temperatures*

How to deal with missing data?

Simple approaches:

1. Drop missing data
2. Impute with statistical measures (*mean, median, mode..*)

More complex approaches:

1. Imputing using an algorithmic approach
2. Impute with machine learning models

Dealing with missing data

```
airquality.head()
```

```
      Date  Temperature  CO2
0  05/03/2005        8.5   2.5
1  23/08/2004       21.8   0.0
2  18/02/2005        6.3   1.0
3  08/02/2005      -31.0   NaN
4  13/03/2005       19.9   0.1
```

Dropping missing values

```
# Drop missing values  
airquality_dropped = airquality.dropna(subset = ['CO2'])  
airquality_dropped.head()
```

	Date	Temperature	CO2
0	05/03/2005	8.5	2.5
1	23/08/2004	21.8	0.0
2	18/02/2005	6.3	1.0
4	13/03/2005	19.9	0.1
5	02/04/2005	17.0	0.8

Replacing with statistical measures

```
co2_mean = airquality['CO2'].mean()  
airquality_imputed = airquality.fillna({'CO2': co2_mean})  
airquality_imputed.head()
```

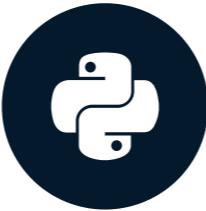
	Date	Temperature	CO2
0	05/03/2005	8.5	2.500000
1	23/08/2004	21.8	0.000000
2	18/02/2005	6.3	1.000000
3	08/02/2005	-31.0	1.739584
4	13/03/2005	19.9	0.100000

Let's practice!

CLEANING DATA IN PYTHON

Comparing strings

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @ DataCamp

In this chapter

Chapter 4 - Record linkage

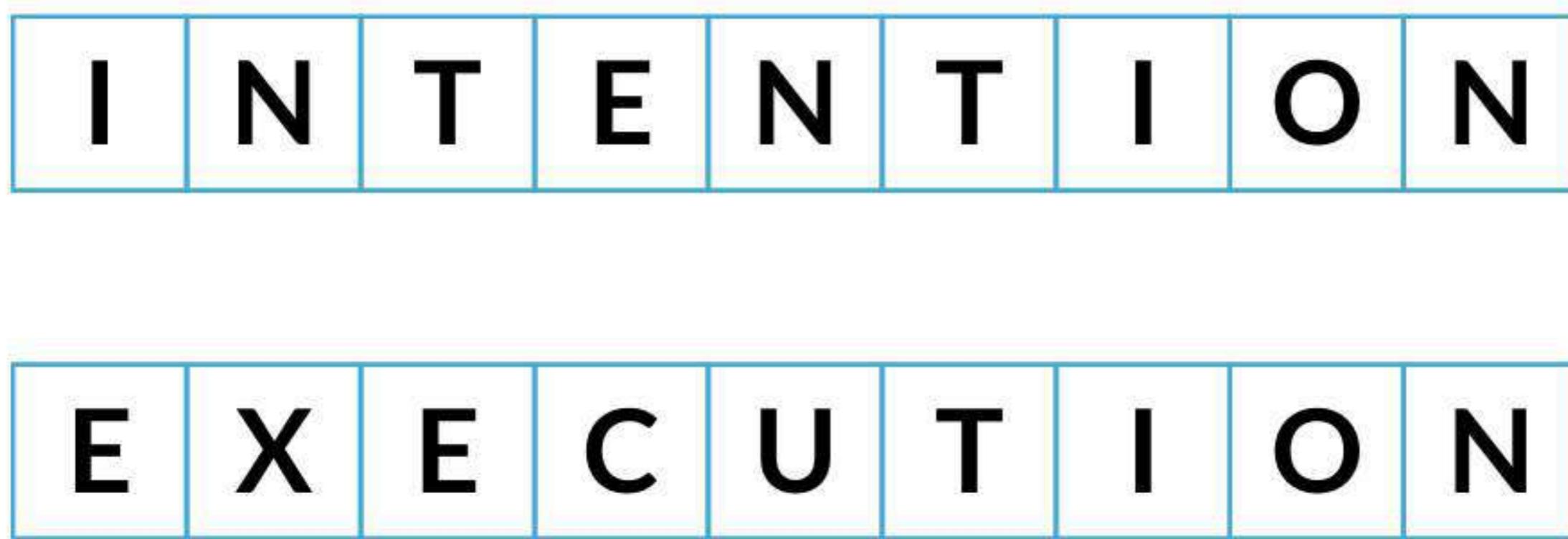
Minimum edit distance

I	N	T	E	N	T	I	O	N
---	---	---	---	---	---	---	---	---

E	X	E	C	U	T	I	O	N
---	---	---	---	---	---	---	---	---

Least possible amount of steps needed to transition from one string to another

Minimum edit distance

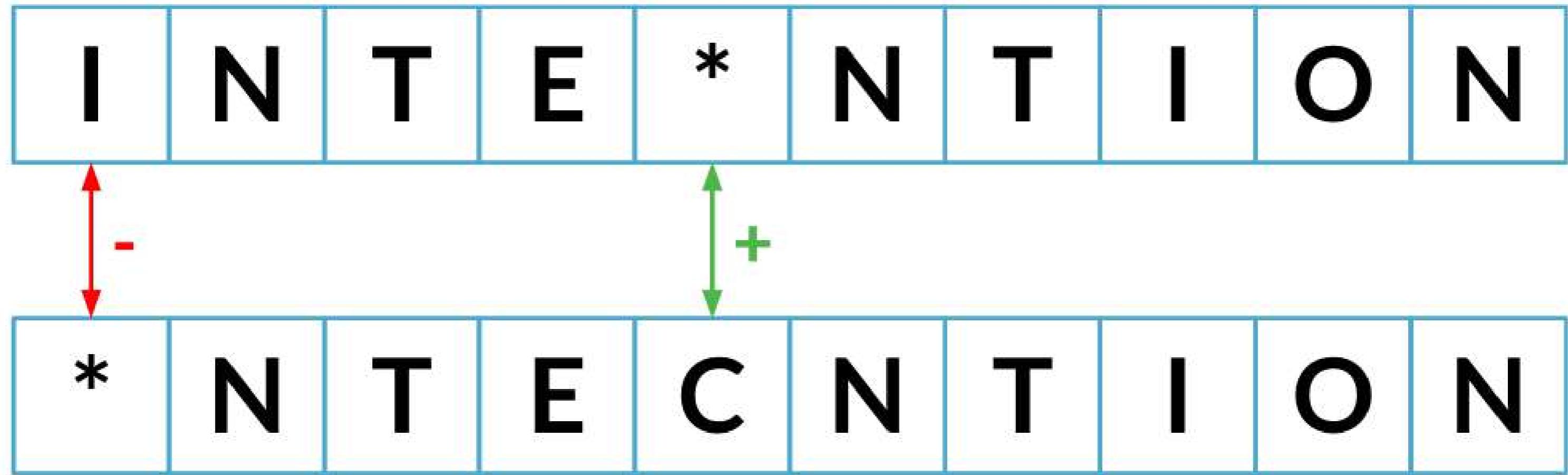


Least possible amount of steps needed to transition from one string to another

Minimum edit distance

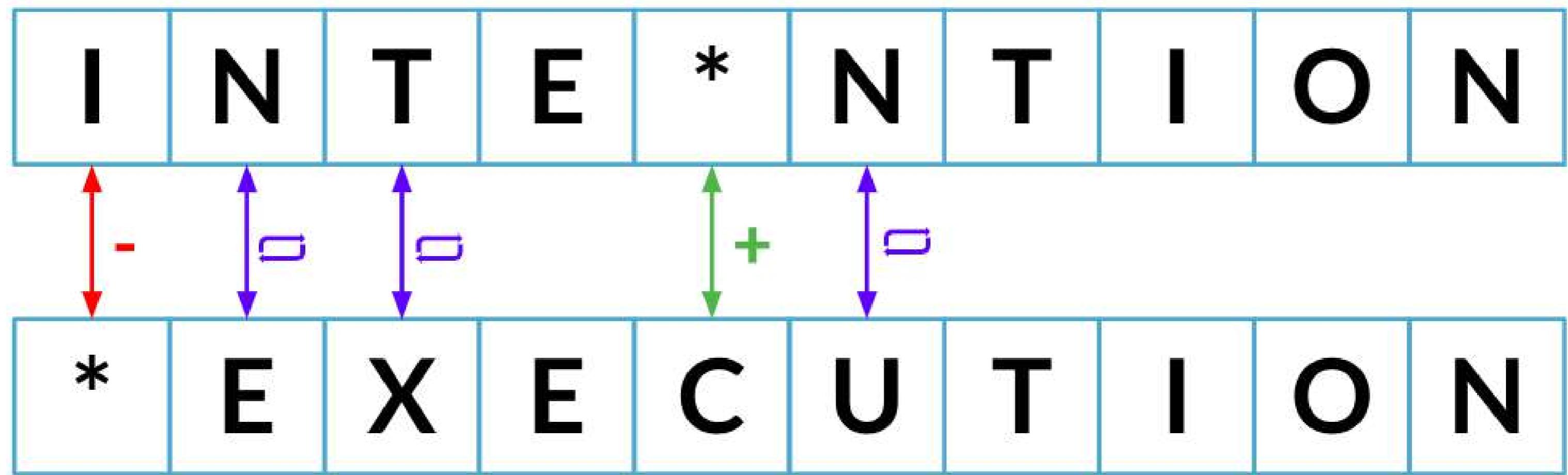
I	N	T	E	N	T	I	O	N
---	---	---	---	---	---	---	---	---

Minimum edit distance



Minimum edit distance so far: 2

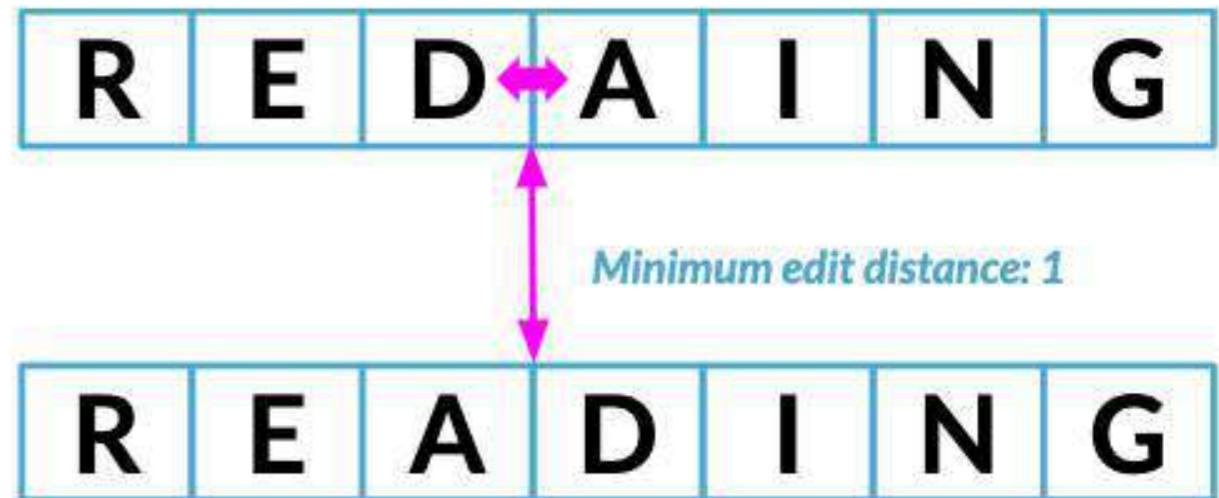
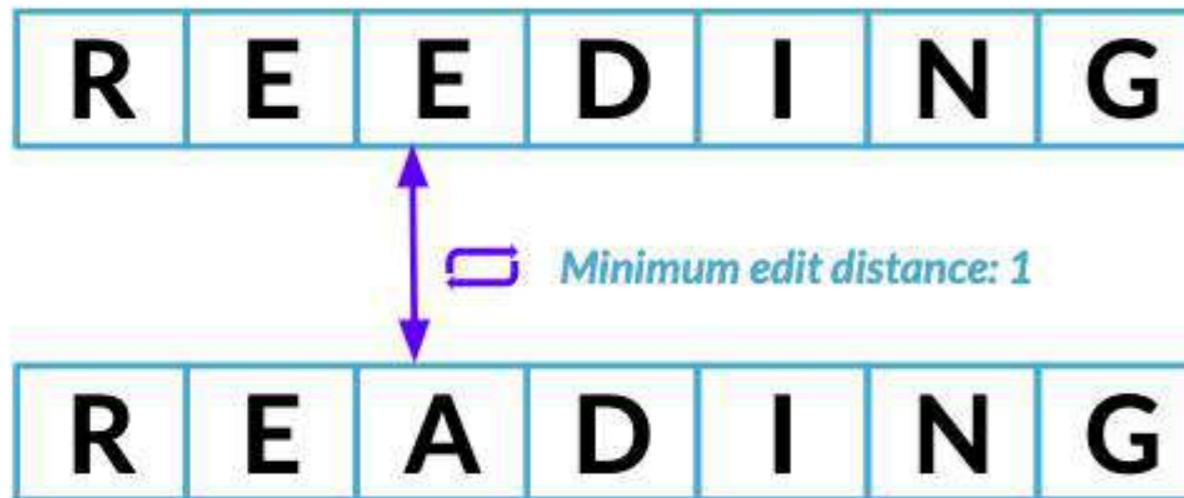
Minimum edit distance



Minimum edit distance: 5

Minimum edit distance

Typos for the word: READING



Minimum edit distance algorithms

Algorithm	Operations
Damerau-Levenshtein	insertion, substitution, deletion, transposition
Levenshtein	insertion, substitution, deletion
Hamming	substitution only
Jaro distance	transposition only
...	...

Possible packages: `nltk` , `thefuzz` , `textdistance` ..

Minimum edit distance algorithms

Algorithm	Operations
Damerau-Levenshtein	insertion, substitution, deletion, transposition
Levenshtein	<i>insertion, substitution, deletion</i>
Hamming	substitution only
Jaro distance	transposition only
...	...

Possible packages: `thefuzz`

Simple string comparison

```
# Lets us compare between two strings
from thefuzz import fuzz

# Compare reeding vs reading
fuzz.WRatio('Reeding', 'Reading')
```

86

Partial strings and different orderings

```
# Partial string comparison  
fuzz.WRatio('Houston Rockets', 'Rockets')
```

90

```
# Partial string comparison with different order  
fuzz.WRatio('Houston Rockets vs Los Angeles Lakers', 'Lakers vs Rockets')
```

86

Comparison with arrays

```
# Import process
from thefuzz import process

# Define string and array of possible matches
string = "Houston Rockets vs Los Angeles Lakers"
choices = pd.Series(['Rockets vs Lakers', 'Lakers vs Rockets',
                     'Houson vs Los Angeles', 'Heat vs Bulls'])

process.extract(string, choices, limit = 2)
```

```
[('Rockets vs Lakers', 86, 0), ('Lakers vs Rockets', 86, 1)]
```

Collapsing categories with string similarity

Chapter 2

Use `.replace()` to collapse "eur" into "Europe"

What if there are too many variations?

"EU" , "eur" , "Europ" , "Europa" , "Erope" , "Evropa" ...

String similarity!

Collapsing categories with string matching

```
print(survey['state'].unique())
```

```
id      state
0      California
1      Cali
2      Calefornia
3      Caleifornie
4      Californie
5      California
6      Calefernia
7      New York
8      New York City
...
...
```

categories

```
state
0 California
1 New York
```

Collapsing all of the state

```
# For each correct category
for state in categories['state']:
    # Find potential matches in states with typos
    matches = process.extract(state, survey['state'], limit = survey.shape[0])
    # For each potential match match
    for potential_match in matches:
        # If high similarity score
        if potential_match[1] >= 80:
            # Replace typo with correct category
            survey.loc[survey['state'] == potential_match[0], 'state'] = state
```

Record linkage

Event	Time	Event	Time
Houston Rockets vs Chicago Bulls	19:00	NBA: Nets vs Magic	8pm
Miami Heat vs Los Angeles Lakers	19:00	NBA: Bulls vs Rockets	9pm
Brooklyn Nets vs Orlando Magic	20:00	NBA: Heat vs Lakers	7pm
Denver Nuggets vs Miami Heat	21:00	NBA: Grizzlies vs Heat	10pm
San Antonio Spurs vs Atlanta Hawks	21:00	NBA: Heat vs Cavaliers	9pm

Let's practice!

CLEANING DATA IN PYTHON

Generating pairs

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @ DataCamp

Motivation

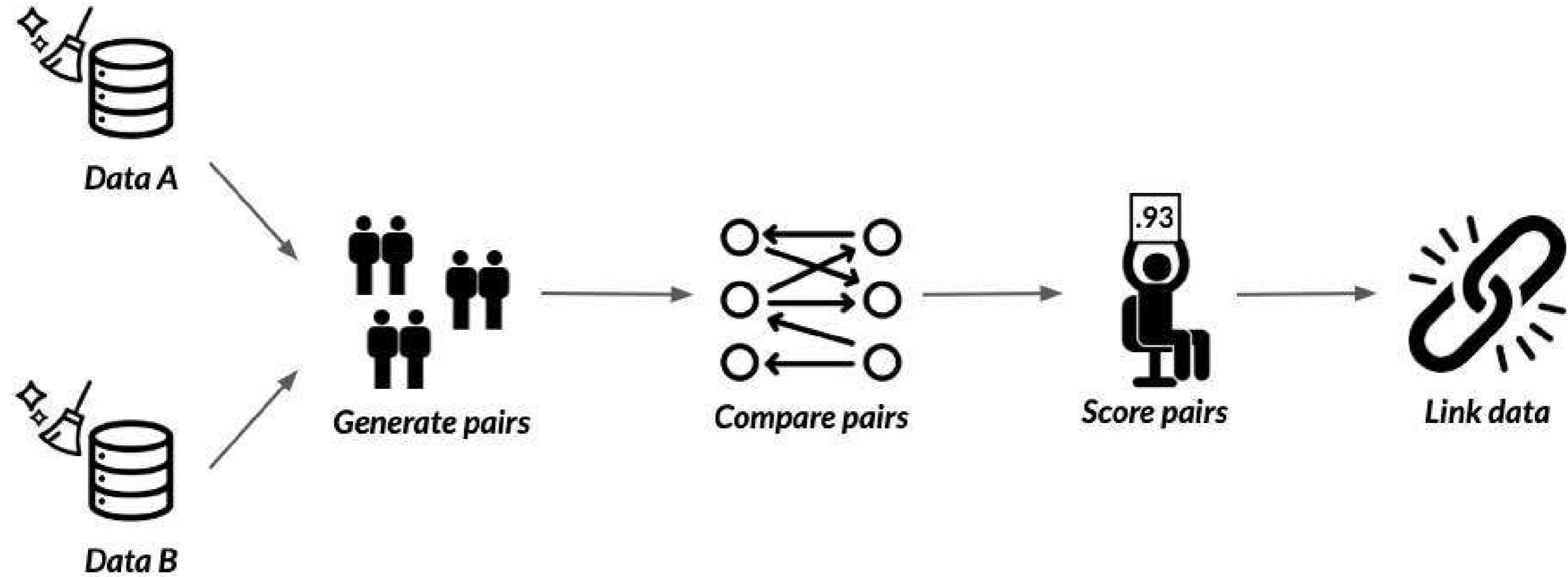
Event	Time
Houston Rockets vs Chicago Bulls	19:00
Miami Heat vs Los Angeles Lakers	19:00
Brooklyn Nets vs Orlando Magic	20:00
Denver Nuggets vs Miami Heat	21:00
San Antonio Spurs vs Atlanta Hawks	21:00

Event	Time
NBA: Nets vs Magic	8pm
NBA: Bulls vs Rockets	9pm
NBA: Heat vs Lakers	7pm
NBA: Grizzlies vs Heat	10pm
NBA: Heat vs Cavaliers	9pm

When joins won't work

Event	Time	Event	Time
Houston Rockets vs Chicago Bulls	19:00	NBA: Nets vs Magic	8pm
Miami Heat vs Los Angeles Lakers	19:00	NBA: Bulls vs Rockets	9pm
Brooklyn Nets vs Orlando Magic	20:00	NBA: Heat vs Lakers	7pm
Denver Nuggets vs Miami Heat	21:00	NBA: Grizzlies vs Heat	10pm
San Antonio Spurs vs Atlanta Hawks	21:00	NBA: Heat vs Cavaliers	9pm

Record linkage



The `recordlinkage` package

Our DataFrames

census_A

	given_name	surname	date_of_birth	suburb	state	address_1
rec_id						
rec-1070-org	michaela	neumann	19151111	winston hills	cal	stanley street
rec-1016-org	courtney	painter	19161214	richlands	txs	pinkerton circuit
...						

census_B

	given_name	surname	date_of_birth	suburb	state	address_1
rec_id						
rec-561-dup-0	elton	NaN	19651013	windermere	ny	light setreet
rec-2642-dup-0	mitchell	maxon	19390212	north ryde	cal	edkins street
...						

Generating pairs

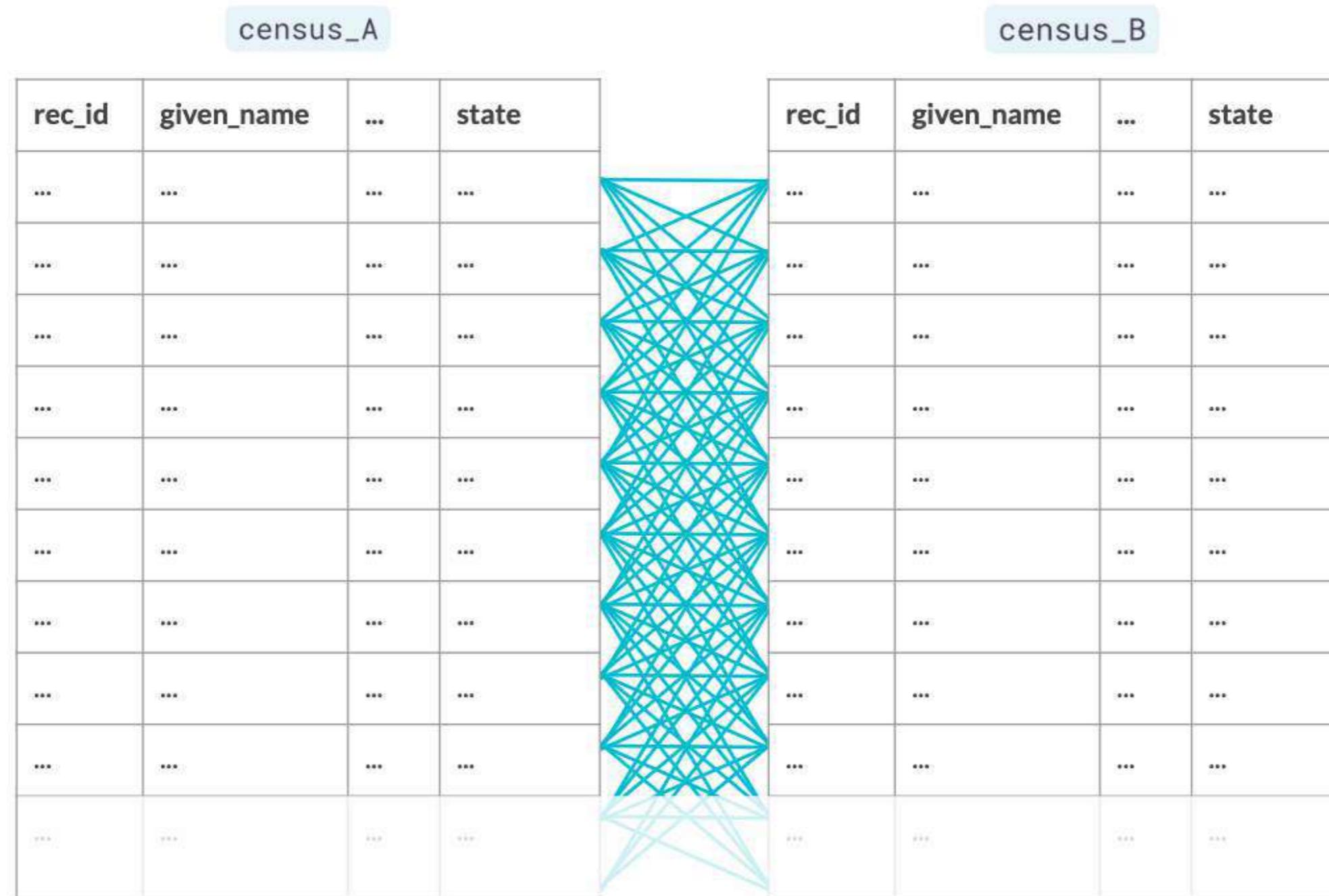
census_A

rec_id	given_name	...	state
...
...
...
...

census_B

rec_id	given_name	...	state
...
...
...
...

Generating pairs



Blocking

census_A

rec_id	given_name	...	state
...	cal
...	ny
...	txs
...	txs

census_A

rec_id	given_name	...	state
...	cal
...	txs
...	ny
...	cal

Generating pairs

```
# Import recordlinkage
import recordlinkage

# Create indexing object
indexer = recordlinkage.Index()

# Generate pairs blocked on state
indexer.block('state')
pairs = indexer.index(census_A, census_B)
```

Generating pairs

```
print(pairs)
```

```
MultiIndex(levels=[['rec-1007-org', 'rec-1016-org', 'rec-1054-org', 'rec-1066-org',
'rec-1070-org', 'rec-1075-org', 'rec-1080-org', 'rec-110-org', 'rec-1146-org',
'rec-1157-org', 'rec-1165-org', 'rec-1185-org', 'rec-1234-org', 'rec-1271-org',
'rec-1280-org', .....,
66, 14, 13, 18, 34, 39, 0, 16, 80, 50, 20, 69, 28, 25, 49, 77, 51, 85, 52, 63, 74, 61,
83, 91, 22, 26, 55, 84, 11, 81, 97, 56, 27, 48, 2, 64, 5, 17, 29, 60, 72, 47, 92, 12,
95, 15, 19, 57, 37, 70, 94]], names=['rec_id_1', 'rec_id_2'])
```

Comparing the DataFrames

```
# Generate the pairs
pairs = indexer.index(census_A, census_B)
# Create a Compare object
compare_cl = recordlinkage.Compare()

# Find exact matches for pairs of date_of_birth and state
compare_cl.exact('date_of_birth', 'date_of_birth', label='date_of_birth')
compare_cl.exact('state', 'state', label='state')
# Find similar matches for pairs of surname and address_1 using string similarity
compare_cl.string('surname', 'surname', threshold=0.85, label='surname')
compare_cl.string('address_1', 'address_1', threshold=0.85, label='address_1')

# Find matches
potential_matches = compare_cl.compute(pairs, census_A, census_B)
```

Finding matching pairs

```
print(potential_matches)
```

rec_id_1	rec_id_2	date_of_birth	state	surname	address_1
rec-1070-org	rec-561-dup-0	0	1	0.0	0.0
	rec-2642-dup-0	0	1	0.0	0.0
	rec-608-dup-0	0	1	0.0	0.0
...					
rec-1631-org	rec-4070-dup-0	0	1	0.0	0.0
	rec-4862-dup-0	0	1	0.0	0.0
	rec-629-dup-0	0	1	0.0	0.0
...					

Finding the only pairs we want

```
potential_matches[potential_matches.sum(axis = 1) >= 2]
```

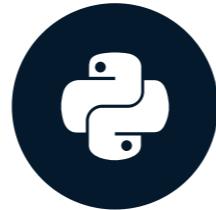
rec_id_1	rec_id_2	date_of_birth	state	surname	address_1
rec-4878-org	rec-4878-dup-0	1	1	1.0	0.0
rec-417-org	rec-2867-dup-0	0	1	0.0	1.0
rec-3964-org	rec-394-dup-0	0	1	1.0	0.0
rec-1373-org	rec-4051-dup-0	0	1	1.0	0.0
	rec-802-dup-0	0	1	1.0	0.0
rec-3540-org	rec-470-dup-0	0	1	1.0	0.0

Let's practice!

CLEANING DATA IN PYTHON

Linking DataFrames

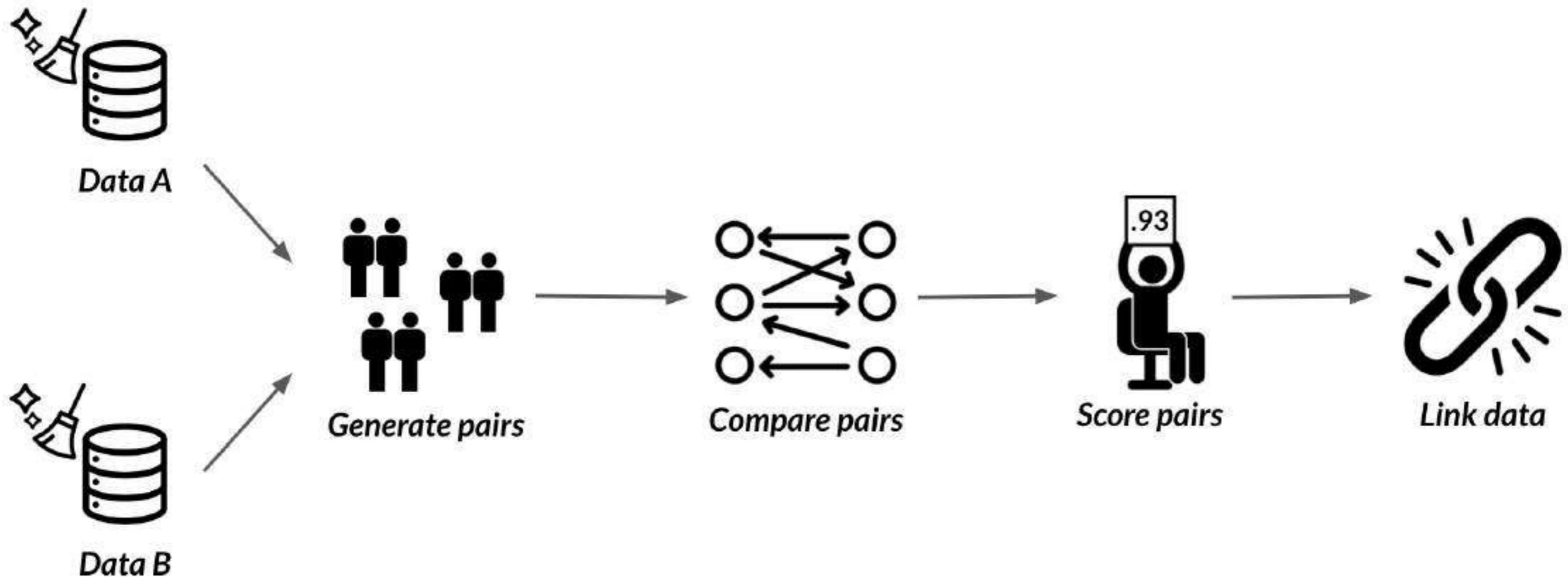
CLEANING DATA IN PYTHON



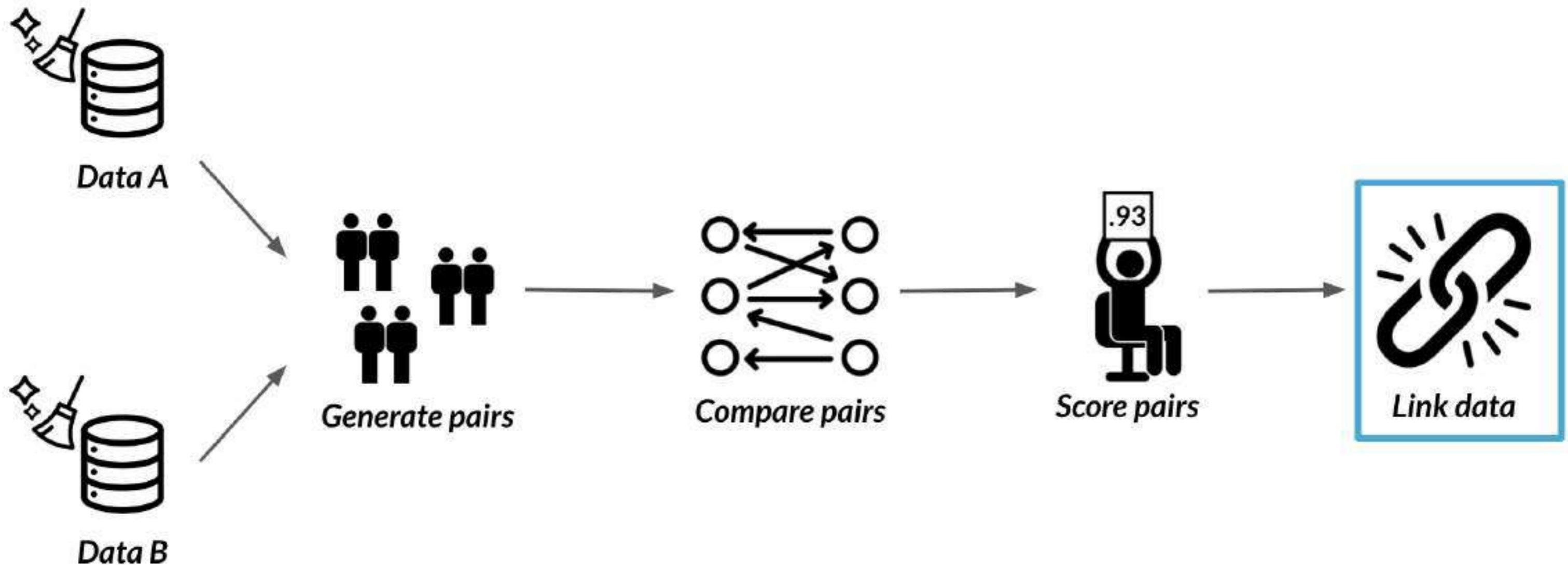
Adel Nehme

Content Developer @ DataCamp

Record linkage



Record linkage



Our DataFrames

census_A

	given_name	surname	date_of_birth	suburb	state	address_1
rec_id						
rec-1070-org	michaela	neumann	19151111	winston hills	nsw	stanley street
rec-1016-org	courtney	painter	19161214	richlands	vic	pinkerton circuit
...						

census_B

	given_name	surname	date_of_birth	suburb	state	address_1
rec_id						
rec-561-dup-0	elton	NaN	19651013	windermere	vic	light setreet
rec-2642-dup-0	mitchell	maxon	19390212	north ryde	nsw	edkins street
...						

What we've already done

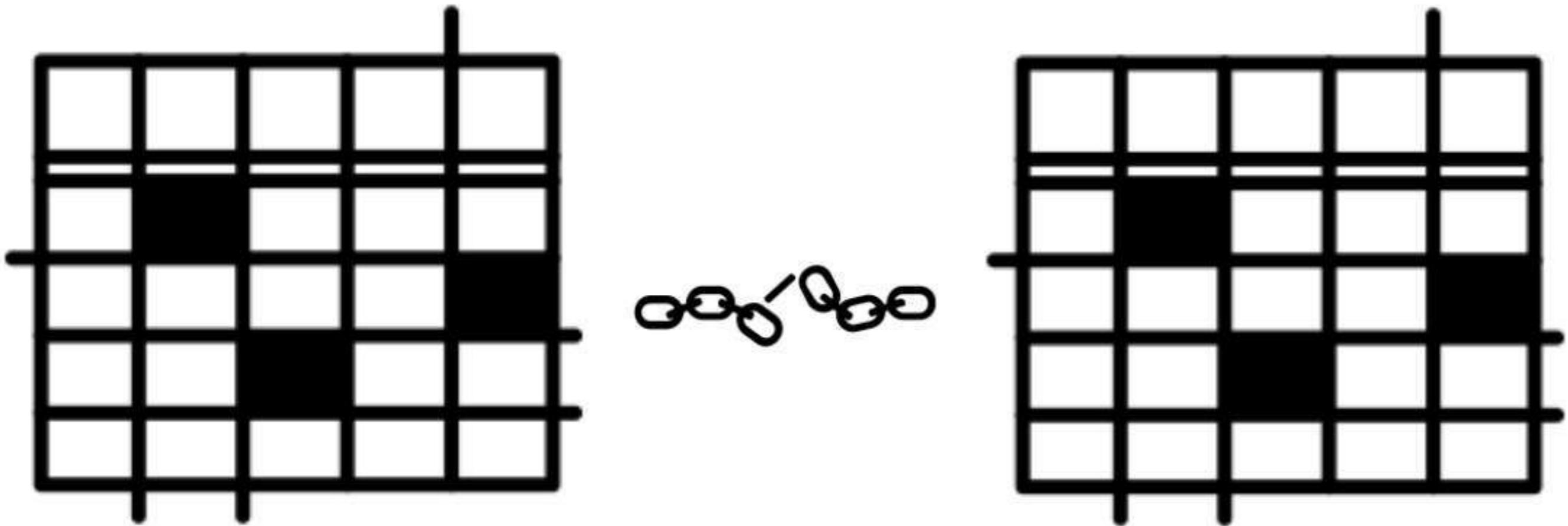
```
# Import recordlinkage and generate full pairs
import recordlinkage

indexer = recordlinkage.Index()
indexer.block('state')
full_pairs = indexer.index(census_A, census_B)

# Comparison step
compare_cl = recordlinkage.Compare()
compare_cl.exact('date_of_birth', 'date_of_birth', label='date_of_birth')
compare_cl.exact('state', 'state', label='state')
compare_cl.string('surname', 'surname', threshold=0.85, label='surname')
compare_cl.string('address_1', 'address_1', threshold=0.85, label='address_1')

potential_matches = compare_cl.compute(full_pairs, census_A, census_B)
```

What we're doing now



census_A

census_B

Our potential matches

potential_matches

rec_id_1	rec_id_2	date_of_birth	state	surname	address_1
rec-1070-org	rec-561-dup-0	0	1	0.0	0.0
	rec-2642-dup-0	0	1	0.0	0.0
	rec-608-dup-0	0	1	0.0	0.0
...	
rec-1631-org	rec-1697-dup-0	0	1	0.0	0.0
	rec-4404-dup-0	0	1	0.0	0.0
	rec-3780-dup-0	0	1	0.0	0.0
...	

Our potential matches

potential_matches

census_A		date_of_birth	state	surname	address_1
rec_id_1	rec_id_2				
rec-1070-org	rec-561-dup-0	0	1	0.0	0.0
	rec-2642-dup-0	0	1	0.0	0.0
	rec-608-dup-0	0	1	0.0	0.0
...	
rec-1631-org	rec-1697-dup-0	0	1	0.0	0.0
	rec-4404-dup-0	0	1	0.0	0.0
	rec-3780-dup-0	0	1	0.0	0.0
...	

Our potential matches

potential_matches

census_A	census_B	date_of_birth	state	surname	address_1
rec_id_1	rec_id_2				
rec-1070-org	rec-561-dup-0	0	1	0.0	0.0
	rec-2642-dup-0	0	1	0.0	0.0
	rec-608-dup-0	0	1	0.0	0.0
...	
rec-1631-org	rec-1697-dup-0	0	1	0.0	0.0
	rec-4404-dup-0	0	1	0.0	0.0
	rec-3780-dup-0	0	1	0.0	0.0
...	

Our potential matches

potential_matches

census_A	census_B	date_of_birth	state	surname	address_1
rec_id_1	rec_id_2				
rec-1070-org	rec-561-dup-0	0	1	0.0	0.0
	rec-2642-dup-0	0	1	0.0	0.0
	rec-608-dup-0	0	1	0.0	0.0
...	
rec-1631-org	rec-1697-dup-0	0	1	0.0	0.0
	rec-4404-dup-0	0	1	0.0	0.0
	rec-3780-dup-0	0	1	0.0	0.0
...	

Probable matches

```
matches = potential_matches[potential_matches.sum(axis = 1) >= 3]
print(matches)
```

rec_id_1	rec_id_2	date_of_birth	state	surname	address_1
rec-2404-org	rec-2404-dup-0	1	1	1.0	1.0
rec-4178-org	rec-4178-dup-0	1	1	1.0	1.0
rec-1054-org	rec-1054-dup-0	1	1	1.0	1.0
...
rec-1234-org	rec-1234-dup-0	1	1	1.0	1.0
rec-1271-org	rec-1271-dup-0	1	1	1.0	1.0

Probable matches

```
matches = potential_matches[potential_matches.sum(axis = 1) >= 3]
print(matches)
```

census_B		date_of_birth	state	surname	address_1
rec_id_1	rec_id_2				
rec-2404-org	rec-2404-dup-0	1	1	1.0	1.0
rec-4178-org	rec-4178-dup-0	1	1	1.0	1.0
rec-1054-org	rec-1054-dup-0	1	1	1.0	1.0
...
rec-1234-org	rec-1234-dup-0	1	1	1.0	1.0
rec-1271-org	rec-1271-dup-0	1	1	1.0	1.0

Get the indices

```
matches.index
```

```
MultiIndex(levels=[[ 'rec-1007-org', 'rec-1016-org', 'rec-1054-org', 'rec-1066-org',
'rec-1070-org', 'rec-1075-org', 'rec-1080-org', 'rec-110-org', ...
```

```
# Get indices from census_B only
duplicate_rows = matches.index.get_level_values(1)
print(census_B_index)
```

```
Index(['rec-2404-dup-0', 'rec-4178-dup-0', 'rec-1054-dup-0', 'rec-4663-dup-0',
'rec-485-dup-0', 'rec-2950-dup-0', 'rec-1234-dup-0', ... , 'rec-299-dup-0'])
```

Linking DataFrames

```
# Finding duplicates in census_B  
census_B_duplicates = census_B[census_B.index.isin(duplicate_rows)]
```

```
# Finding new rows in census_B  
census_B_new = census_B[~census_B.index.isin(duplicate_rows)]
```

```
# Link the DataFrames!  
full_census = census_A.append(census_B_new)
```

```
# Import recordlinkage and generate pairs and compare across columns
...
# Generate potential matches
potential_matches = compare_cl.compute(full_pairs, census_A, census_B)

# Isolate matches with matching values for 3 or more columns
matches = potential_matches[potential_matches.sum(axis = 1) >= 3]

# Get index for matching census_B rows only
duplicate_rows = matches.index.get_level_values(1)

# Finding new rows in census_B
census_B_new = census_B[~census_B.index.isin(duplicate_rows)]

# Link the DataFrames!
full_census = census_A.append(census_B_new)
```

Let's practice!

CLEANING DATA IN PYTHON

Congratulations!

CLEANING DATA IN PYTHON



Adel Nehme

Content Developer @ DataCamp

What we've learned



Diagnose dirty
data



Side effects of
dirty data



Clean data

What we've learned



**Data Type
Constraints**

Strings
Numeric data

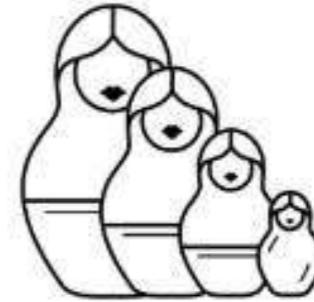
...



**Data Range
Constraints**

Out of range data
Out of range dates

...



**Uniqueness
Constraints**

Finding duplicates
Treating them

...

Chapter 1 - Common data problems

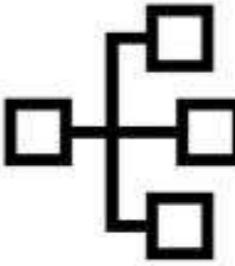
What we've learned



Membership Constraints

*Finding inconsistent categories
Treating them with joins*

...



Categorical Variables

*Finding inconsistent categories
Collapsing them into less*

...



Cleaning Text Data

*Unifying formats
Finding lengths*

...

Chapter 2 - Text and categorical data problems

What we've learned



Uniformity

*Unifying currency formats
Unifying date formats*

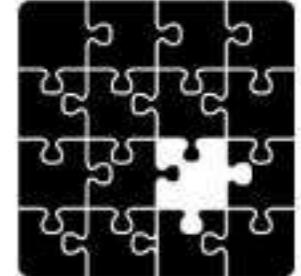
...



Cross field validation

*Summing across rows
Building assert functions*

...



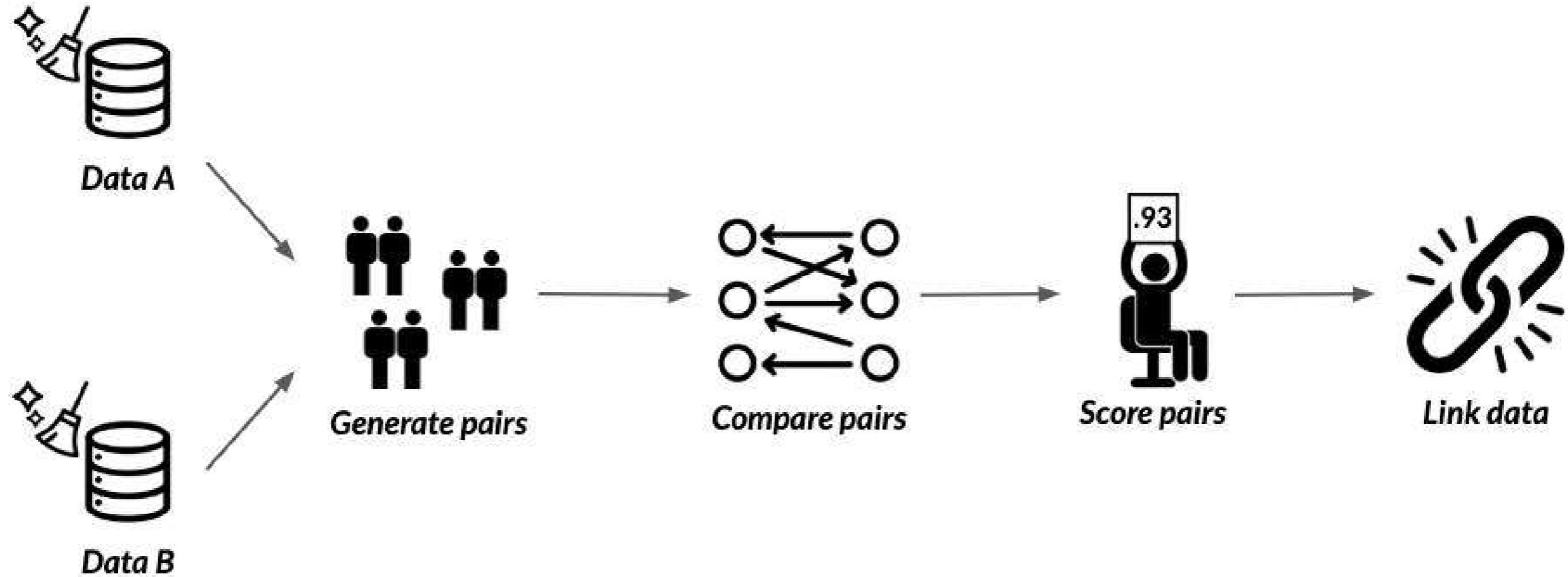
Completeness

*Finding missing data
Treating them*

...

Chapter 3 - Advanced data problems

What we've learned

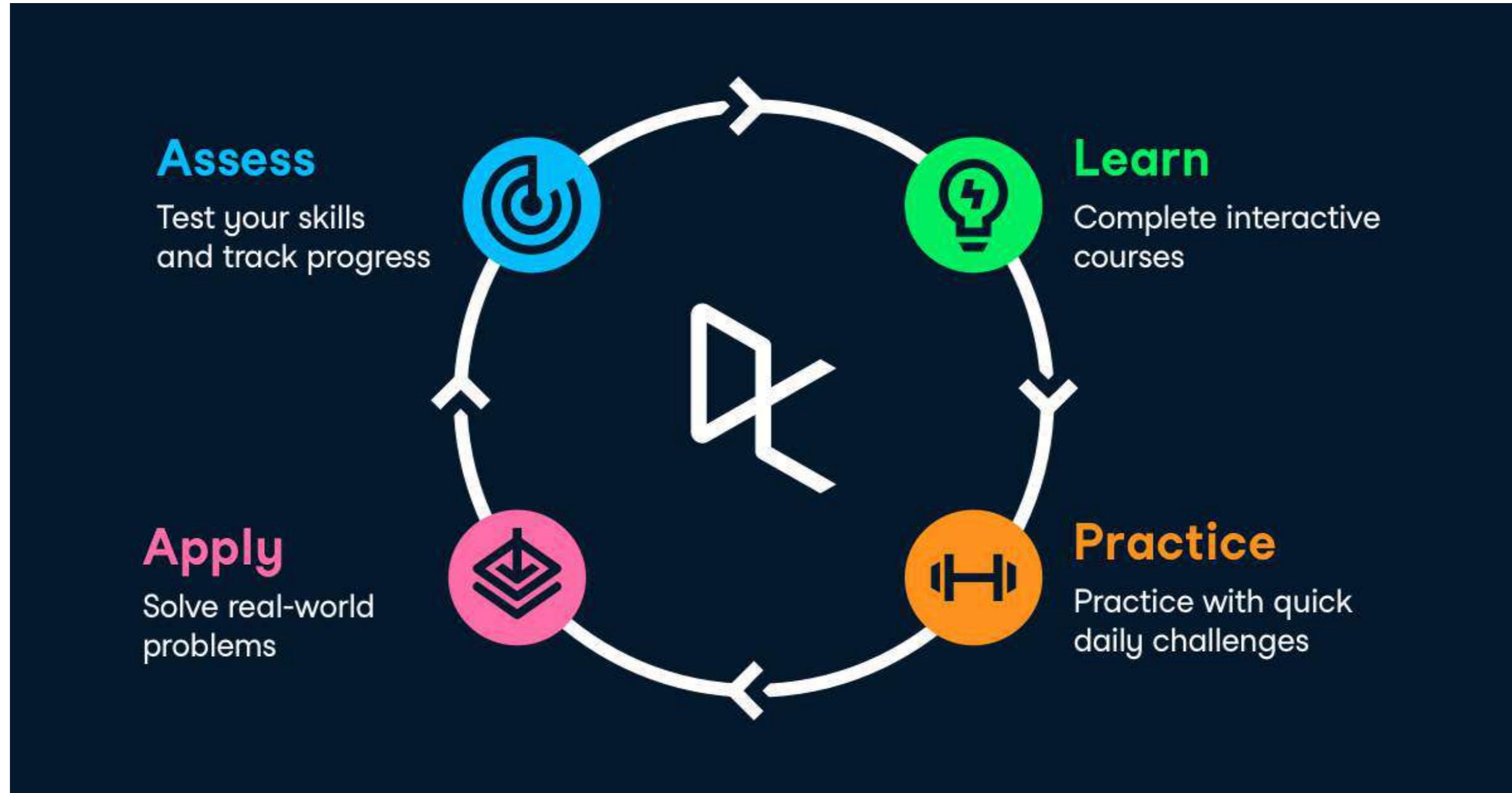


Chapter 4 - Record linkage

More to learn on DataCamp!

- [Working with Dates and Times in Python](#)
- [Regular Expressions in Python](#)
- [Dealing with Missing Data in Python](#)
- And more!

More to learn!



More to learn!

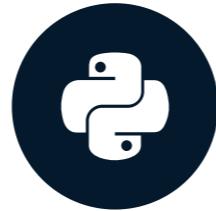


Thank you!

CLEANING DATA IN PYTHON

What is statistics?

INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

Content Developer, DataCamp

What is statistics?

- **The field of statistics** - the practice and study of collecting and analyzing data
- **A summary statistic** - a fact about or summary of some data

What can statistics do?

What is statistics?

- **The field of statistics** - the practice and study of collecting and analyzing data
- **A summary statistic** - a fact about or summary of some data

What can statistics do?

- How likely is someone to purchase a product? Are people more likely to purchase it if they can use a different payment system?
- How many occupants will your hotel have? How can you optimize occupancy?
- How many sizes of jeans need to be manufactured so they can fit 95% of the population? Should the same number of each size be produced?
- A/B tests: Which ad is more effective in getting people to purchase a product?

What can't statistics do?

- *Why* is *Game of Thrones* so popular?

Instead...

- Are series with more violent scenes viewed by more people?

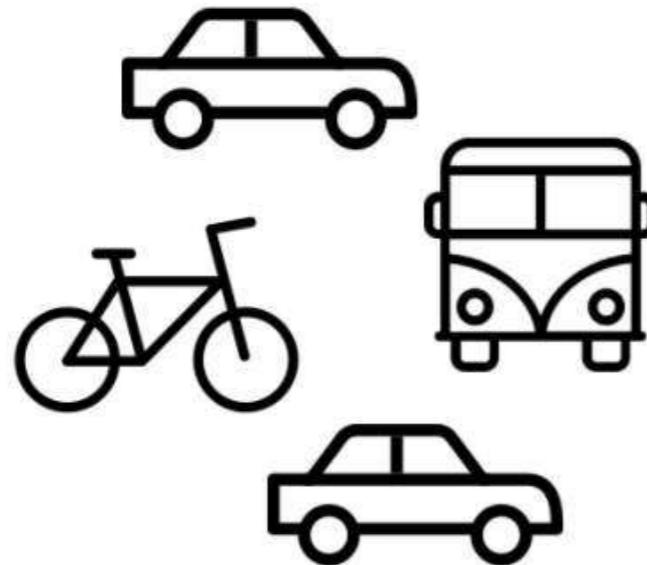
But...

- Even so, this can't tell us if more violent scenes lead to more views

Types of statistics

Descriptive statistics

- *Describe* and summarize data



- 50% of friends drive to work
- 25% take the bus
- 25% bike

Inferential statistics

- Use a sample of data to make *inferences* about a larger population



What percent of people drive to work?

Types of data

Numeric (Quantitative)

- Continuous (Measured)
 - Airplane speed
 - Time spent waiting in line
- Discrete (Counted)
 - Number of pets
 - Number of packages shipped

Categorical (Qualitative)

- Nominal (Unordered)
 - Married/unmarried
 - Country of residence
- Ordinal (Ordered)
 - Strongly disagree
 - Somewhat disagree
 - Neither agree nor disagree
 - Somewhat agree
 - Strongly agree

Categorical data can be represented as numbers

Nominal (Unordered)

- Married/unmarried (1 / 0)
- Country of residence (1 , 2 , ...)

Ordinal (Ordered)

- Strongly disagree (1)
- Somewhat disagree (2)
- Neither agree nor disagree (3)
- Somewhat agree (4)
- Strongly agree (5)

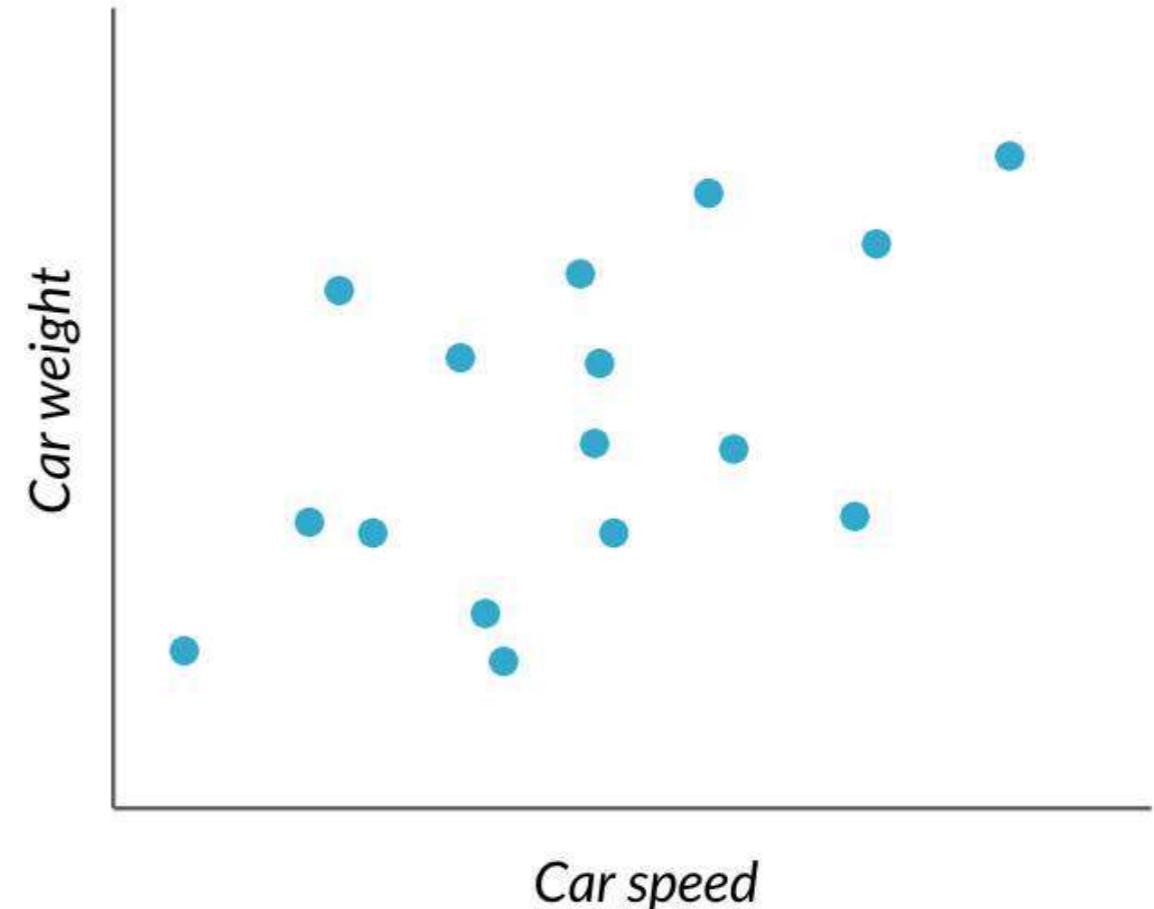
Why does data type matter?

Summary statistics

```
import numpy as np  
np.mean(car_speeds['speed_mph'])
```

40.09062

Plots



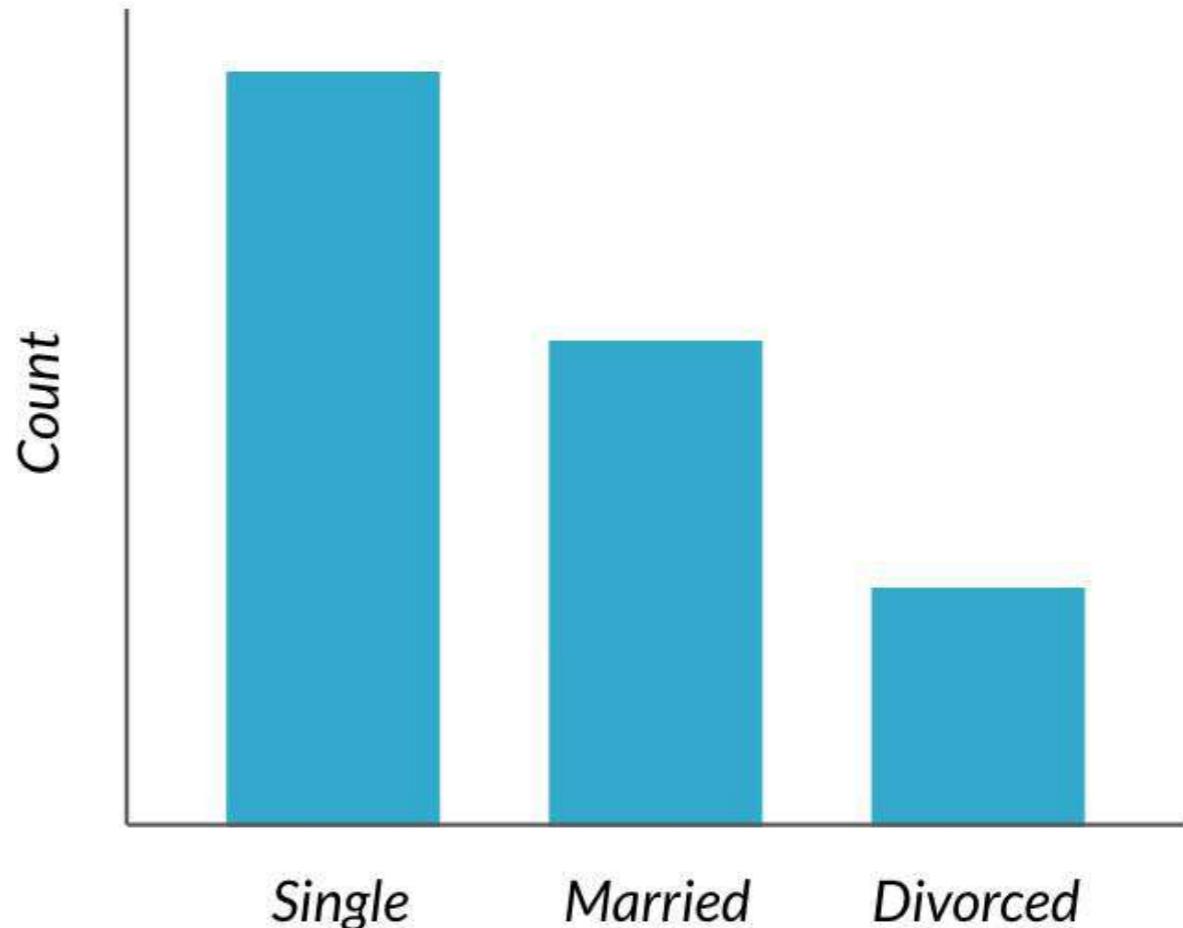
Why does data type matter?

Summary statistics

```
demographics['marriage_status'].value_counts()
```

```
single      188  
married     143  
divorced    124  
dtype: int64
```

Plots

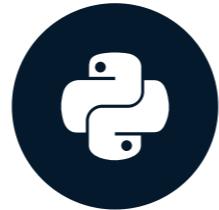


Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

Measures of center

INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

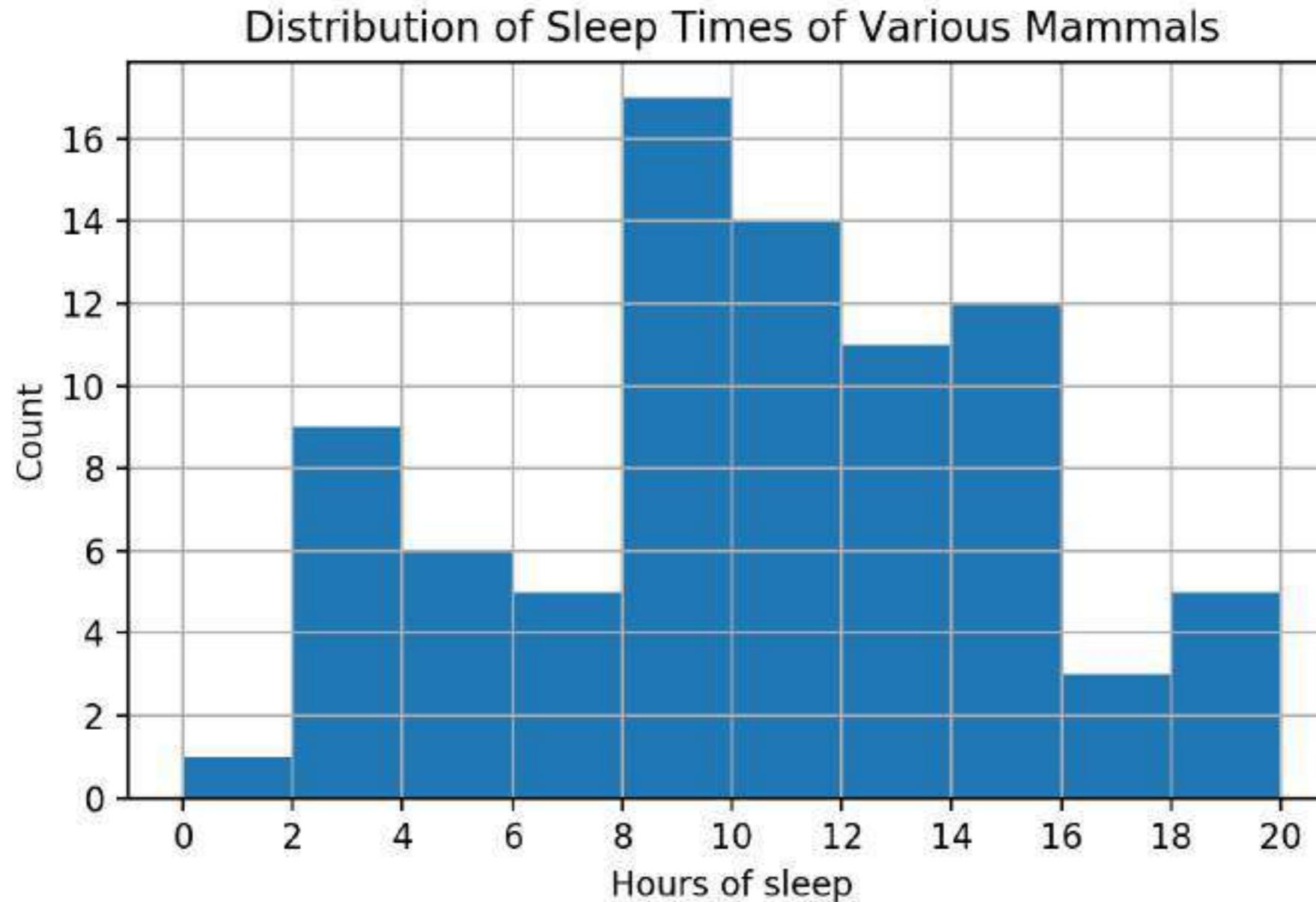
Content Developer, DataCamp

Mammal sleep data

```
print(msleep)
```

	name	genus	vore	order	...	sleep_cycle	awake	brainwt	bodywt
1	Cheetah	Acinonyx	carni	Carnivora	...		NaN	11.9	NaN
2	Owl monkey	Aotus	omni	Primates	...		NaN	7.0	0.01550
3	Mountain beaver	Aplodontia	herbi	Rodentia	...		NaN	9.6	NaN
4	Greater short-ta...	Blarina	omni	Soricomorpha	...	0.133333	9.1	0.00029	0.019
5	Cow	Bos	herbi	Artiodactyla	...	0.666667	20.0	0.42300	600.000
...
79	Tree shrew	Tupaia	omni	Scandentia	...	0.233333	15.1	0.00250	0.104
80	Bottle-nosed do...	Tursiops	carni	Cetacea	...		NaN	18.8	NaN
81	Genet	Genetta	carni	Carnivora	...		NaN	17.7	0.01750
82	Arctic fox	Vulpes	carni	Carnivora	...		NaN	11.5	0.04450
83	Red fox	Vulpes	carni	Carnivora	...	0.350000	14.2	0.05040	4.230

Histograms

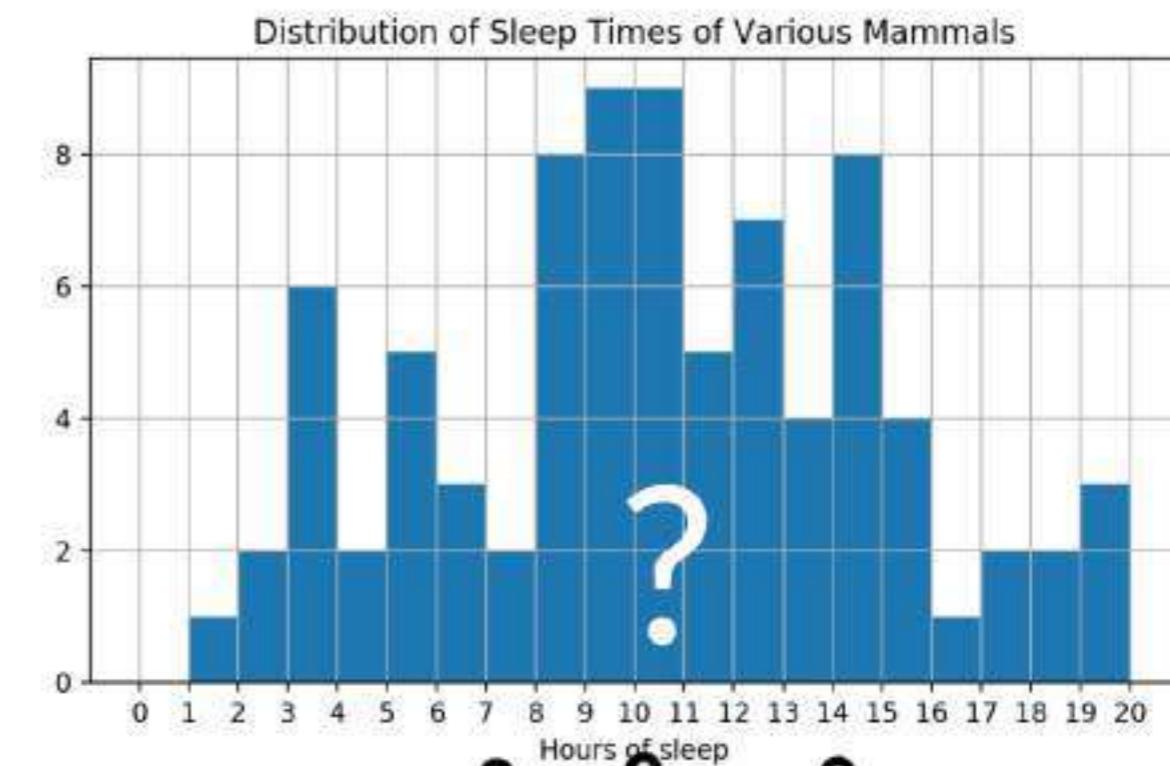


How long do mammals in this dataset typically sleep?

What's a typical value?

Where is the center of the data?

- Mean
- Median
- Mode



Measures of center: mean

	name	sleep_total
1	Cheetah	12.1
2	Owl monkey	17.0
3	Mountain beaver	14.4
4	Greater short-t...	14.9
5	Cow	4.0
...

```
import numpy as np  
np.mean(msleep['sleep_total'])
```

```
10.43373
```

Mean sleep time =

$$\frac{12.1 + 17.0 + 14.4 + 14.9 + \dots}{83} = 10.43$$

Measures of center: median

```
msleep['sleep_total'].sort_values()
```

```
29      1.9  
30      2.7  
22      2.9  
9       3.0  
23      3.1  
...  
19      18.0  
61      18.1  
36      19.4  
21      19.7  
42      19.9
```

```
msleep['sleep_total'].sort_values().iloc[41]
```

```
10.1
```

```
np.median(msleep['sleep_total'])
```

```
10.1
```

Measures of center: mode

Most frequent value

```
msleep['sleep_total'].value_counts()
```

```
12.5      4  
10.1      3  
14.9      2  
11.0      2  
8.4       2  
...  
14.3      1  
17.0      1  
  
Name: sleep_total, Length: 65, dtype: int64
```

```
msleep['vore'].value_counts()
```

```
herbi      32  
omni       20  
carni      19  
insecti    5  
  
Name: vore, dtype: int64
```

```
import statistics  
statistics.mode(msleep['vore'])
```

```
'herbi'
```

Adding an outlier

```
msleep[msleep['vore'] == 'insecti']
```

	name	genus	vore	order	sleep_total
22	Big brown bat	Eptesicus	insecti	Chiroptera	19.7
43	Little brown bat	Myotis	insecti	Chiroptera	19.9
62	Giant armadillo	Priodontes	insecti	Cingulata	18.1
67	Eastern american mole	Scalopus	insecti	Soricomorpha	8.4

Adding an outlier

```
msleep[msleep['vore'] == "insectivore"]['sleep_total'].agg([np.mean, np.median])
```

```
mean      16.53
median    18.9
Name: sleep_total, dtype: float64
```

Adding an outlier

```
msleep[msleep['vore'] == 'insecti']
```

	name	genus	vore	order	sleep_total
22	Big brown bat	Eptesicus	insecti	Chiroptera	19.7
43	Little brown bat	Myotis	insecti	Chiroptera	19.9
62	Giant armadillo	Priodontes	insecti	Cingulata	18.1
67	Eastern american mole	Scalopus	insecti	Soricomorpha	8.4
84	Mystery insectivore	...	insecti	...	0.0

Adding an outlier

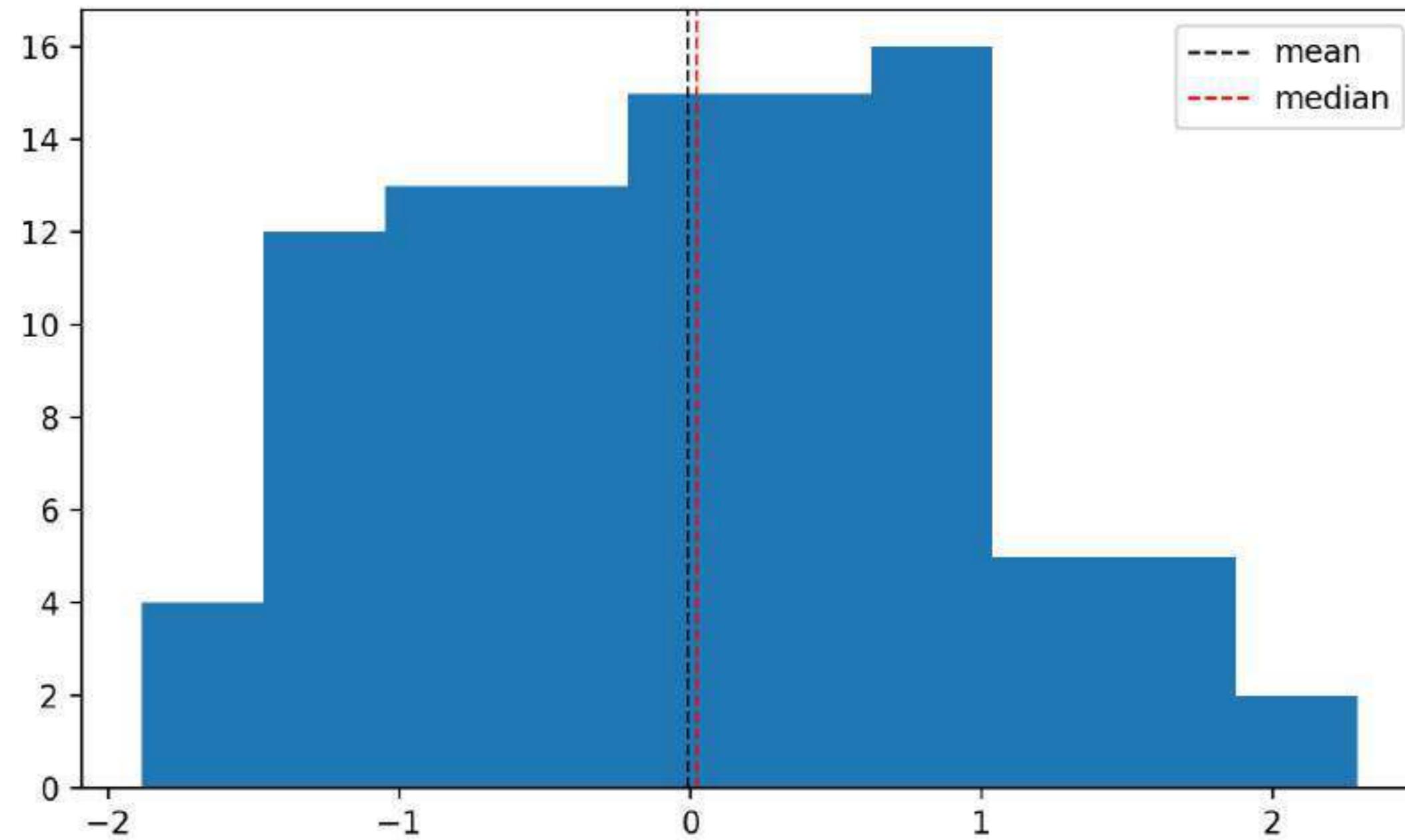
```
msleep[msleep['vore'] == "insectivore"]['sleep_total'].agg([np.mean, np.median])
```

```
mean      13.22  
median    18.1  
Name: sleep_total, dtype: float64
```

Mean: 16.5 → 13.2

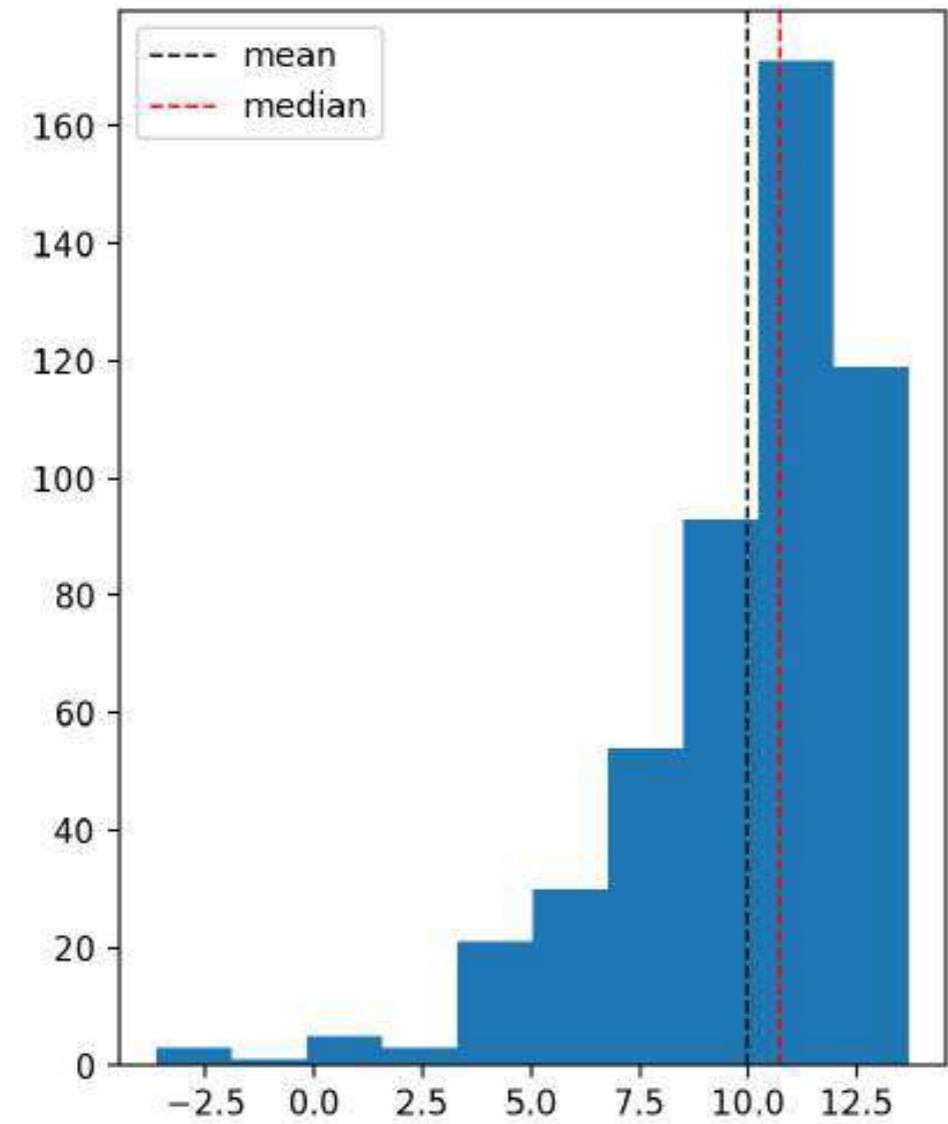
Median: 18.9 → 18.1

Which measure to use?

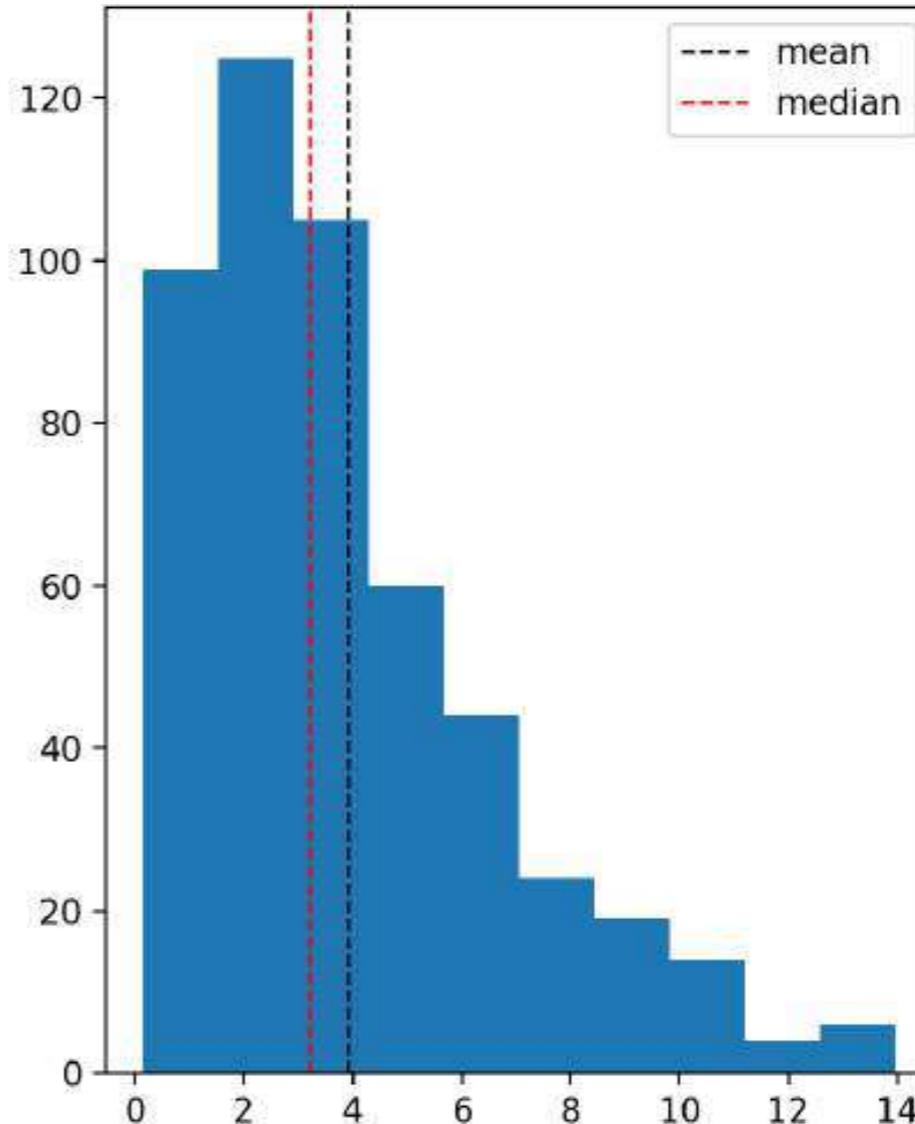


Skew

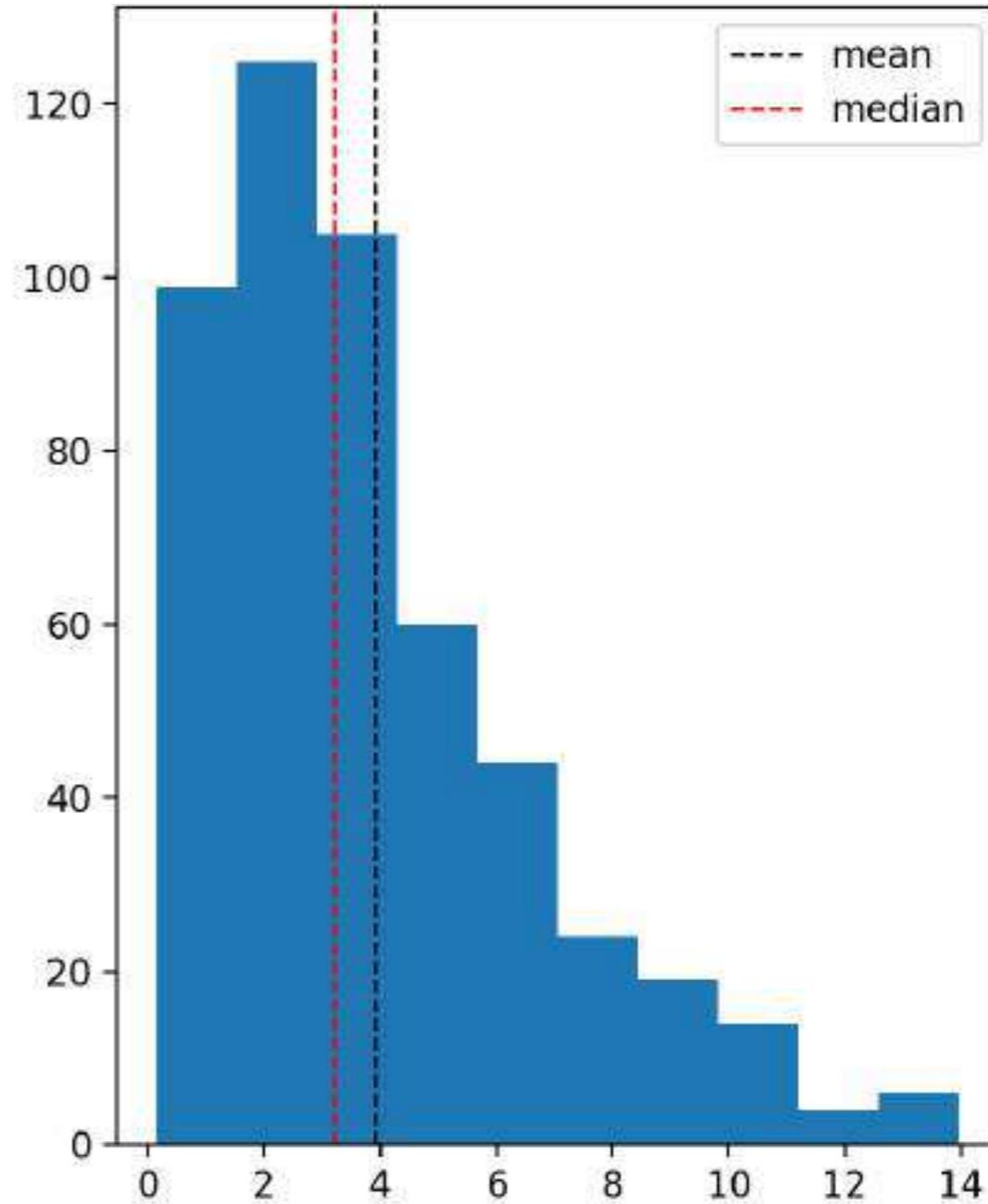
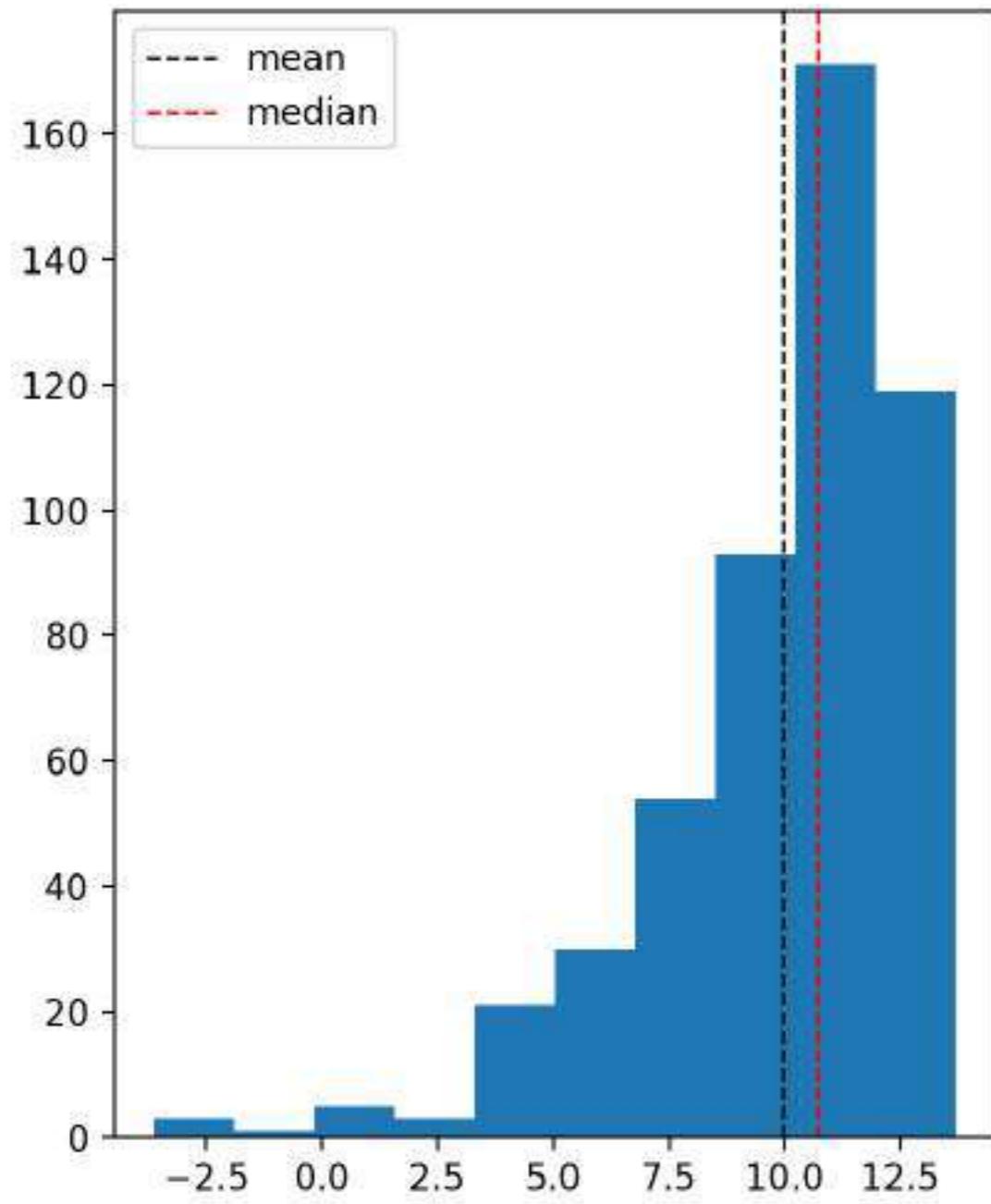
Left-skewed



Right-skewed



Which measure to use?



Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

Measures of spread

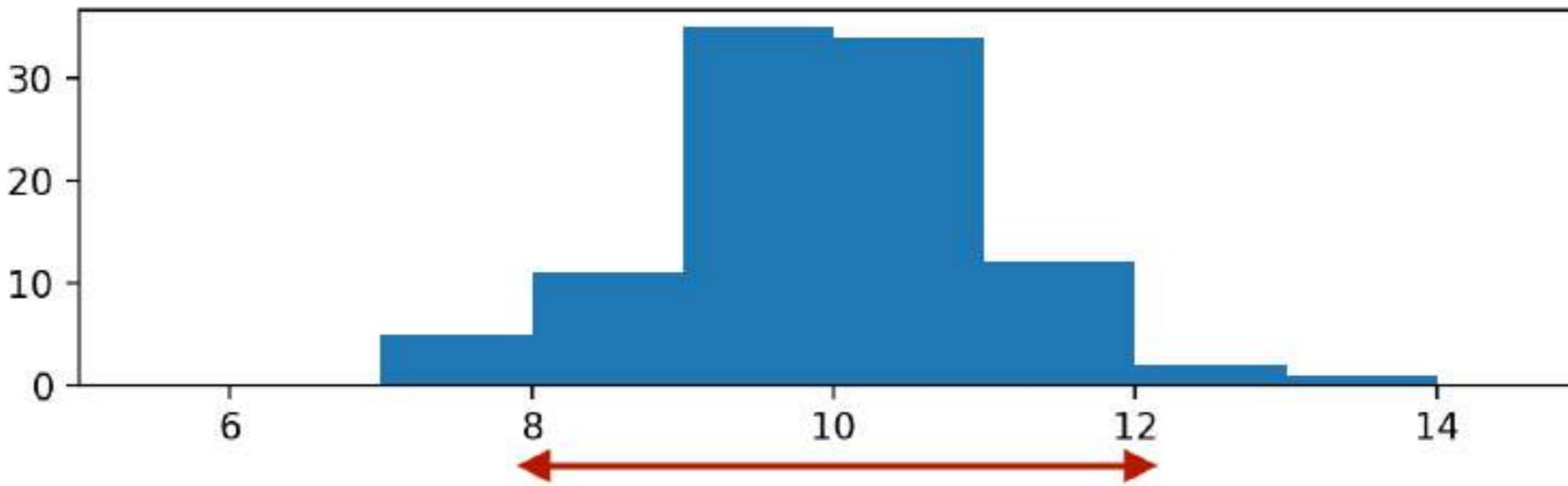
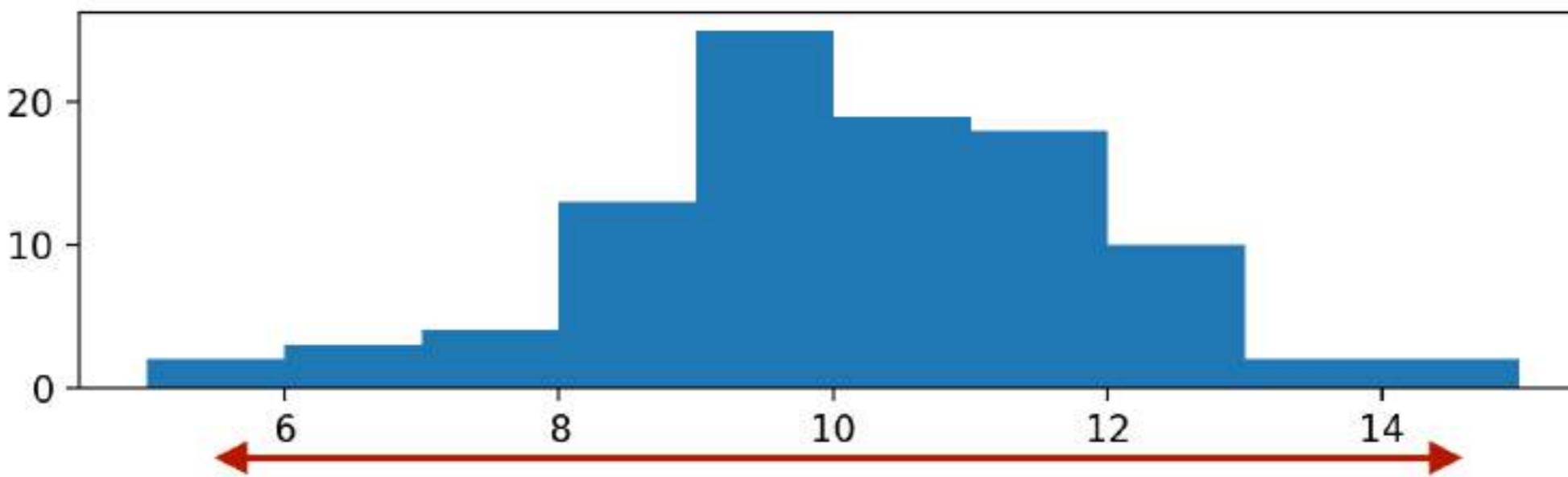
INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

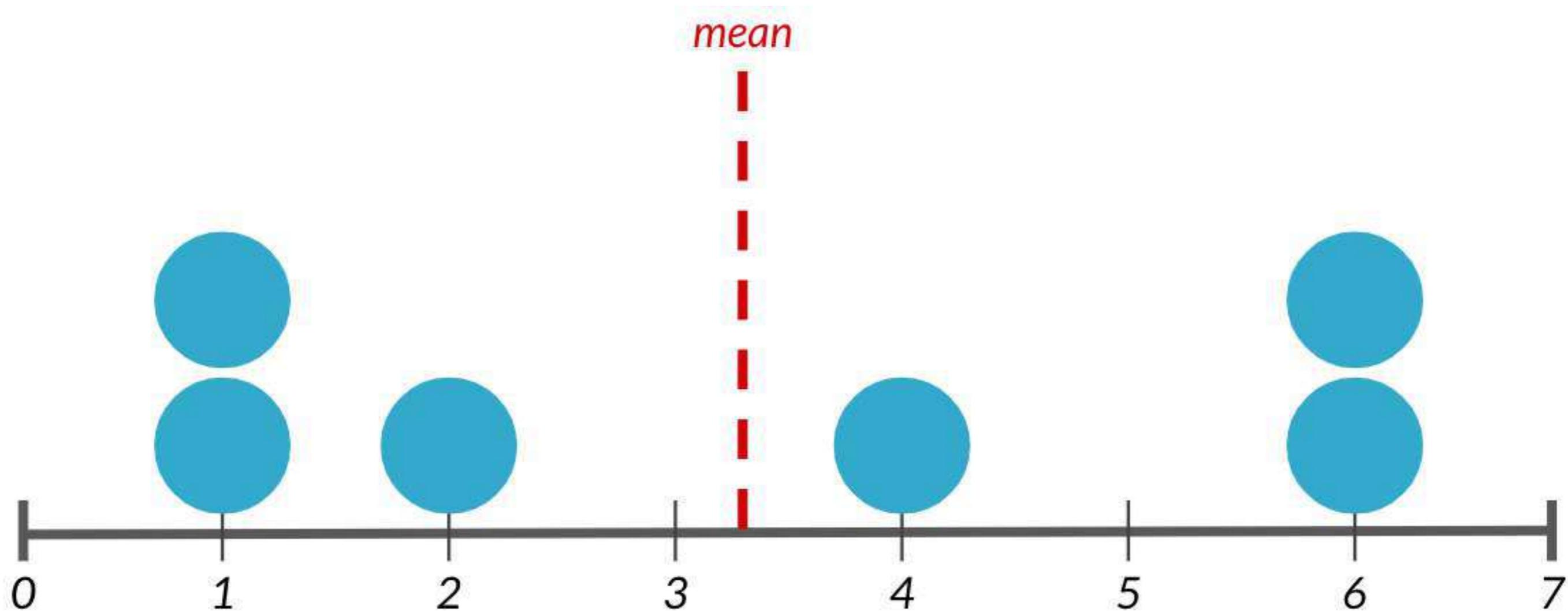
Content Developer, DataCamp

What is spread?



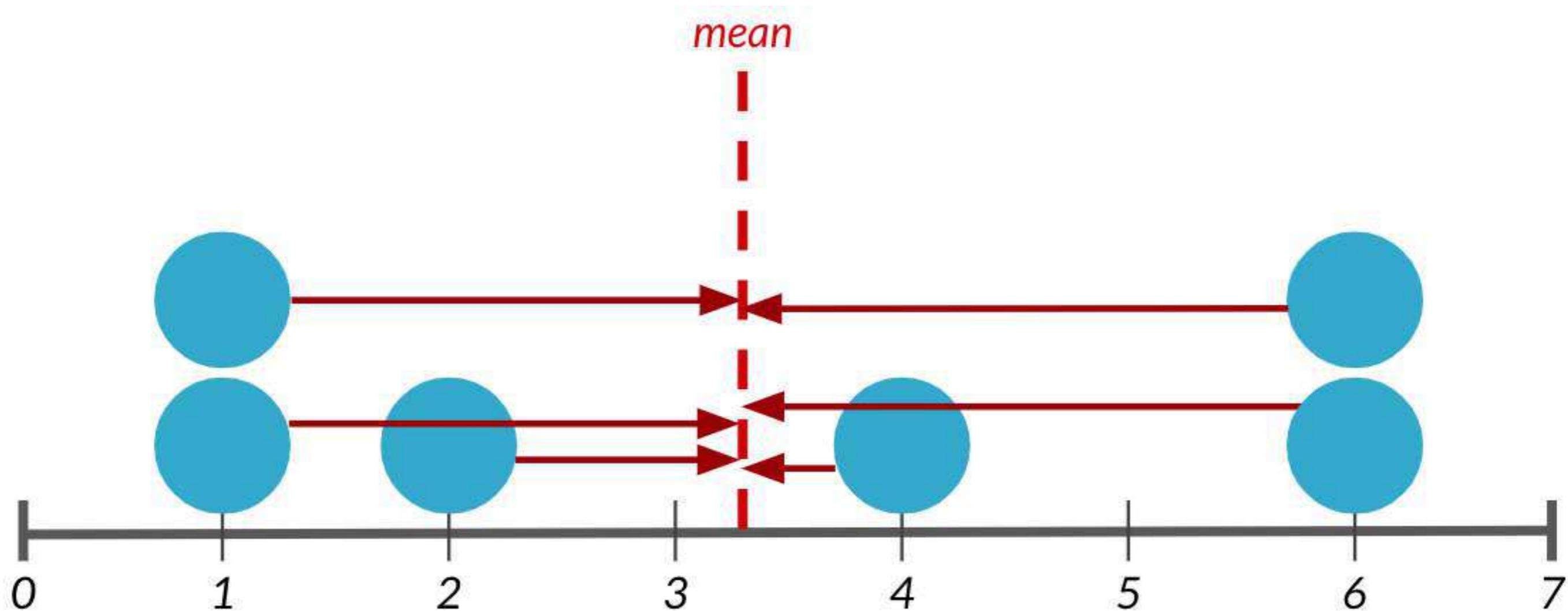
Variance

Average distance from each data point to the data's mean



Variance

Average distance from each data point to the data's mean



Calculating variance

1. Subtract mean from each data point

```
dists = msleep['sleep_total'] -  
        np.mean(msleep['sleep_total'])  
  
print(dists)
```

```
0    1.666265  
1    6.566265  
2    3.966265  
3    4.466265  
4   -6.433735  
  
...
```

2. Square each distance

```
sq_dists = dists ** 2  
  
print(sq_dists)
```

```
0      2.776439  
1     43.115837  
2     15.731259  
3     19.947524  
4     41.392945  
...  
...
```

Calculating variance

3. Sum squared distances

```
sum_sq_dists = np.sum(sq_dists)  
print(sum_sq_dists)
```

1624.065542

4. Divide by number of data points - 1

```
variance = sum_sq_dists / (83 - 1)  
print(variance)
```

19.805677

Use `np.var()`

```
np.var(msleep['sleep_total'], ddof=1)
```

19.805677

Without `ddof=1`, population variance is calculated instead of sample variance:

```
np.var(msleep['sleep_total'])
```

19.567055

Standard deviation

```
np.sqrt(np.var(msleep['sleep_total'], ddof=1))
```

```
4.450357
```

```
np.std(msleep['sleep_total'], ddof=1)
```

```
4.450357
```

Mean absolute deviation

```
dists = msleep['sleep_total'] - mean(msleep$sleep_total)  
np.mean(np.abs(dists))
```

3.566701

Standard deviation vs. mean absolute deviation

- Standard deviation squares distances, penalizing longer distances more than shorter ones.
- Mean absolute deviation penalizes each distance equally.
- One isn't better than the other, but SD is more common than MAD.

Quantiles

```
np.quantile(msleep['sleep_total'], 0.5)
```

```
10.1
```

0.5 quantile = median

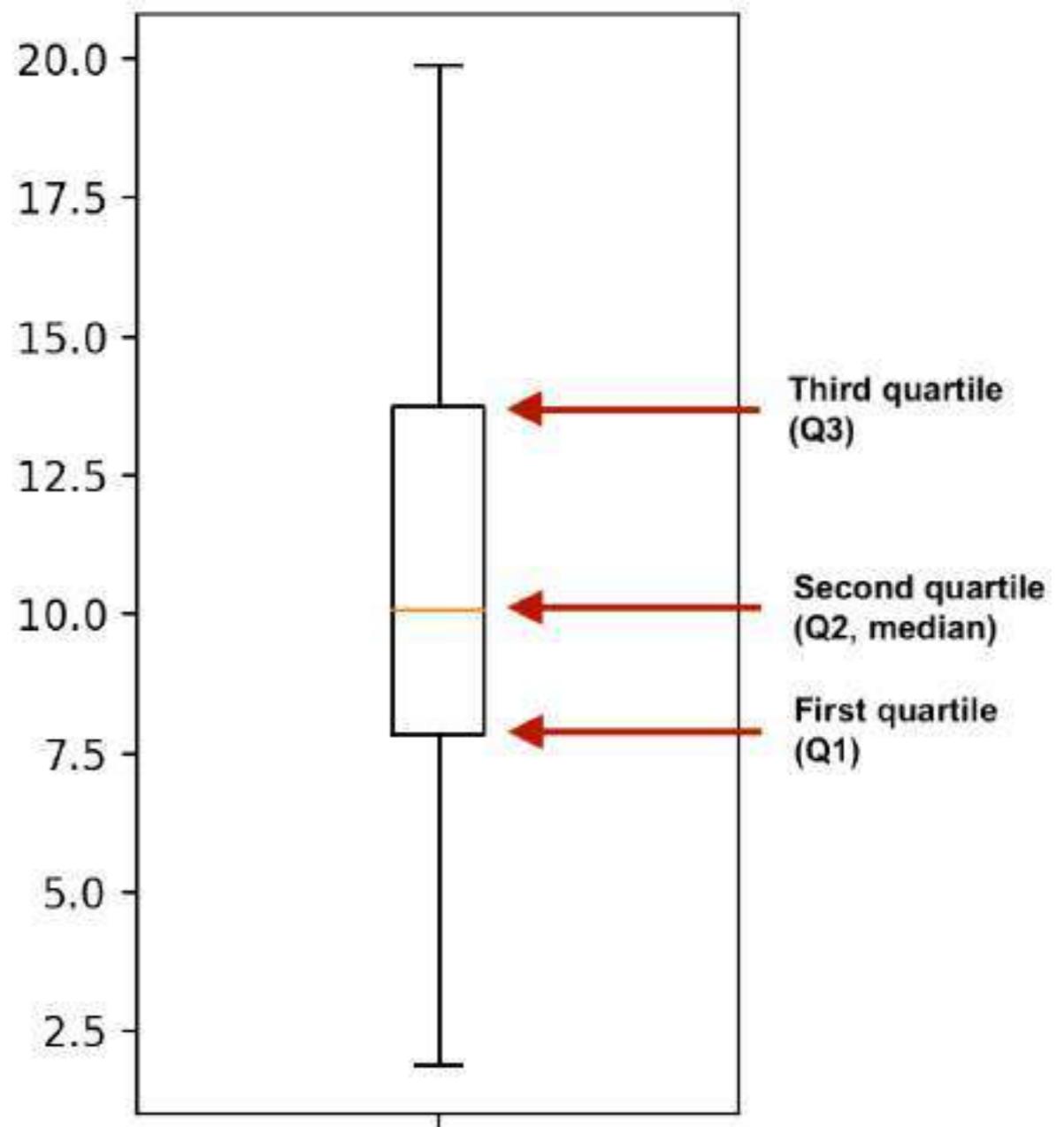
Quartiles:

```
np.quantile(msleep['sleep_total'], [0, 0.25, 0.5, 0.75, 1])
```

```
array([ 1.9 ,  7.85, 10.1 , 13.75, 19.9 ])
```

Boxplots use quartiles

```
import matplotlib.pyplot as plt  
plt.boxplot(msleep['sleep_total'])  
plt.show()
```



Quantiles using np.linspace()

```
np.quantile(msleep['sleep_total'], [0, 0.2, 0.4, 0.6, 0.8, 1])
```

```
array([ 1.9 ,  6.24,  9.48, 11.14, 14.4 , 19.9 ])
```

```
np.linspace(start, stop, num)
```

```
np.quantile(msleep['sleep_total'], np.linspace(0, 1, 5))
```

```
array([ 1.9 ,  7.85, 10.1 , 13.75, 19.9 ])
```

Interquartile range (IQR)

Height of the box in a boxplot

```
np.quantile(msleep['sleep_total'], 0.75) - np.quantile(msleep['sleep_total'], 0.25)
```

```
5.9
```

```
from scipy.stats import iqr  
iqr(msleep['sleep_total'])
```

```
5.9
```

Outliers

Outlier: data point that is substantially different from the others

How do we know what a substantial difference is? A data point is an outlier if:

- $\text{data} < Q1 - 1.5 \times \text{IQR}$ or
- $\text{data} > Q3 + 1.5 \times \text{IQR}$

Finding outliers

```
from scipy.stats import iqr  
  
iqr = iqr(msleep['bodywt'])  
  
lower_threshold = np.quantile(msleep['bodywt'], 0.25) - 1.5 * iqr  
upper_threshold = np.quantile(msleep['bodywt'], 0.75) + 1.5 * iqr
```

```
msleep[(msleep['bodywt'] < lower_threshold) | (msleep['bodywt'] > upper_threshold)]
```

		name	vore	sleep_total	bodywt
4		Cow	herbi	4.0	600.000
20		Asian elephant	herbi	3.9	2547.000
22		Horse	herbi	2.9	521.000
...					

All in one go

```
msleep['bodywt'].describe()
```

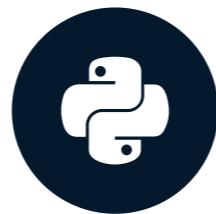
```
count      83.000000
mean      166.136349
std       786.839732
min       0.005000
25%      0.174000
50%      1.670000
75%     41.750000
max     6654.000000
Name: bodywt, dtype: float64
```

Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

What are the chances?

INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

Content Developer, DataCamp

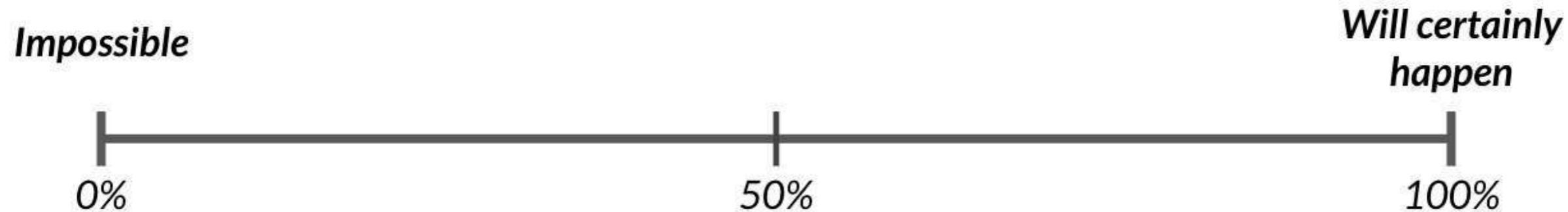
Measuring chance

What's the probability of an event?

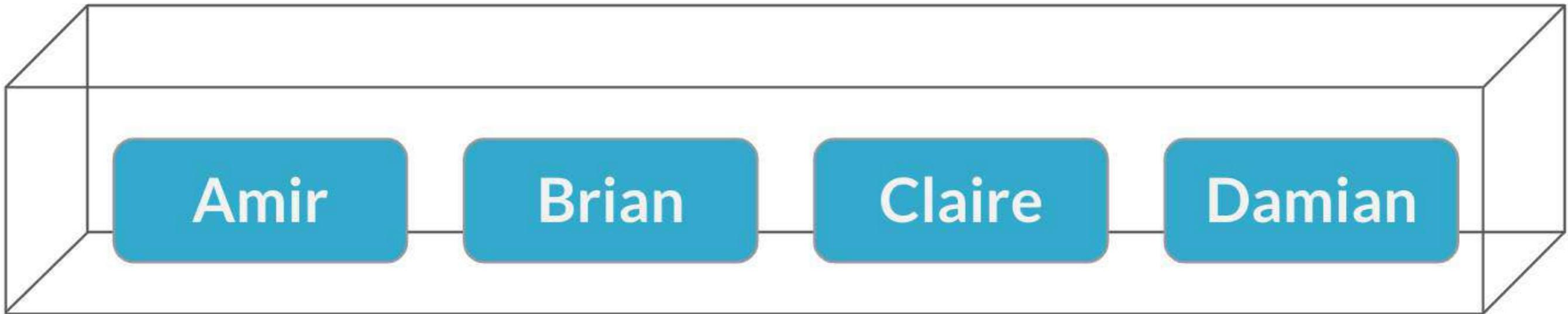
$$P(\text{event}) = \frac{\# \text{ ways event can happen}}{\text{total } \# \text{ of possible outcomes}}$$

Example: a coin flip

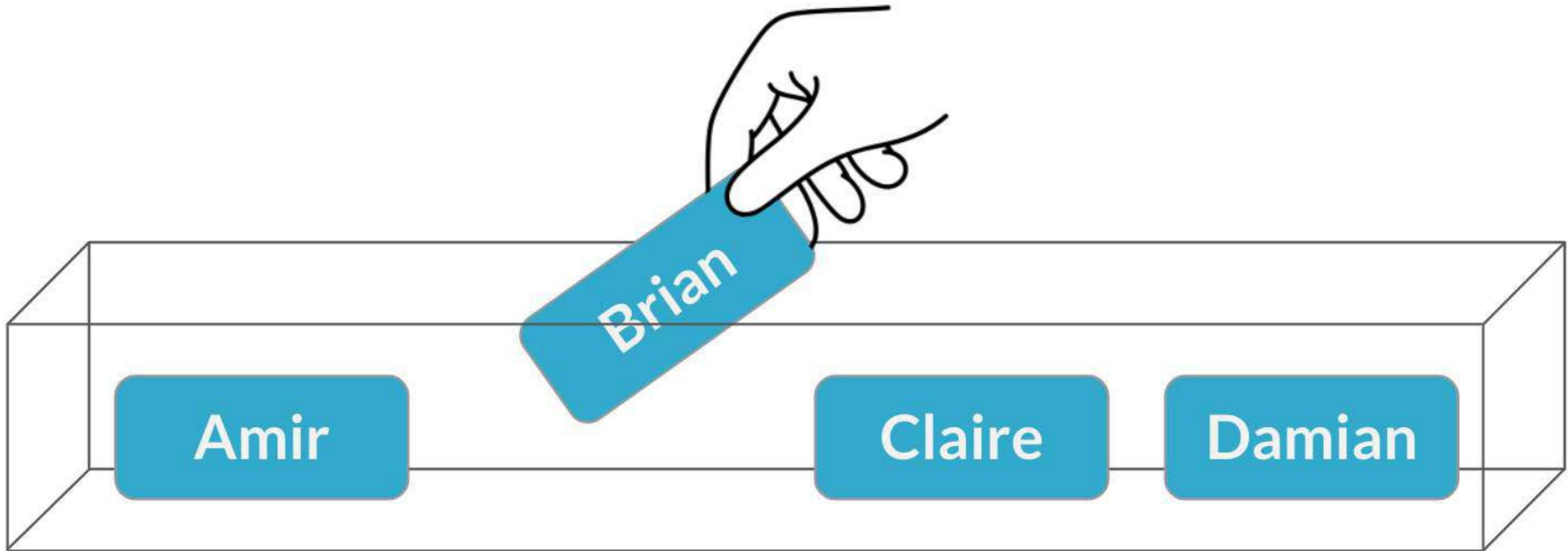
$$P(\text{heads}) = \frac{1 \text{ way to get heads}}{2 \text{ possible outcomes}} = \frac{1}{2} = 50\%$$



Assigning salespeople



Assigning salespeople



$$P(\text{Brian}) = \frac{1}{4} = 25\%$$

Sampling from a DataFrame

```
print(sales_counts)
```

```
      name  n_sales  
0   Amir     178  
1  Brian     128  
2  Claire      75  
3  Damian      69
```

```
sales_counts.sample()
```

```
      name  n_sales  
1  Brian     128
```

```
sales_counts.sample()
```

```
      name  n_sales  
2  Claire      75
```

Setting a random seed

```
np.random.seed(10)  
sales_counts.sample()
```

```
name    n_sales  
1  Brian        128
```

```
np.random.seed(10)  
sales_counts.sample()
```

```
name    n_sales  
1  Brian        128
```

```
np.random.seed(10)  
sales_counts.sample()
```

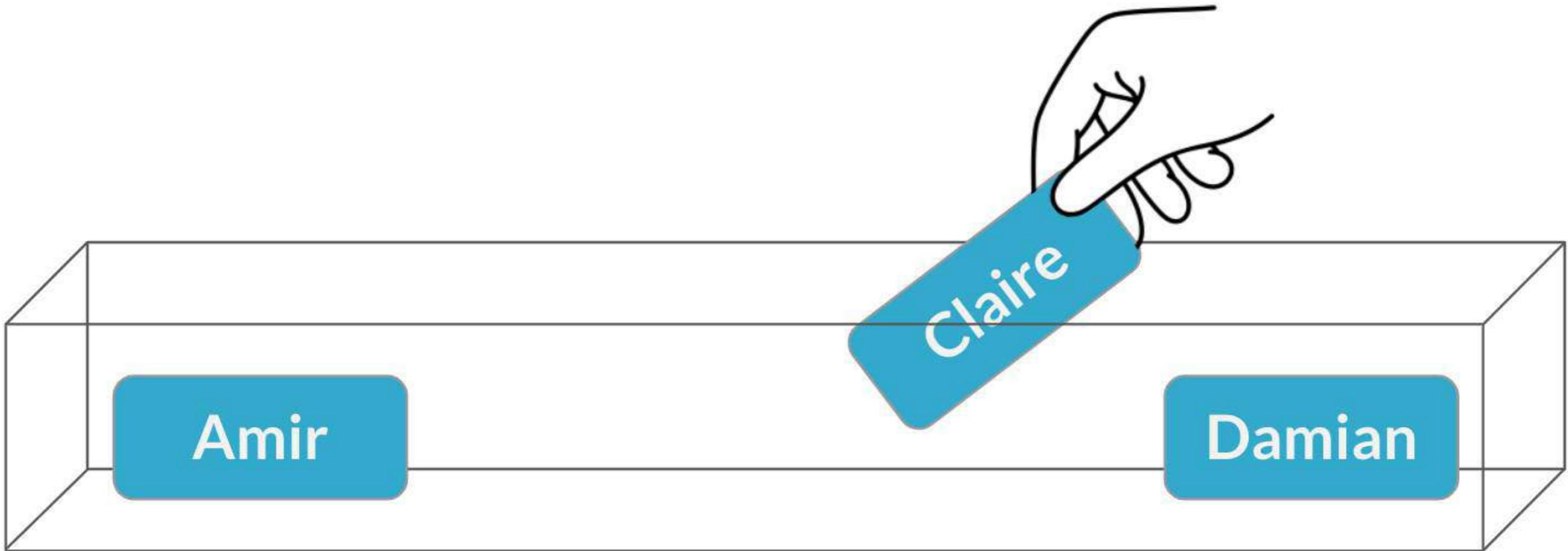
```
name    n_sales  
1  Brian        128
```

A second meeting

Sampling without replacement



A second meeting



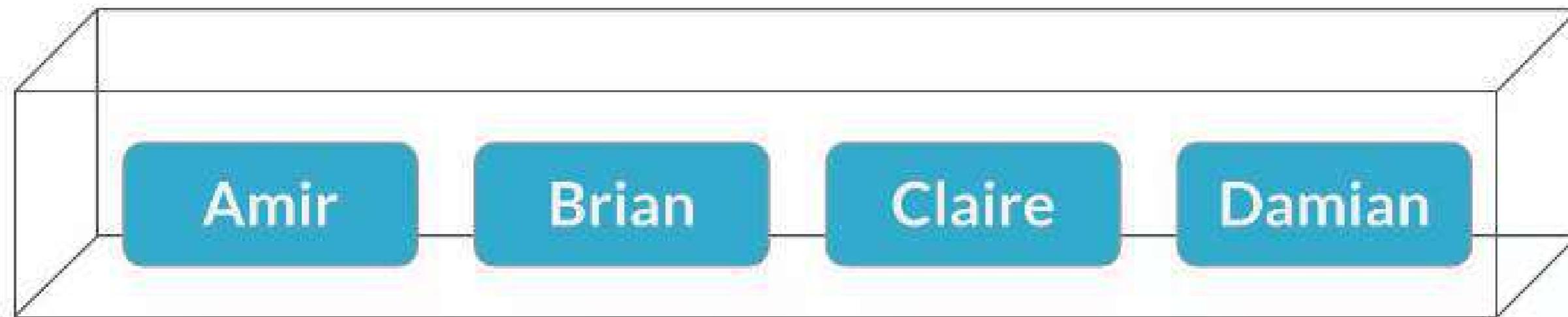
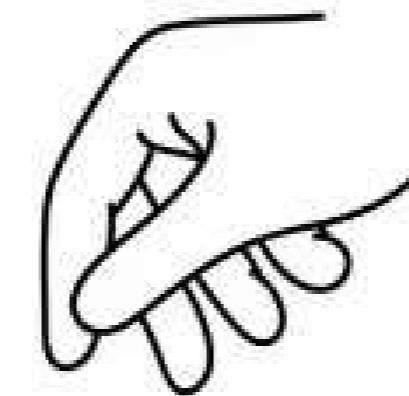
$$P(\text{Claire}) = \frac{1}{3} = 33\%$$

Sampling twice in Python

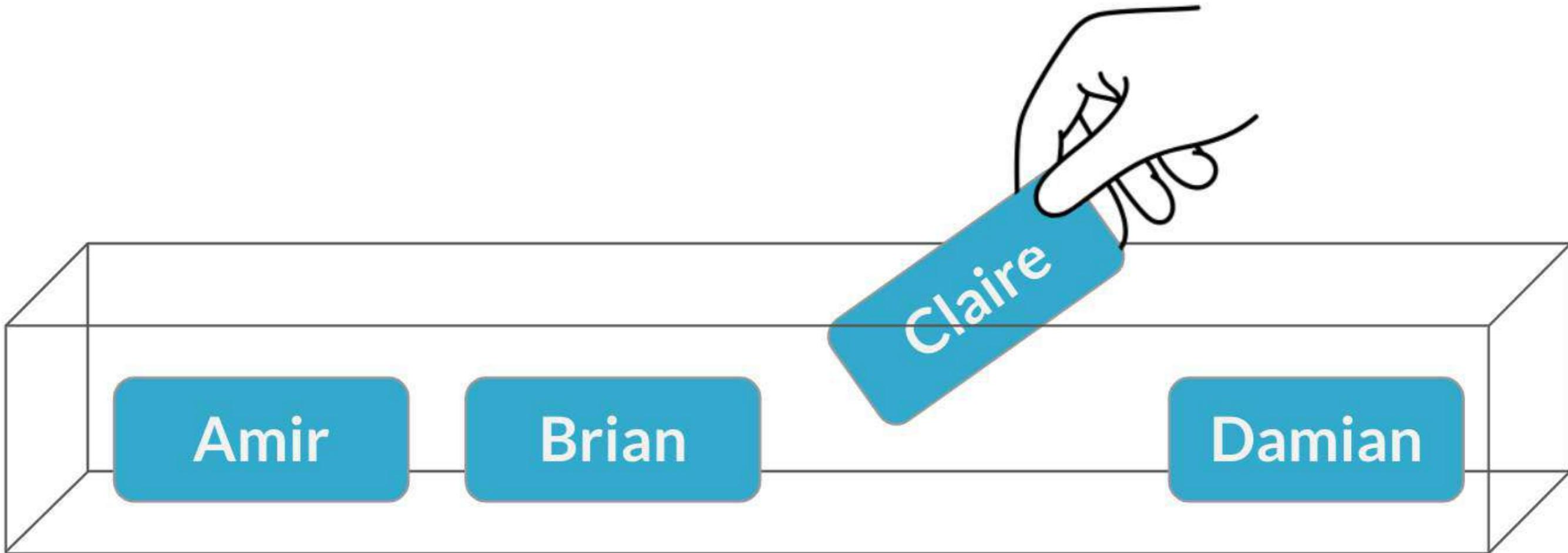
```
sales_counts.sample(2)
```

```
    name  n_sales  
1  Brian      128  
2 Claire       75
```

Sampling with replacement



Sampling with replacement



$$P(\text{Claire}) = \frac{1}{4} = 25\%$$

Sampling with/without replacement in Python

```
sales_counts.sample(5, replace = True)
```

```
      name  n_sales
1  Brian      128
2  Claire      75
1  Brian      128
3  Damian      69
0  Amir       178
```

Independent events

*Two events are **independent** if the probability of the second event **isn't** affected by the outcome of the first event.*

Sampling with Replacement

First pick

Second pick

Amir

Brian

Claire

Damian

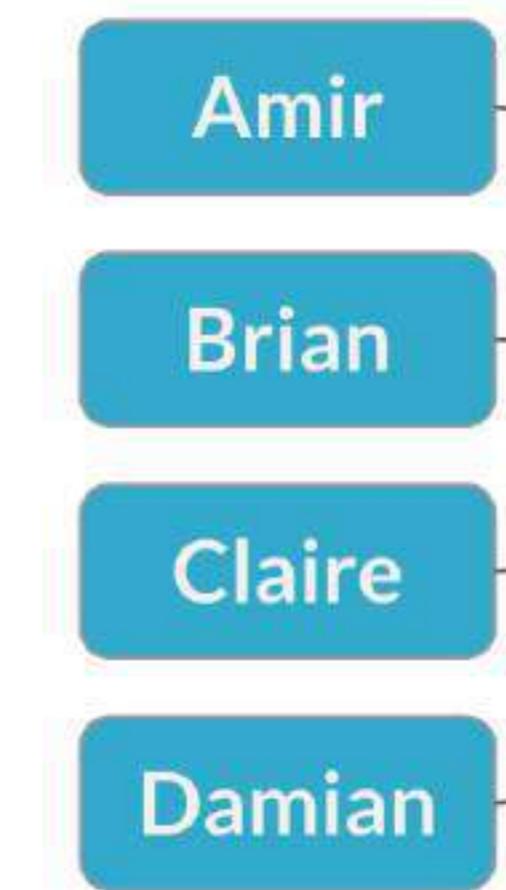
Independent events

Two events are *independent* if the probability of the second event *isn't* affected by the outcome of the first event.

Sampling with replacement = each pick is independent

Sampling with Replacement

First pick



Second pick

Dependent events

*Two events are **dependent** if the probability of the second event **is** affected by the outcome of the first event.*

Sampling without Replacement

First pick

Second pick

Amir

Brian

Damian

Claire

Dependent events

*Two events are **dependent** if the probability of the second event **is affected** by the outcome of the first event.*

Sampling without Replacement

First pick

Amir

Second pick

Brian

Damian

Claire

Claire

0%

Dependent events

*Two events are **dependent** if the probability of the second event **is** affected by the outcome of the first event.*

Sampling without replacement = each pick is dependent

Sampling without Replacement

First pick

Amir

Brian

Damian

Claire

Second pick

Claire

33%

Claire

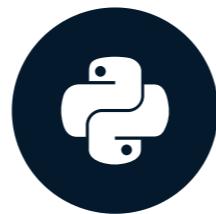
0%

Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

Discrete distributions

INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

Content Developer, DataCamp

Rolling the dice



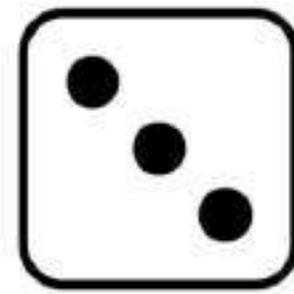
Rolling the dice



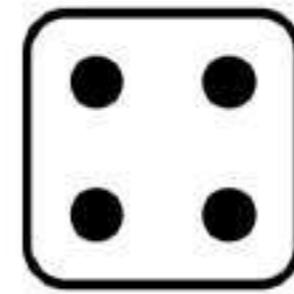
$\frac{1}{6}$



$\frac{1}{6}$



$\frac{1}{6}$



$\frac{1}{6}$

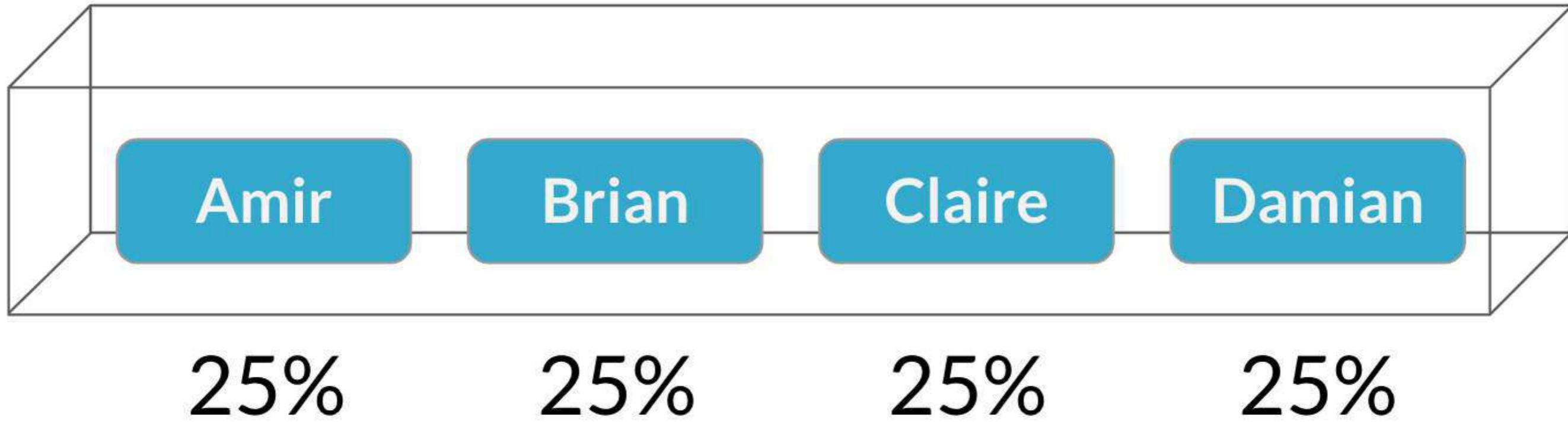


$\frac{1}{6}$



$\frac{1}{6}$

Choosing salespeople



Probability distribution

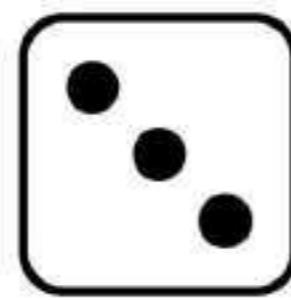
Describes the probability of each possible outcome in a scenario



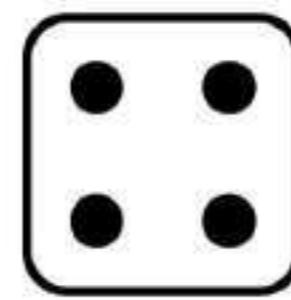
$\frac{1}{6}$



$\frac{1}{6}$



$\frac{1}{6}$



$\frac{1}{6}$



$\frac{1}{6}$



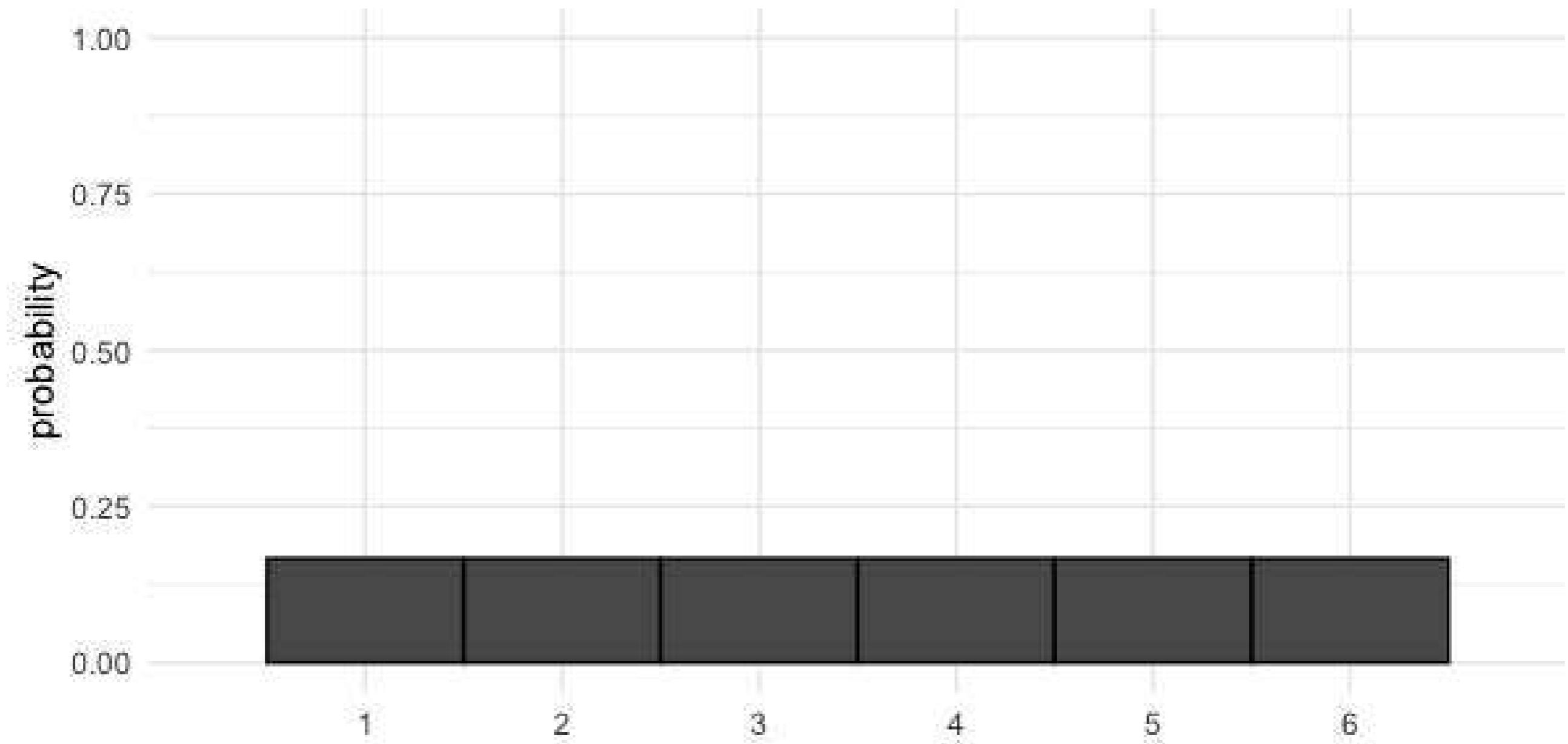
$\frac{1}{6}$

Expected value: mean of a probability distribution

Expected value of a fair die roll =

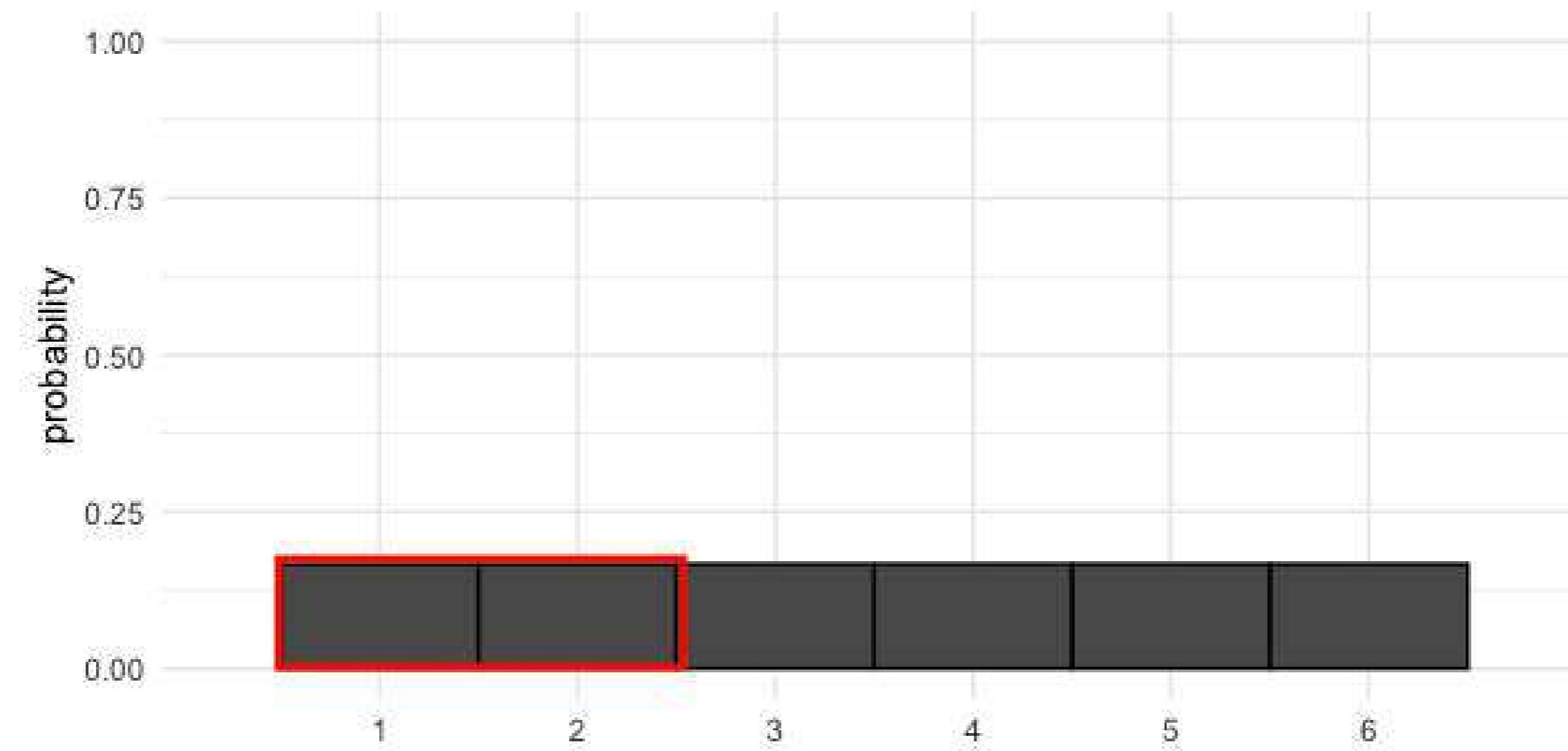
$$(1 \times \frac{1}{6}) + (2 \times \frac{1}{6}) + (3 \times \frac{1}{6}) + (4 \times \frac{1}{6}) + (5 \times \frac{1}{6}) + (6 \times \frac{1}{6}) = 3.5$$

Visualizing a probability distribution



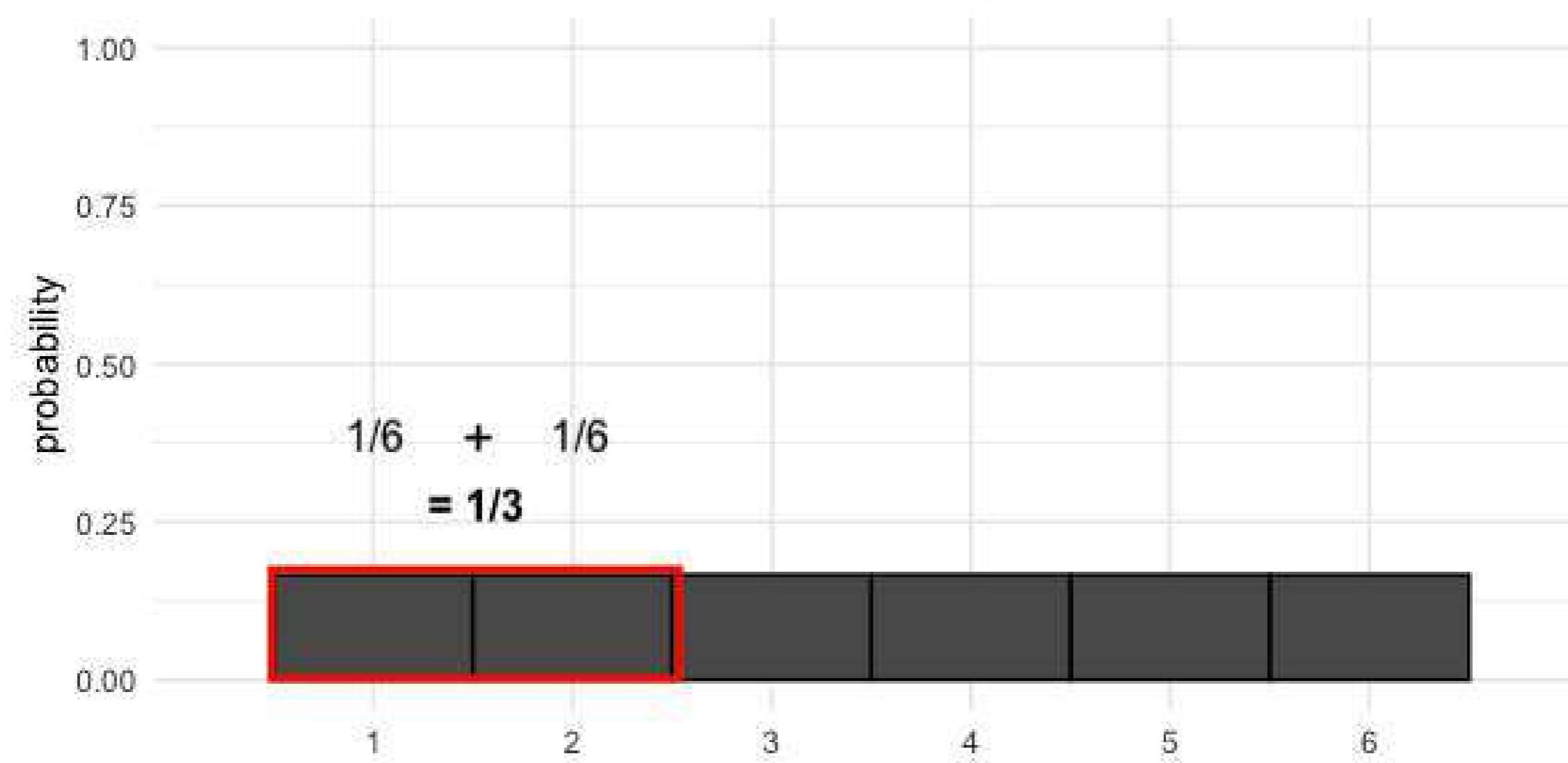
Probability = area

$$P(\text{die roll}) \leq 2 = ?$$

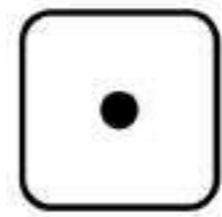


Probability = area

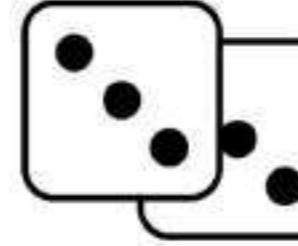
$$P(\text{die roll}) \leq 2 = 1/3$$



Uneven die



$\frac{1}{6}$



$\frac{1}{3}$



$\frac{1}{6}$



$\frac{1}{6}$

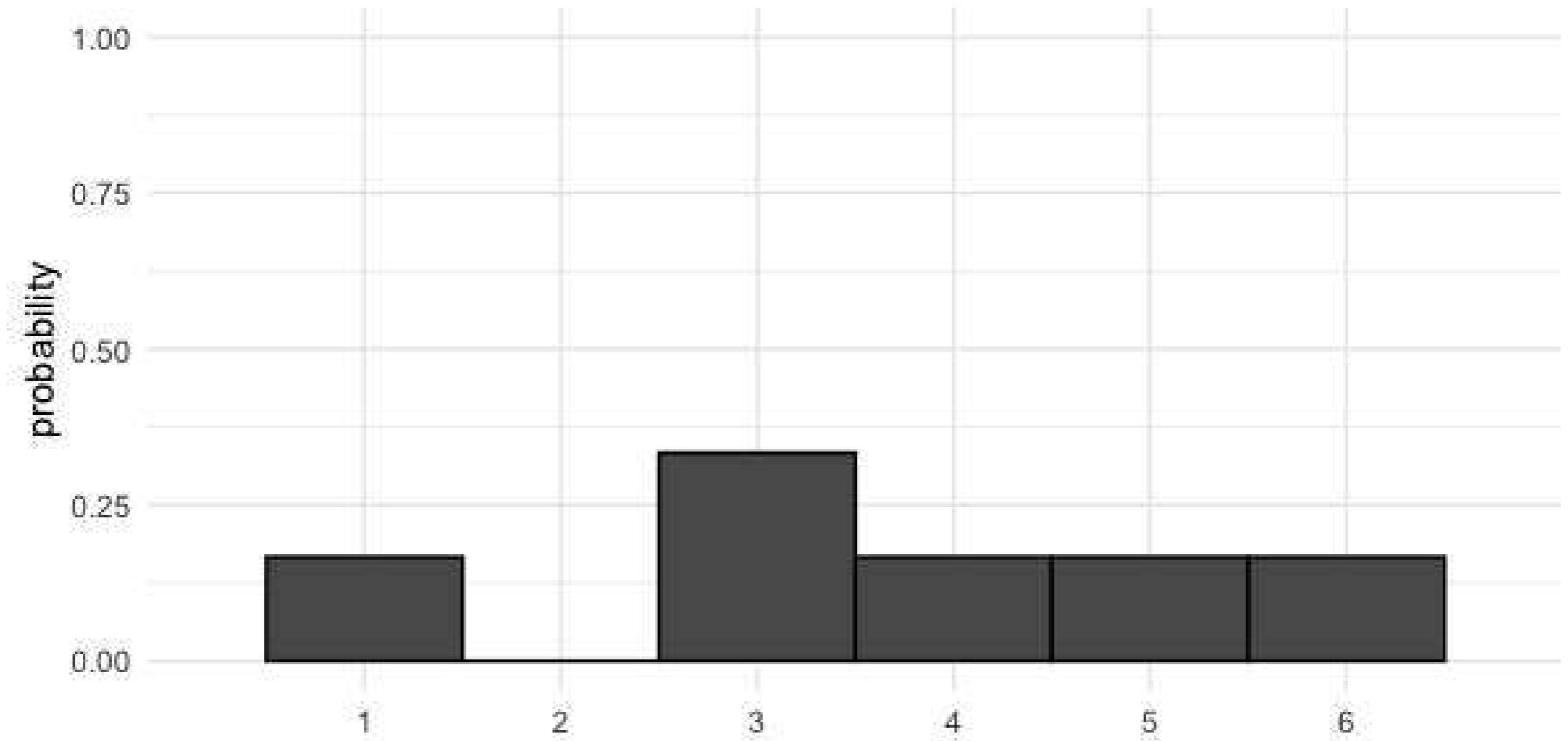


$\frac{1}{6}$

Expected value of uneven die roll =

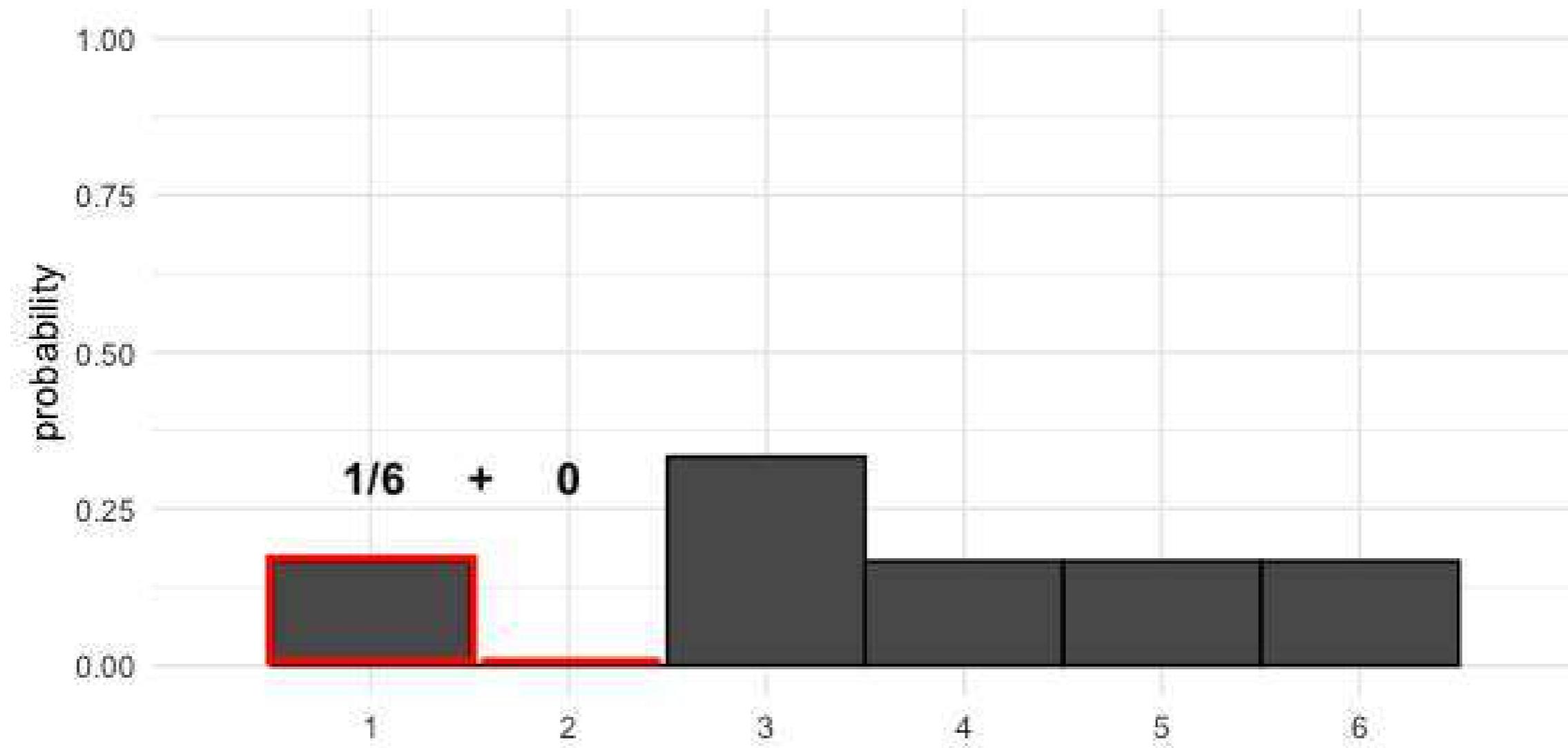
$$(1 \times \frac{1}{6}) + (2 \times 0) + (3 \times \frac{1}{3}) + (4 \times \frac{1}{6}) + (5 \times \frac{1}{6}) + (6 \times \frac{1}{6}) = 3.67$$

Visualizing uneven probabilities



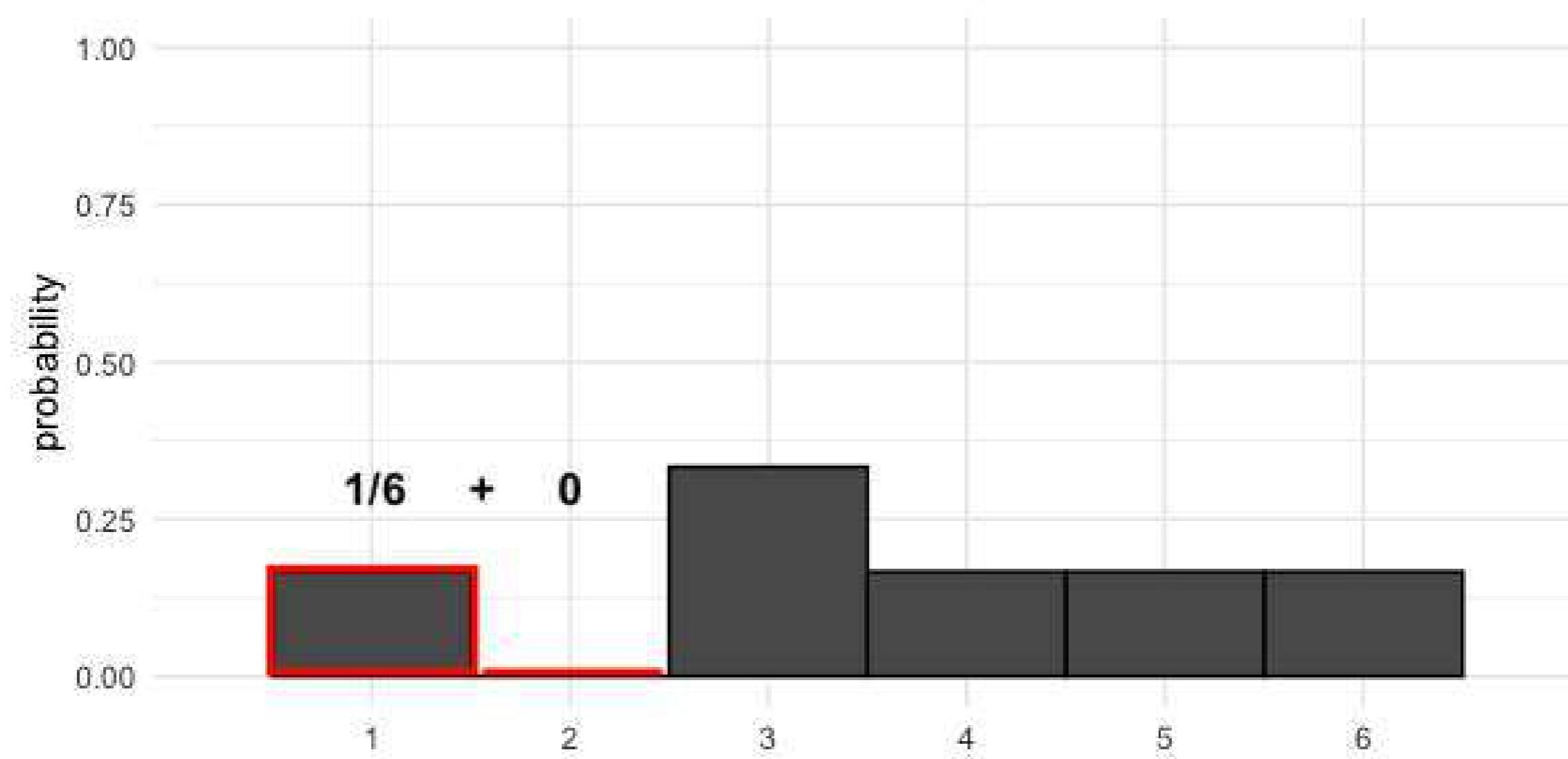
Adding areas

$P(\text{uneven die roll}) \leq 2 = ?$



Adding areas

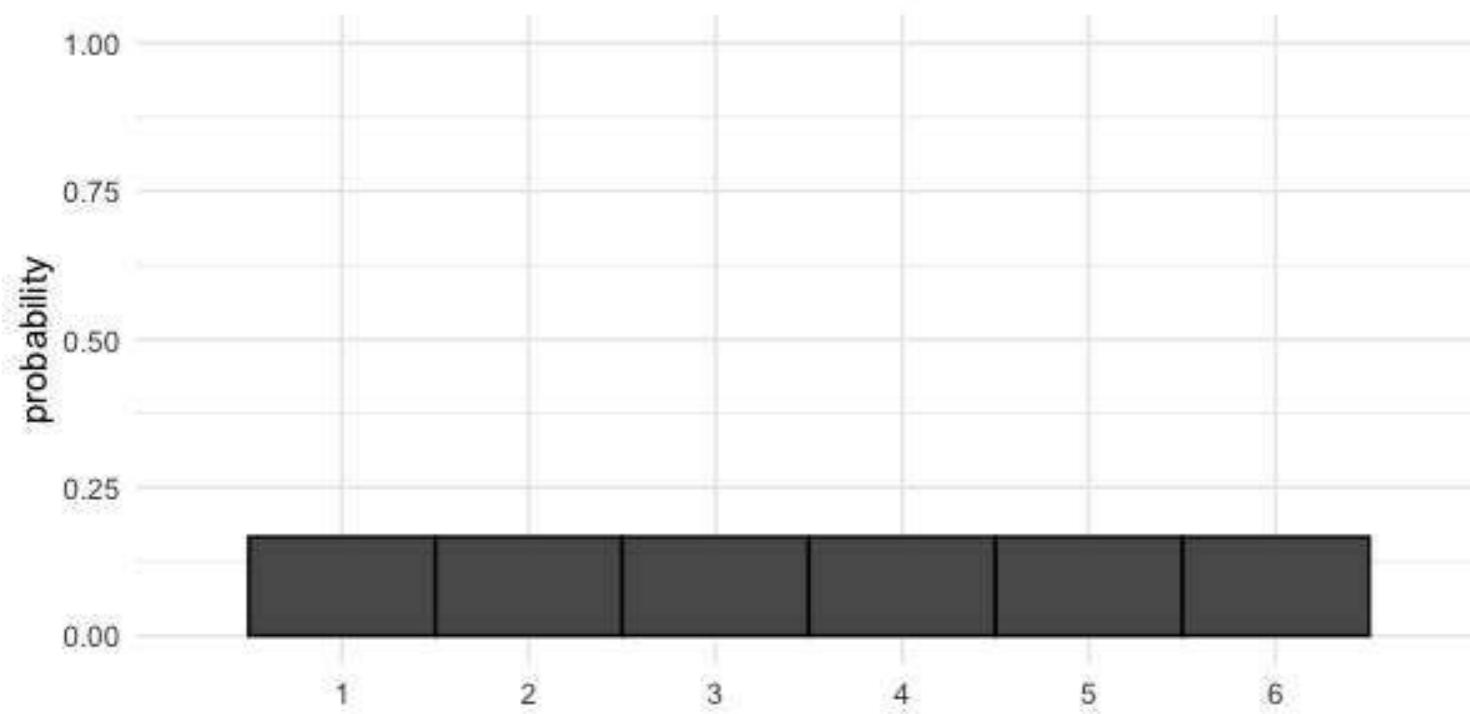
$$P(\text{uneven die roll}) \leq 2 = 1/6$$



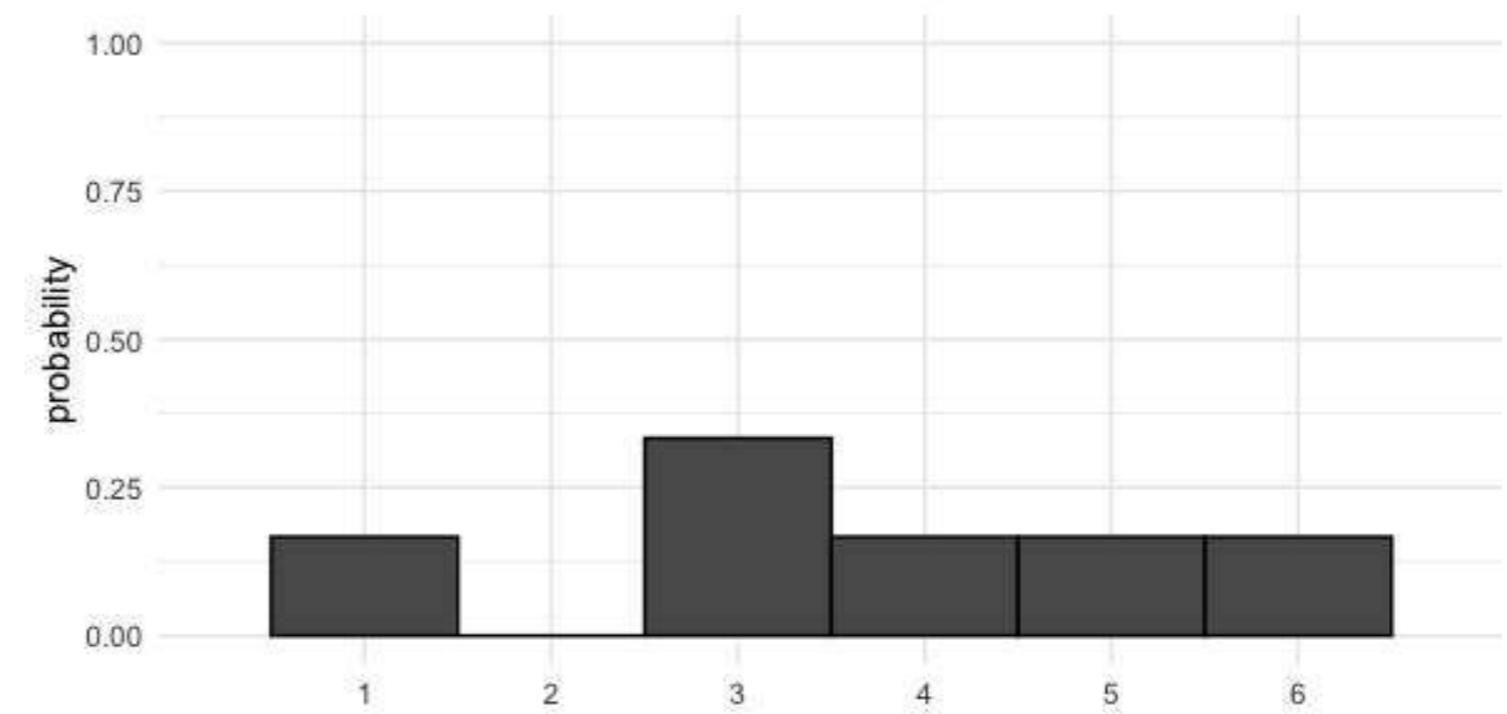
Discrete probability distributions

Describe probabilities for discrete outcomes

Fair die



Uneven die



Discrete uniform distribution

Sampling from discrete distributions

```
print(die)
```

```
number      prob
0          1  0.166667
1          2  0.166667
2          3  0.166667
3          4  0.166667
4          5  0.166667
5          6  0.166667
```

```
np.mean(die['number'])
```

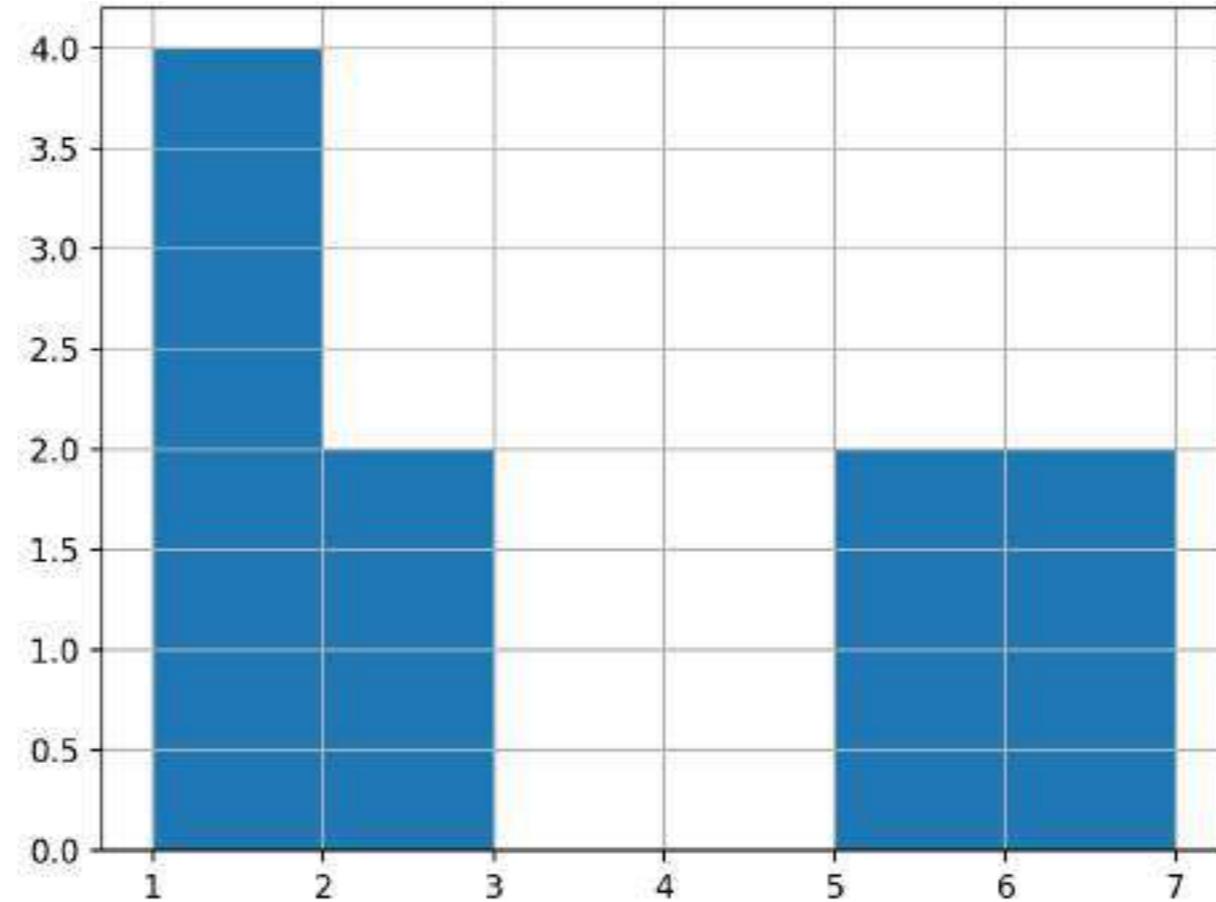
```
3.5
```

```
rolls_10 = die.sample(10, replace = True)
rolls_10
```

```
number      prob
0          1  0.166667
0          1  0.166667
4          5  0.166667
1          2  0.166667
0          1  0.166667
0          1  0.166667
5          6  0.166667
5          6  0.166667
...
...
```

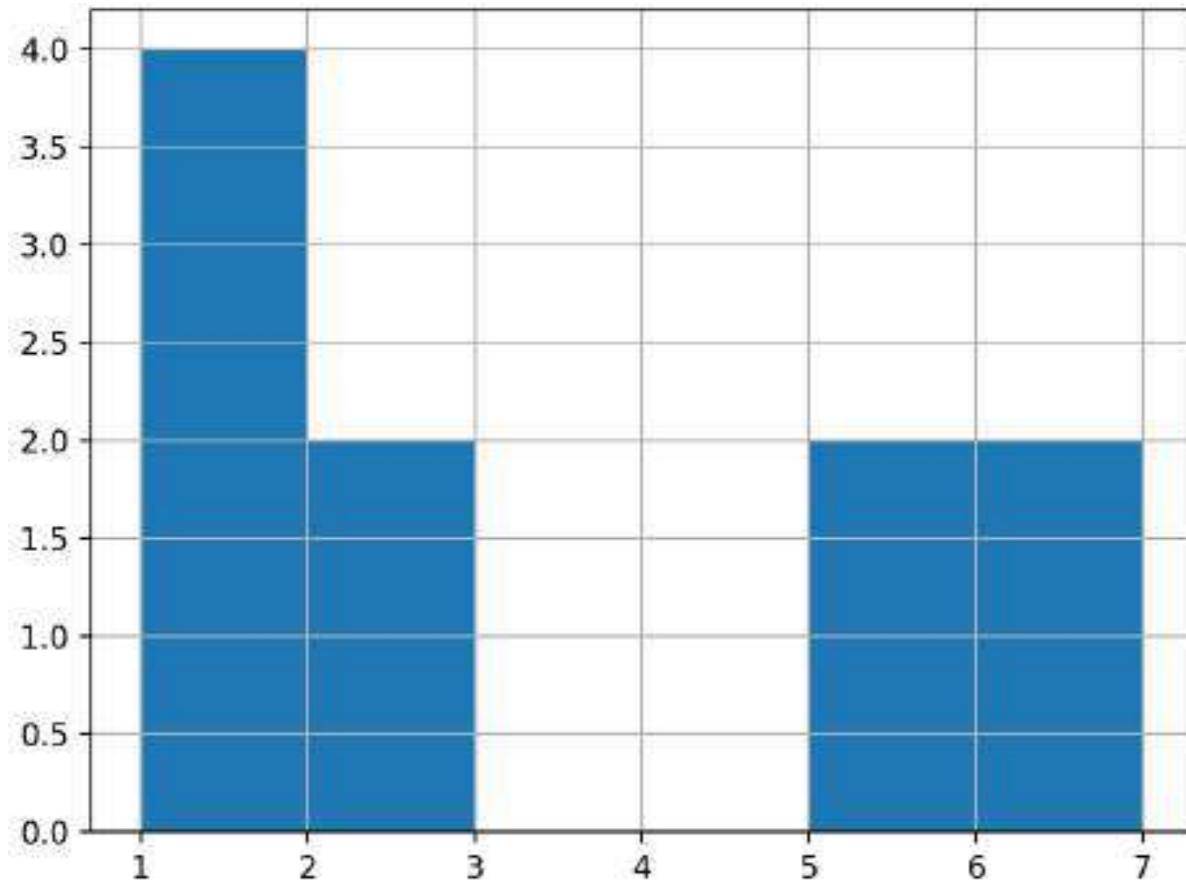
Visualizing a sample

```
rolls_10['number'].hist(bins=np.linspace(1,7,7))  
plt.show()
```



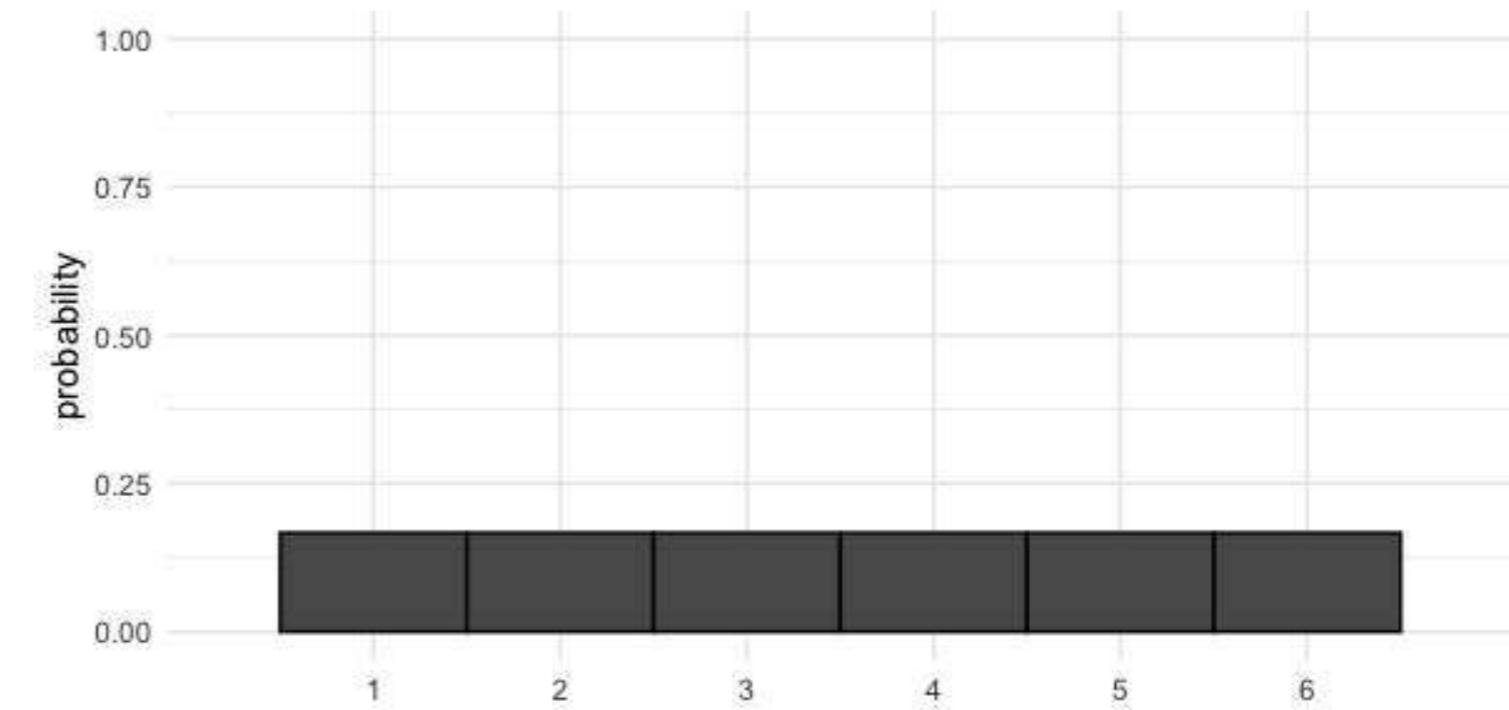
Sample distribution vs. theoretical distribution

Sample of 10 rolls



```
np.mean(rolls_10['number']) = 3.0
```

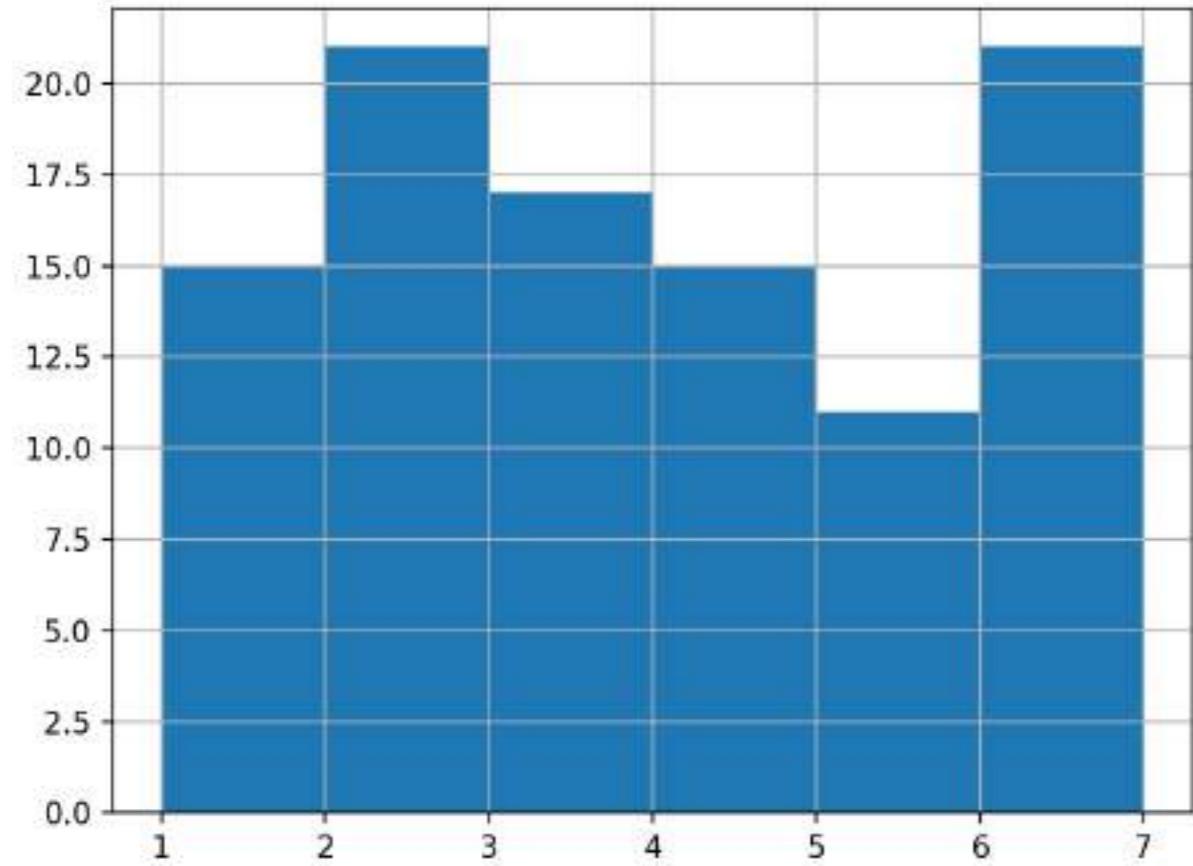
Theoretical probability distribution



```
mean(die['number']) = 3.5
```

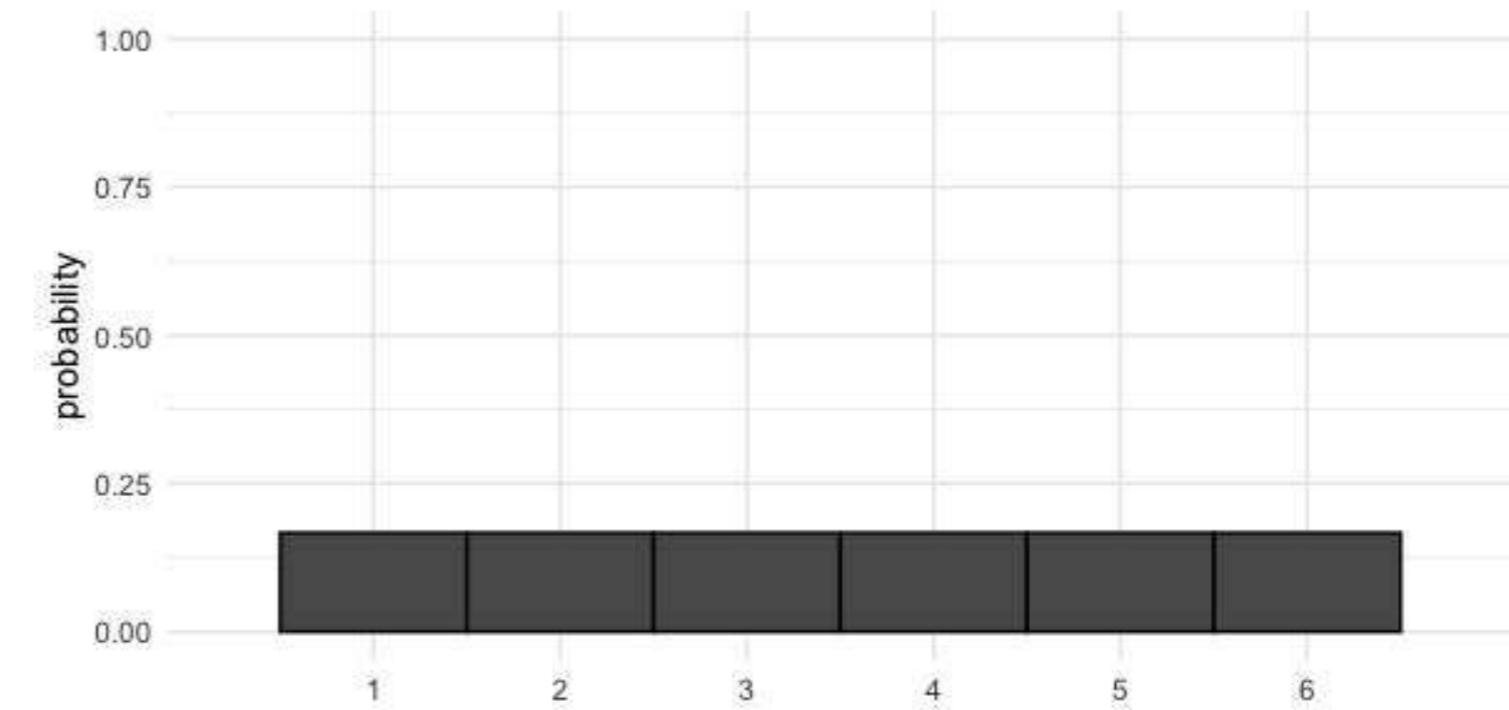
A bigger sample

Sample of 100 rolls



```
np.mean(rolls_100['number']) = 3.4
```

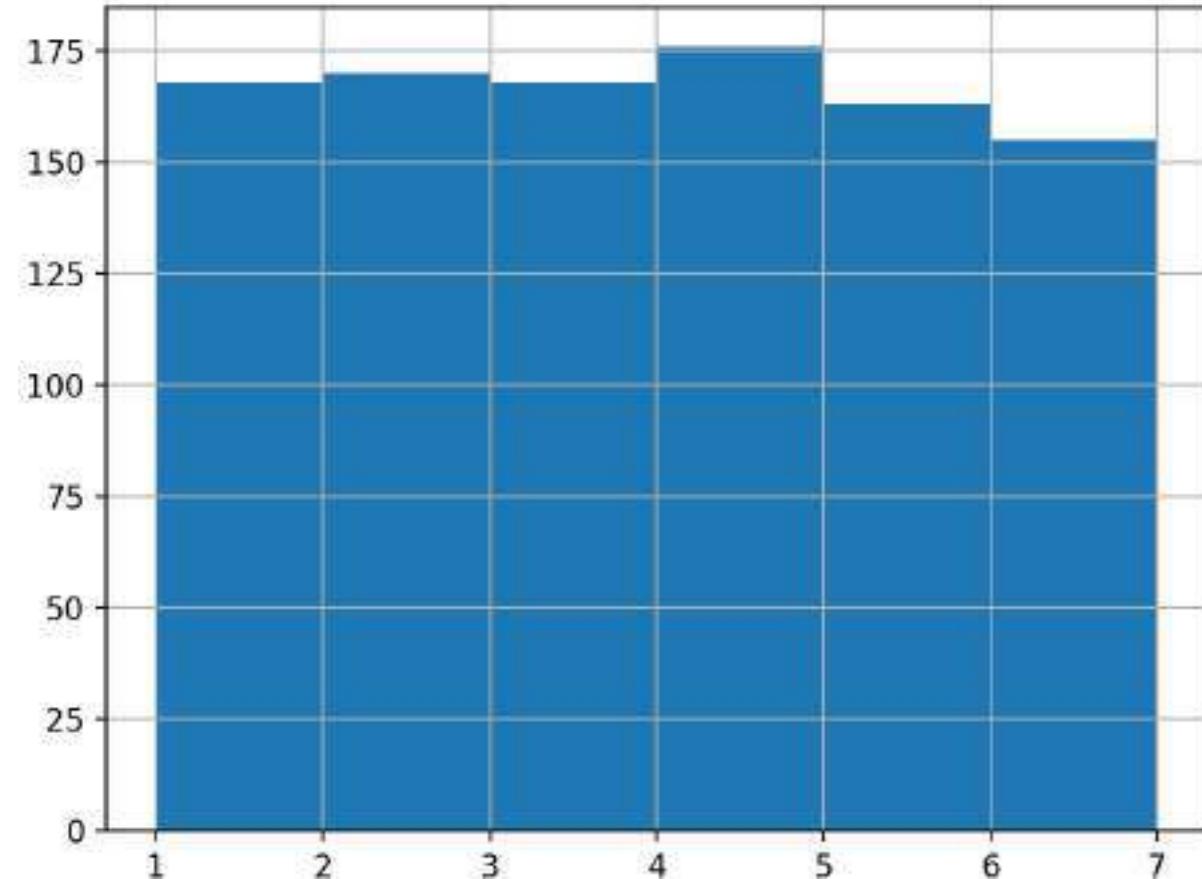
Theoretical probability distribution



```
mean(die['number']) = 3.5
```

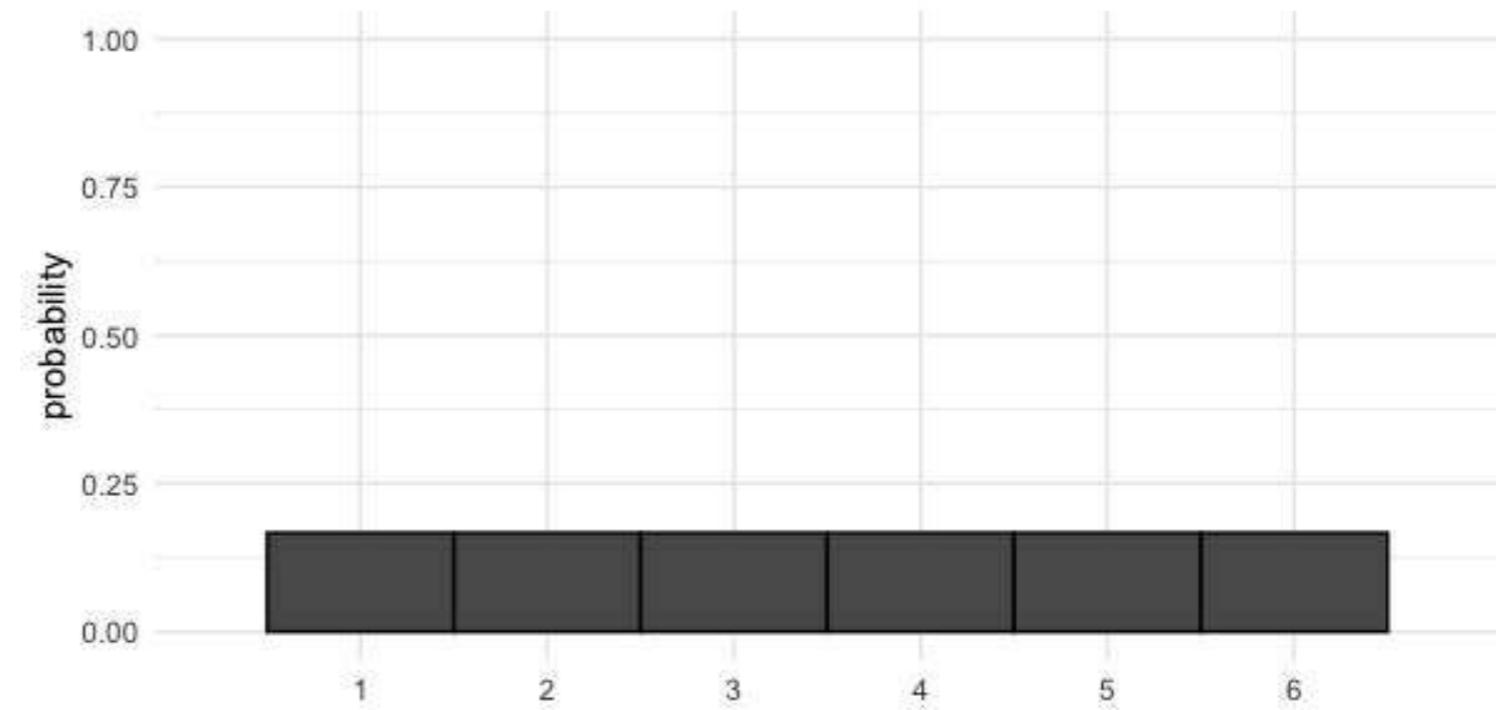
An even bigger sample

Sample of 1000 rolls



```
np.mean(rolls_1000['number']) = 3.48
```

Theoretical probability distribution



```
mean(die['number']) = 3.5
```

Law of large numbers

As the size of your sample increases, the sample mean will approach the expected value.

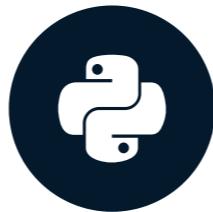
Sample size	Mean
10	3.00
100	3.40
1000	3.48

Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

Continuous distributions

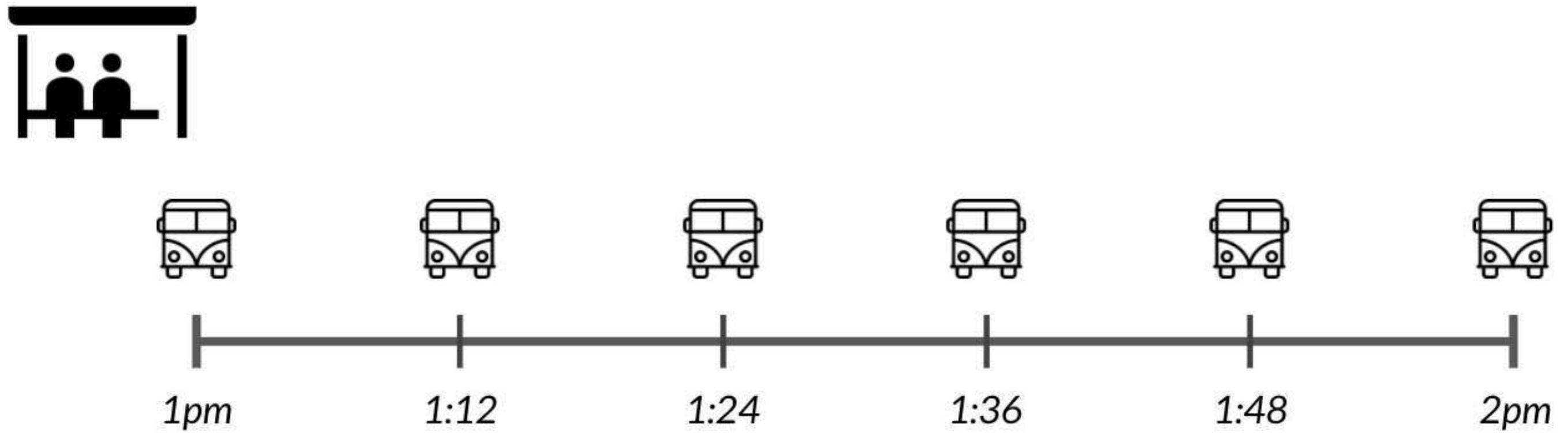
INTRODUCTION TO STATISTICS IN PYTHON



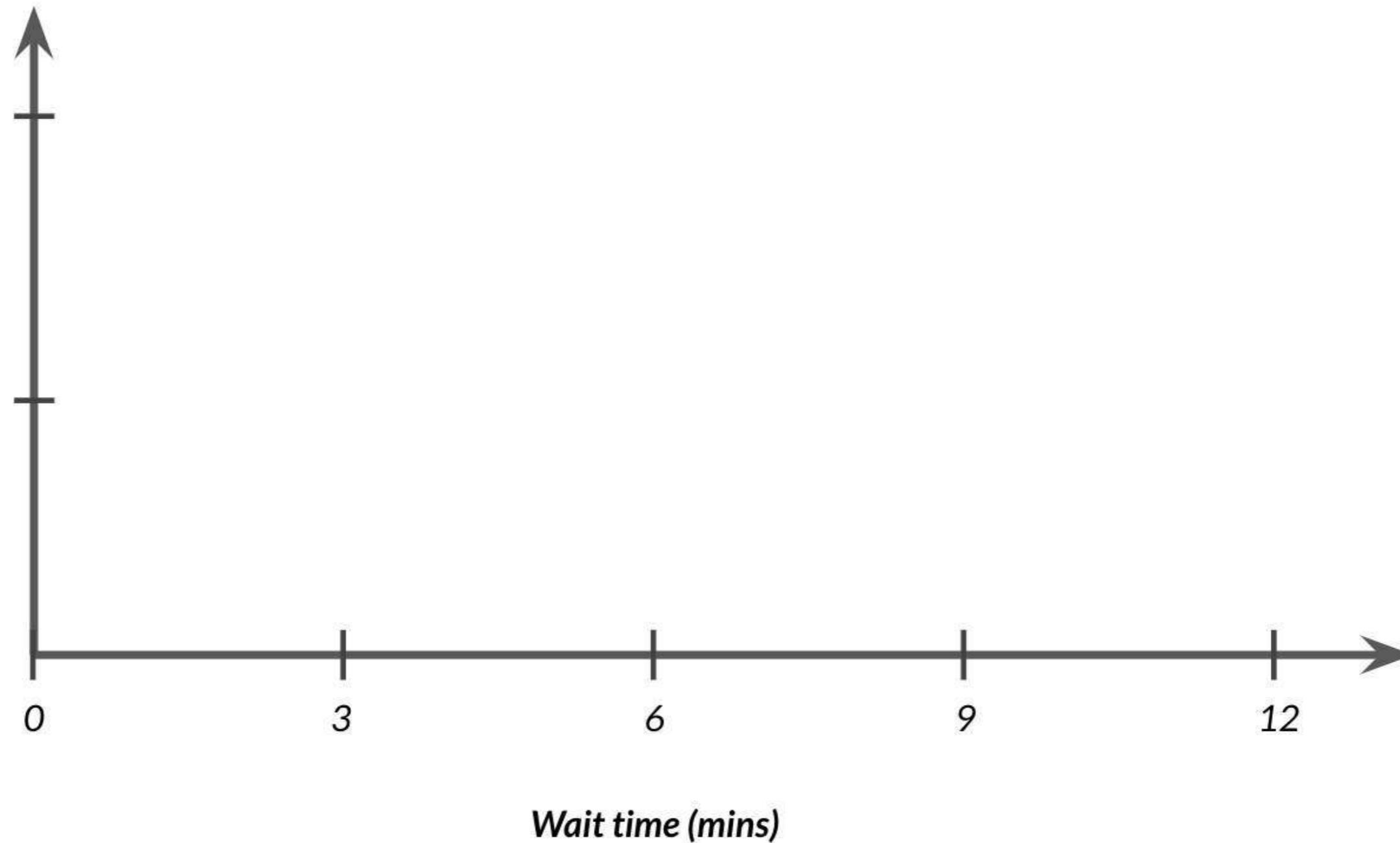
Maggie Matsui

Content Developer, DataCamp

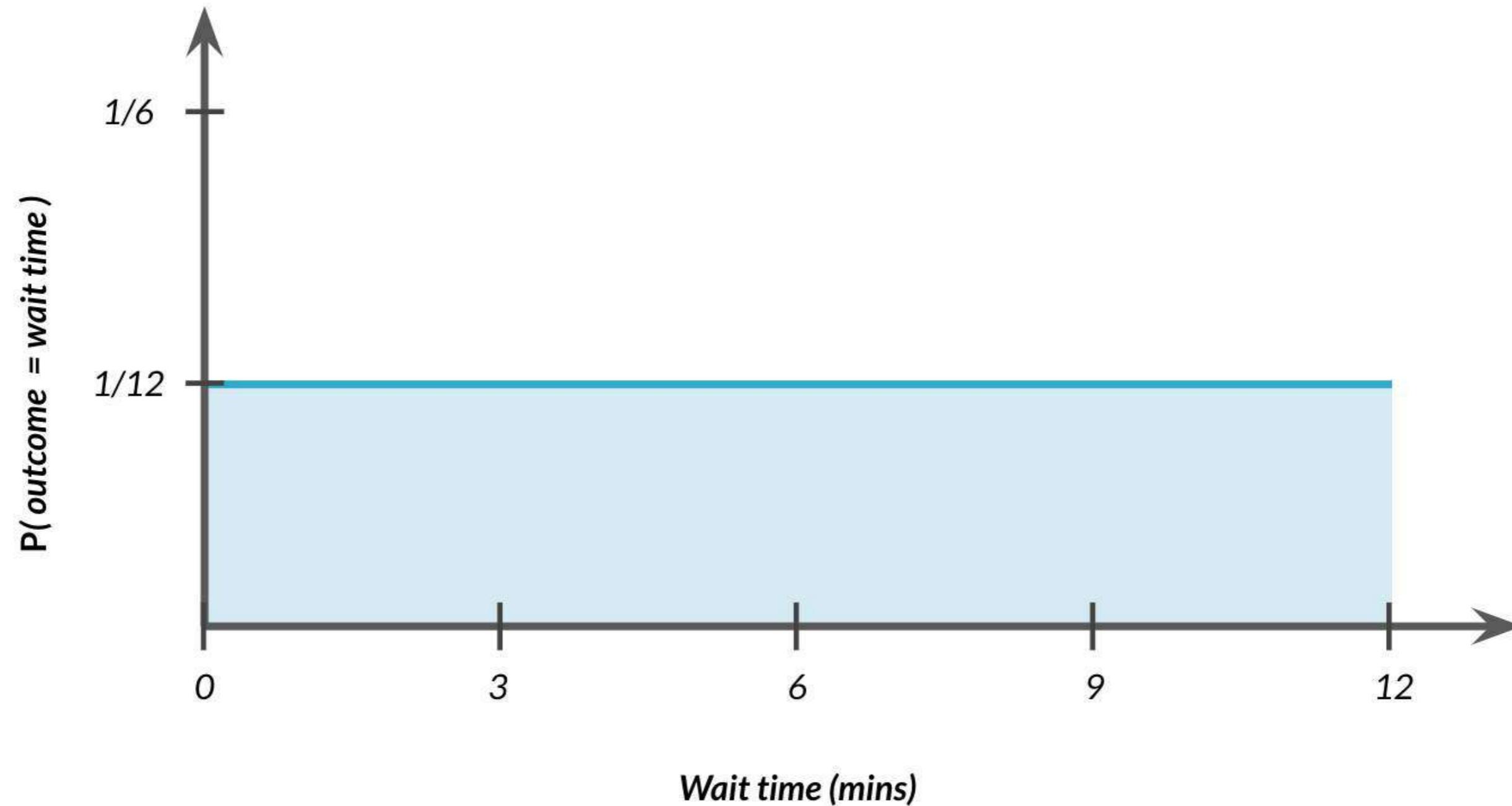
Waiting for the bus



Continuous uniform distribution

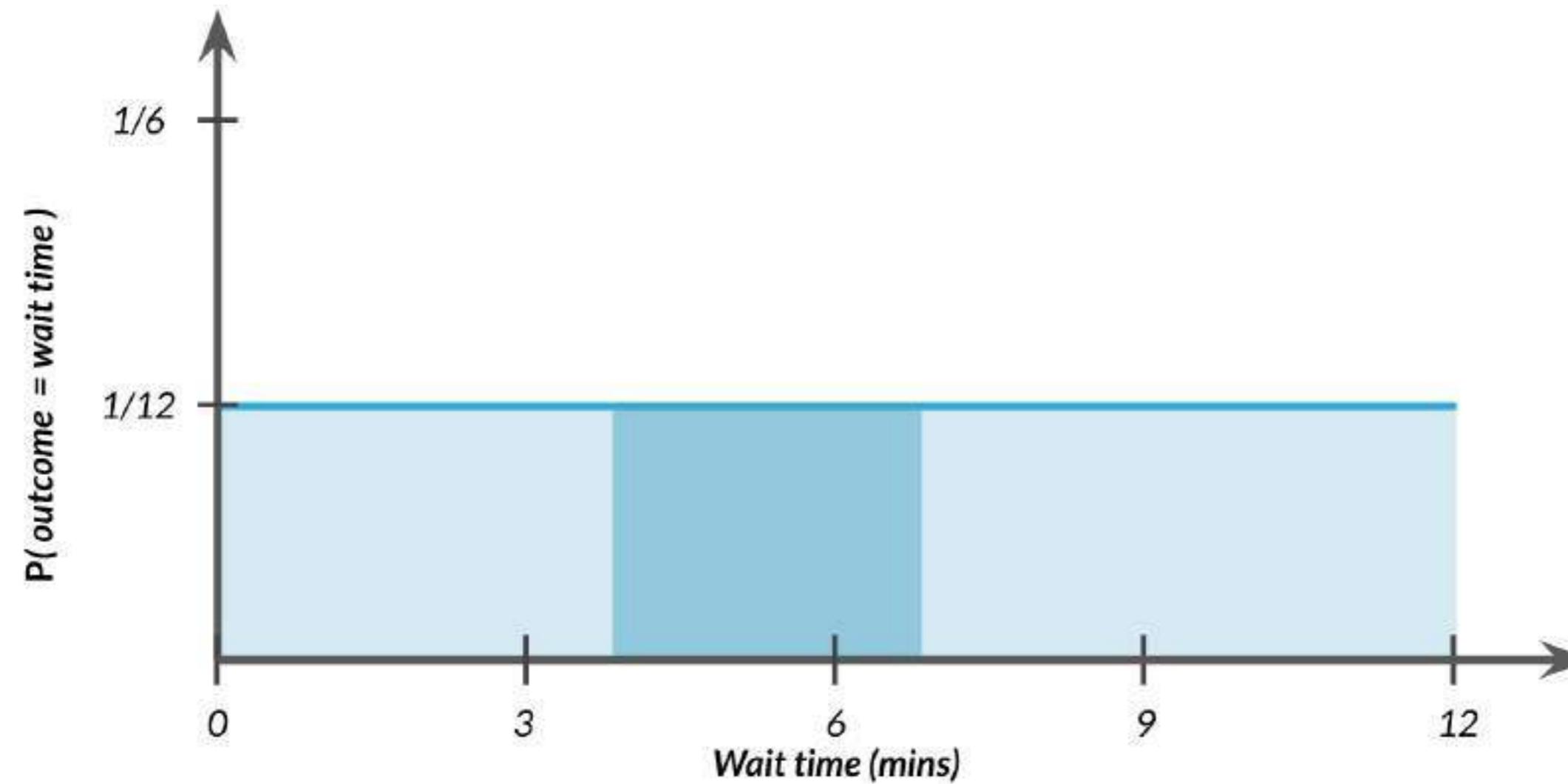


Continuous uniform distribution



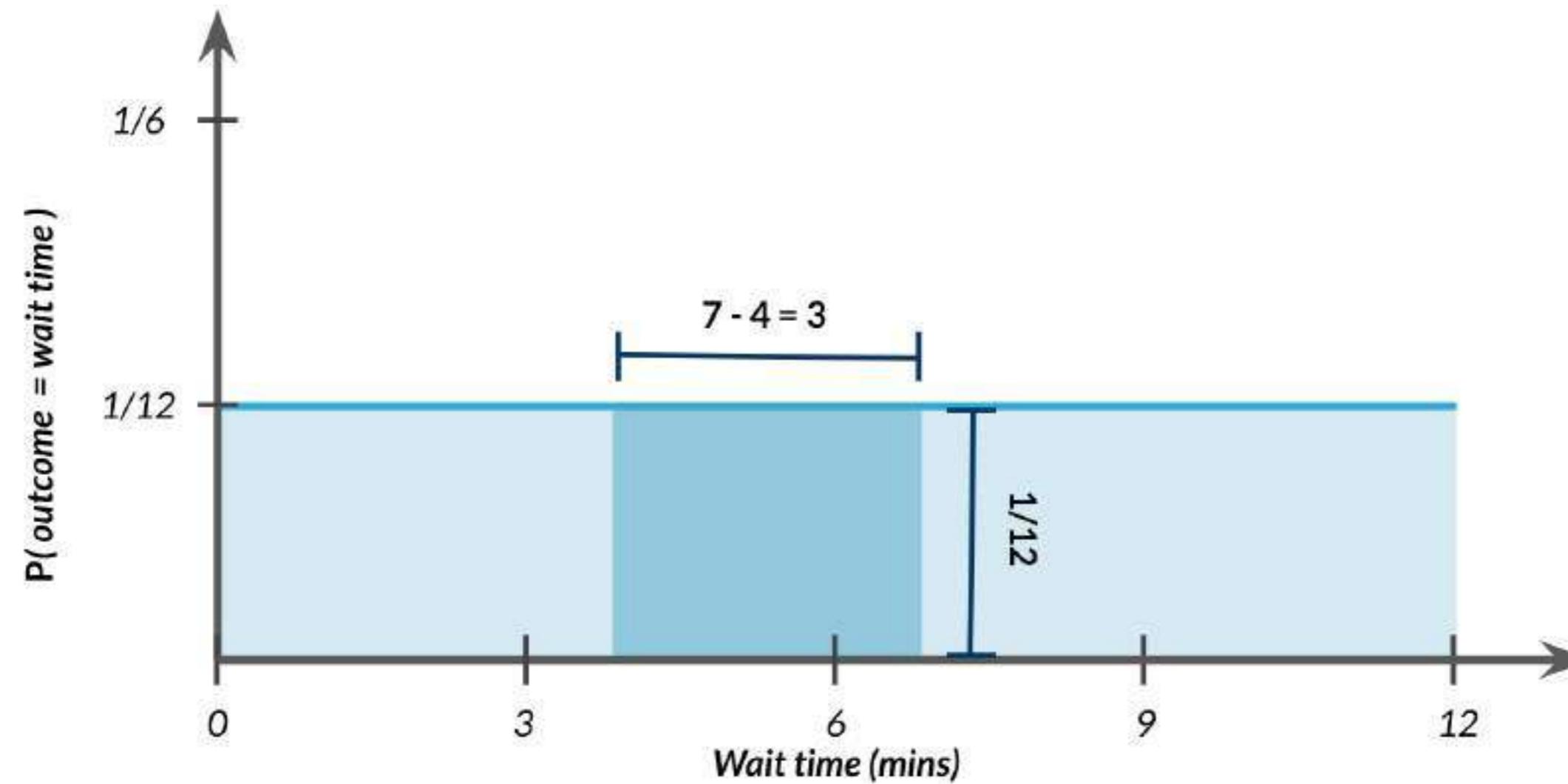
Probability still = area

$$P(4 \leq \text{wait time} \leq 7) = ?$$



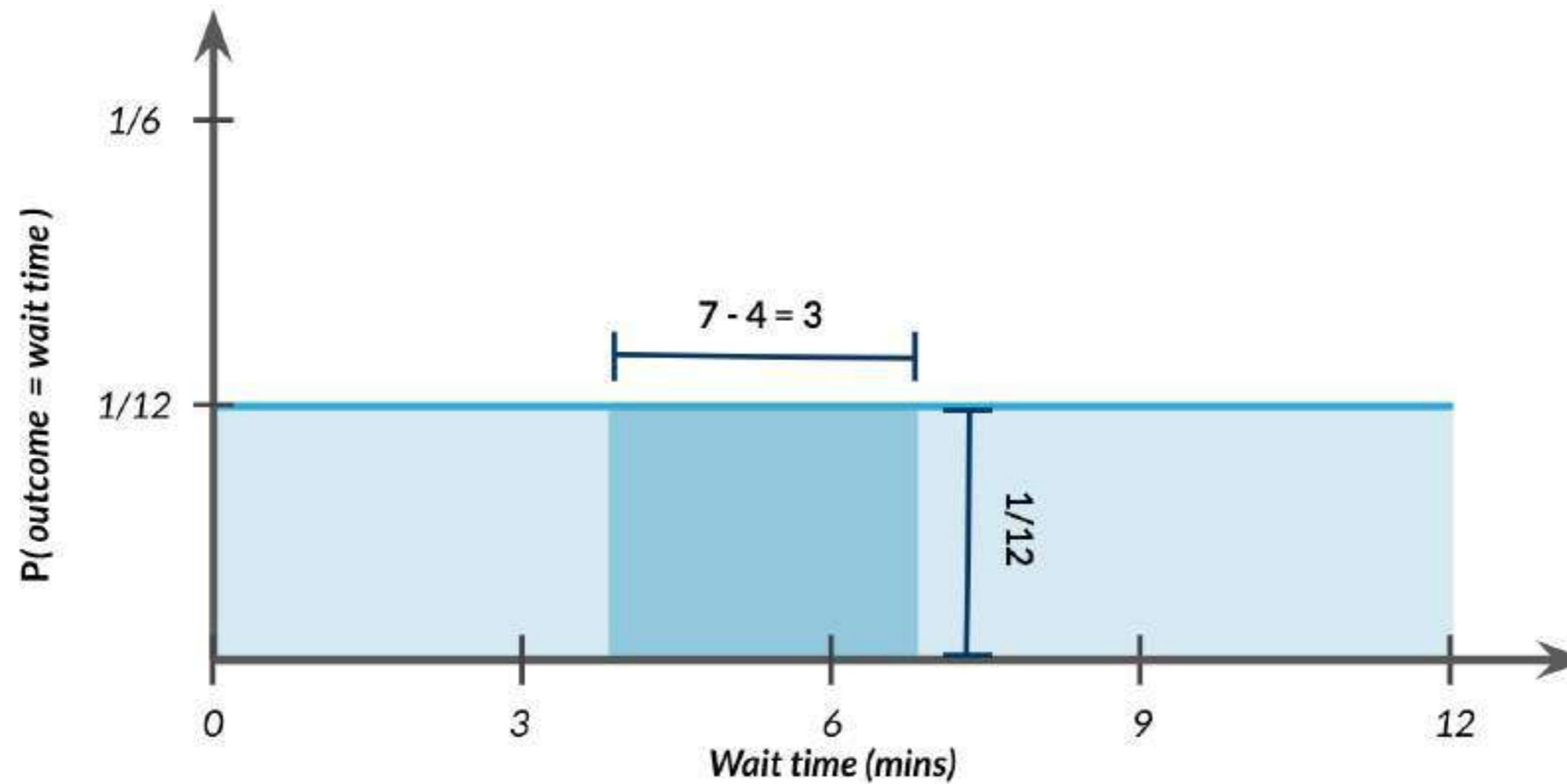
Probability still = area

$$P(4 \leq \text{wait time} \leq 7) = ?$$



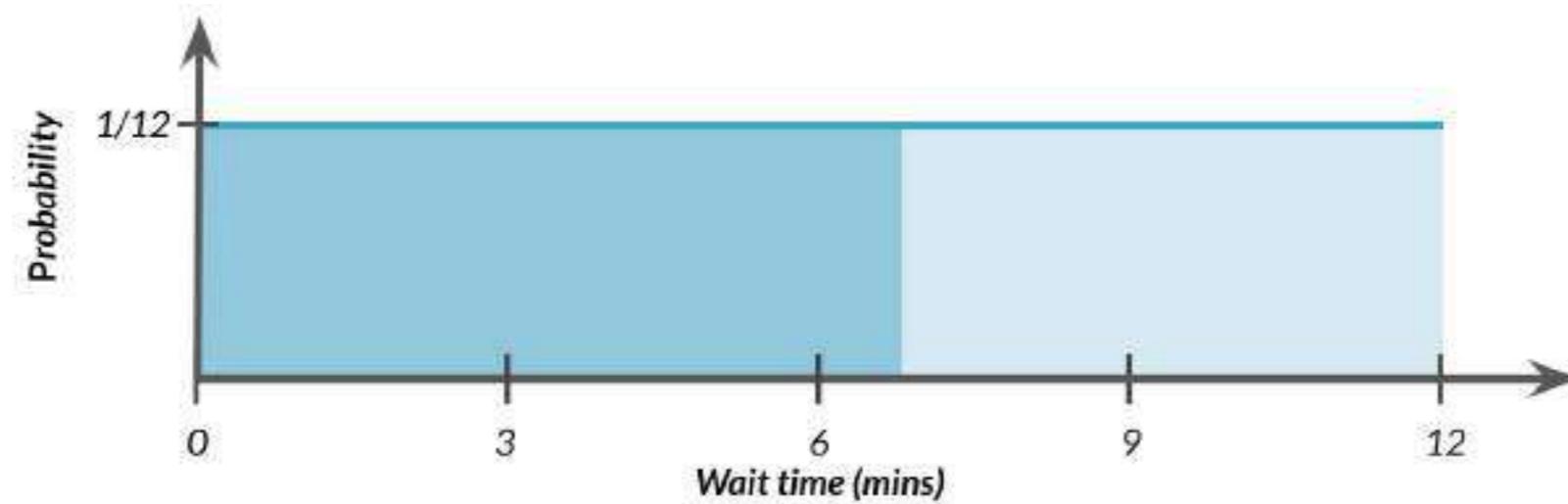
Probability still = area

$$P(4 \leq \text{wait time} \leq 7) = 3 \times 1/12 = 3/12$$



Uniform distribution in Python

$$P(\text{wait time} \leq 7)$$

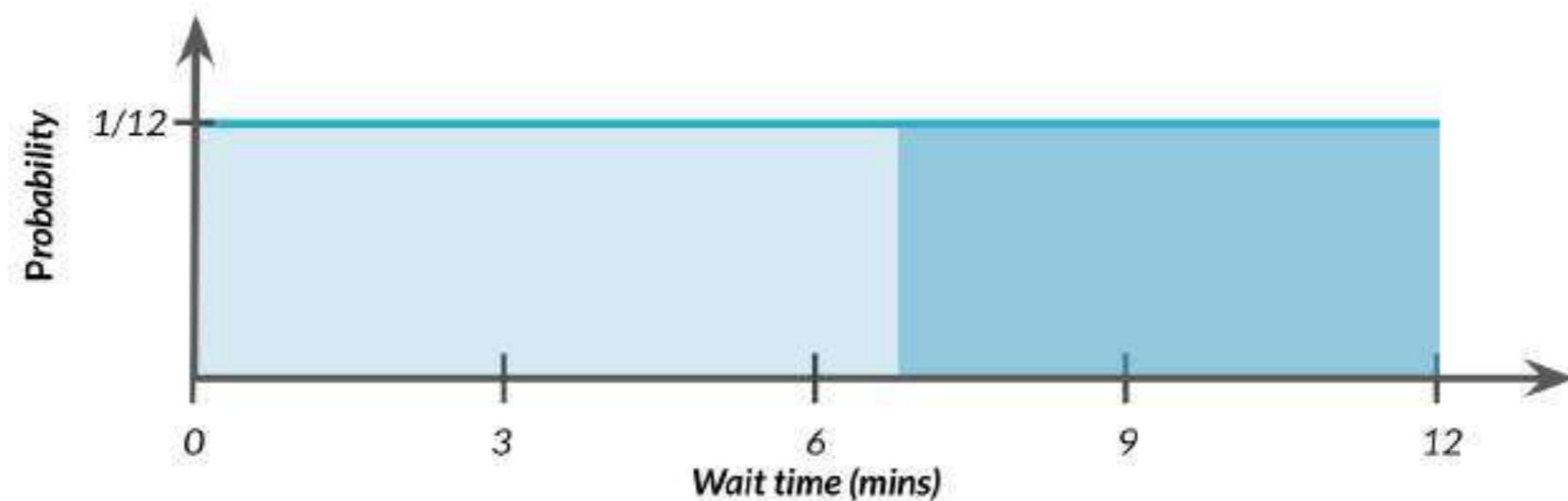


```
from scipy.stats import uniform  
uniform.cdf(7, 0, 12)
```

0.583333

"Greater than" probabilities

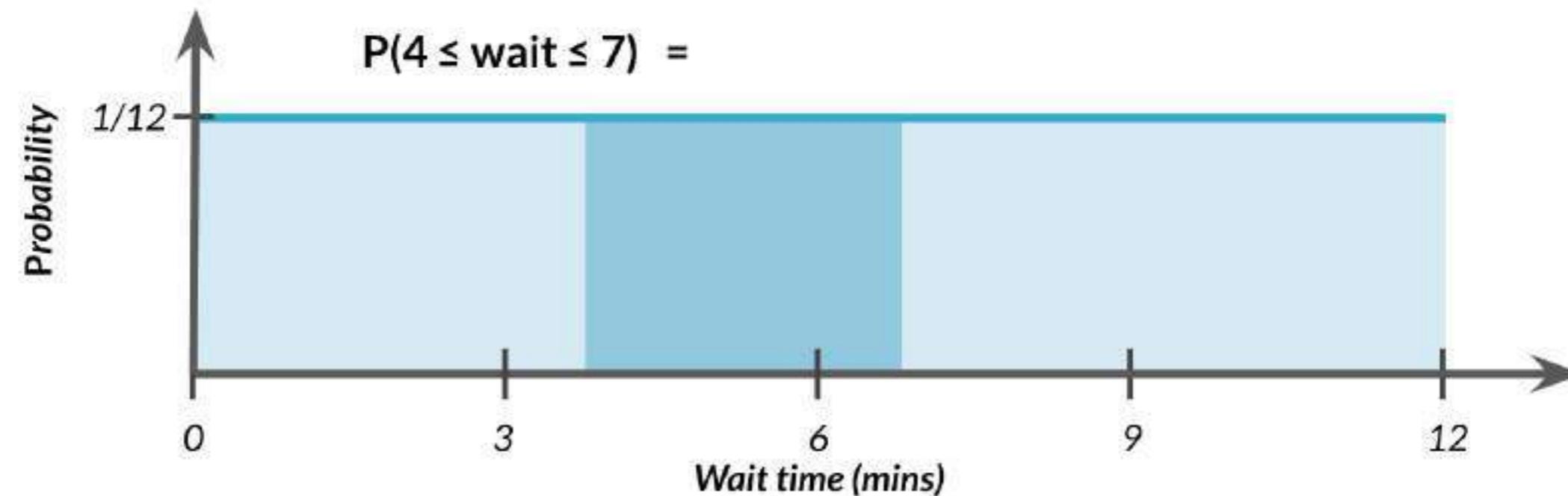
$$P(\text{wait time} \geq 7) = 1 - P(\text{wait time} \leq 7)$$



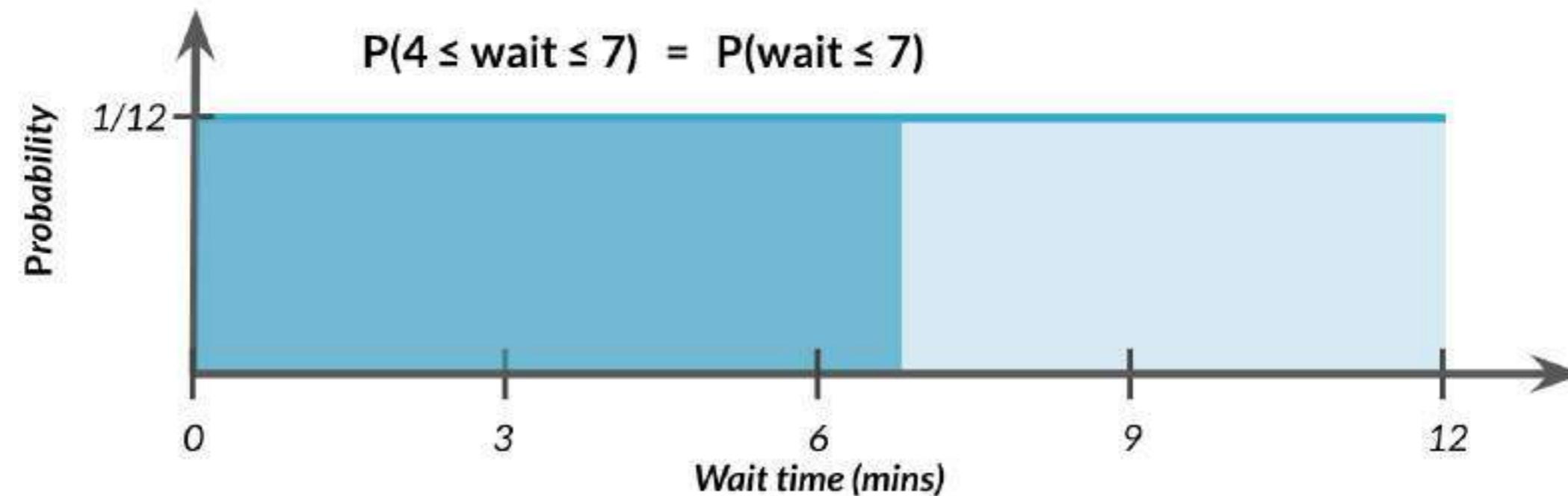
```
from scipy.stats import uniform  
1 - uniform.cdf(7, 0, 12)
```

0.416667

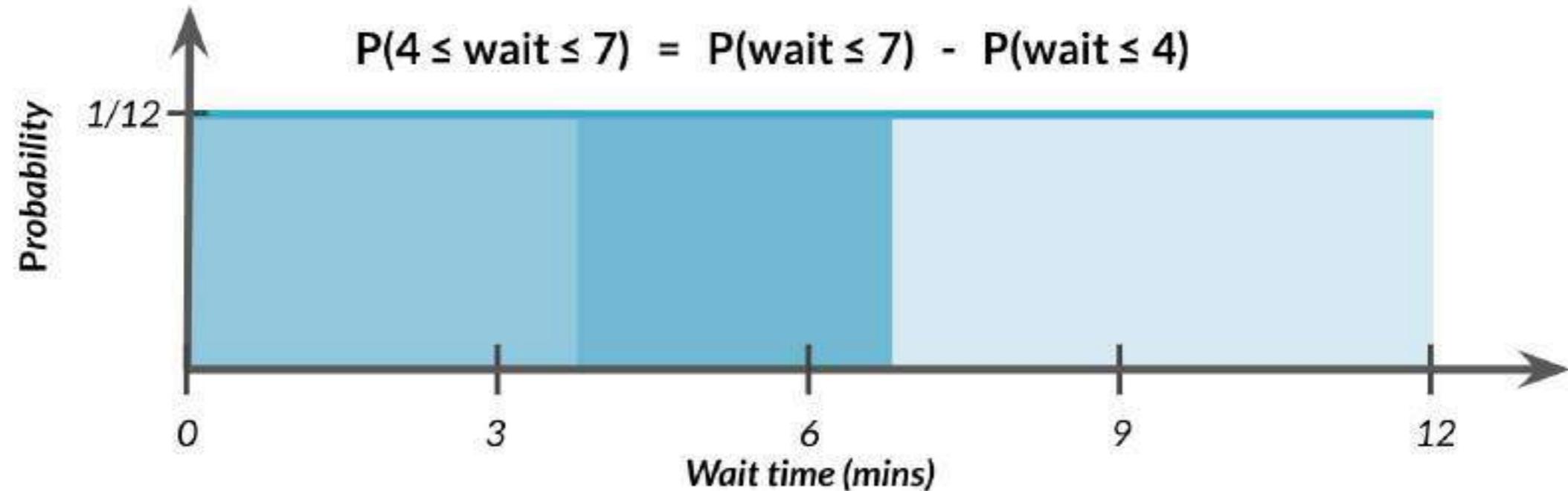
$$P(4 \leq \text{wait time} \leq 7)$$



$$P(4 \leq \text{wait time} \leq 7)$$



$$P(4 \leq \text{wait time} \leq 7)$$

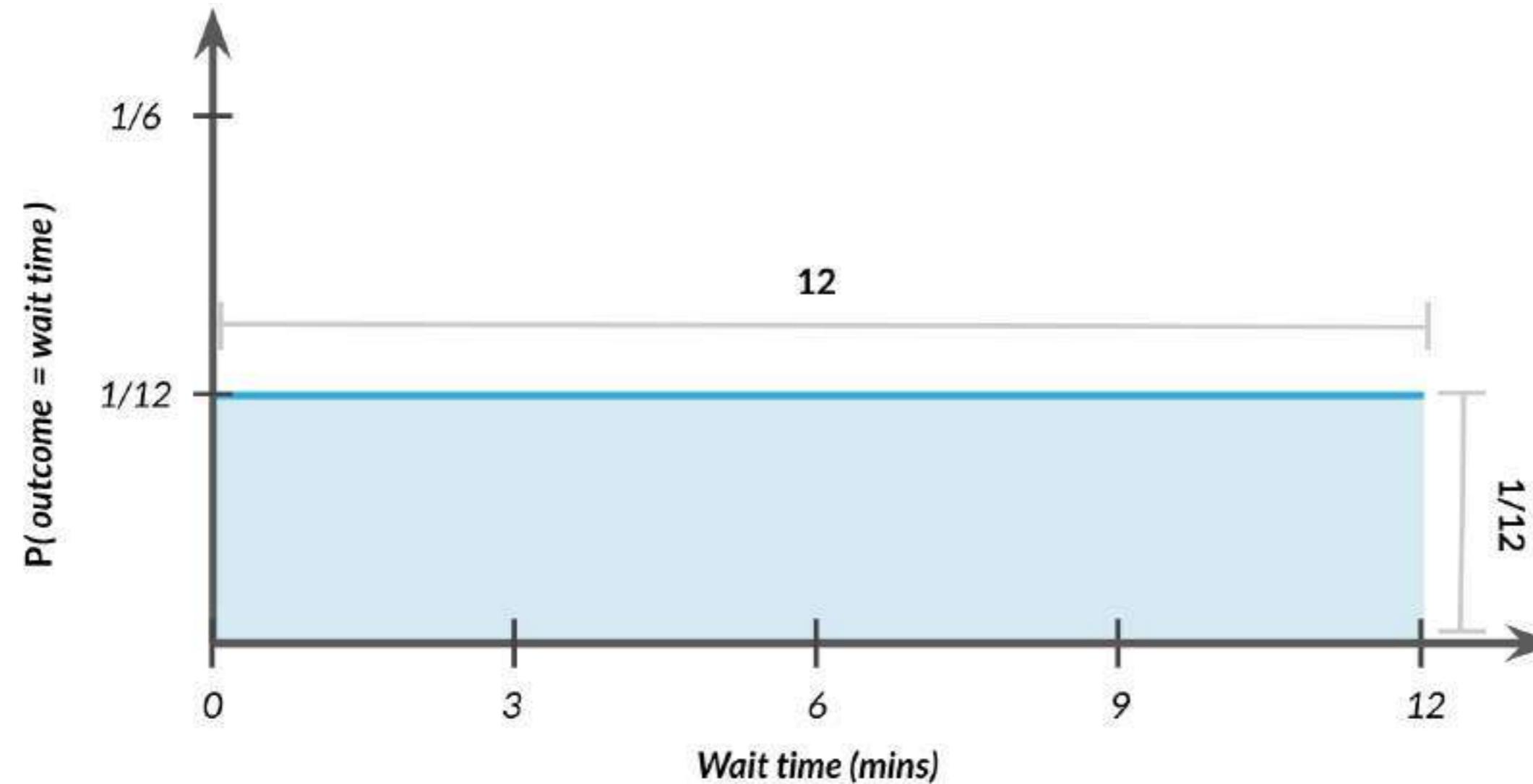


```
from scipy.stats import uniform  
uniform.cdf(7, 0, 12) - uniform.cdf(4, 0, 12)
```

0.25

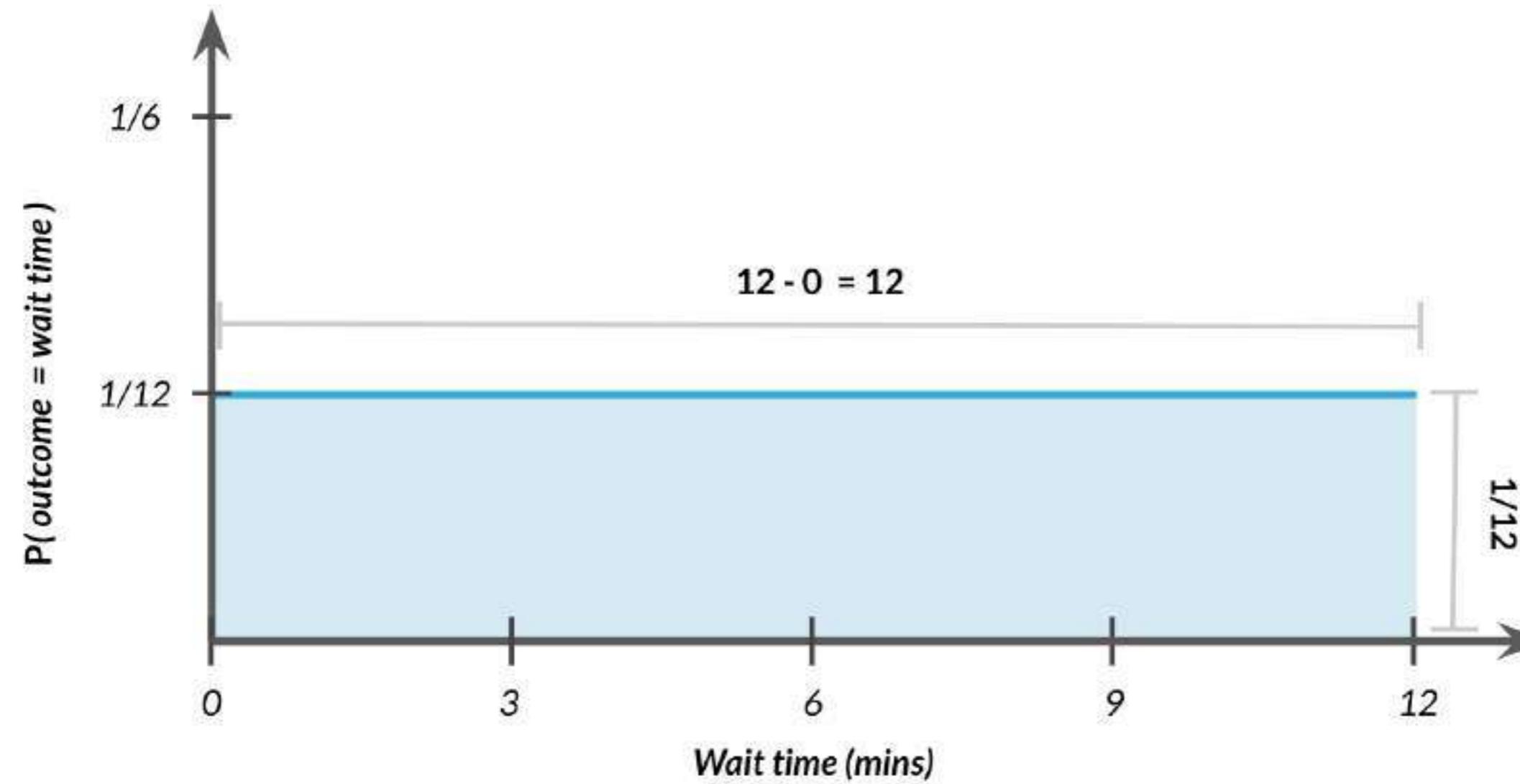
Total area = 1

$$P(0 \leq \text{wait time} \leq 12) = ?$$



Total area = 1

$$P(0 \leq \text{outcome} \leq 12) = 12 \times 1/12 = 1$$

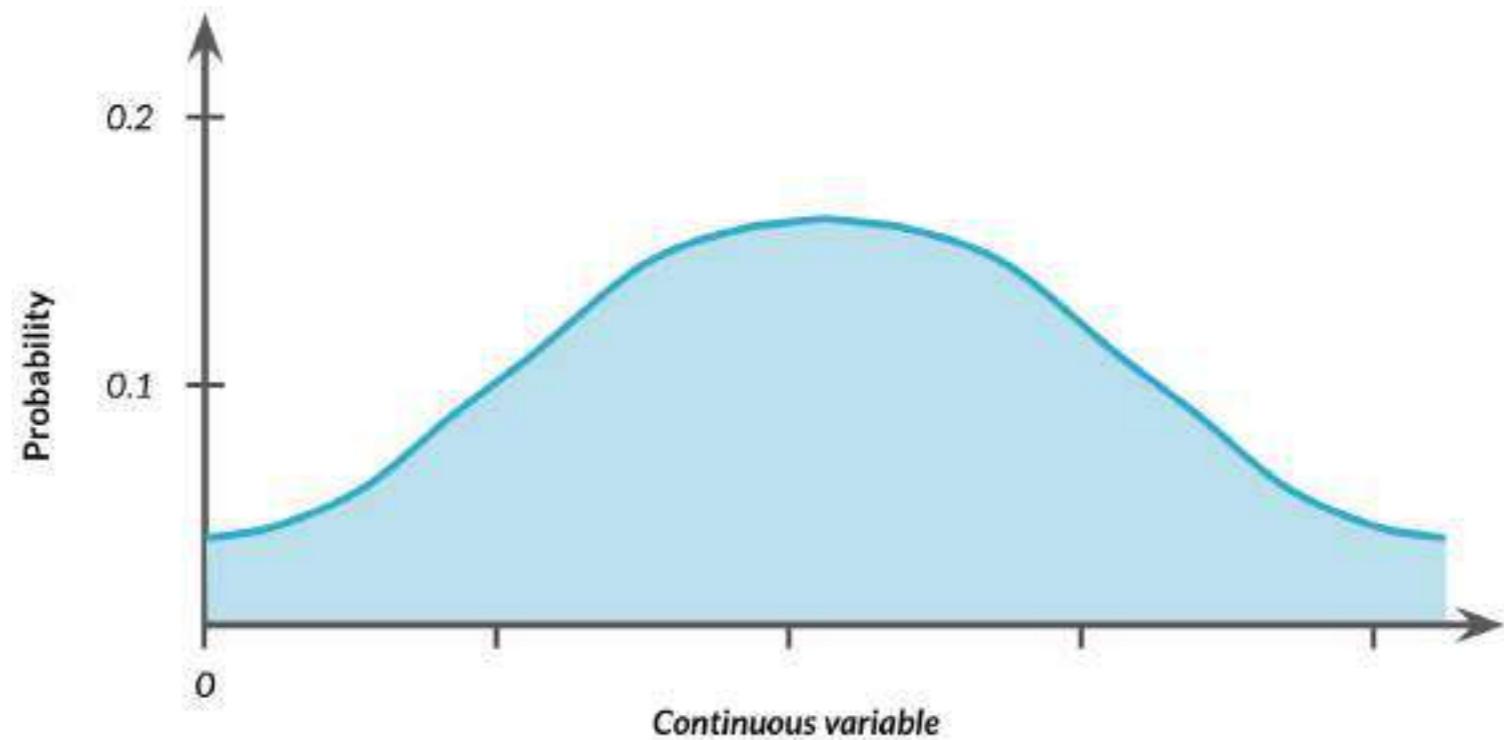
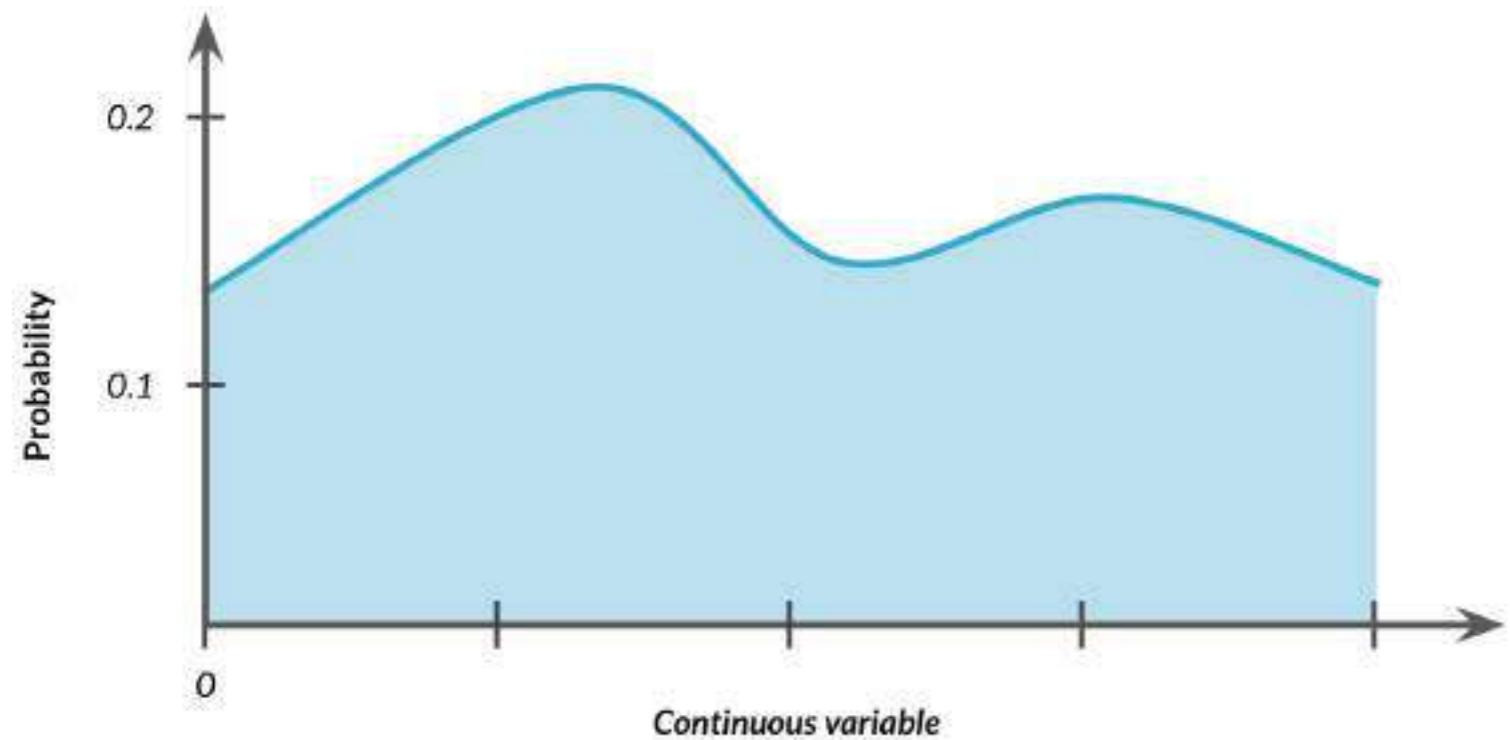


Generating random numbers according to uniform distribution

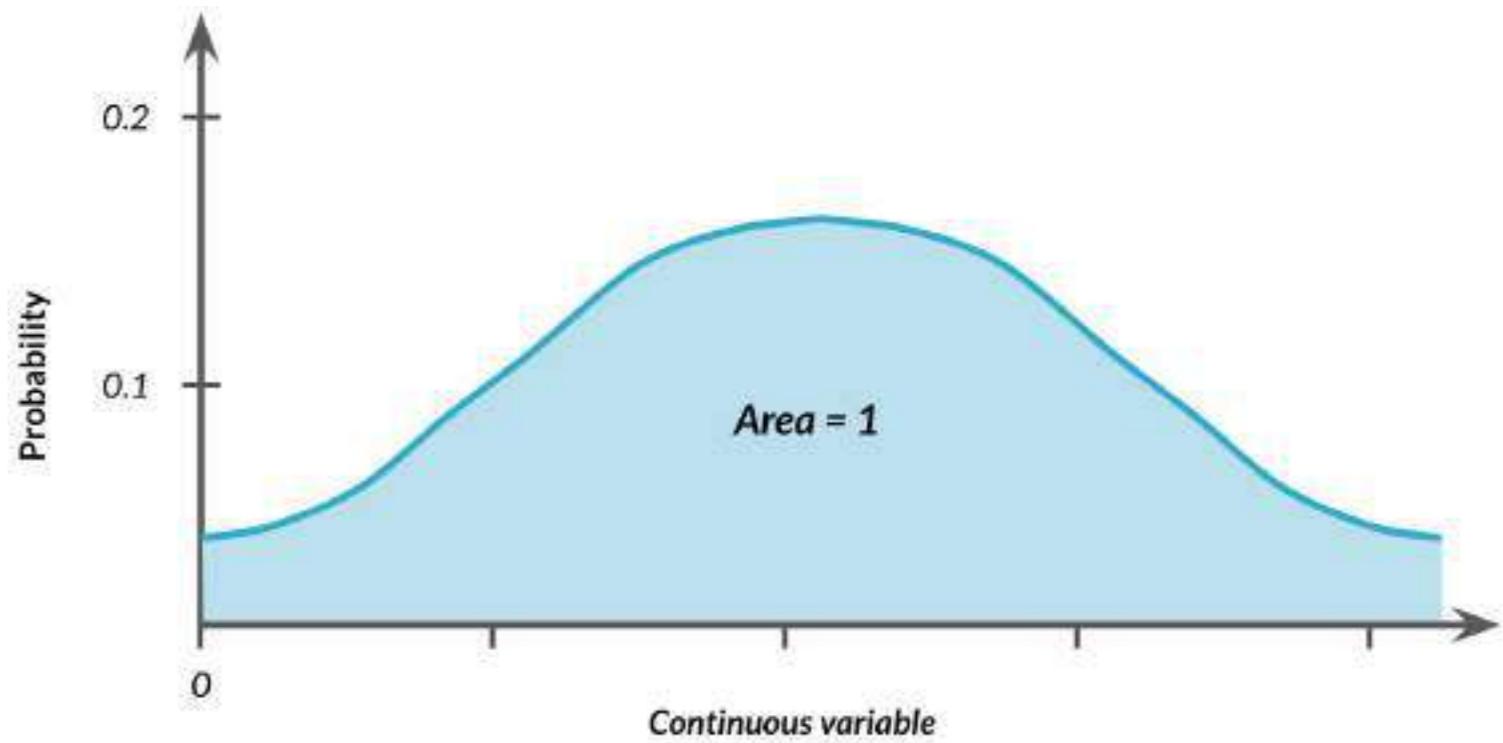
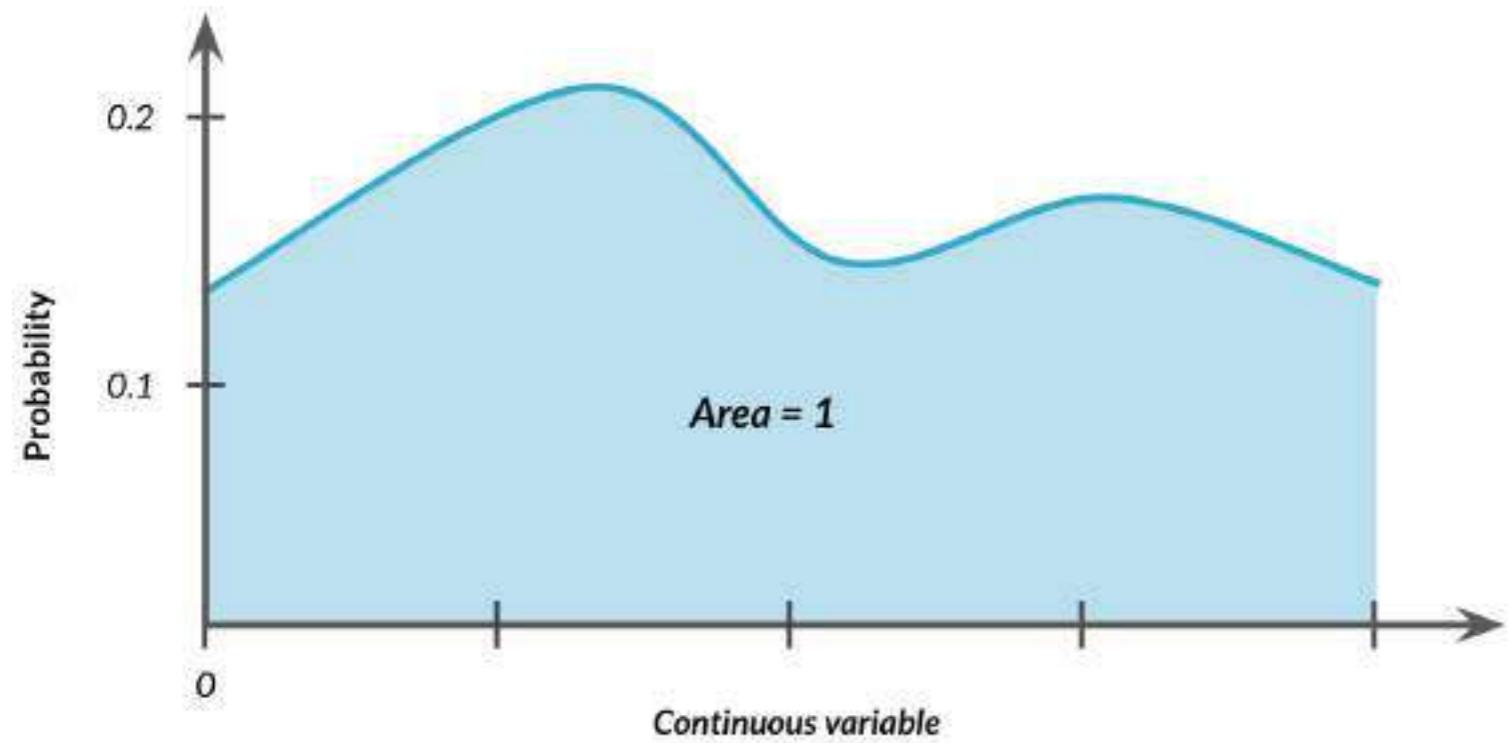
```
from scipy.stats import uniform  
uniform.rvs(0, 5, size=10)
```

```
array([1.89740094, 4.70673196, 0.33224683, 1.0137103 , 2.31641255,  
      3.49969897, 0.29688598, 0.92057234, 4.71086658, 1.56815855])
```

Other continuous distributions

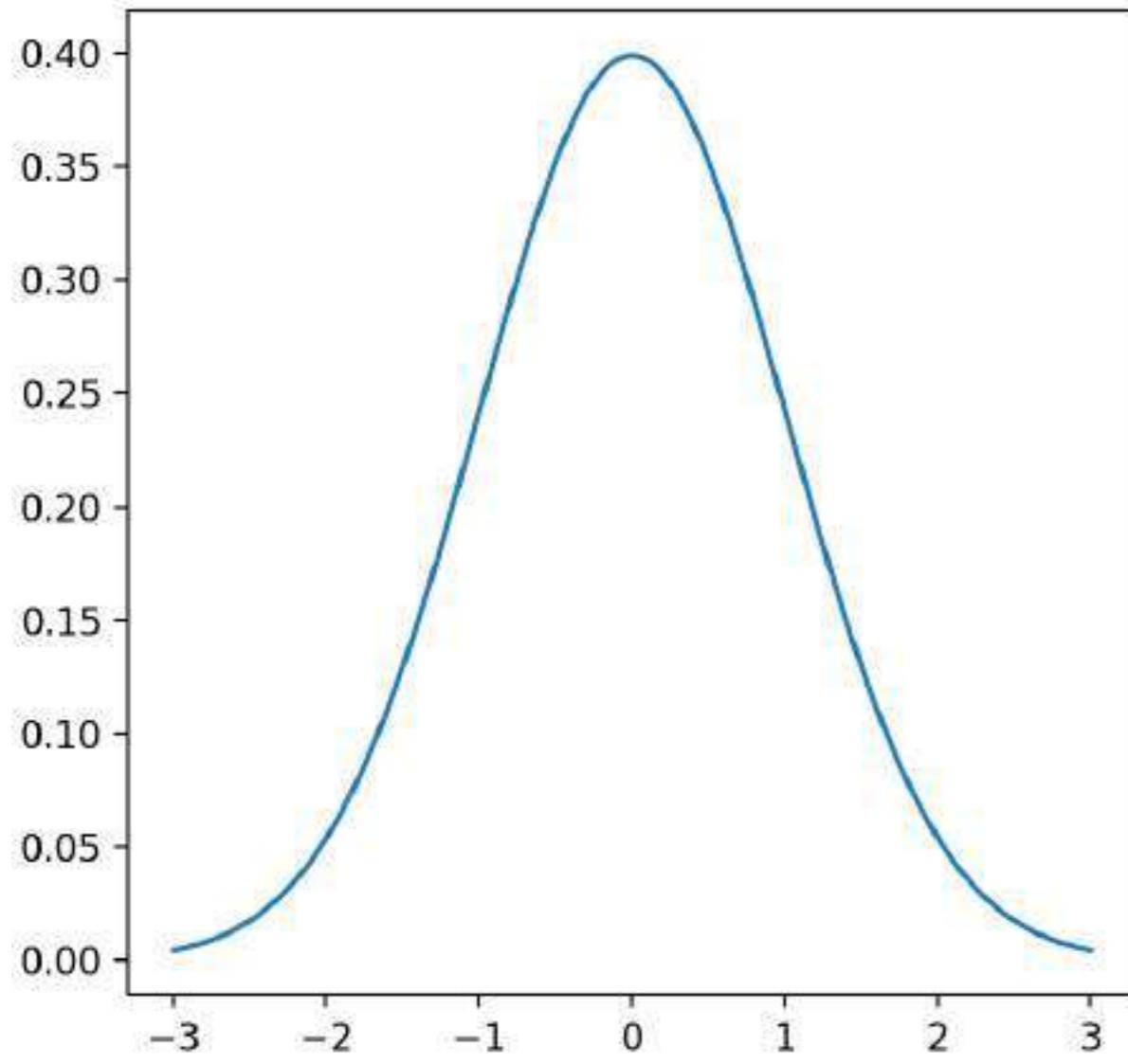


Other continuous distributions

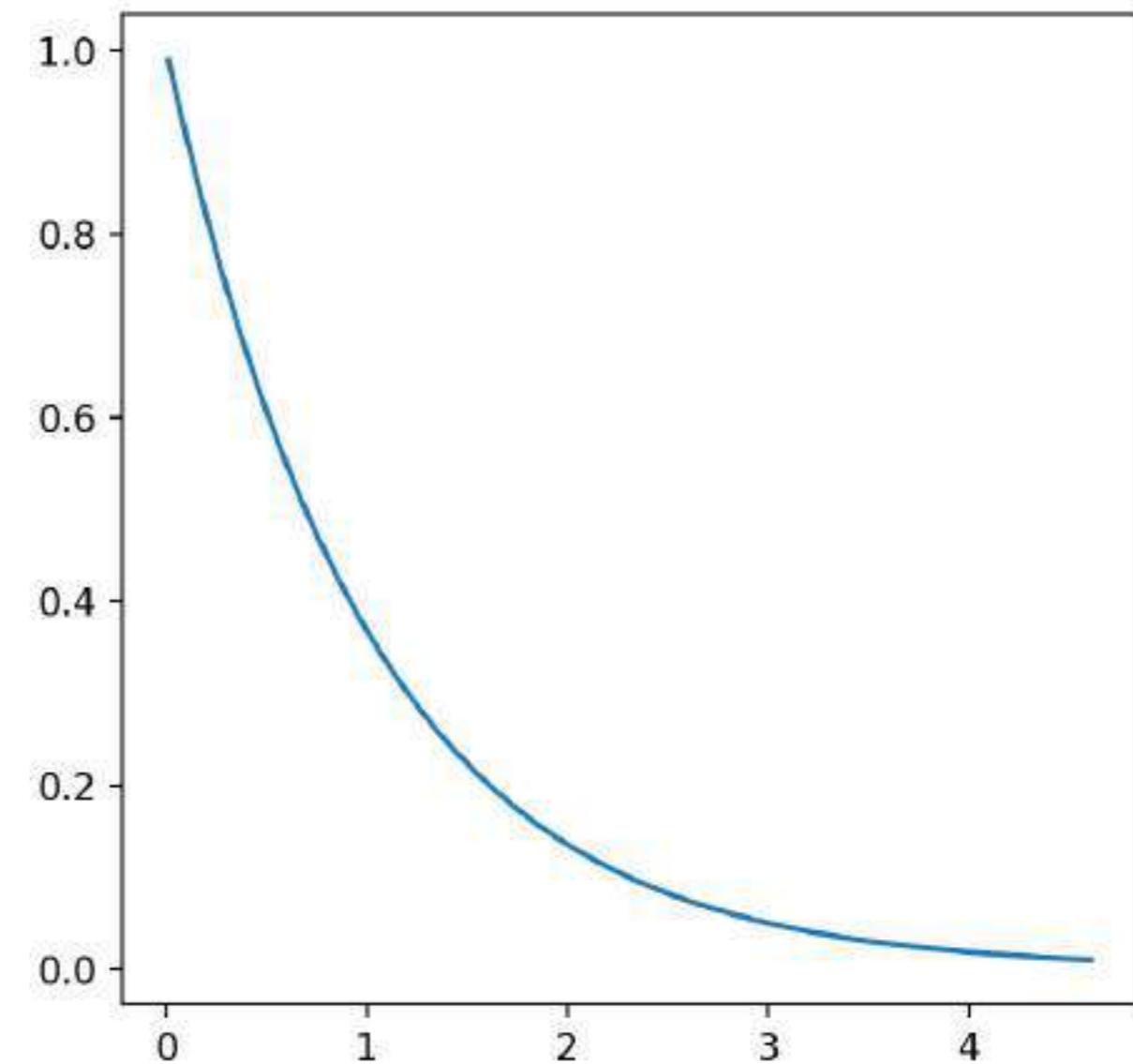


Other special types of distributions

Normal distribution



Exponential distribution

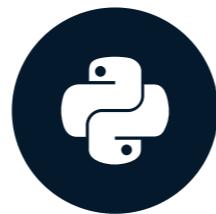


Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

The binomial distribution

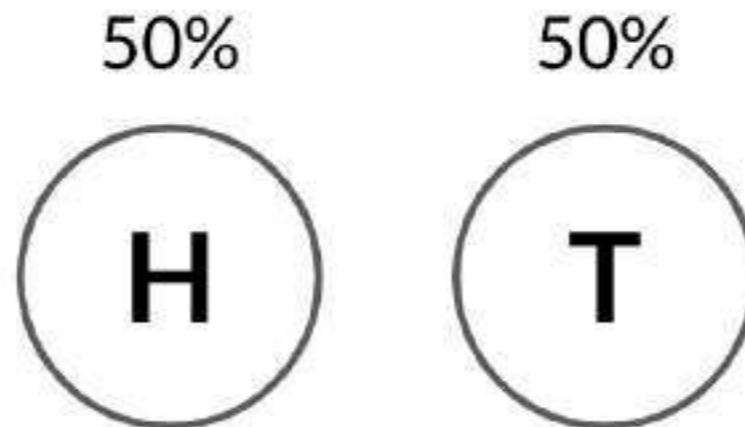
INTRODUCTION TO STATISTICS IN PYTHON



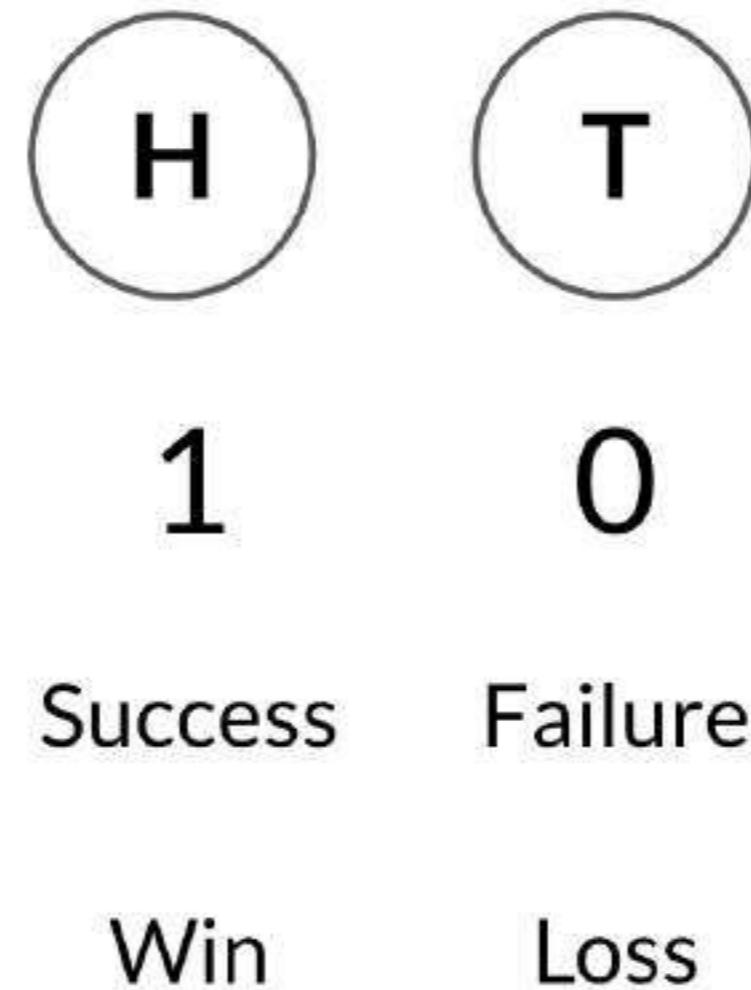
Maggie Matsui

Content Developer, DataCamp

Coin flipping



Binary outcomes



A single flip

```
binom.rvs(# of coins, probability of heads/success, size=# of trials)
```

1 = head, 0 = tails

```
from scipy.stats import binom  
binom.rvs(1, 0.5, size=1)
```

```
array([1])
```

One flip many times

```
binom.rvs(1, 0.5, size=8)
```

```
array([0, 1, 1, 0, 1, 0, 1, 1])
```

binom.rvs(1, 0.5, size = 8)

Flip 1 coin with 50% chance of success 8 times

Many flips one time

```
binom.rvs(8, 0.5, size=1)
```

```
array([5])
```

binom.rvs(8, 0.5, size = 1)

Flip 8 coins with 50% chance of success 1 time

Many flips many times

```
binom.rvs(3, 0.5, size=10)
```

```
array([0, 3, 2, 1, 3, 0, 2, 2, 0, 0])
```

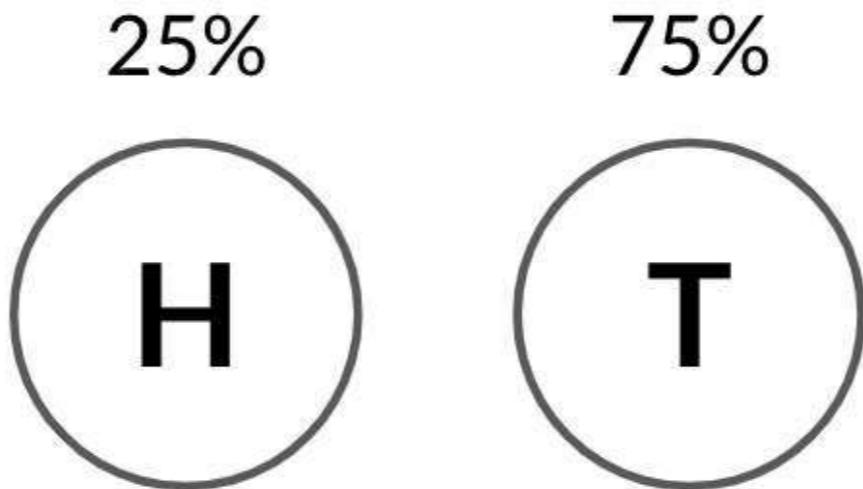
```
binom.rvs(3, 0.5, size = 10)
```

Flip 3 coins with 50% chance of success 10 times

Other probabilities

```
binom.rvs(3, 0.25, size=10)
```

```
array([1, 1, 1, 1, 0, 0, 2, 0, 1, 0])
```



Binomial distribution

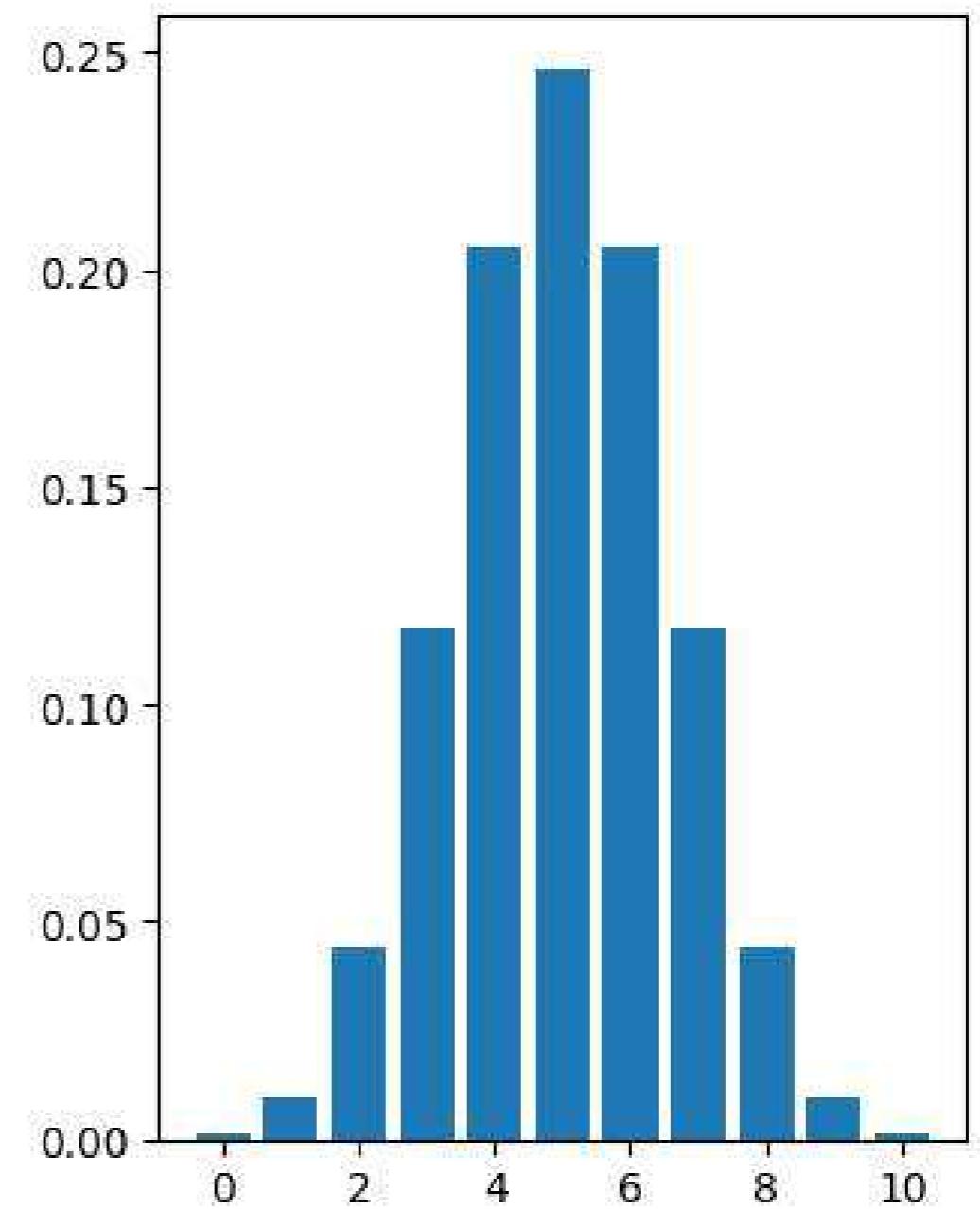
Probability distribution of the number of successes in a sequence of independent trials

E.g. Number of heads in a sequence of coin flips

Described by n and p

- n : total number of trials
- p : probability of success

```
          p           n  
binom.rvs(3, 0.5, size = 10)
```



What's the probability of 7 heads?

$$P(\text{heads} = 7)$$

```
# binom.pmf(num heads, num trials, prob of heads)
binom.pmf(7, 10, 0.5)
```

0.1171875

What's the probability of 7 or fewer heads?

$$P(\text{heads} \leq 7)$$

```
binom.cdf(7, 10, 0.5)
```

```
0.9453125
```

What's the probability of more than 7 heads?

$P(\text{heads} > 7)$

```
1 - binom.cdf(7, 10, 0.5)
```

```
0.0546875
```

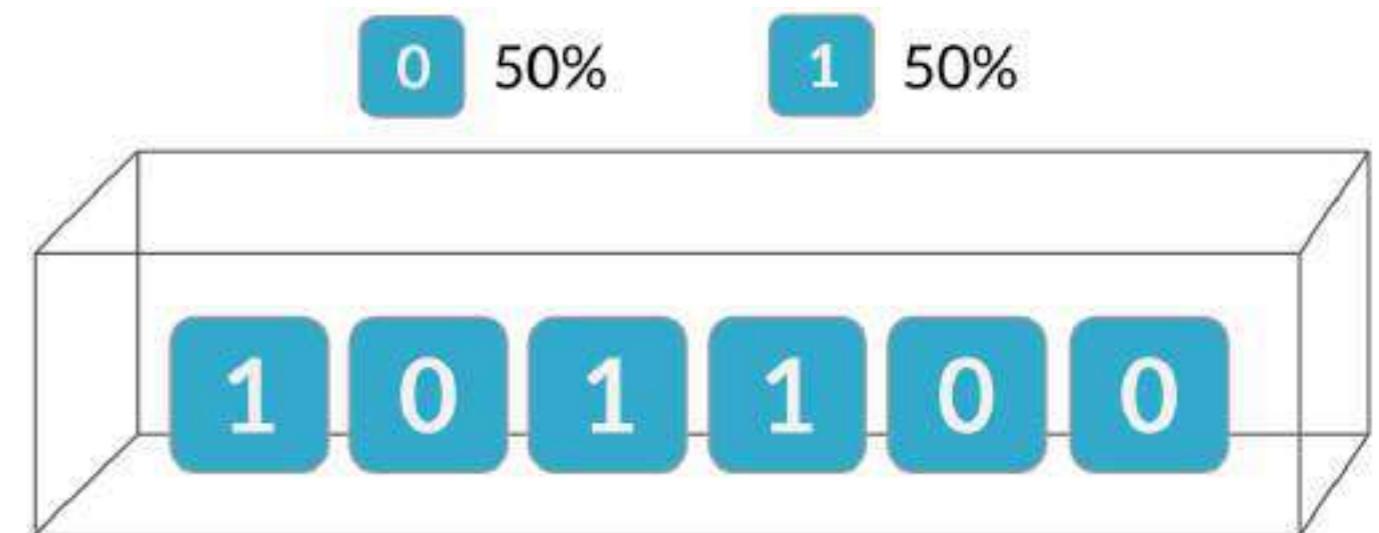
Expected value

Expected value = $n \times p$

Expected number of heads out of 10 flips = $10 \times 0.5 = 5$

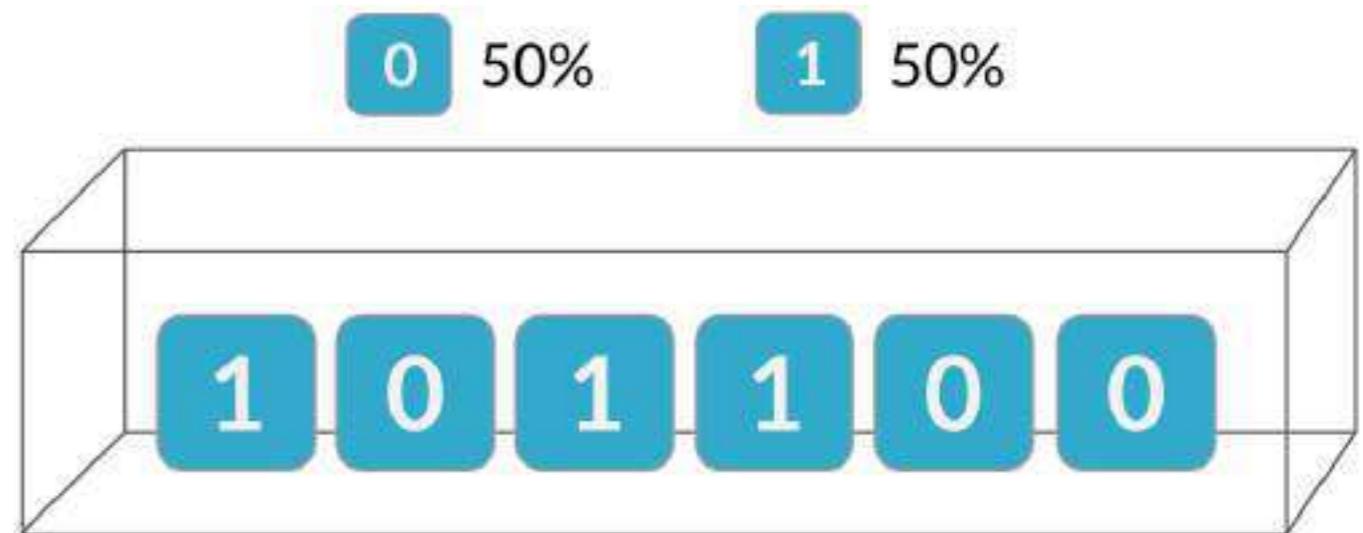
Independence

*The binomial distribution is a probability distribution of the number of successes in a sequence of **independent** trials*

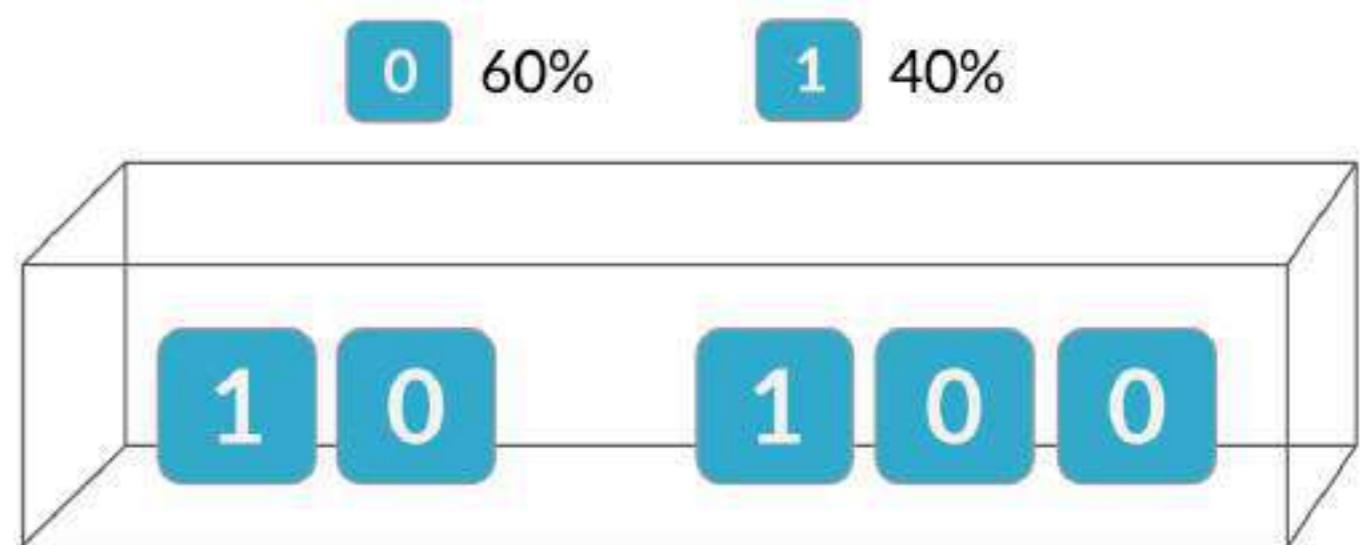


Independence

*The binomial distribution is a probability distribution of the number of successes in a sequence of **independent** trials*



Probabilities of second trial are altered due to outcome of the first



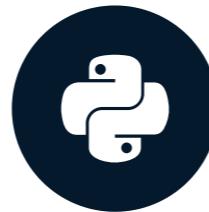
If trials are not independent, the binomial distribution does not apply!

Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

The normal distribution

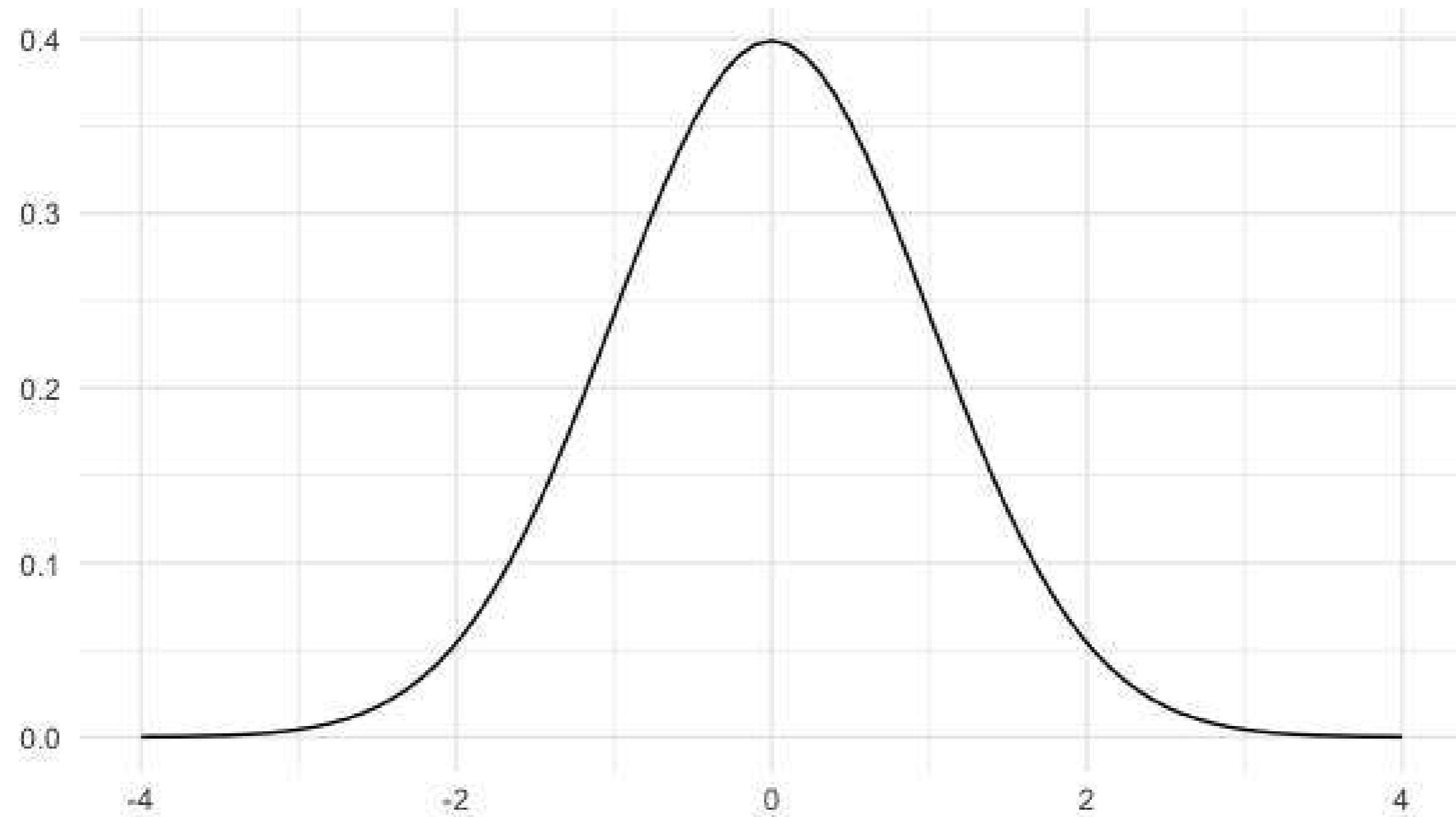
INTRODUCTION TO STATISTICS IN PYTHON



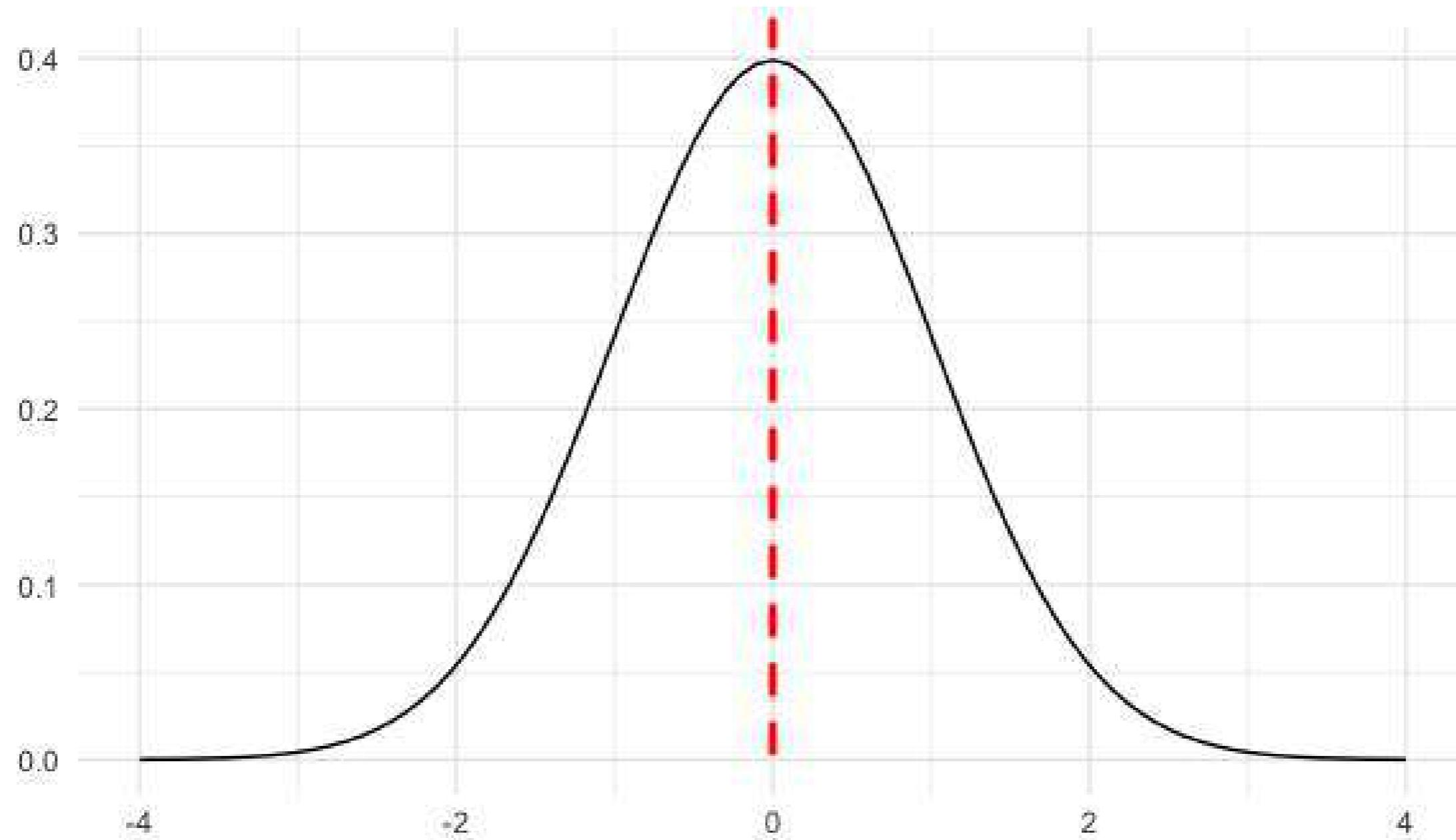
Maggie Matsui

Content Developer, DataCamp

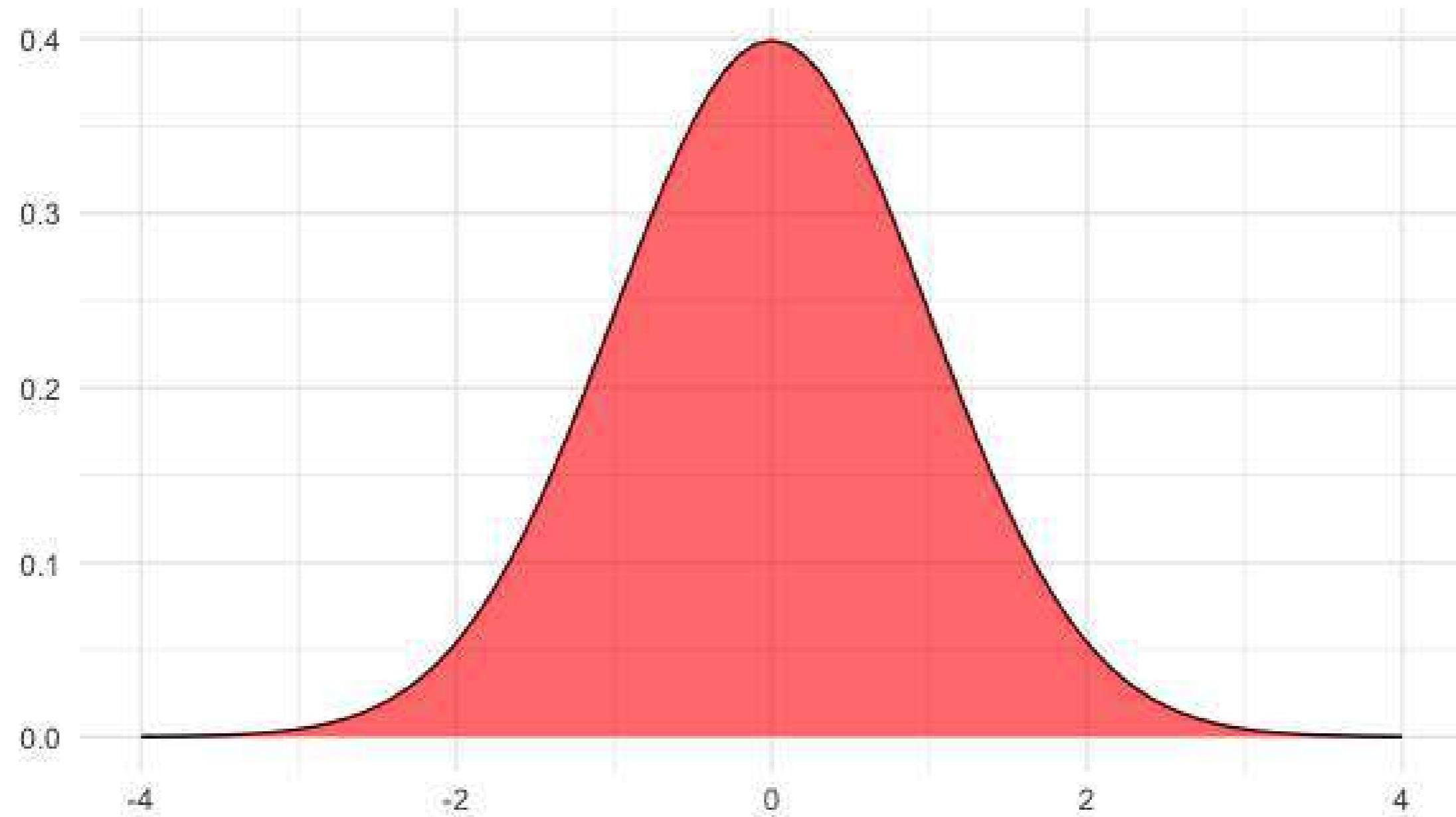
What is the normal distribution?



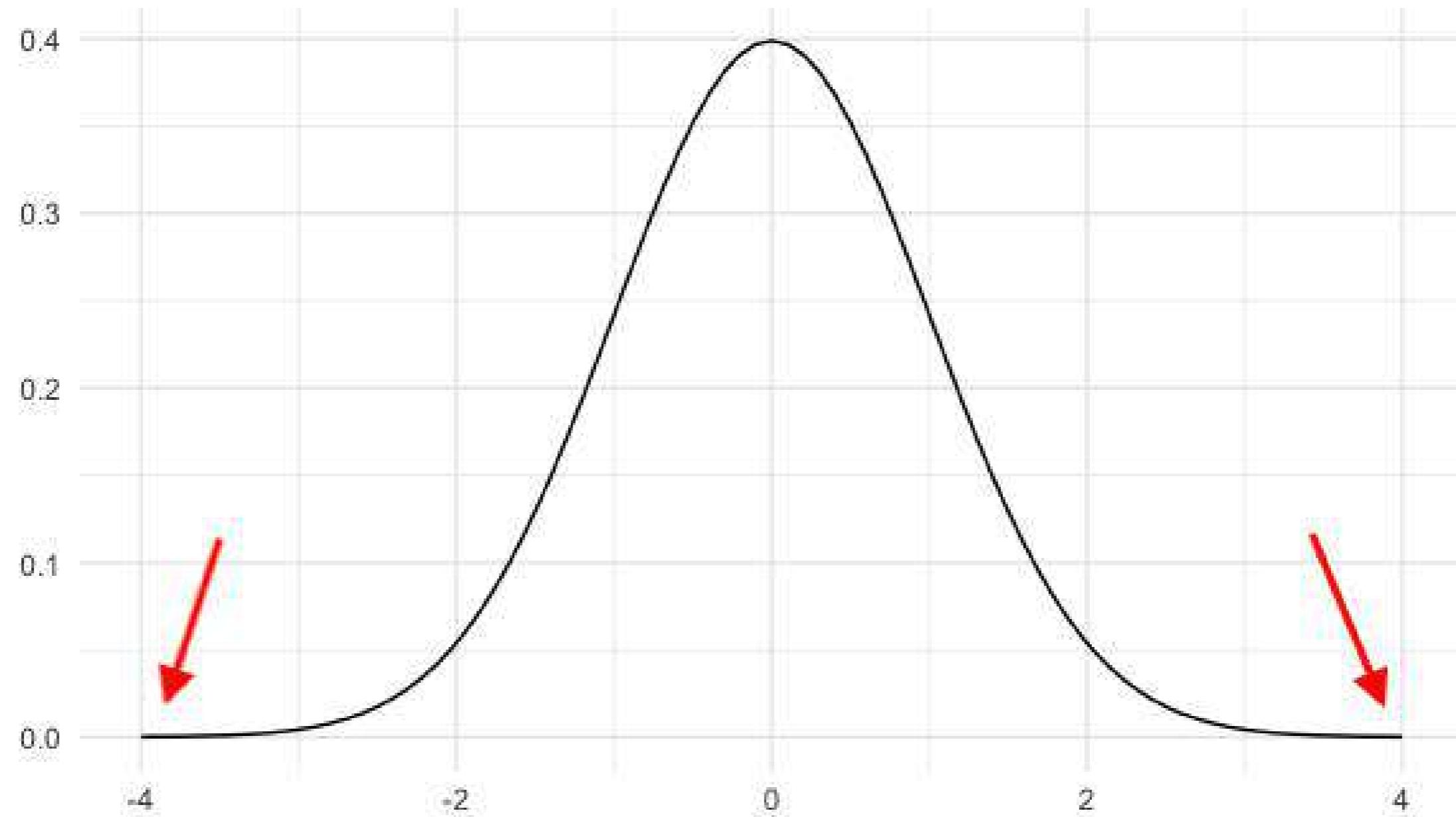
Symmetrical



Area = 1



Curve never hits 0



Described by mean and standard deviation

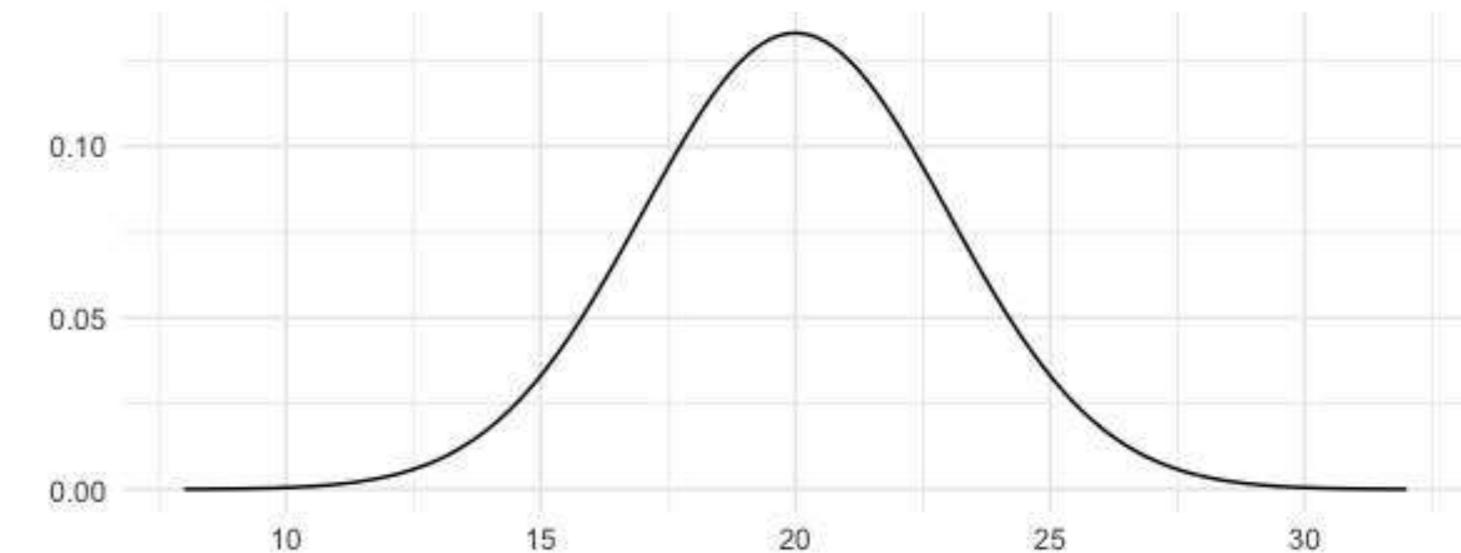
3

distribution

1

Mean: 20

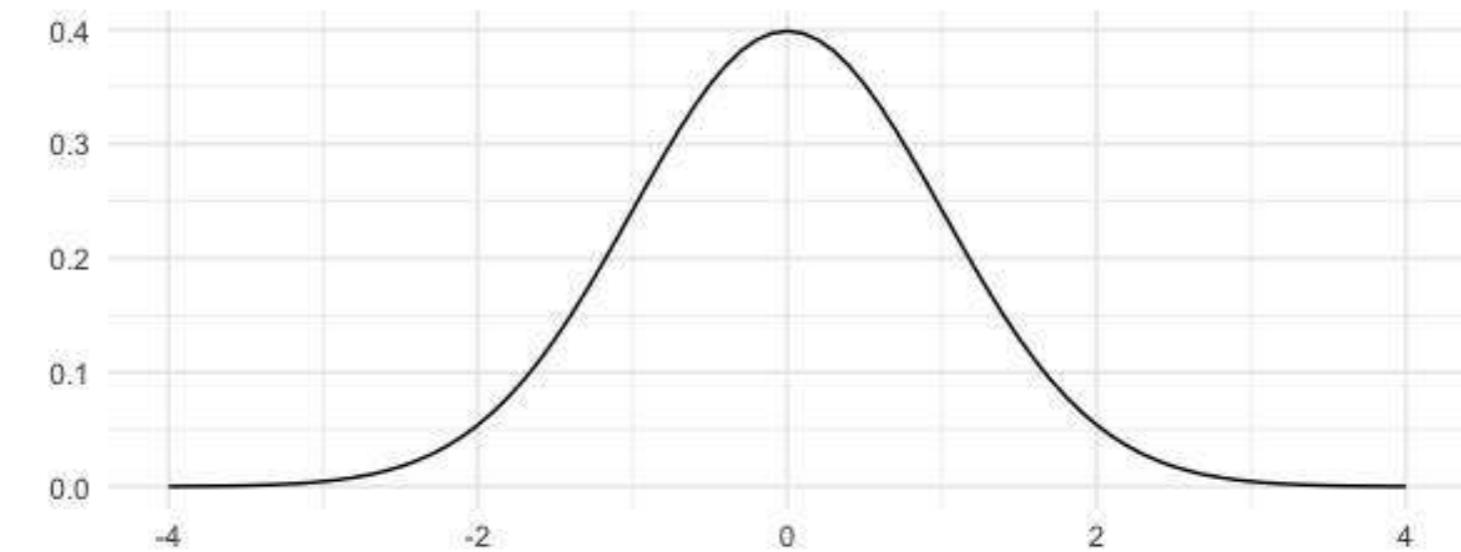
Standard deviation:



Standard normal

Mean: 0

Standard deviation:



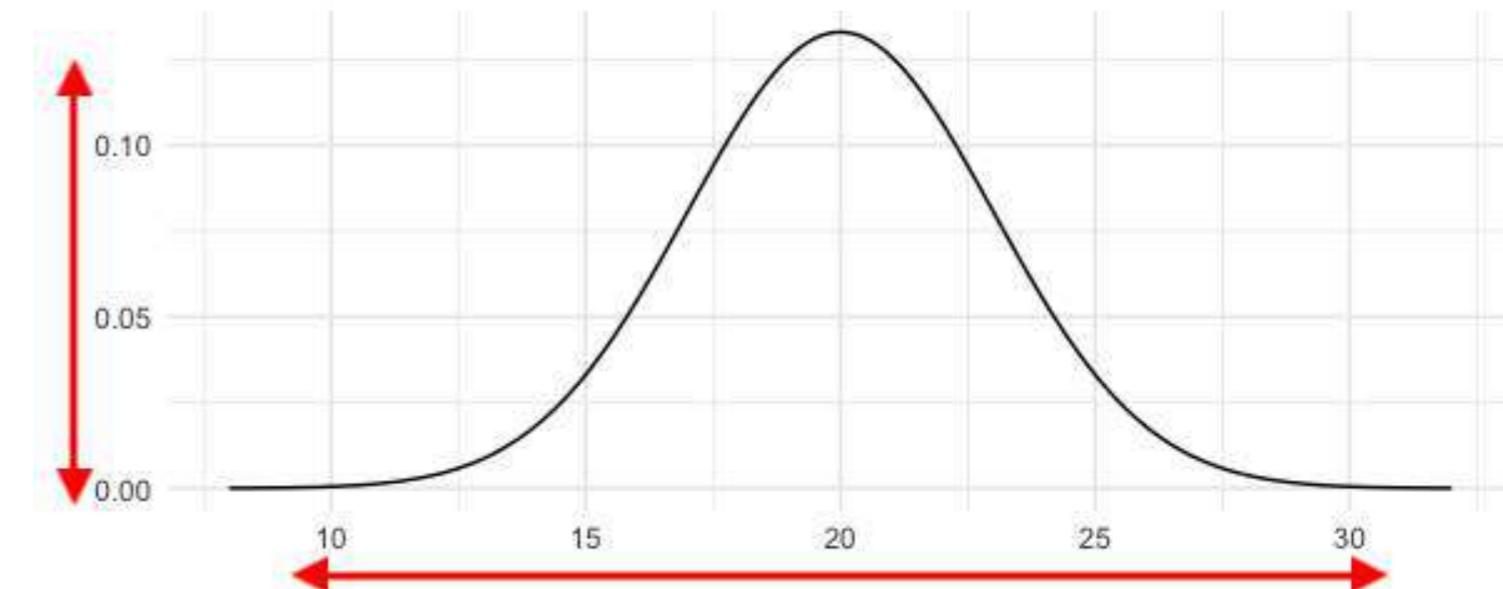
Described by mean and standard deviation

3

distribution

Mean: 20

Standard deviation:

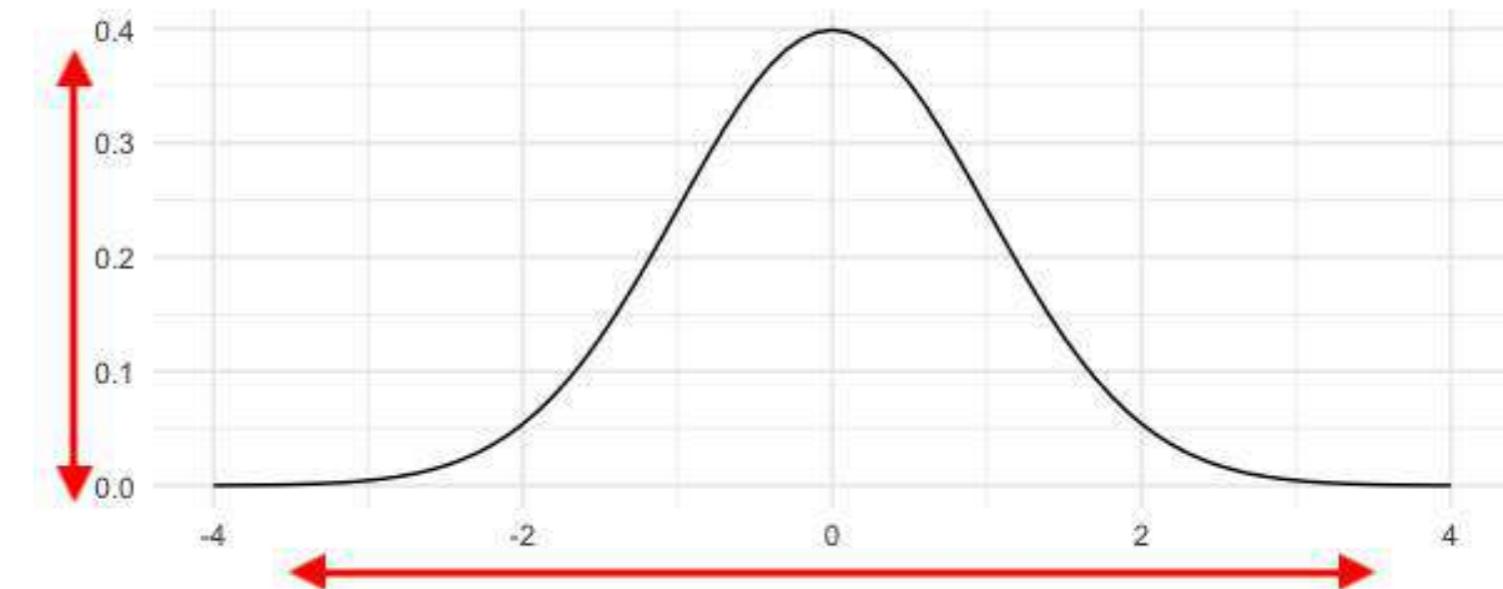


1

Standard normal

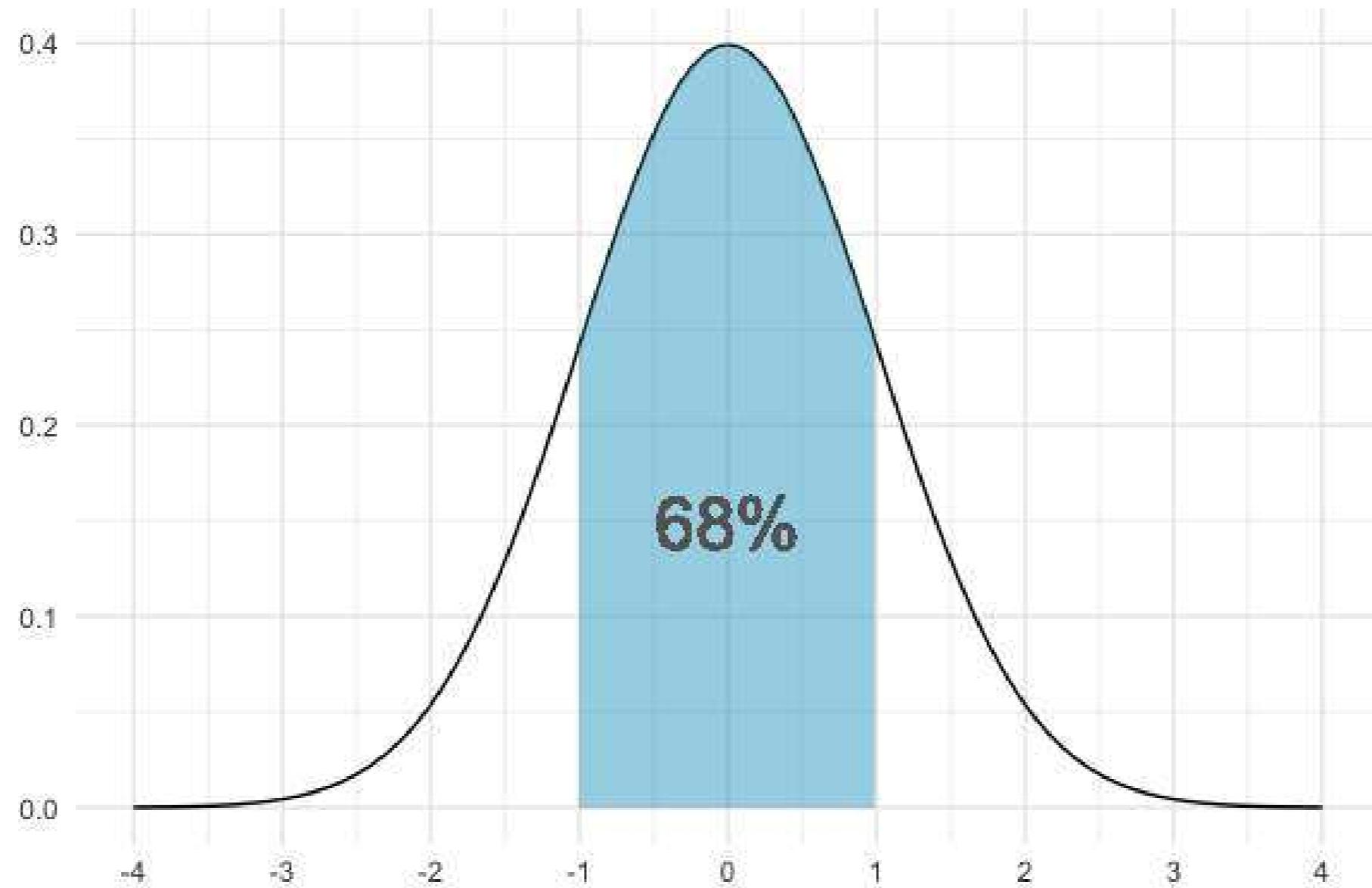
Mean: 0

Standard deviation:



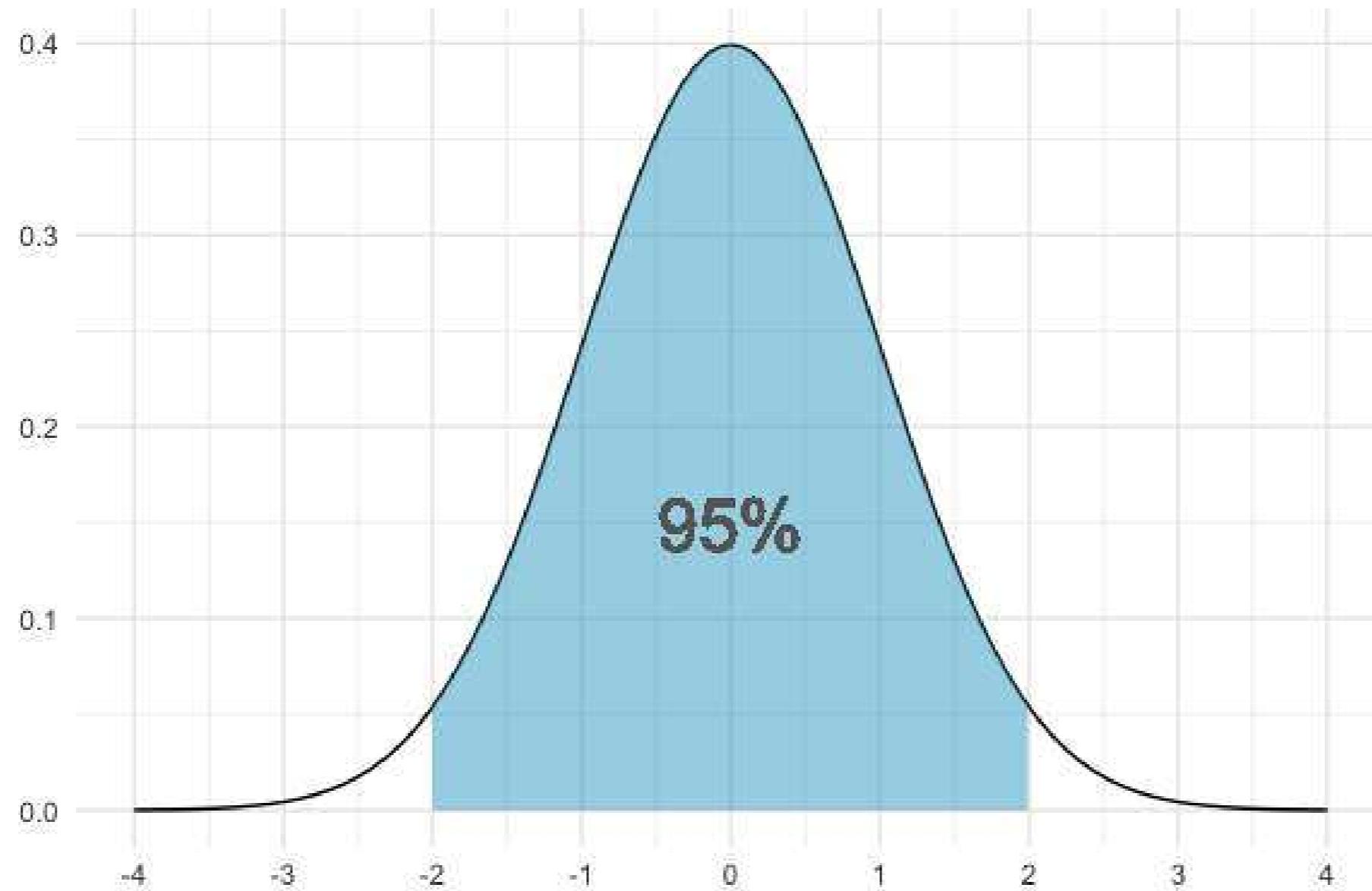
Areas under the normal distribution

68% falls within 1 standard deviation



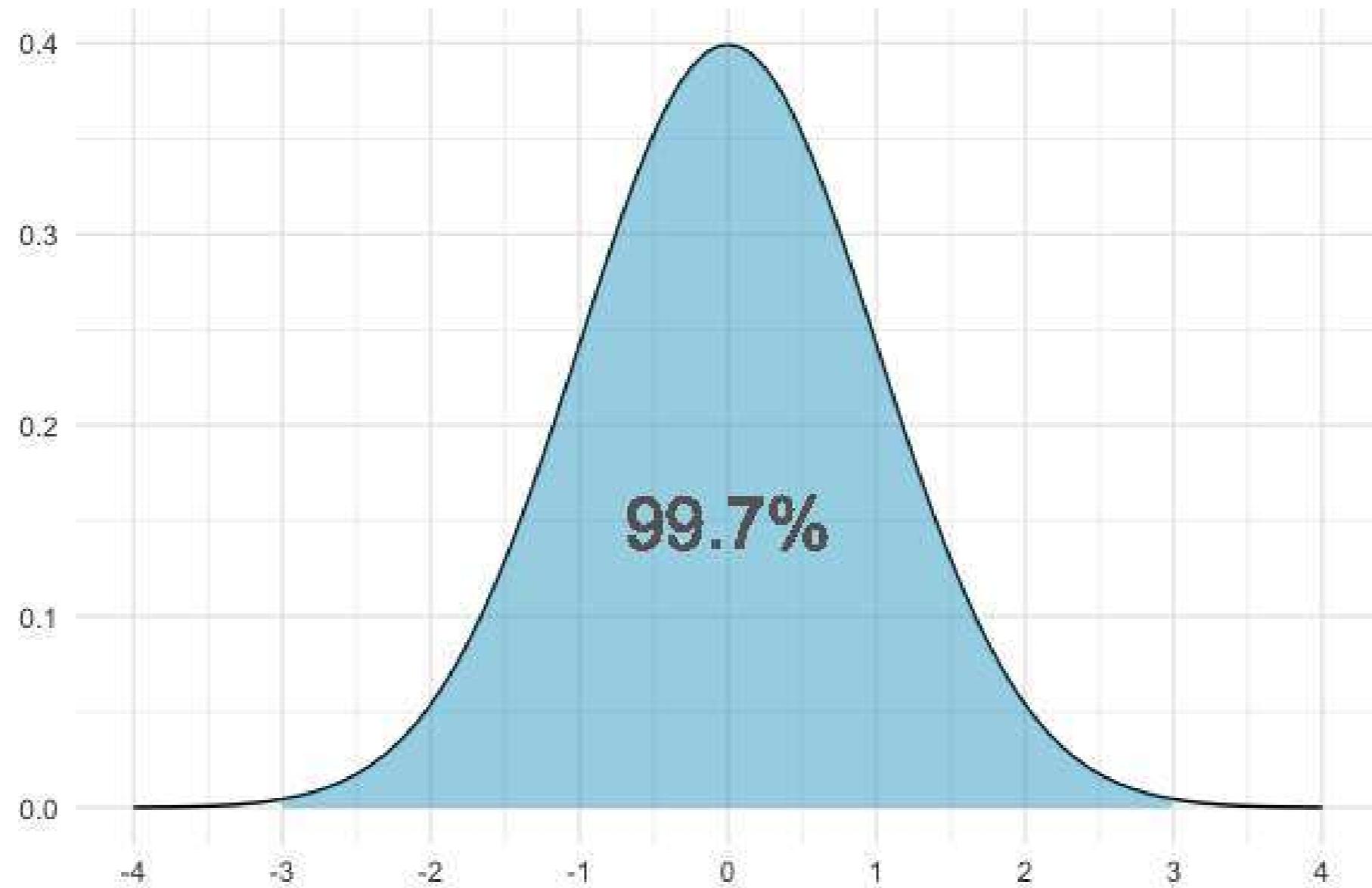
Areas under the normal distribution

95% falls within 2 standard deviations



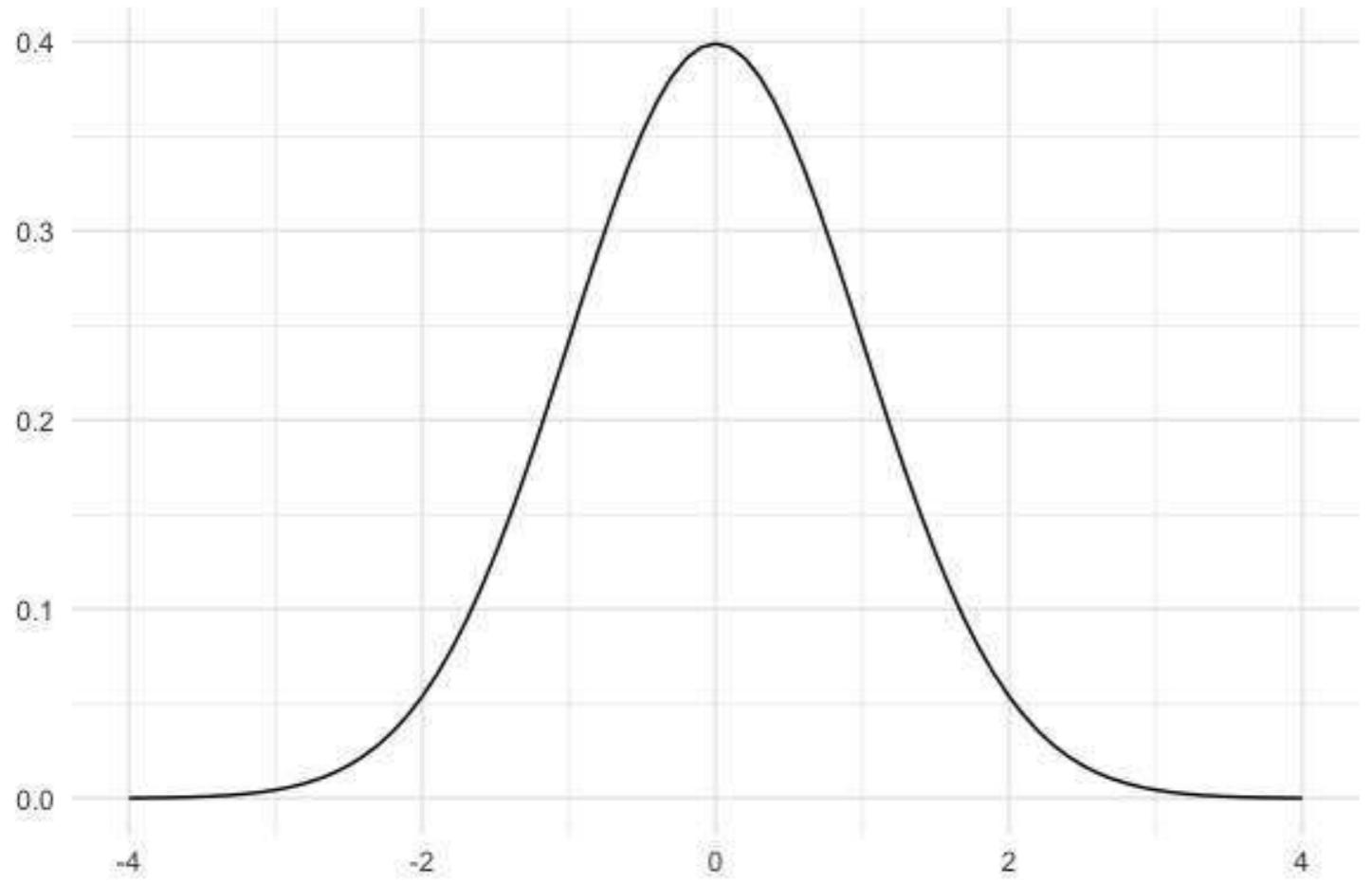
Areas under the normal distribution

99.7% falls within 3 standard deviations

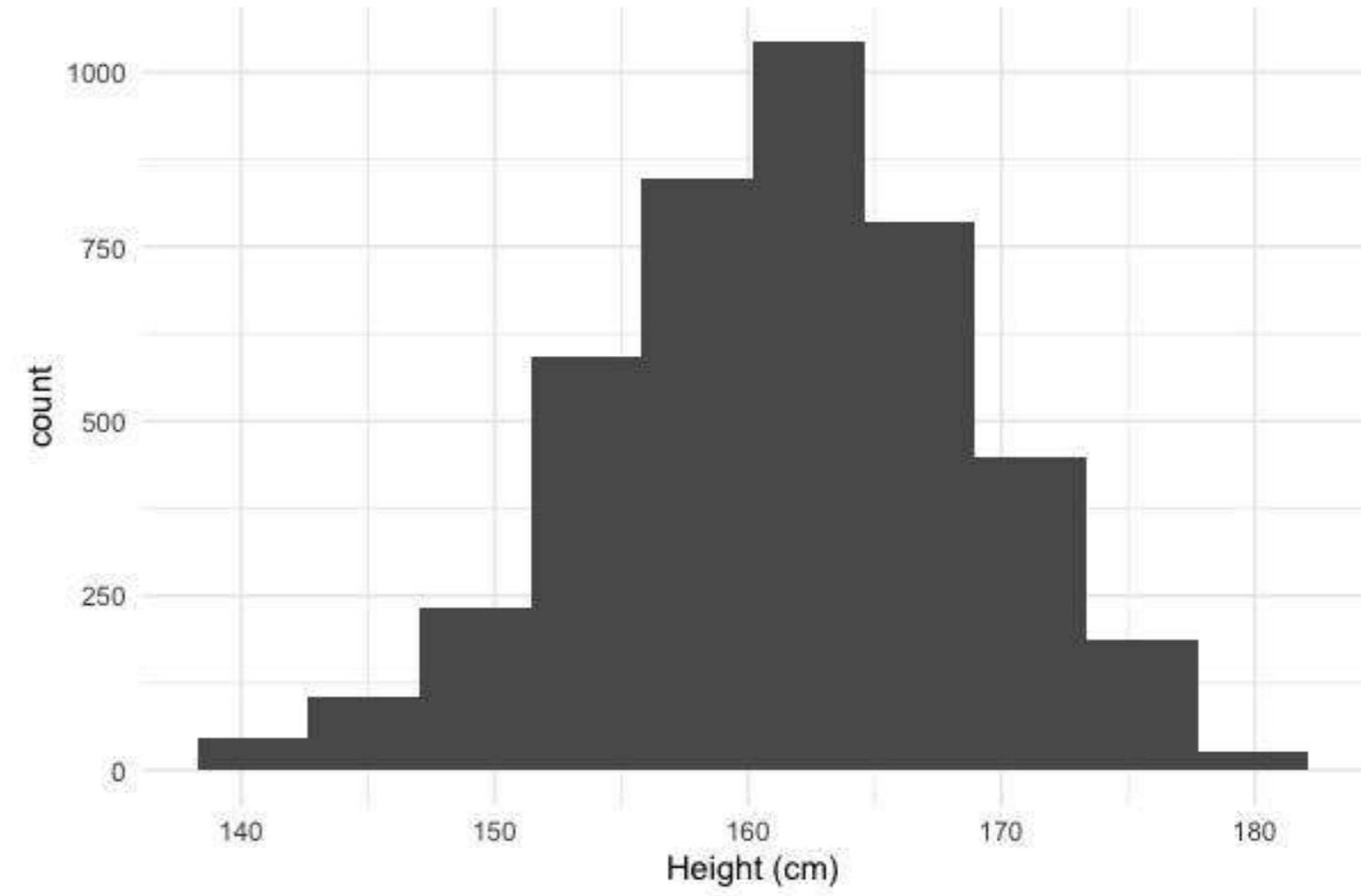


Lots of histograms look normal

Normal distribution



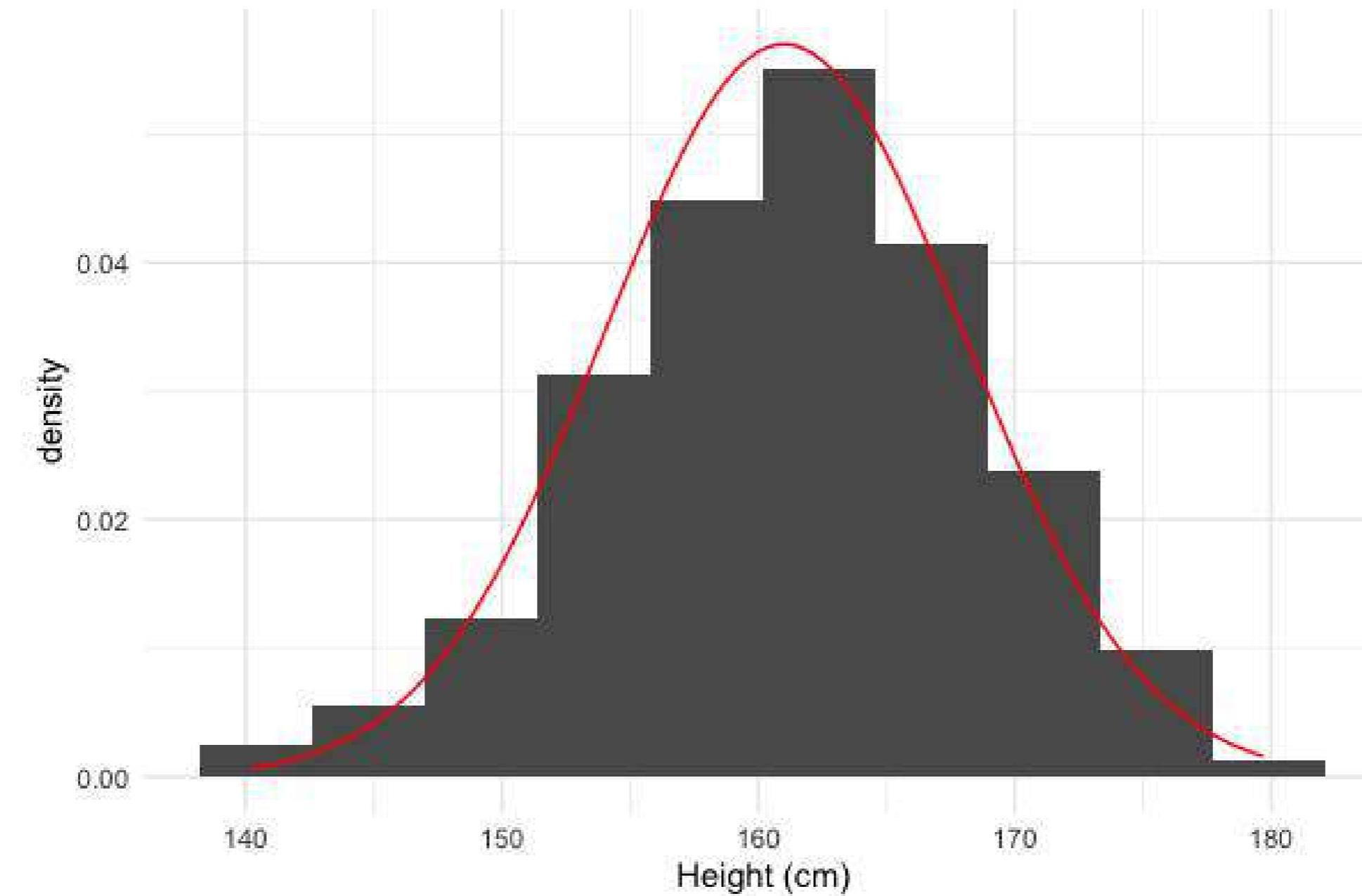
Women's heights from NHANES



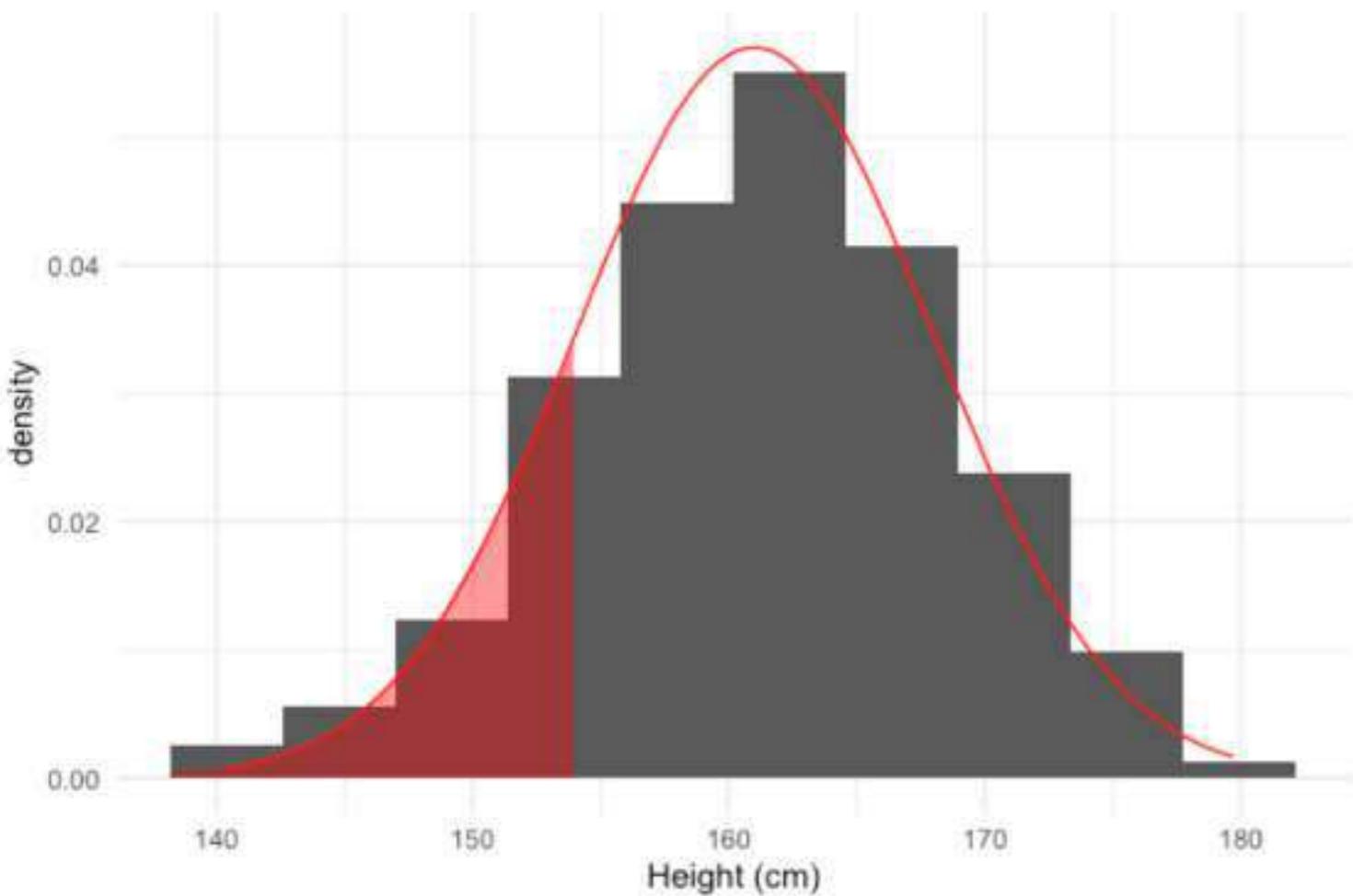
*Mean: 161 cm
deviation: 7 cm*

Standard

Approximating data with the normal distribution



What percent of women are shorter than 154 cm?

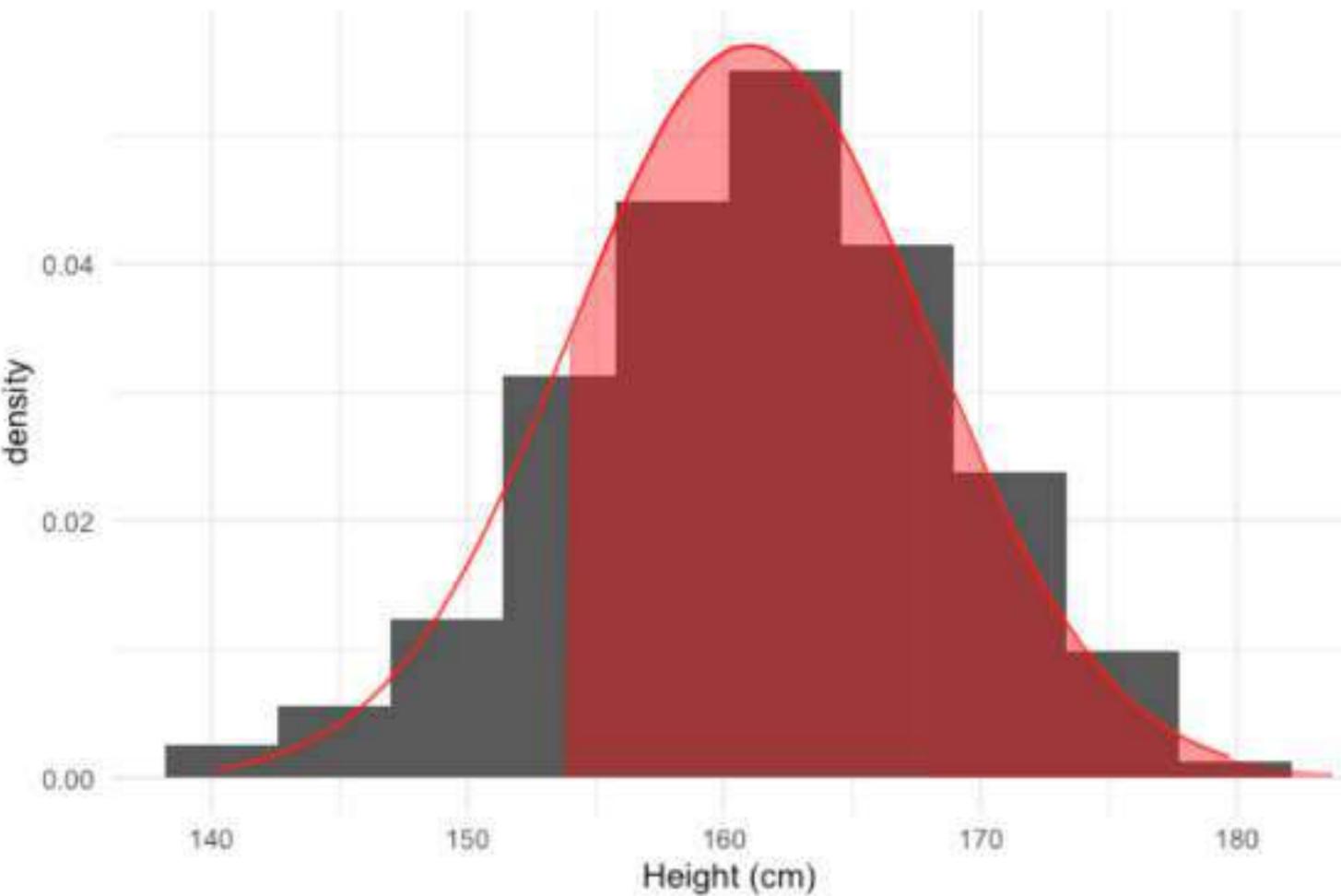


```
from scipy.stats import norm  
norm.cdf(154, 161, 7)
```

0.158655

16% of women in the survey are shorter than 154 cm

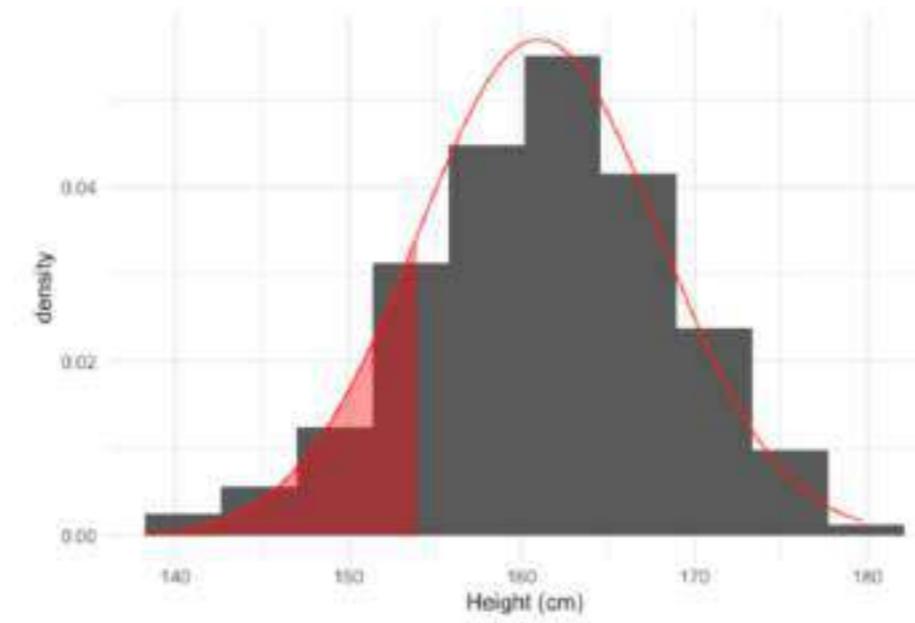
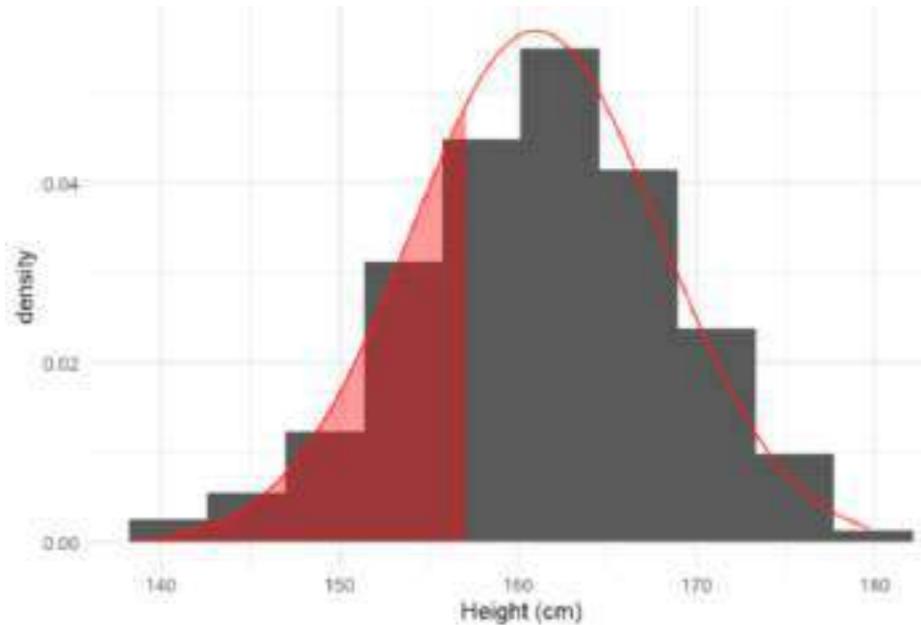
What percent of women are taller than 154 cm?



```
from scipy.stats import norm  
1 - norm.cdf(154, 161, 7)
```

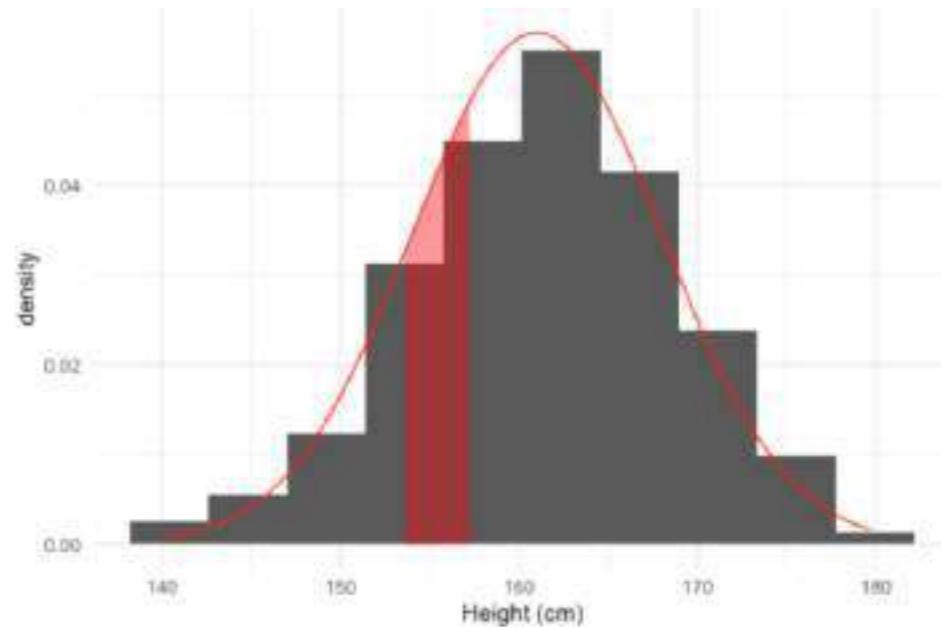
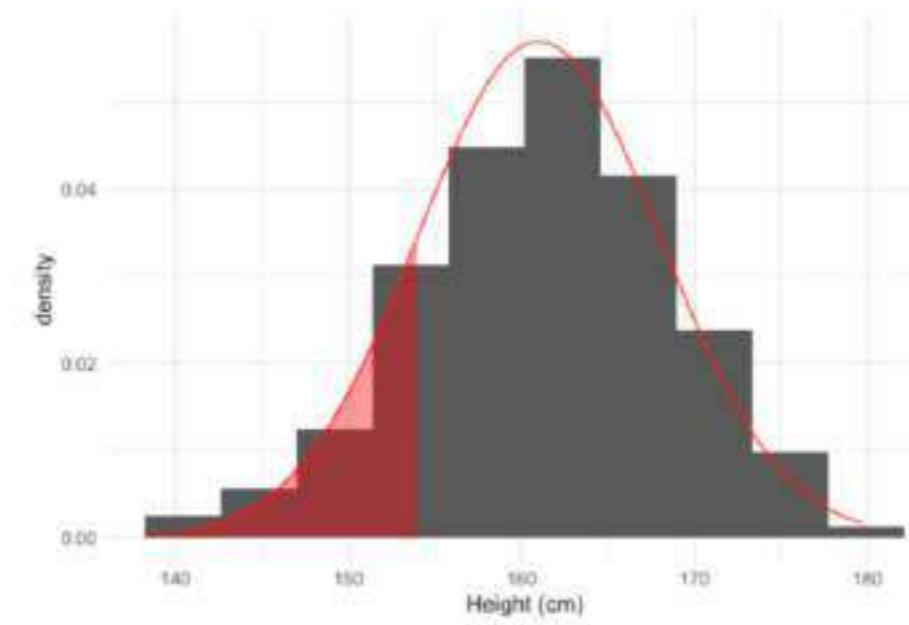
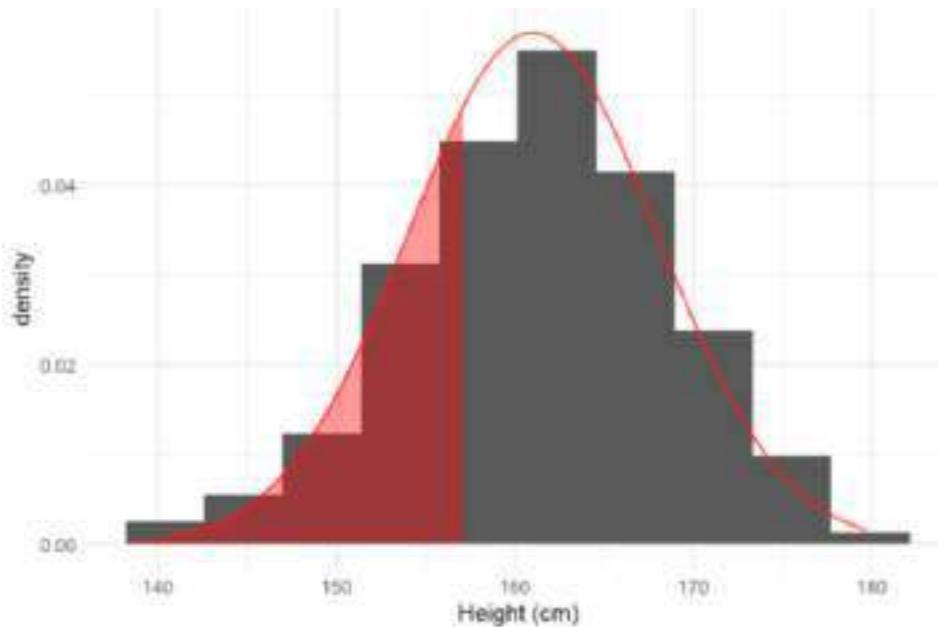
0.841345

What percent of women are 154-157 cm?



```
norm.cdf(157, 161, 7) - norm.cdf(154, 161, 7)
```

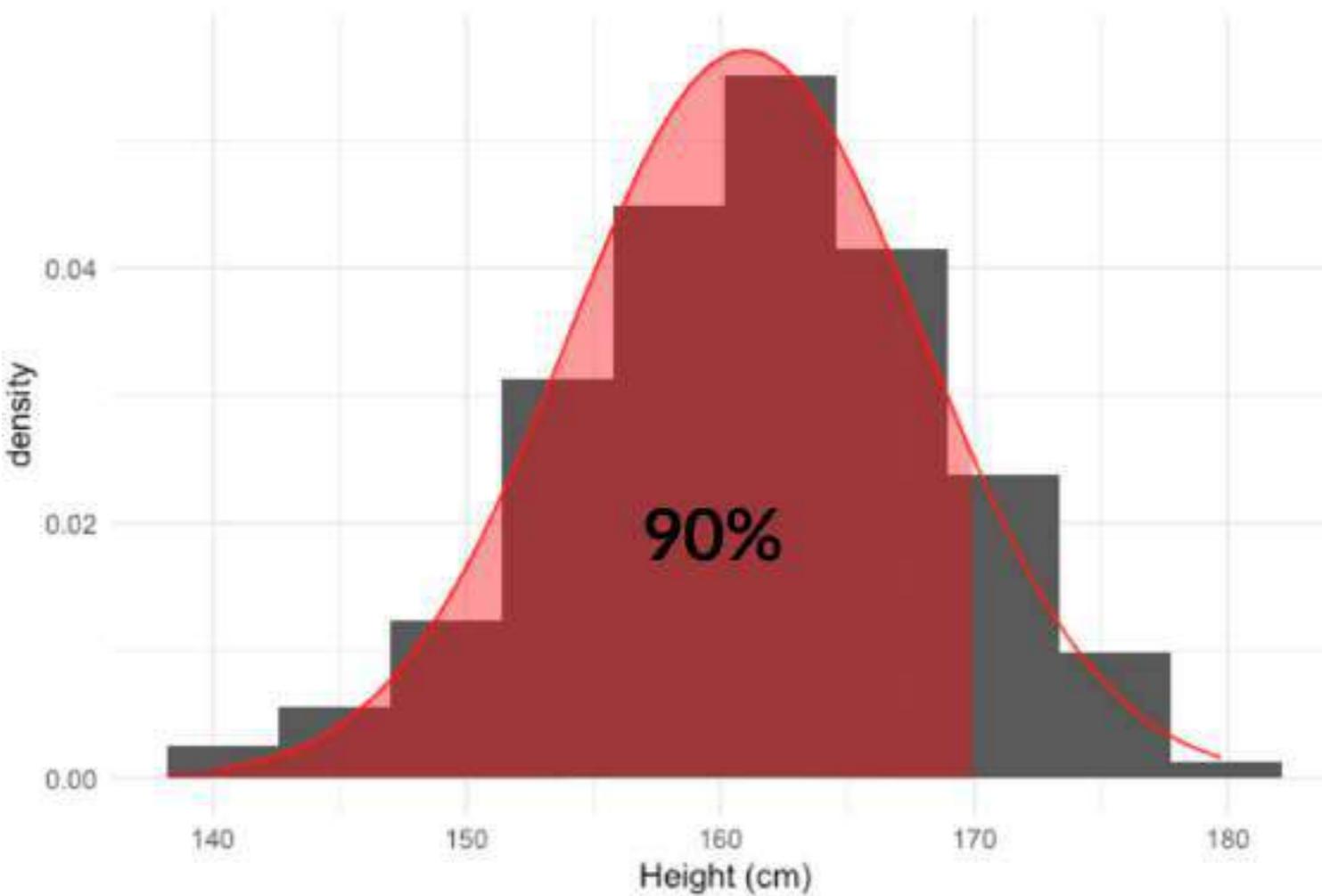
What percent of women are 154-157 cm?



```
norm.cdf(157, 161, 7) - norm.cdf(154, 161, 7)
```

0.1252

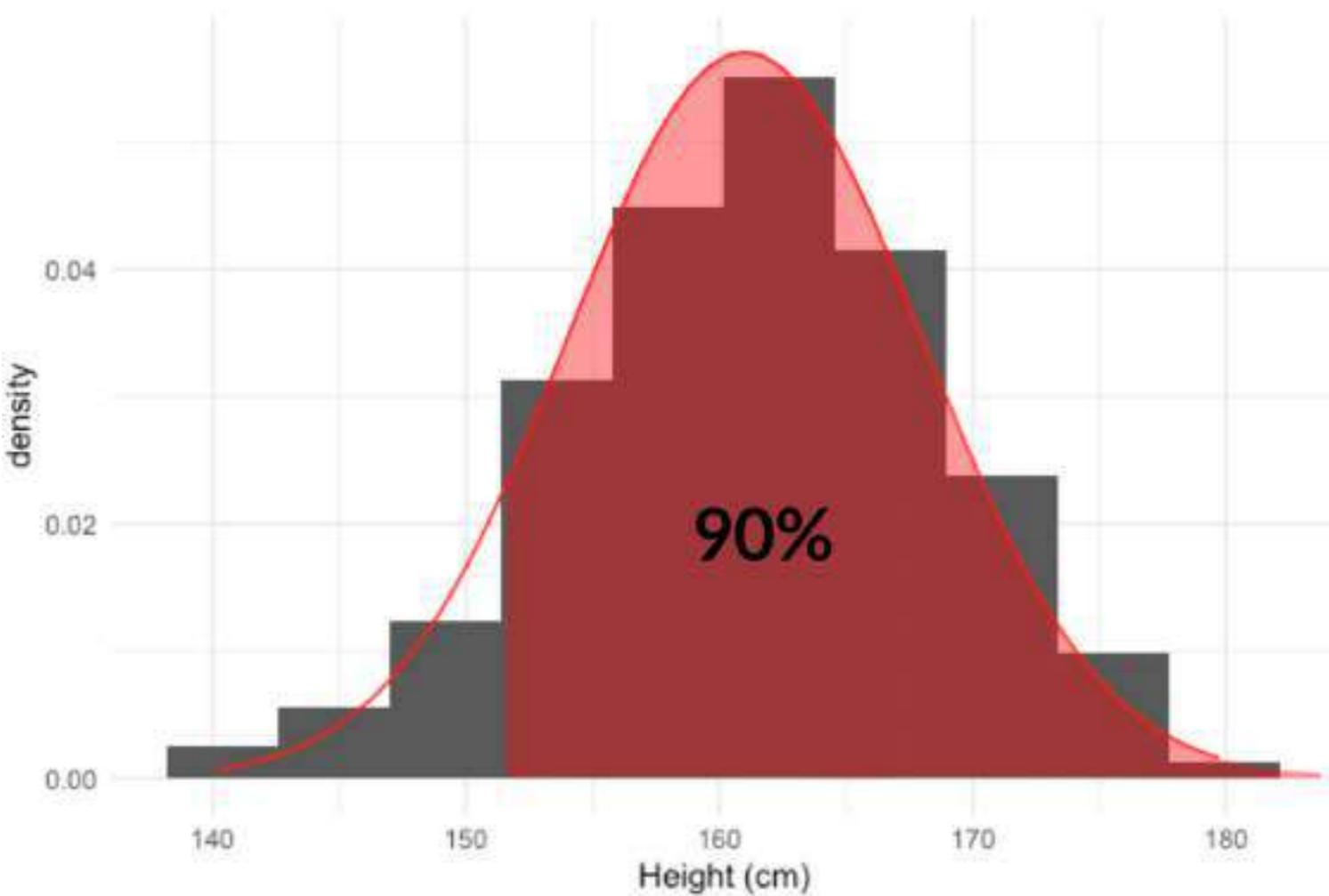
What height are 90% of women shorter than?



```
norm.ppf(0.9, 161, 7)
```

```
169.97086
```

What height are 90% of women taller than?



```
norm.ppf((1-0.9), 161, 7)
```

```
152.029
```

Generating random numbers

```
# Generate 10 random heights  
norm.rvs(161, 7, size=10)
```

```
array([155.5758223 , 155.13133235, 160.06377097, 168.33345778,  
       165.92273375, 163.32677057, 165.13280753, 146.36133538,  
       149.07845021, 160.5790856 ])
```

Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

The central limit theorem

INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

Content Developer, DataCamp

Rolling the dice 5 times

```
die = pd.Series([1, 2, 3, 4, 5, 6])
# Roll 5 times
samp_5 = die.sample(5, replace=True)
print(samp_5)
```

```
array([3, 1, 4, 1, 1])
```

```
np.mean(samp_5)
```

```
2.0
```



Rolling the dice 5 times

```
# Roll 5 times and take mean  
samp_5 = die.sample(5, replace=True)  
np.mean(samp_5)
```

4.4

```
samp_5 = die.sample(5, replace=True)  
np.mean(samp_5)
```

3.8

Rolling the dice 5 times 10 times

Repeat 10 times:

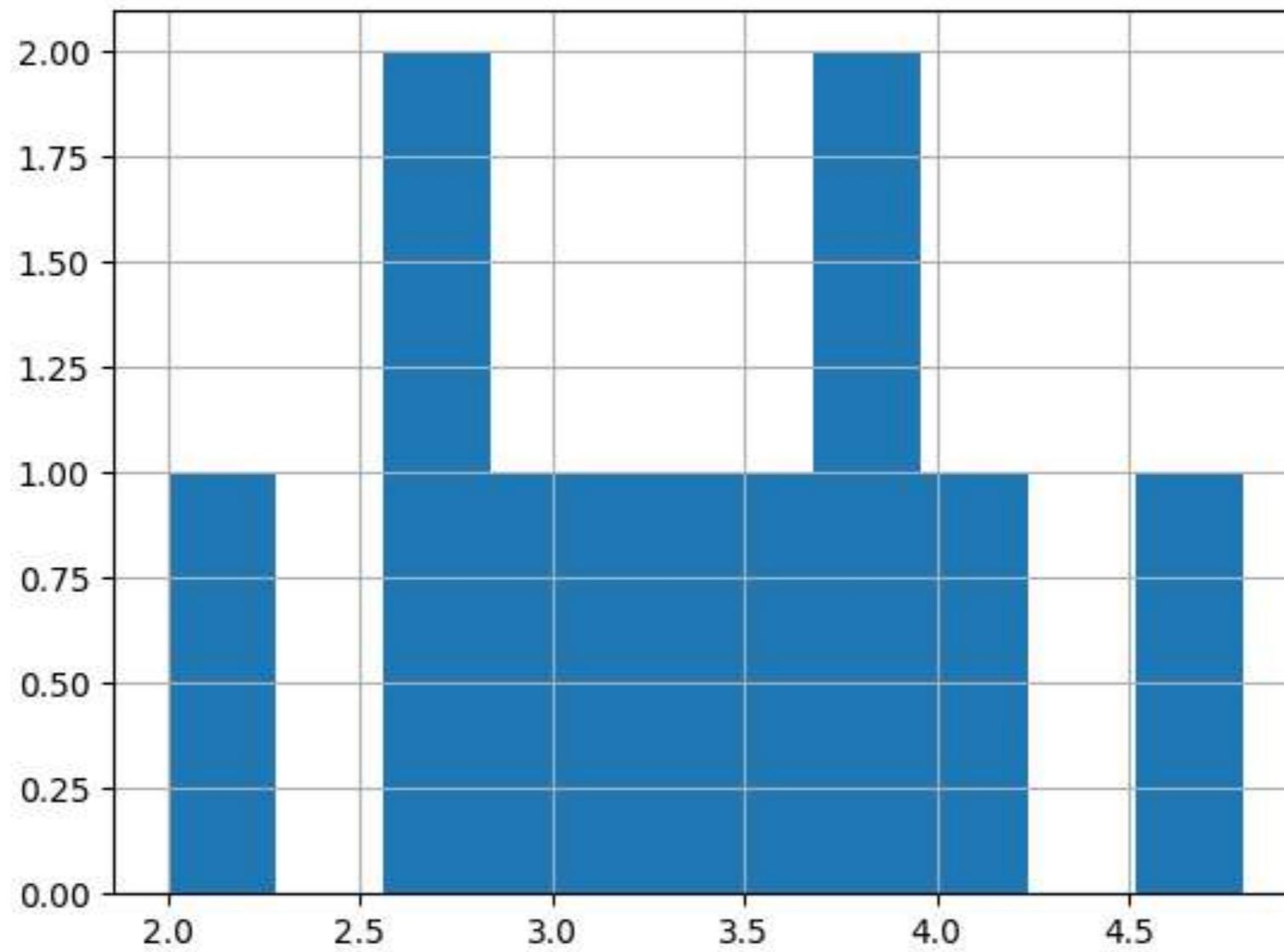
- Roll 5 times
- Take the mean

```
sample_means = []
for i in range(10):
    samp_5 = die.sample(5, replace=True)
    sample_means.append(np.mean(samp_5))
print(sample_means)
```

```
[3.8, 4.0, 3.8, 3.6, 3.2, 4.8, 2.6,
3.0, 2.6, 2.0]
```

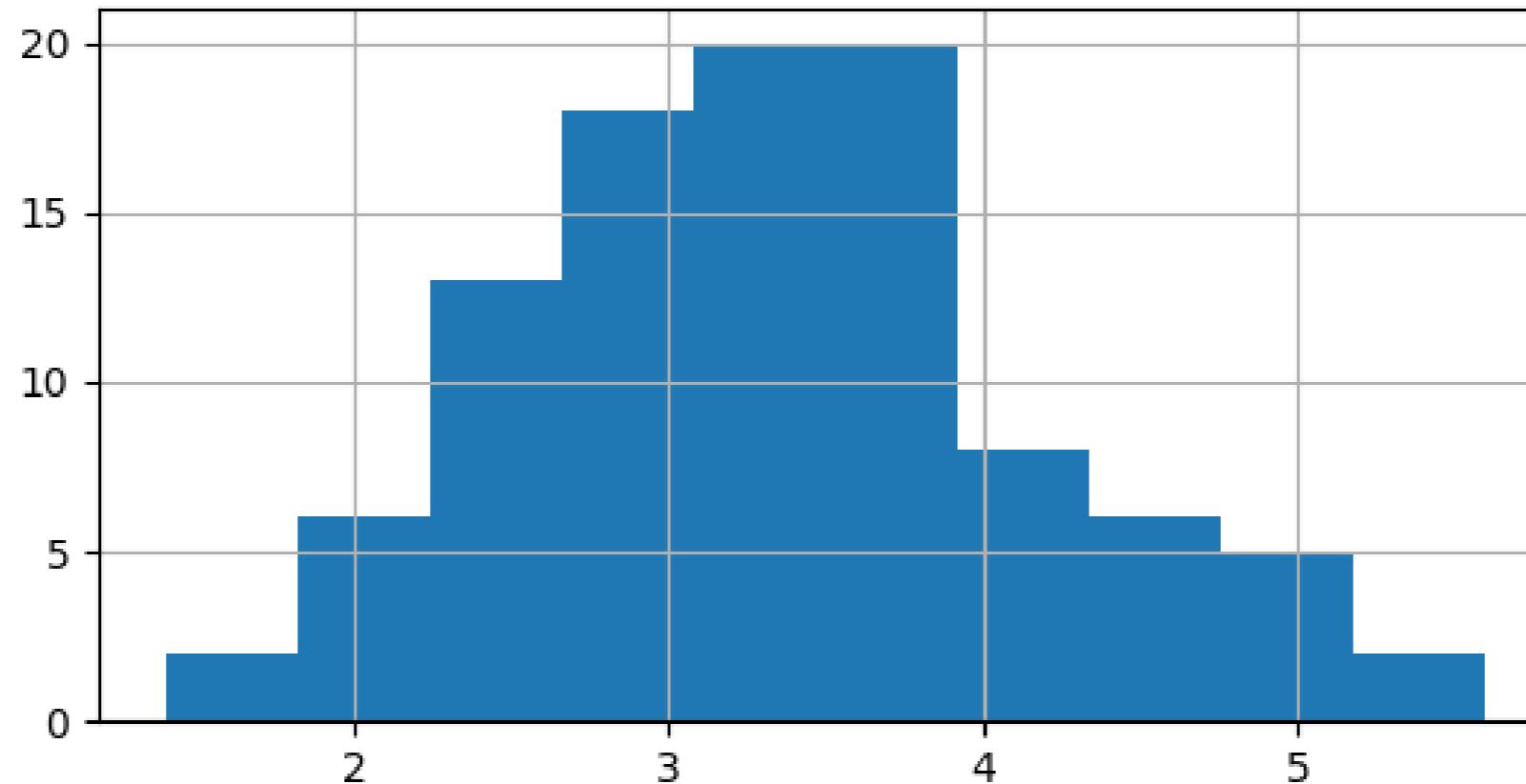
Sampling distributions

Sampling distribution of the sample mean



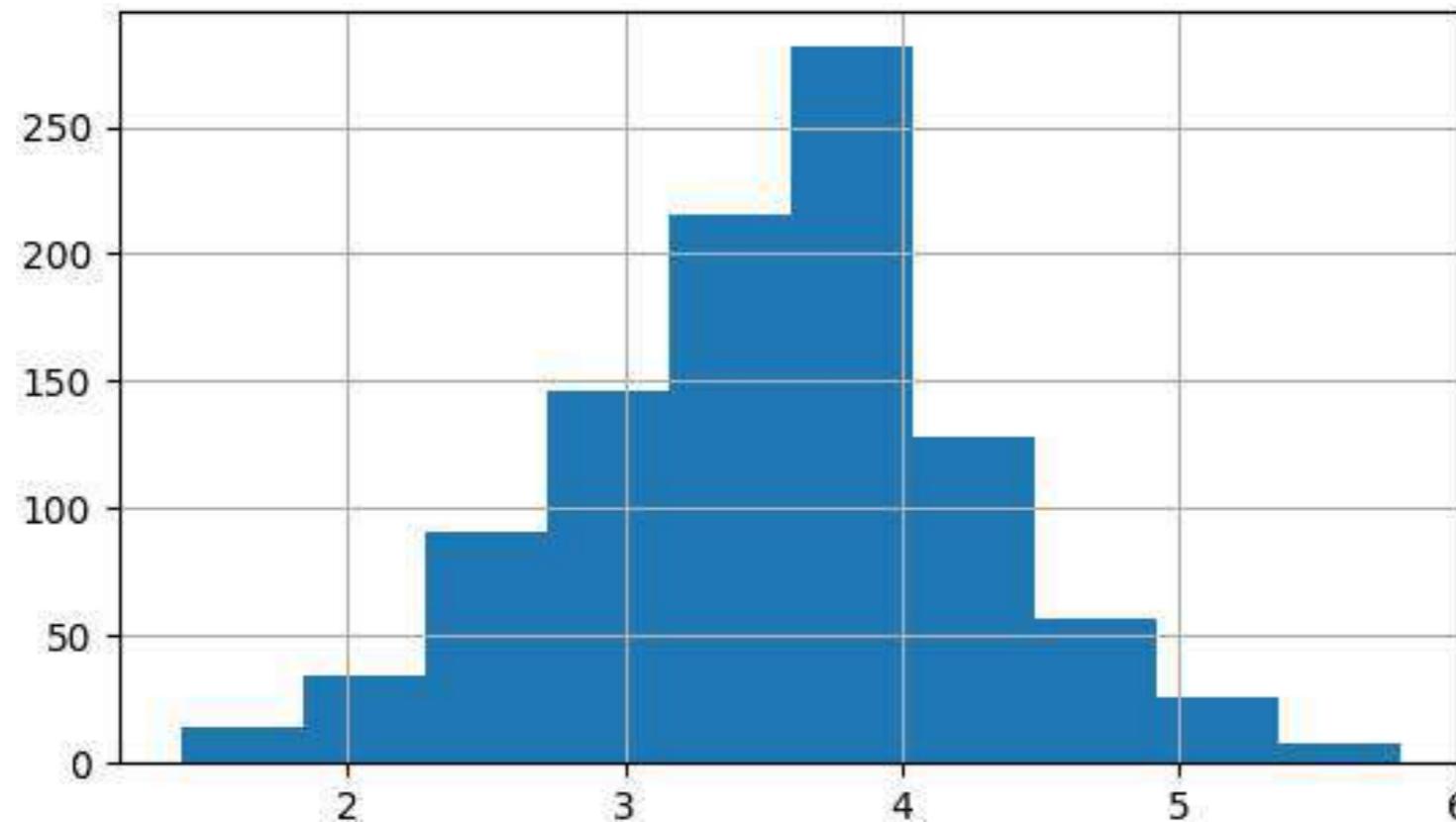
100 sample means

```
sample_means = []
for i in range(100):
    sample_means.append(np.mean(die.sample(5, replace=True)))
```



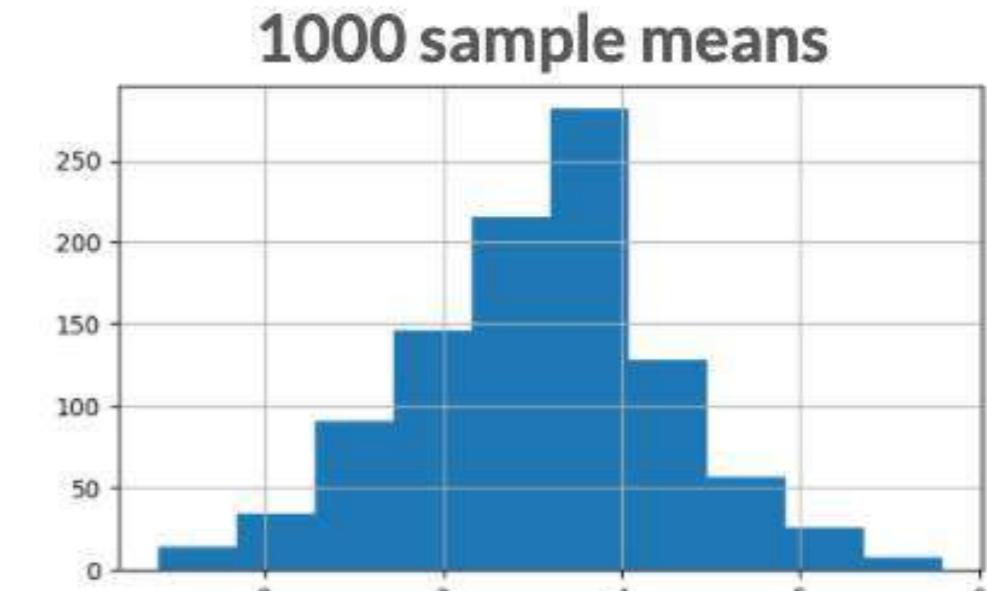
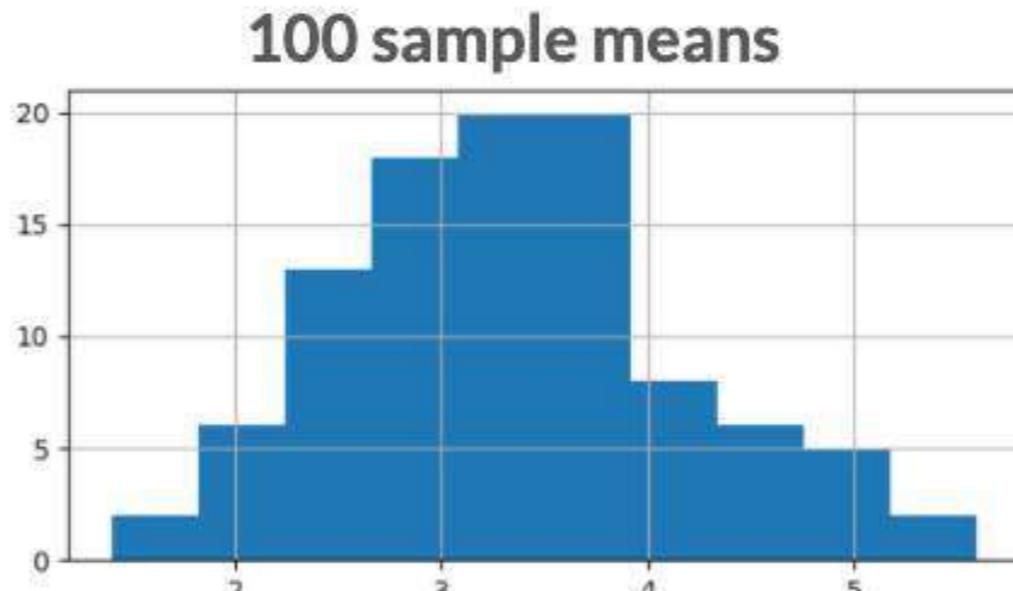
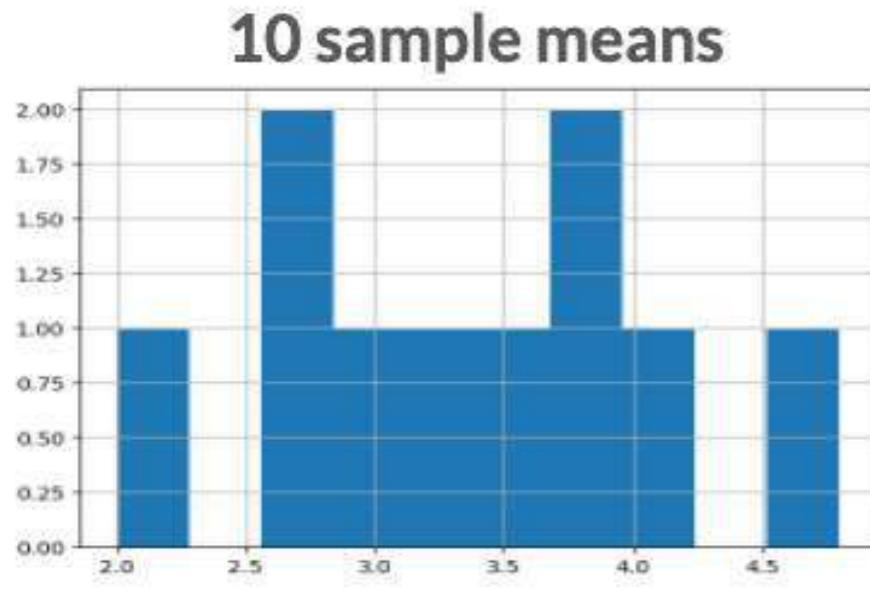
1000 sample means

```
sample_means = []
for i in range(1000):
    sample_means.append(np.mean(die.sample(5, replace=True)))
```



Central limit theorem

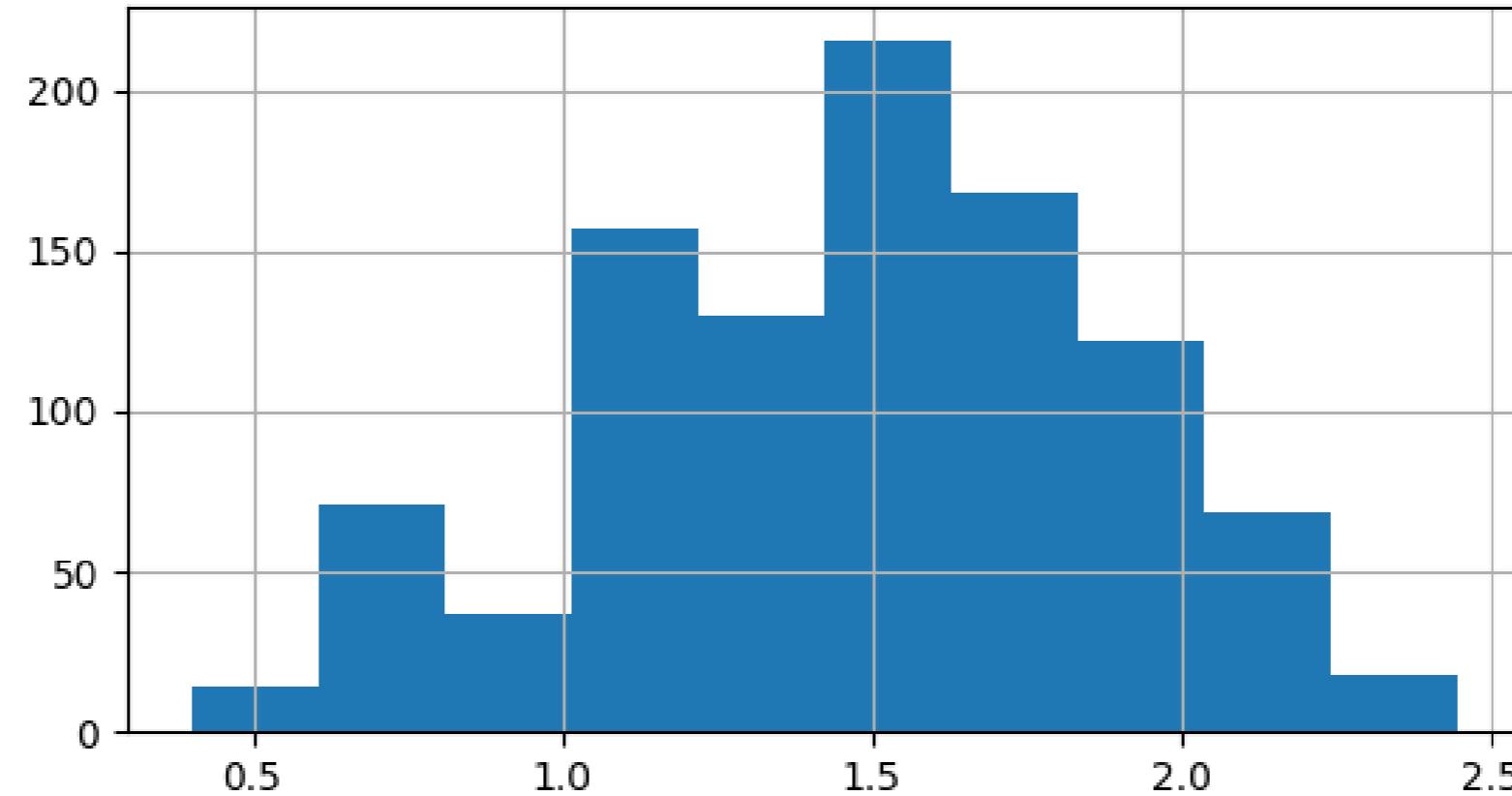
The sampling distribution of a statistic becomes closer to the normal distribution as the number of trials increases.



* Samples should be random and independent

Standard deviation and the CLT

```
sample_sds = []
for i in range(1000):
    sample_sds.append(np.std(die.sample(5, replace=True)))
```



Proportions and the CLT

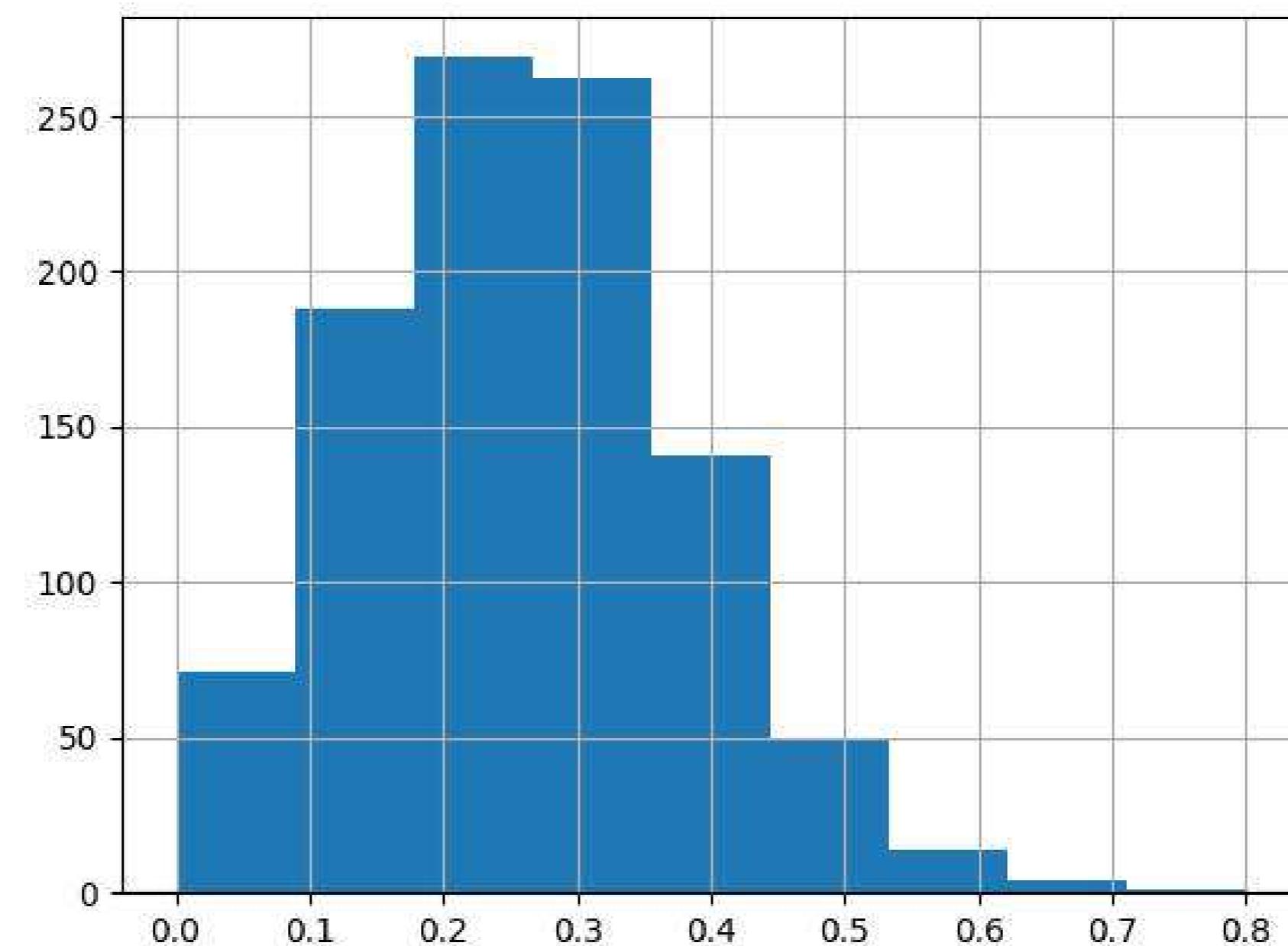
```
sales_team = pd.Series(["Amir", "Brian", "Claire", "Damian"])
sales_team.sample(10, replace=True)
```

```
array(['Claire', 'Damian', 'Brian', 'Damian', 'Damian', 'Amir', 'Amir', 'Amir',
       'Amir'], dtype=object)
```

```
sales_team.sample(10, replace=True)
```

```
array(['Brian', 'Amir', 'Brian', 'Claire', 'Brian', 'Damian', 'Claire', 'Brian',
       'Claire', 'Claire'], dtype=object)
```

Sampling distribution of proportion



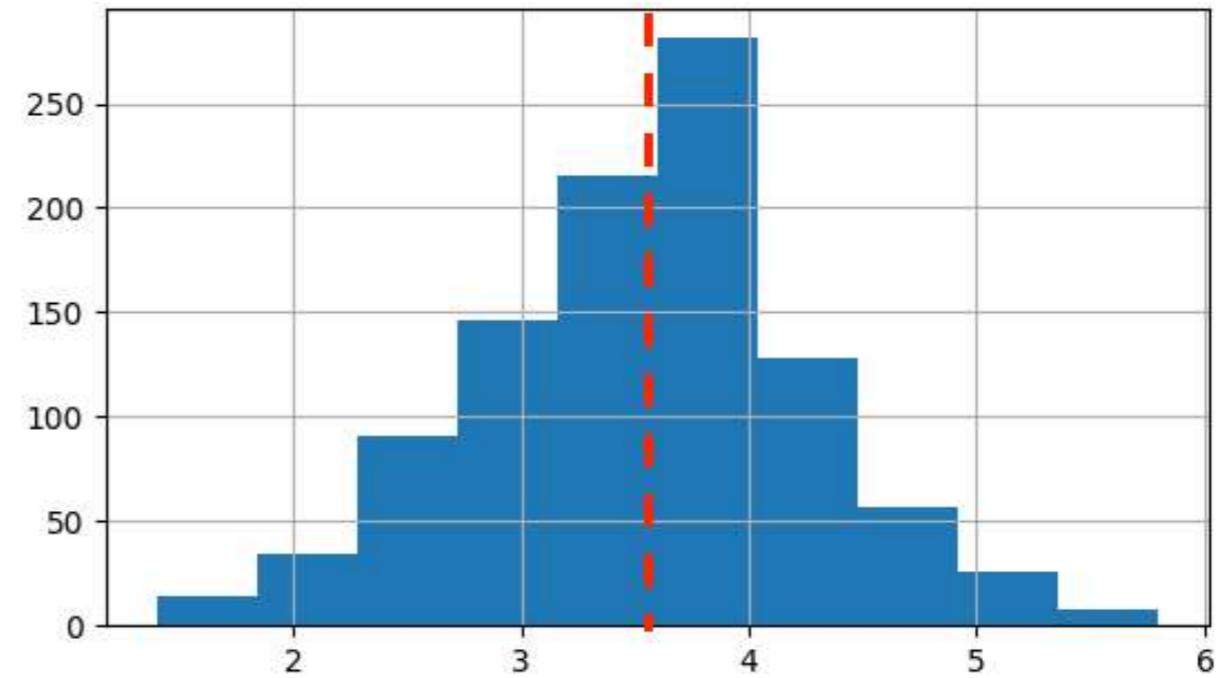
Mean of sampling distribution

```
# Estimate expected value of die  
np.mean(sample_means)
```

3.48

```
# Estimate proportion of "Claire's"  
np.mean(sample_props)
```

0.26



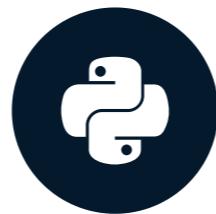
- Estimate characteristics of unknown underlying distribution
- More easily estimate characteristics of large populations

Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

The Poisson distribution

INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

Content Developer, DataCamp

Poisson processes

- Events appear to happen at a certain rate, but completely at random
- Examples
 - Number of animals adopted from an animal shelter per week
 - Number of people arriving at a restaurant per hour
 - Number of earthquakes in California per year
- Time unit is irrelevant, as long as you use the same unit when talking about the same situation

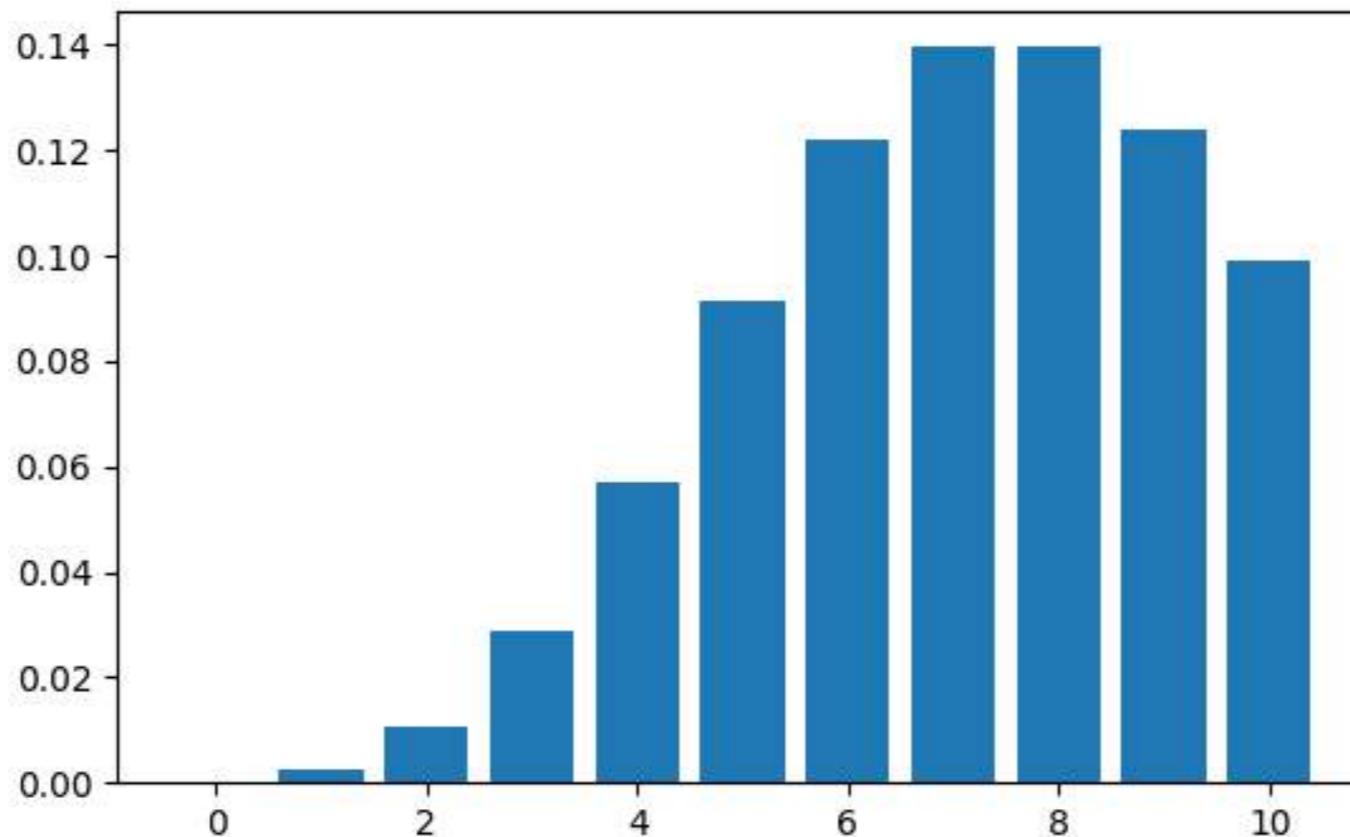


Poisson distribution

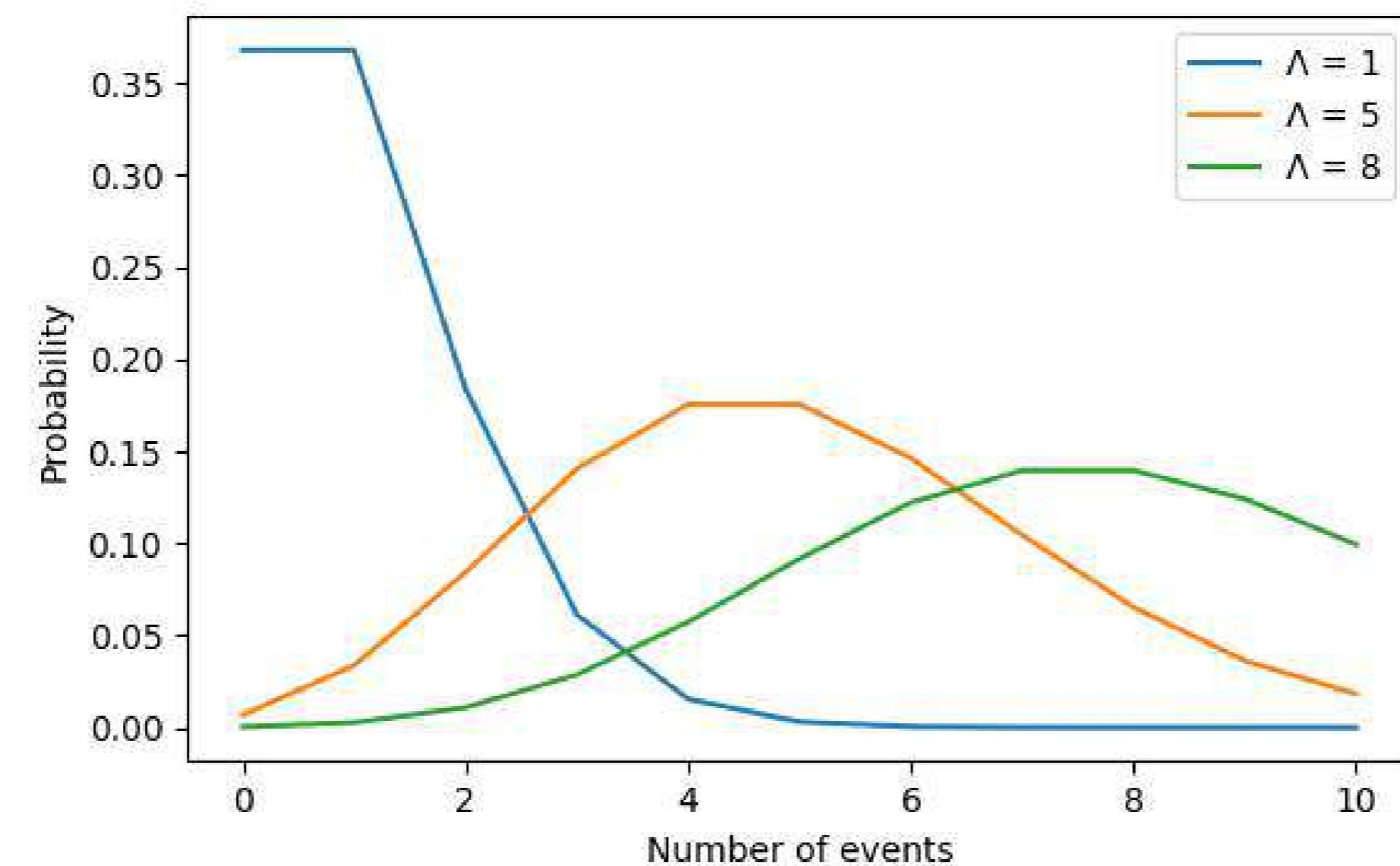
- Probability of some # of events occurring over a fixed period of time
- Examples
 - Probability of ≥ 5 animals adopted from an animal shelter per week
 - Probability of 12 people arriving at a restaurant per hour
 - Probability of < 20 earthquakes in California per year

Lambda (λ)

- λ = average number of events per time interval
 - Average number of adoptions per week = 8



Lambda is the distribution's peak



Probability of a single value

If the average number of adoptions per week is 8, what is $P(\# \text{ adoptions in a week} = 5)$?

```
from scipy.stats import poisson  
poisson.pmf(5, 8)
```

0.09160366

Probability of less than or equal to

If the average number of adoptions per week is 8, what is $P(\# \text{ adoptions in a week} \leq 5)$?

```
from scipy.stats import poisson  
poisson.cdf(5, 8)
```

0.1912361

Probability of greater than

If the average number of adoptions per week is 8, what is $P(\# \text{ adoptions in a week} > 5)$?

```
1 - poisson.cdf(5, 8)
```

0.8087639

If the average number of adoptions per week is 10, what is $P(\# \text{ adoptions in a week} > 5)$?

```
1 - poisson.cdf(5, 10)
```

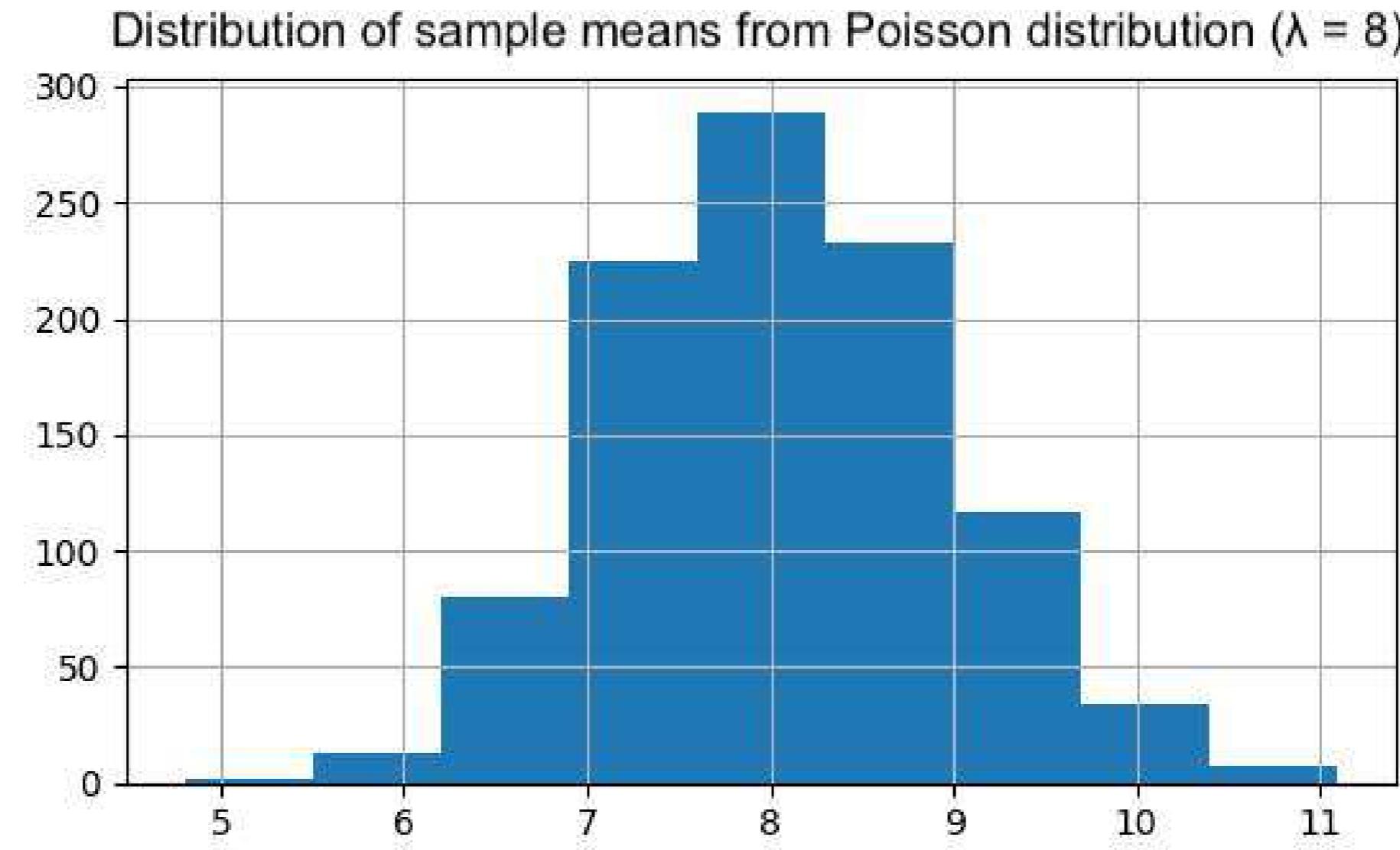
0.932914

Sampling from a Poisson distribution

```
from scipy.stats import poisson  
poisson.rvs(8, size=10)
```

```
array([ 9,  9,  8,  7, 11,  3, 10,  6,  8, 14])
```

The CLT still applies!

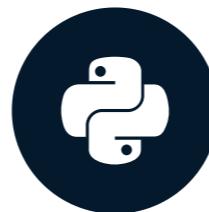


Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

More probability distributions

INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

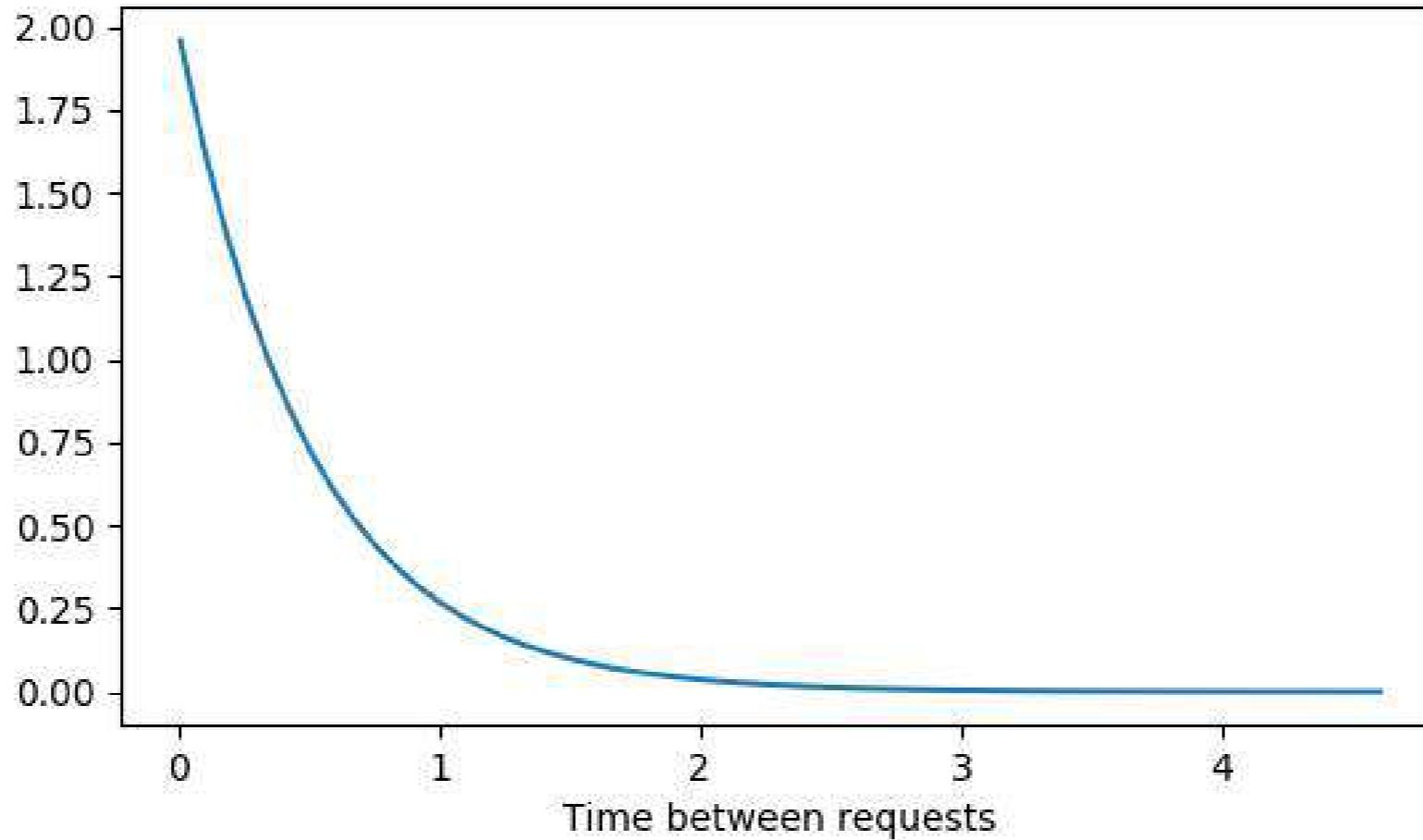
Content Developer, DataCamp

Exponential distribution

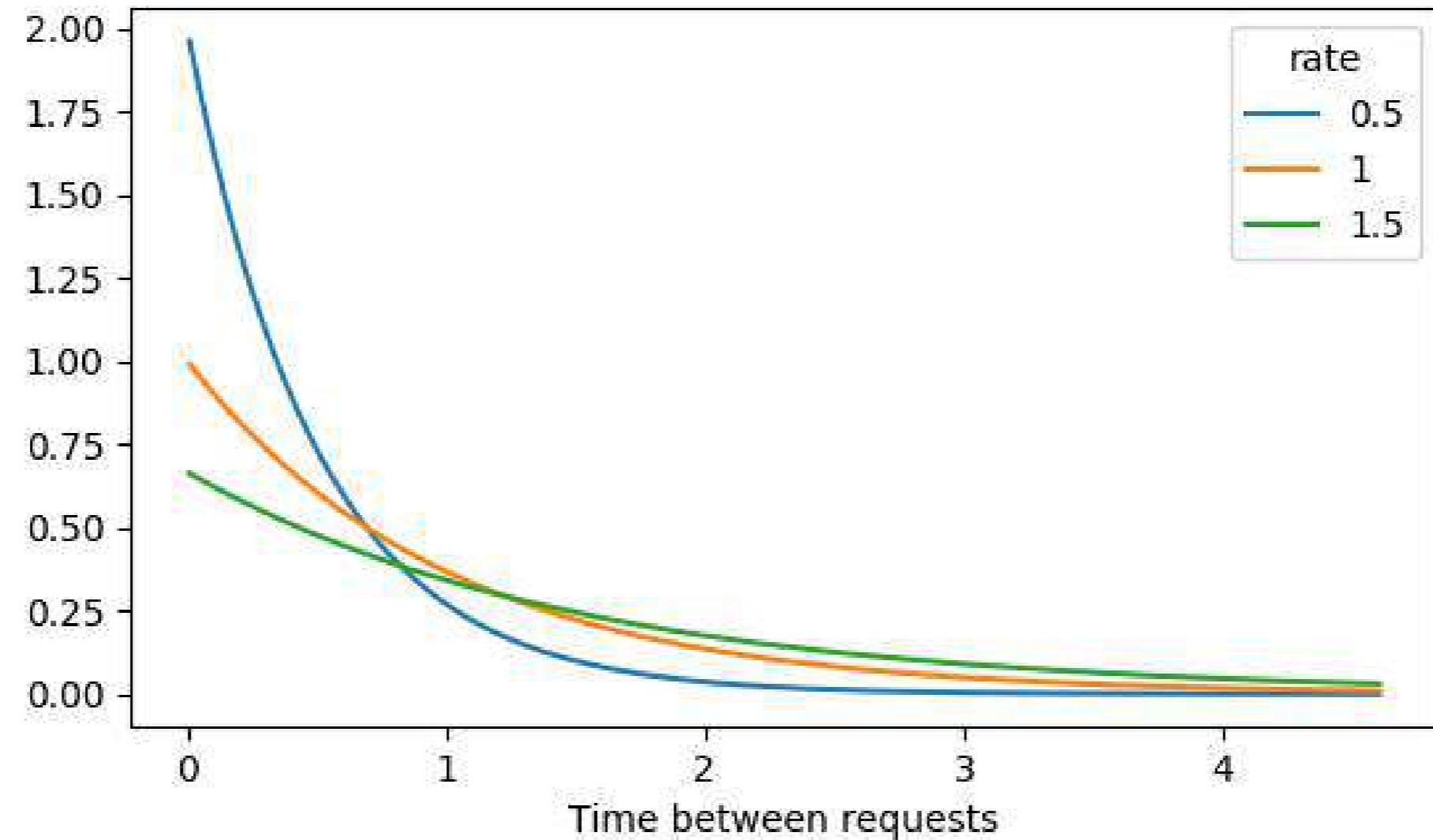
- Probability of time between Poisson events
- Examples
 - Probability of > 1 day between adoptions
 - Probability of < 10 minutes between restaurant arrivals
 - Probability of 6-8 months between earthquakes
- Also uses lambda (rate)
- Continuous (time)

Customer service requests

- On average, one customer service ticket is created every 2 minutes
 - $\lambda = 0.5$ customer service tickets created each minute



Lambda in exponential distribution



Expected value of exponential distribution

In terms of rate (Poisson):

- $\lambda = 0.5$ requests per minute

In terms of time between events (exponential):

- $1/\lambda = 1$ request per 2 minutes
- $1/0.5 = 2$

How long until a new request is created?

$$P(\text{wait} < 1 \text{ min}) =$$

```
from scipy.stats import expon
```

- `scale` = $1/\lambda = 1/0.5 = 2$

```
expon.cdf(1, scale=2)
```

```
0.3934693402873666
```

$$P(\text{wait} > 4 \text{ min}) =$$

```
1 - expon.cdf(4, scale=2)
```

```
0.1353352832366127
```

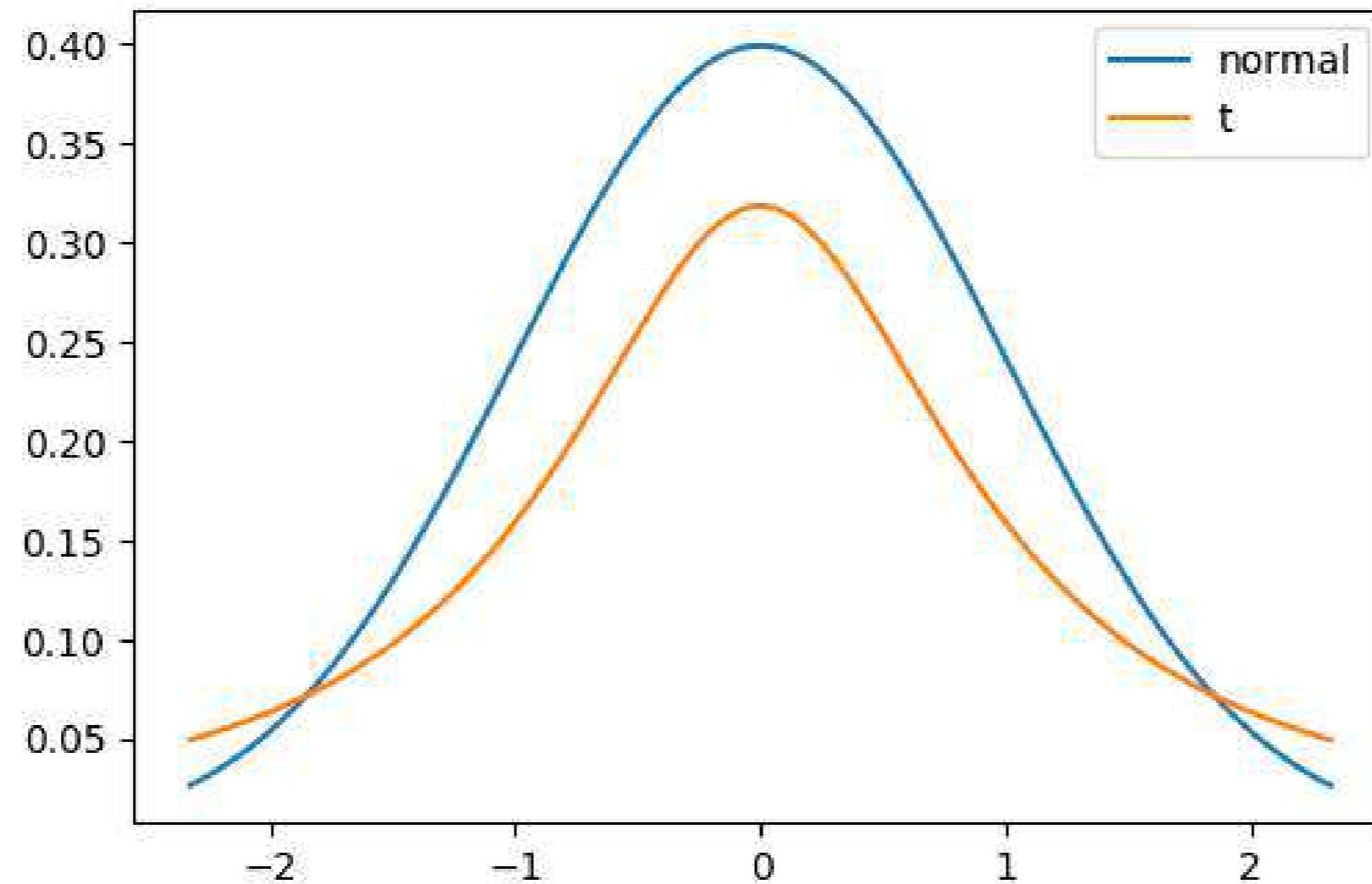
$$P(1 \text{ min} < \text{wait} < 4 \text{ min}) =$$

```
expon.cdf(4, scale=2) - expon.cdf(1, scale=2)
```

```
0.4711953764760207
```

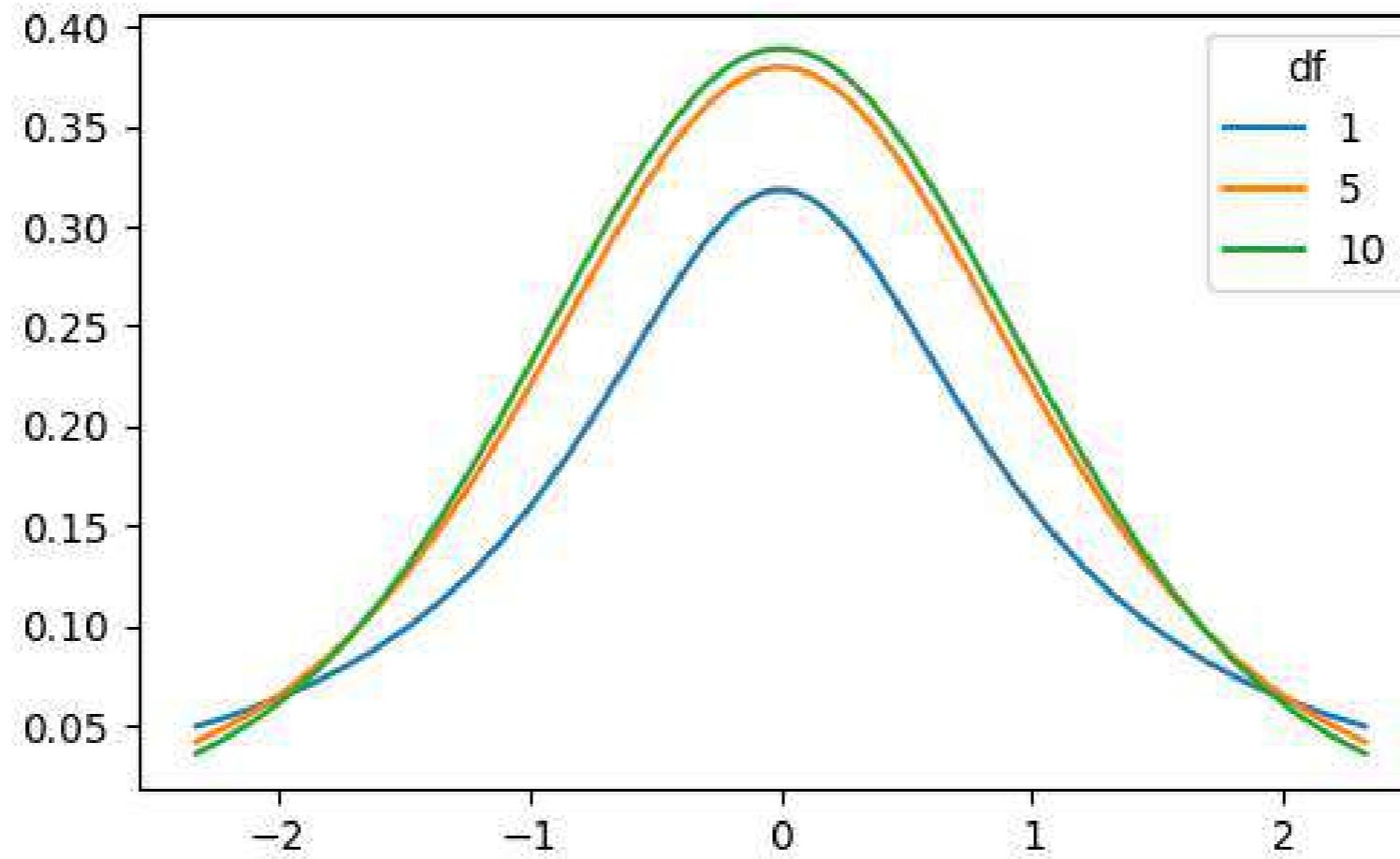
(Student's) t-distribution

- Similar shape as the normal distribution



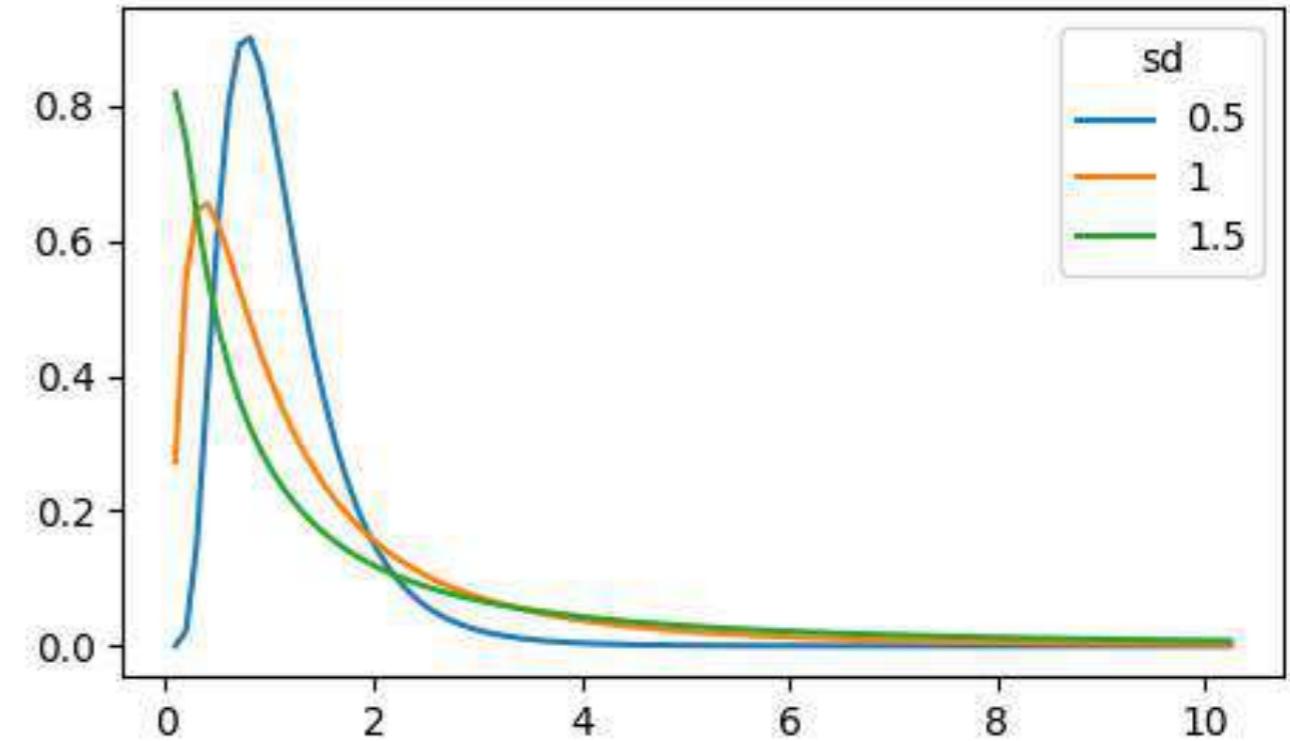
Degrees of freedom

- Has parameter degrees of freedom (df) which affects the thickness of the tails
 - Lower df = thicker tails, higher standard deviation
 - Higher df = closer to normal distribution



Log-normal distribution

- Variable whose logarithm is normally distributed
- Examples:
 - Length of chess games
 - Adult blood pressure
 - Number of hospitalizations in the 2003 SARS outbreak



Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

Correlation

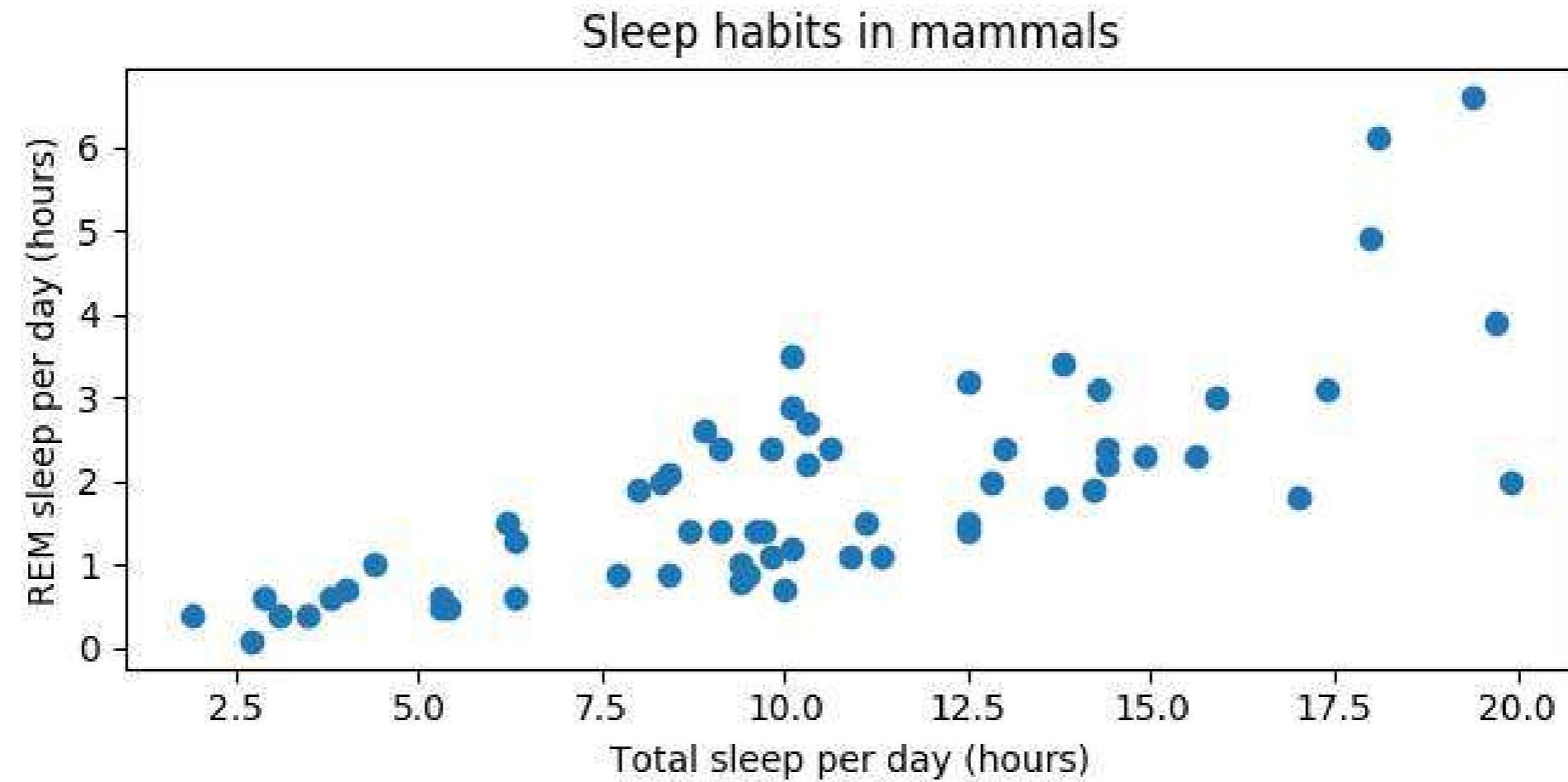
INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

Content Developer, DataCamp

Relationships between two variables



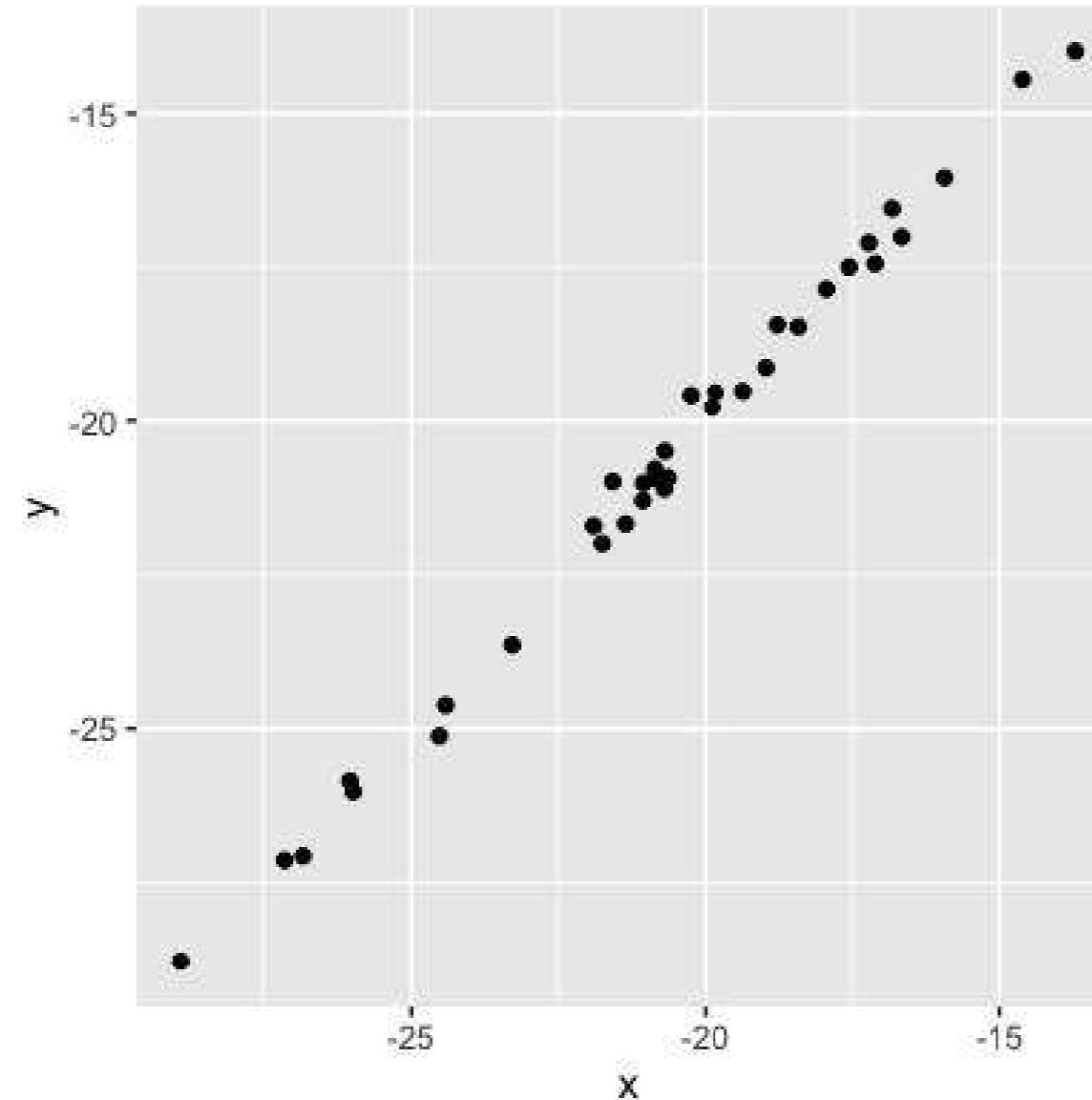
- x = explanatory/independent variable
- y = response/dependent variable

Correlation coefficient

- Quantifies the linear relationship between two variables
- Number between -1 and 1
- Magnitude corresponds to strength of relationship
- Sign (+ or -) corresponds to direction of relationship

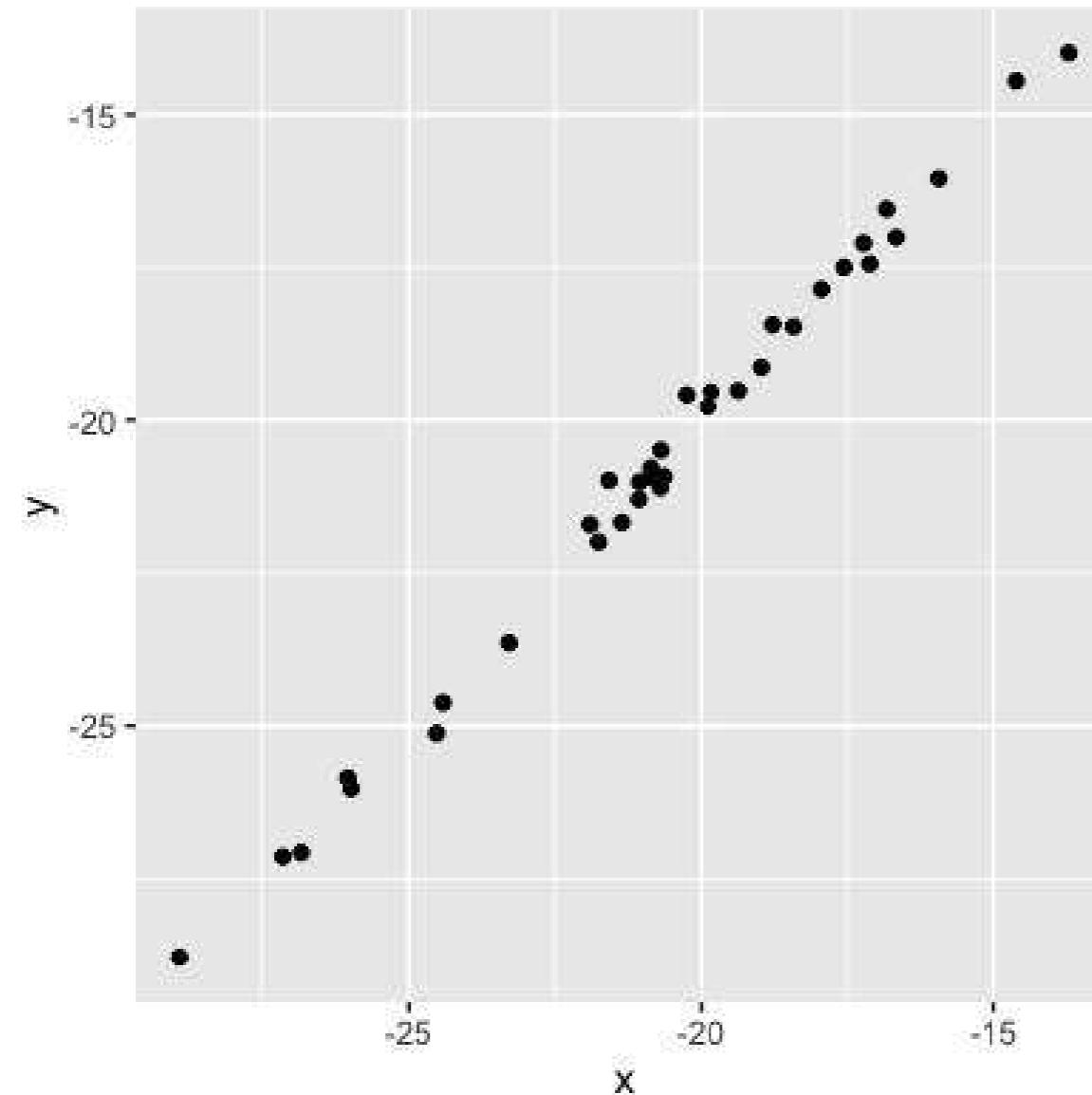
Magnitude = strength of relationship

0.99 (very strong
relationship)

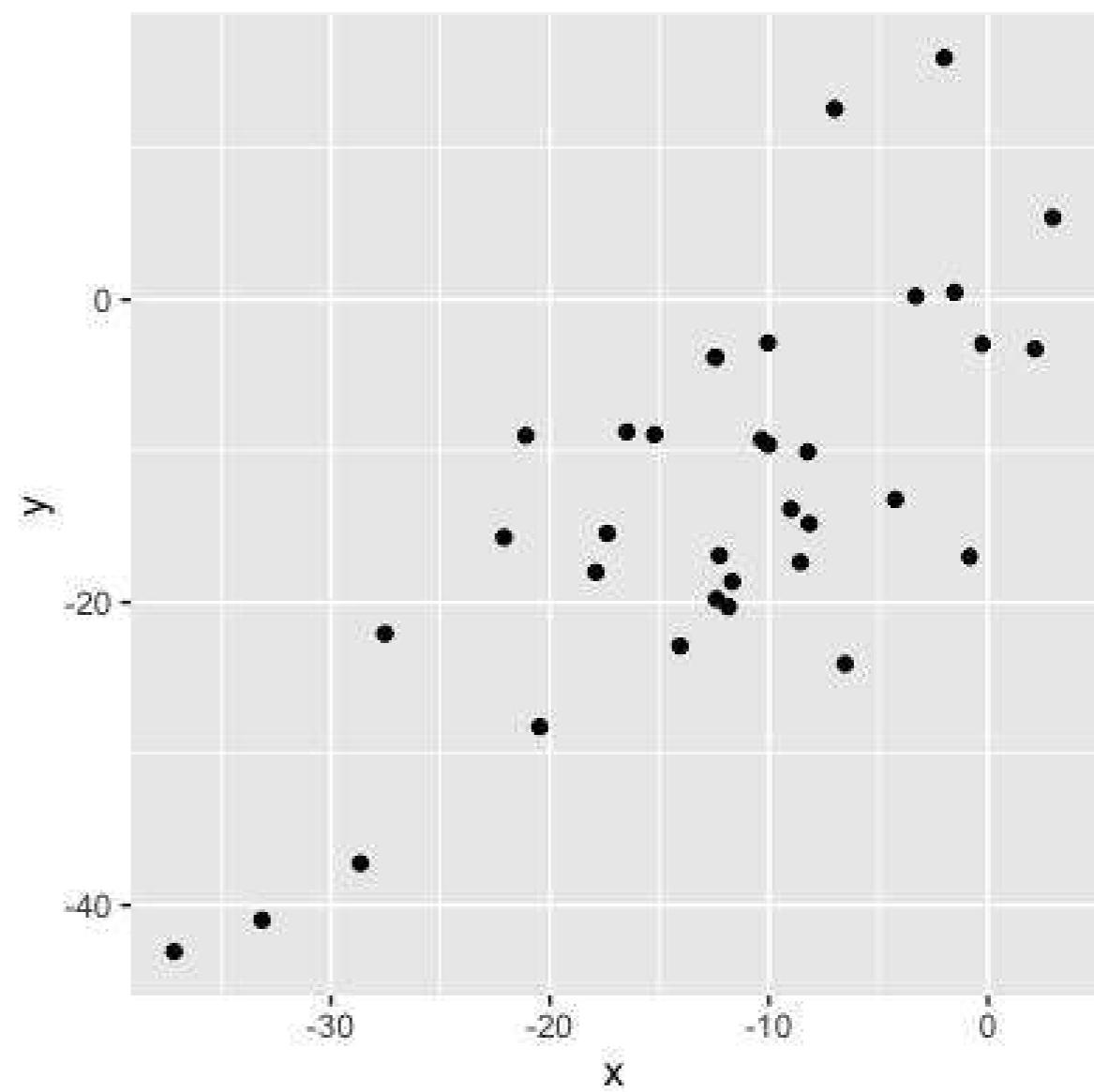


Magnitude = strength of relationship

0.99 (very strong
relationship)

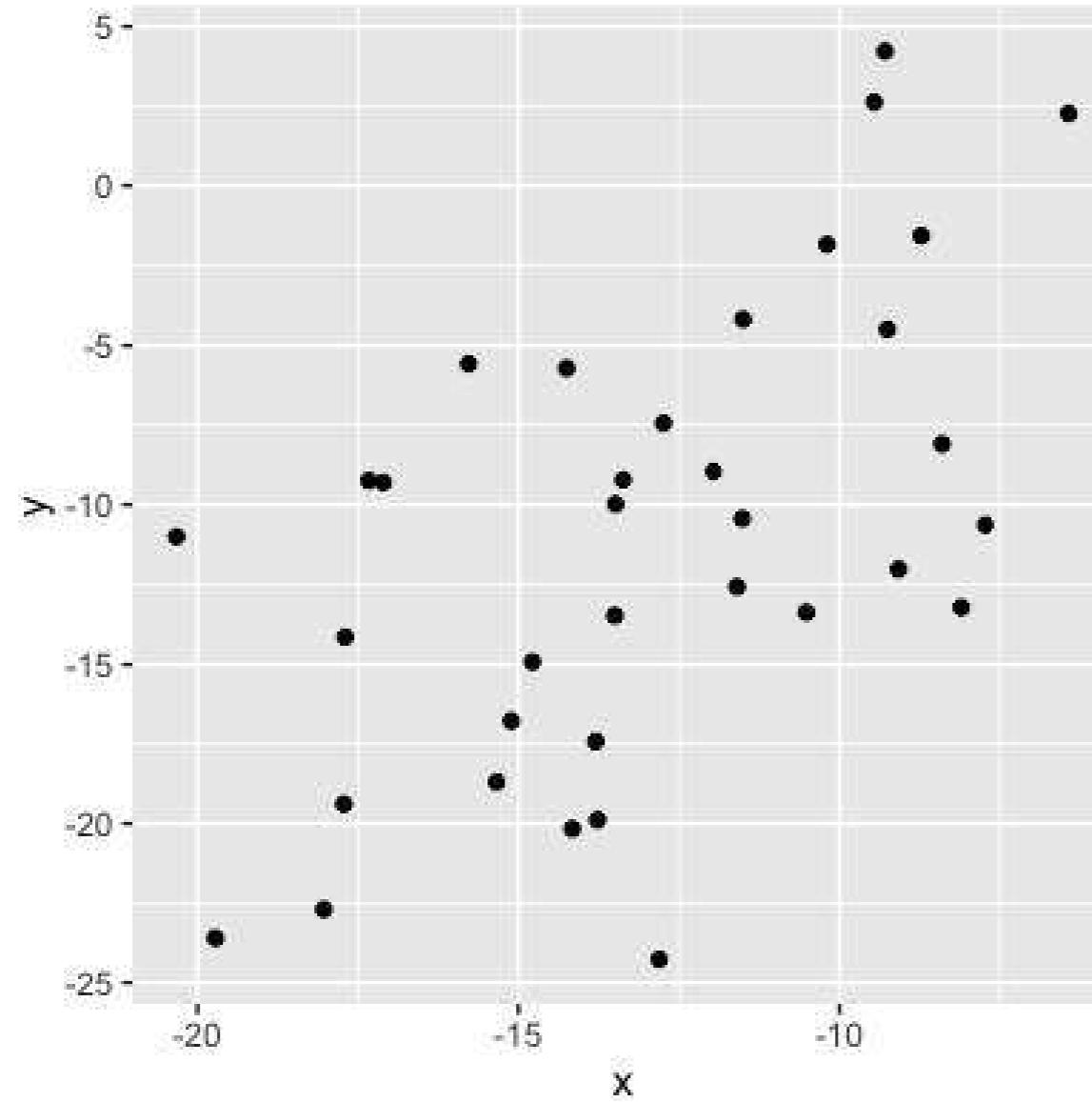


0.75 (strong relationship)



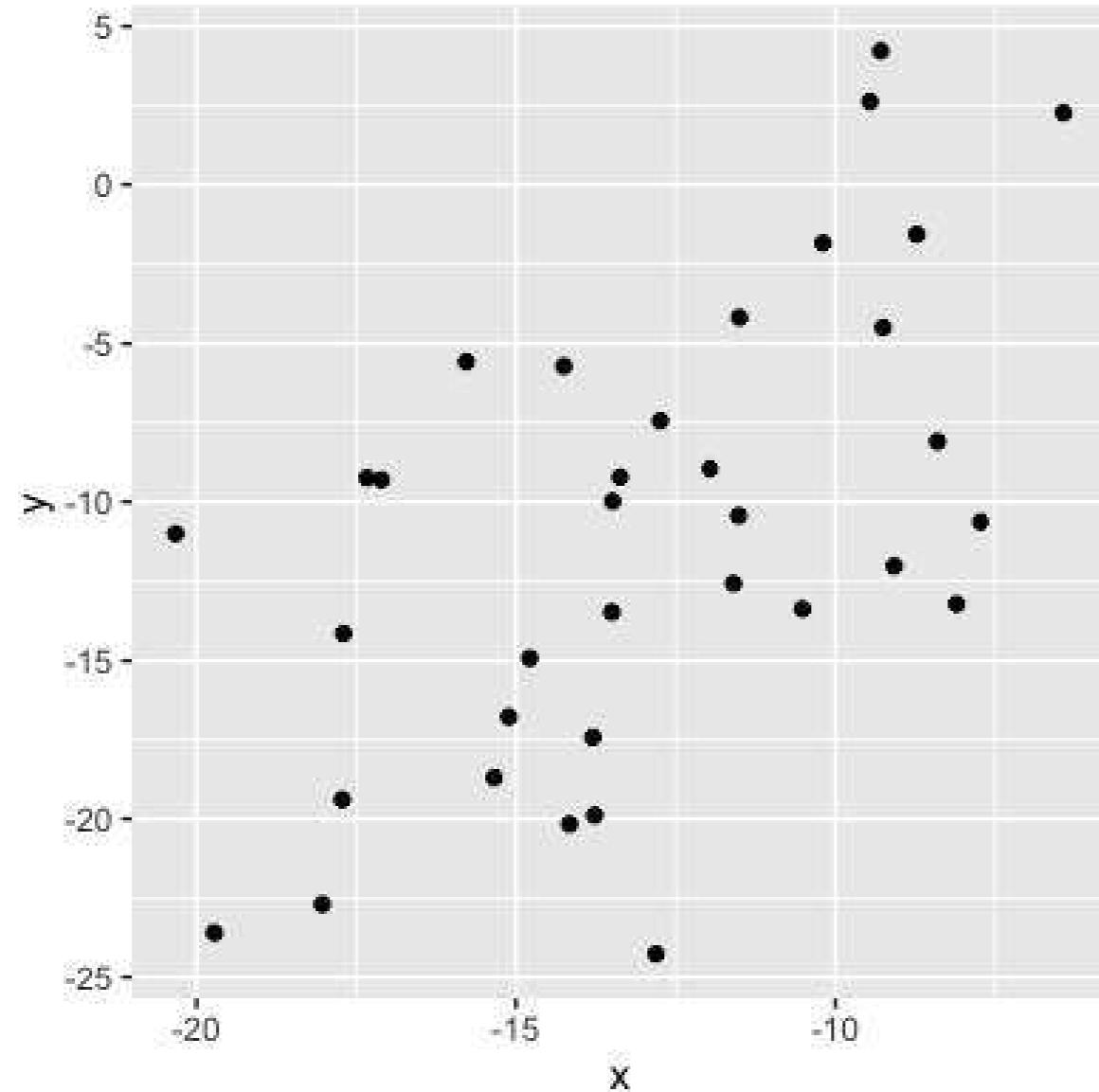
Magnitude = strength of relationship

0.56 (moderate relationship)

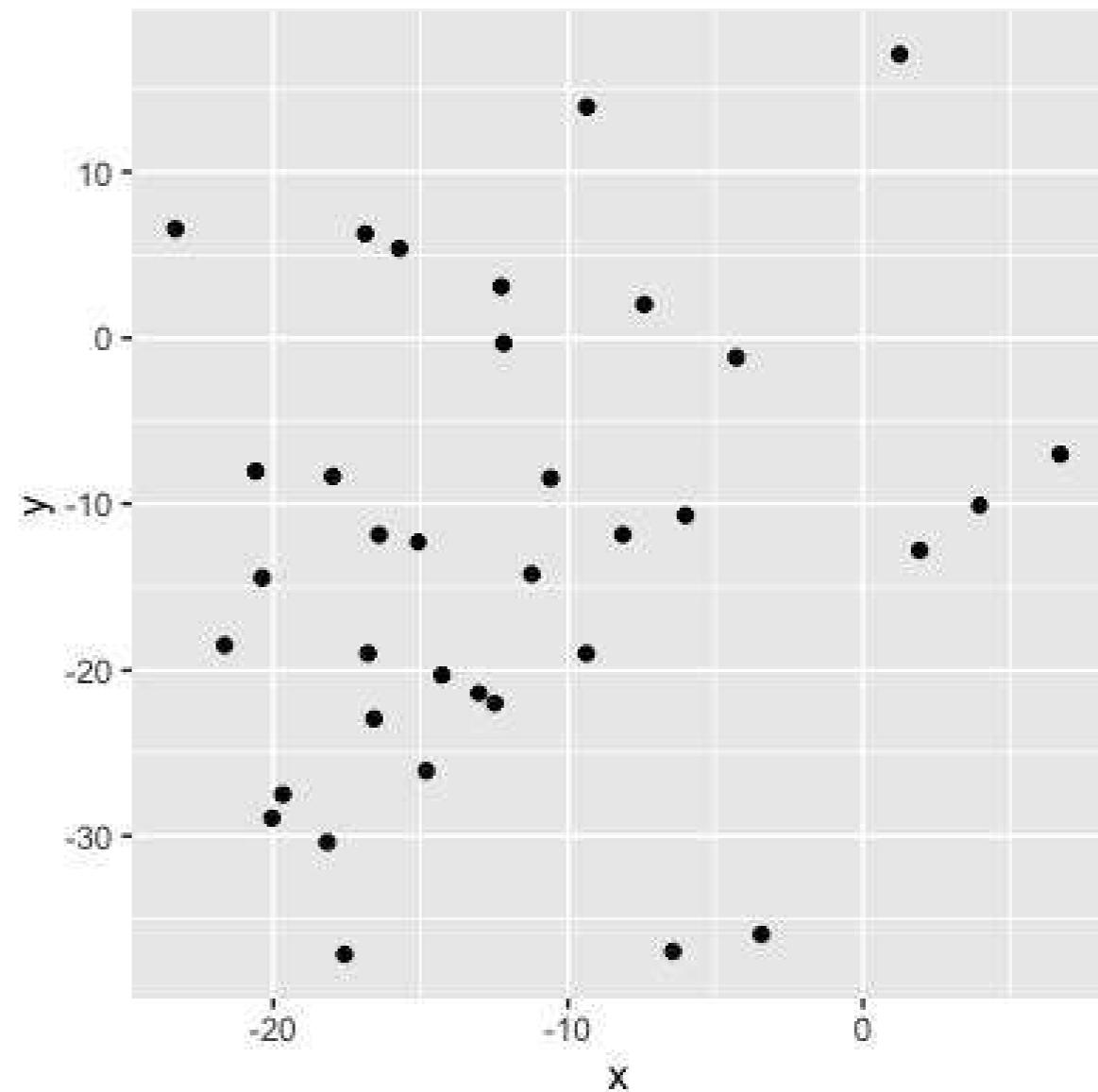


Magnitude = strength of relationship

0.56 (moderate relationship)

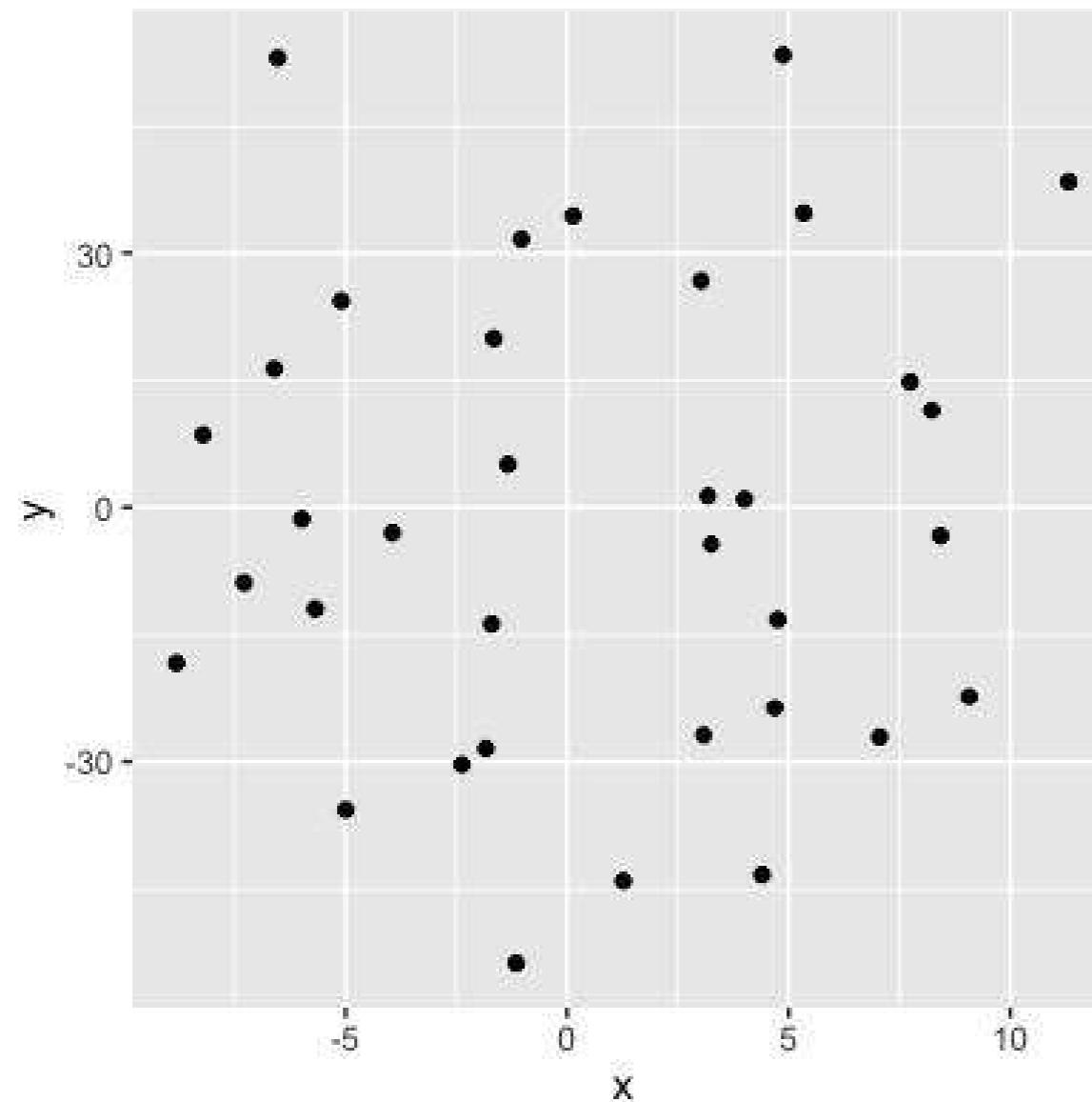


0.21 (weak relationship)



Magnitude = strength of relationship

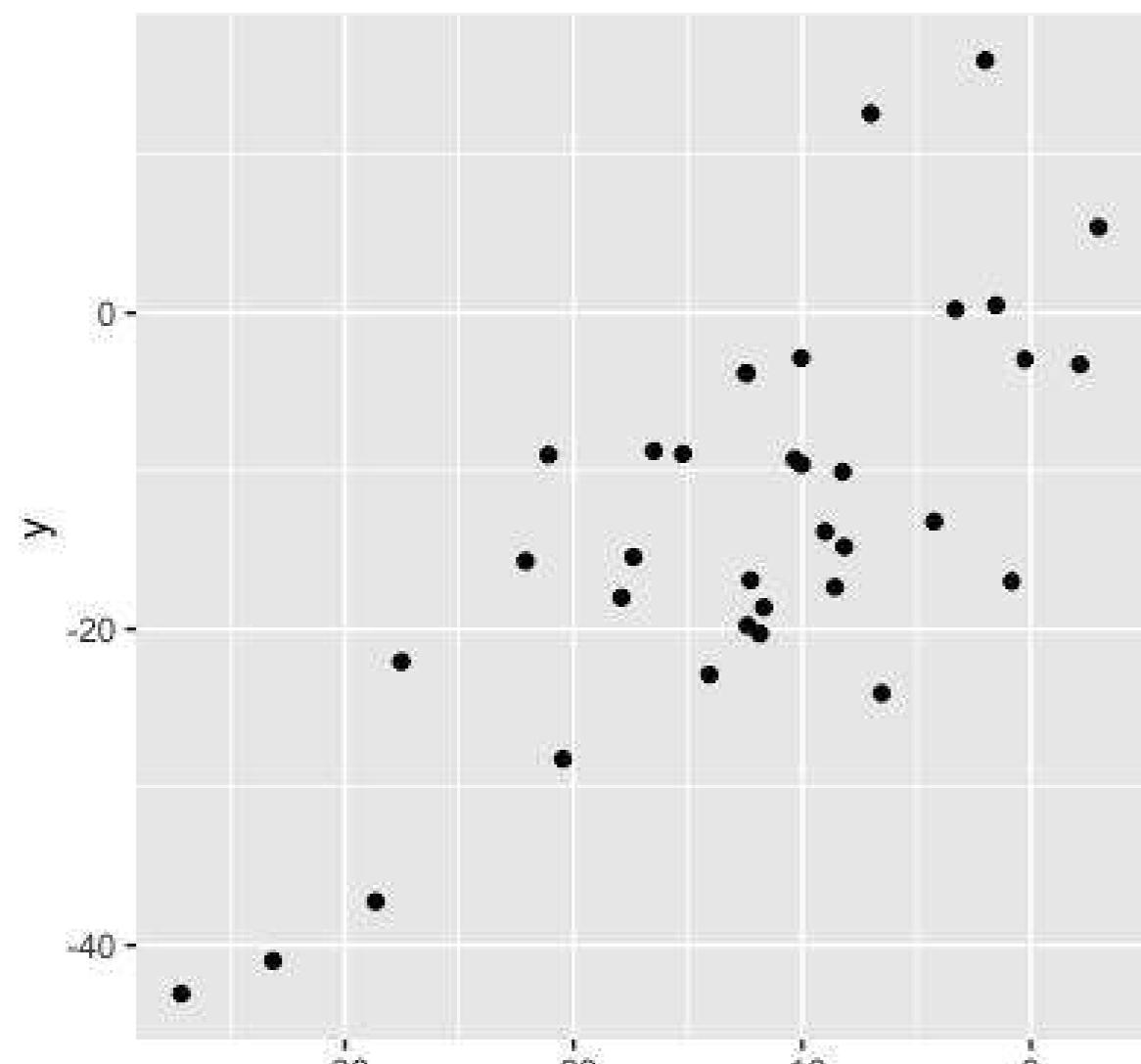
0.04 (no relationship)



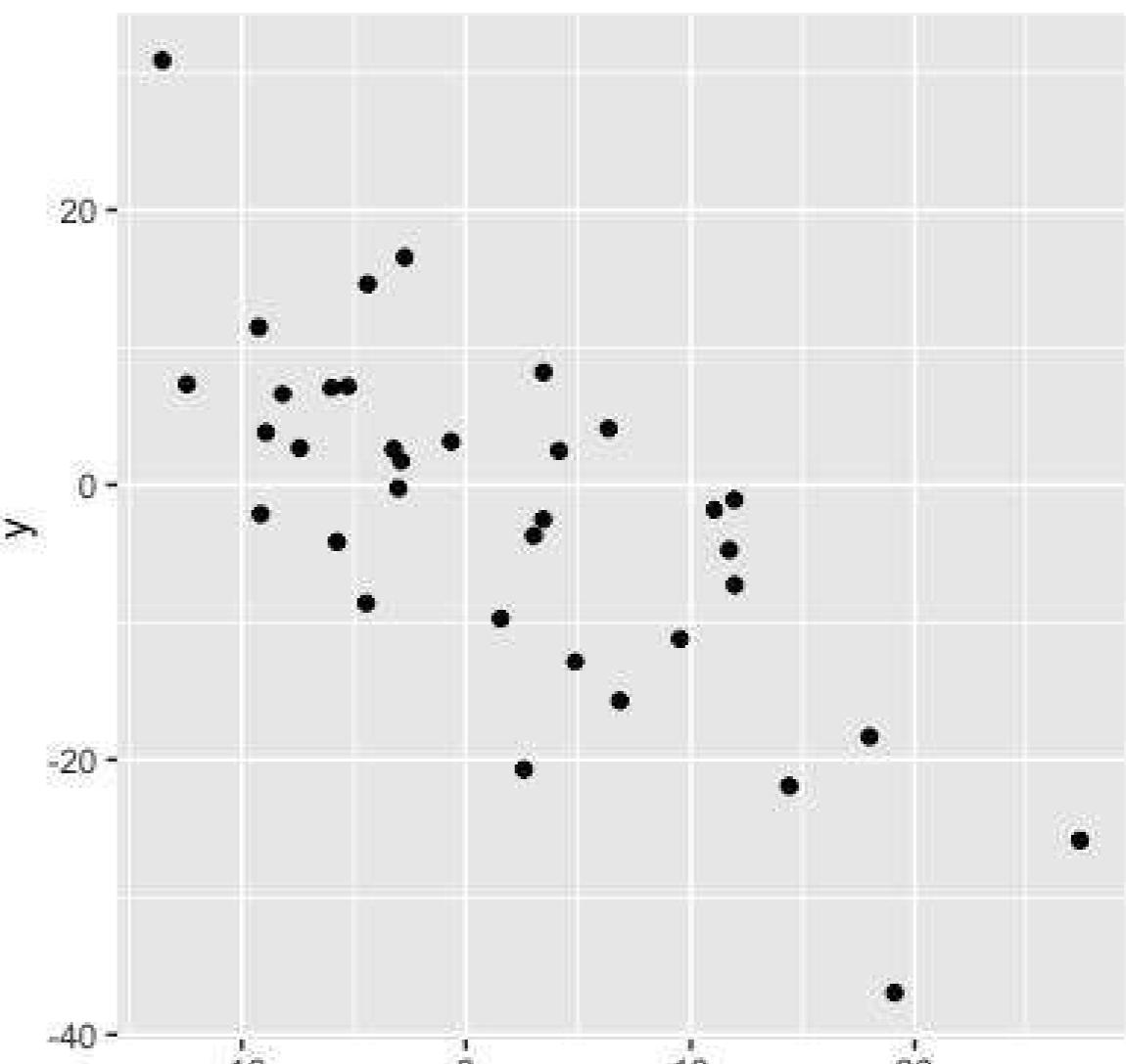
- Knowing the value of x doesn't tell us anything about y

Sign = direction

0.75: as x increases, y increases

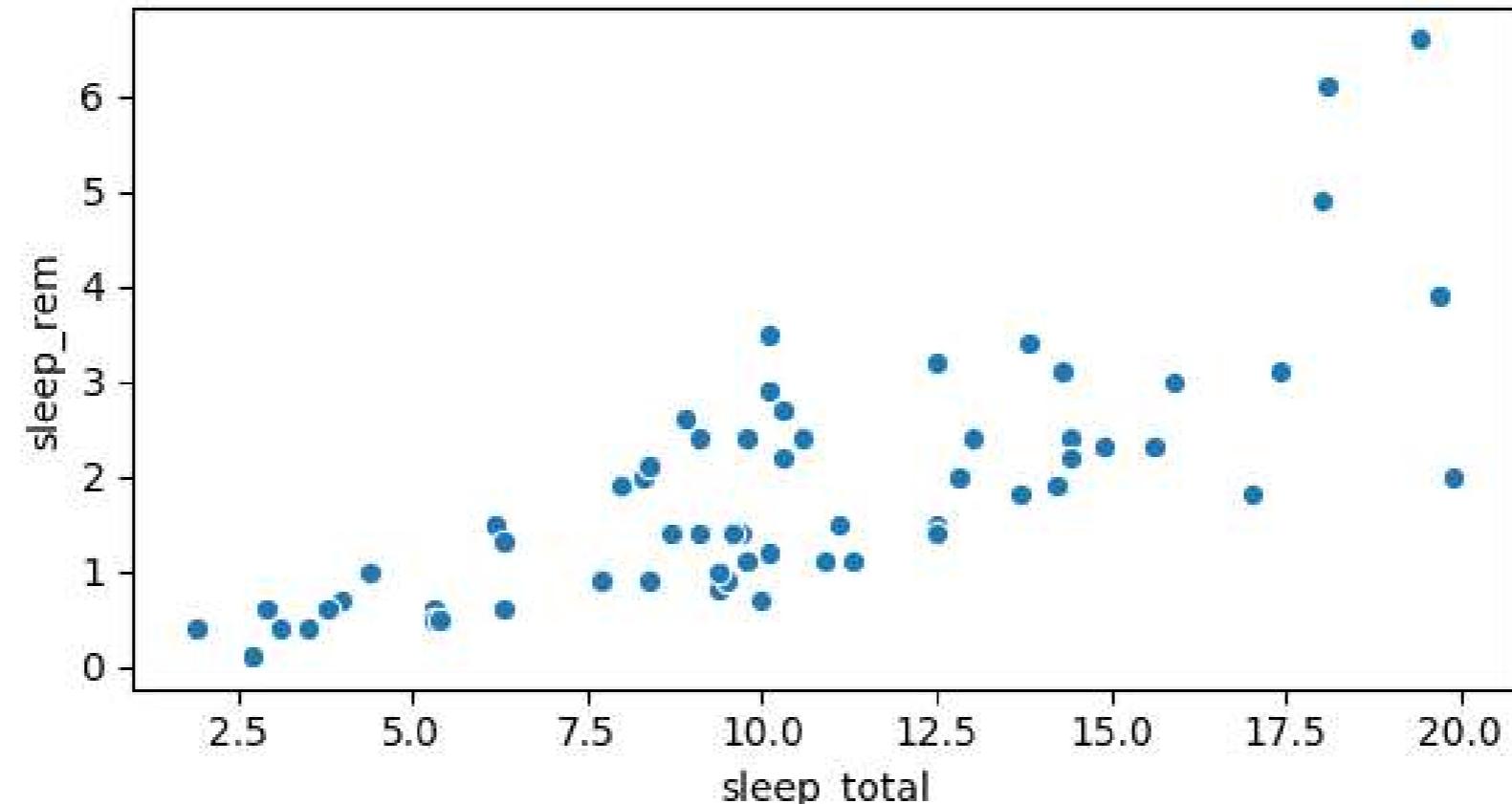


-0.75: as x increases, y decreases



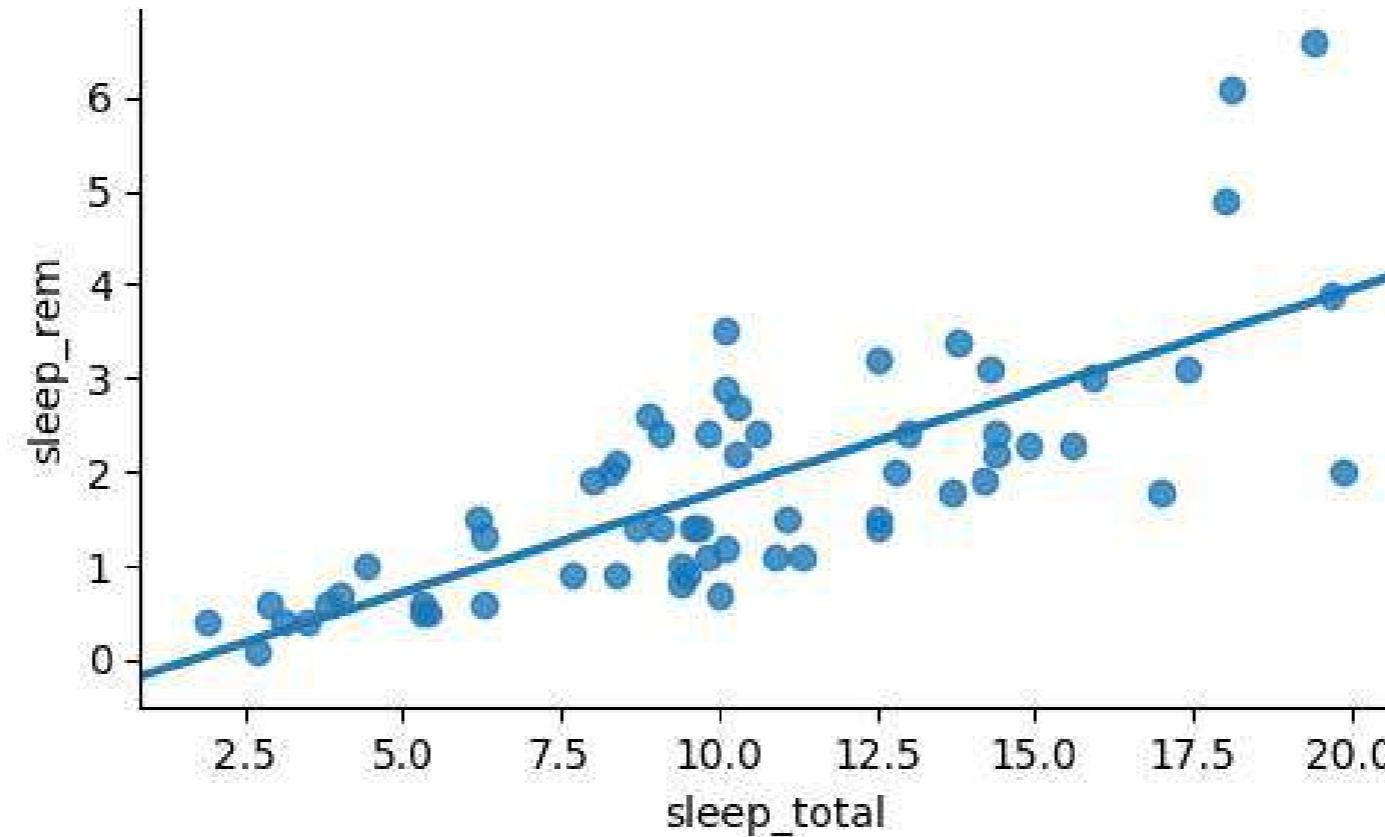
Visualizing relationships

```
import seaborn as sns  
sns.scatterplot(x="sleep_total", y="sleep_rem", data=msleep)  
plt.show()
```



Adding a trendline

```
import seaborn as sns  
sns.lmplot(x="sleep_total", y="sleep_rem", data=msleep, ci=None)  
plt.show()
```



Computing correlation

```
msleep['sleep_total'].corr(msleep['sleep_rem'])
```

```
0.751755
```

```
msleep['sleep_rem'].corr(msleep['sleep_total'])
```

```
0.751755
```

Many ways to calculate correlation

- Used in this course: Pearson product-moment correlation (r)
 - Most common
 - \bar{x} = mean of x
 - σ_x = standard deviation of x

$$r = \sum_{i=1}^n \frac{(x_i - \bar{x})(y_i - \bar{y})}{\sigma_x \times \sigma_y}$$

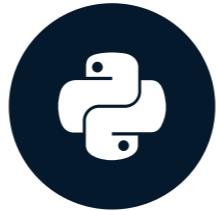
- Variations on this formula:
 - Kendall's tau
 - Spearman's rho

Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

Correlation caveats

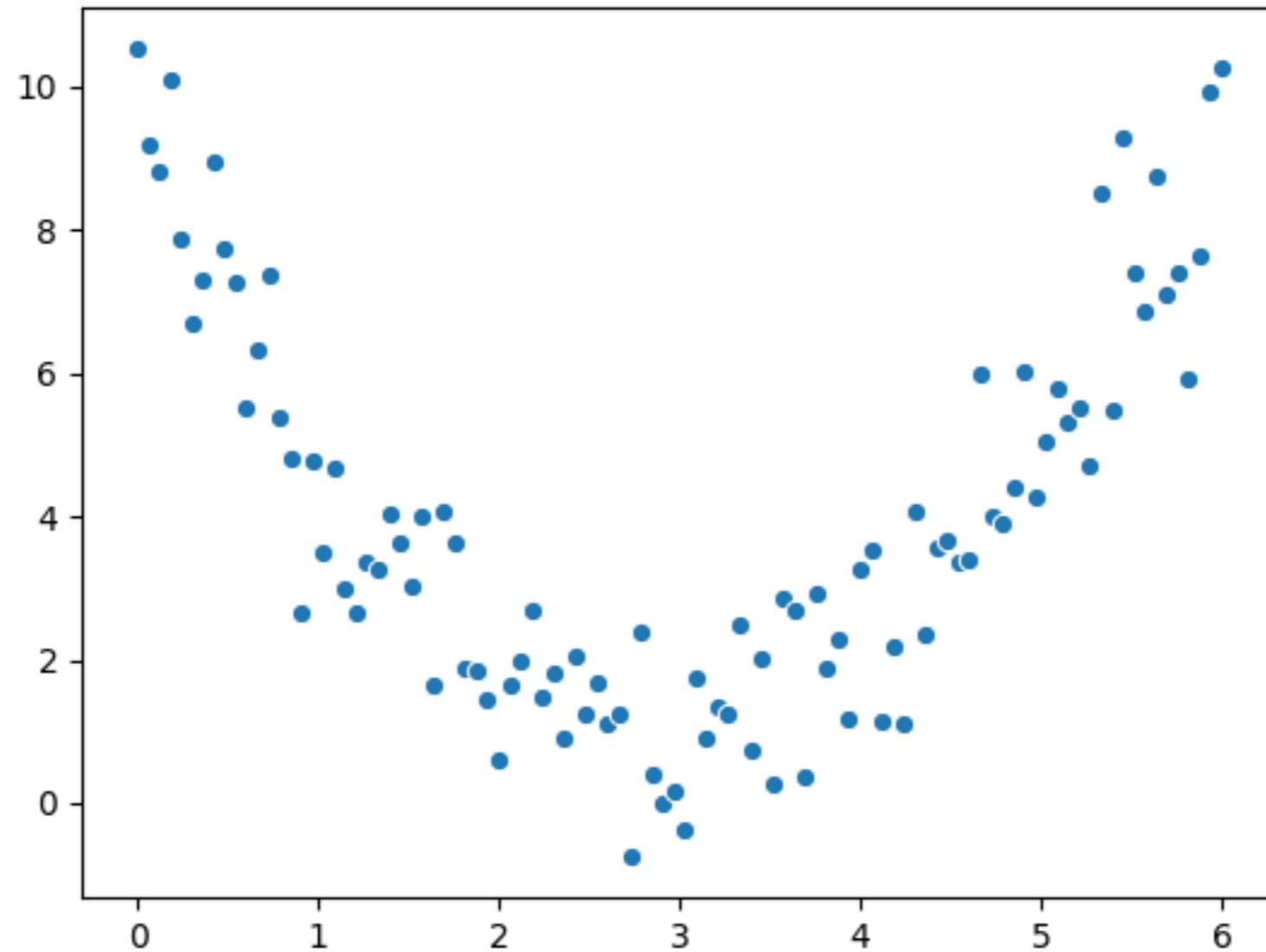
INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

Content Developer, DataCamp

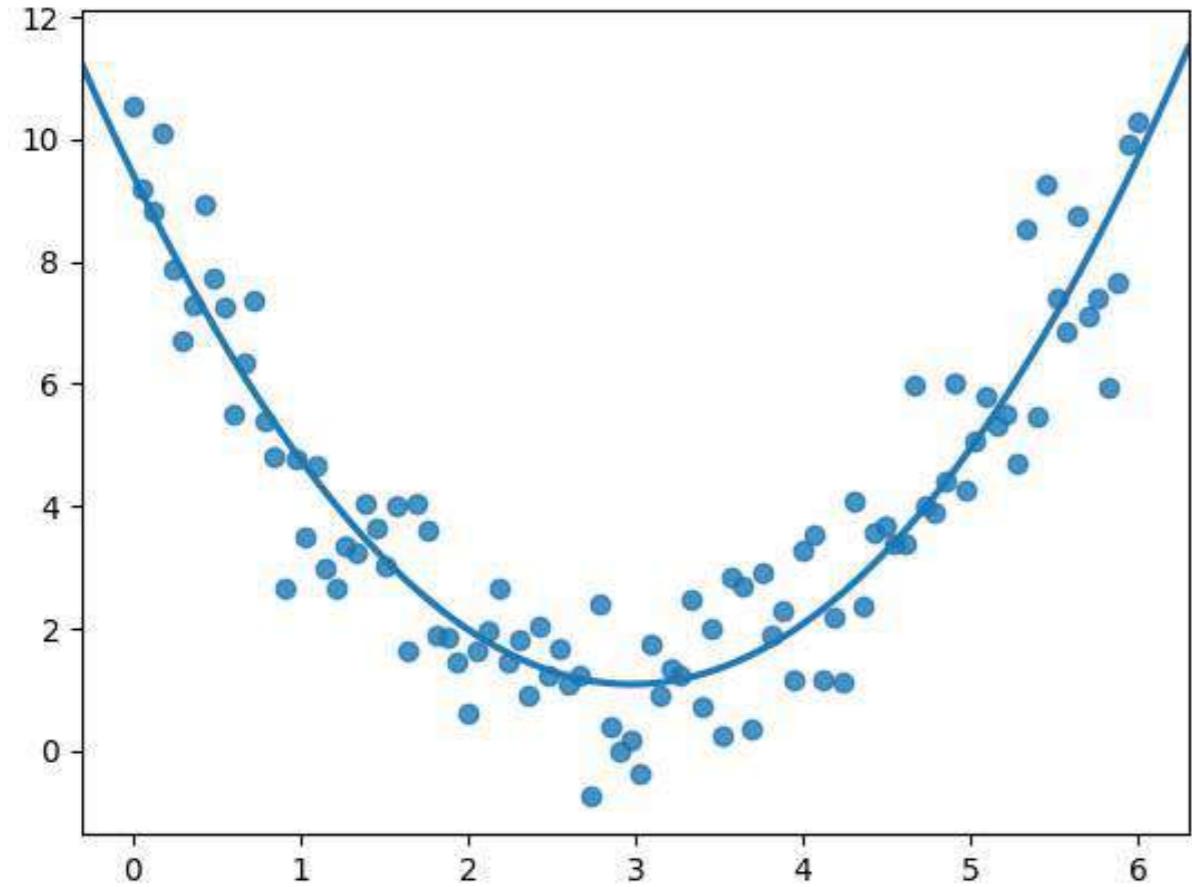
Non-linear relationships



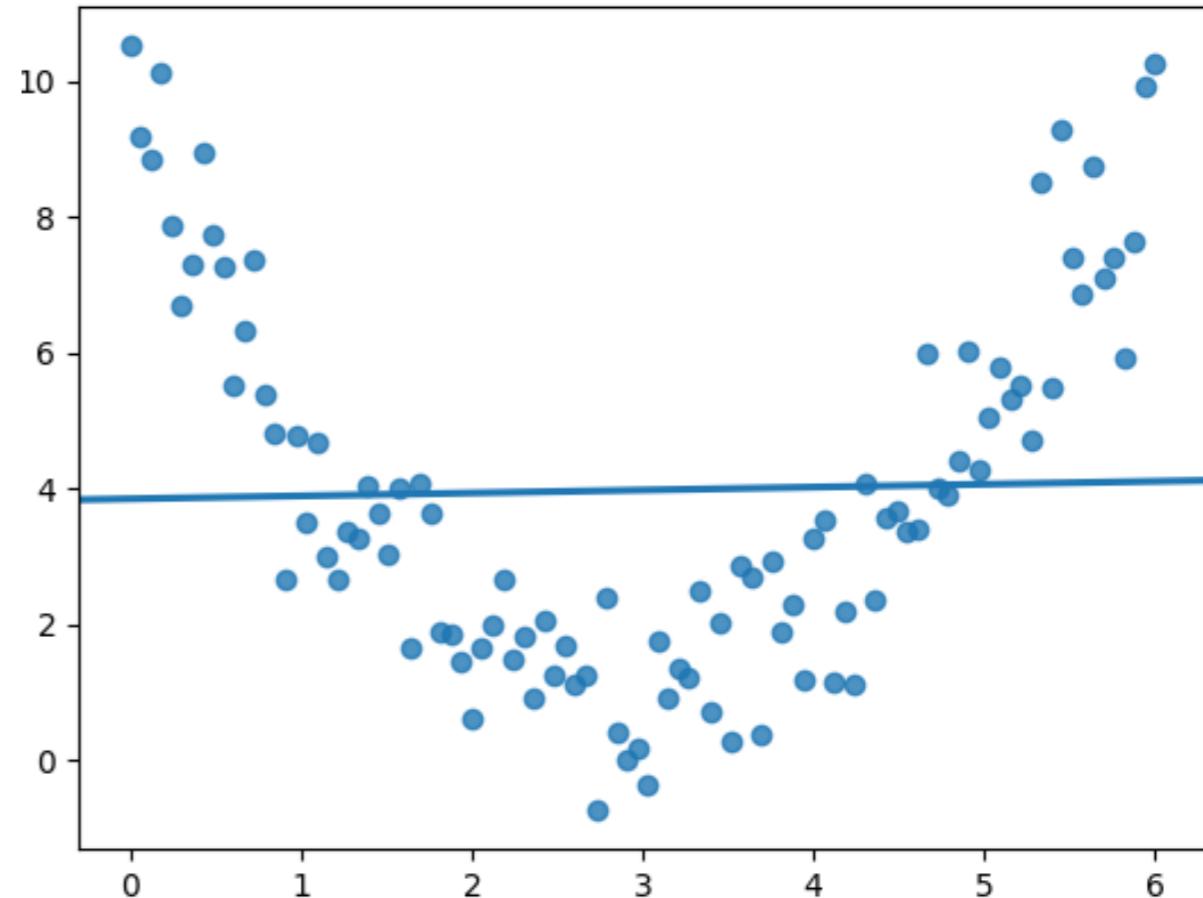
$$r = 0.18$$

Non-linear relationships

What we see:



What the correlation coefficient sees:



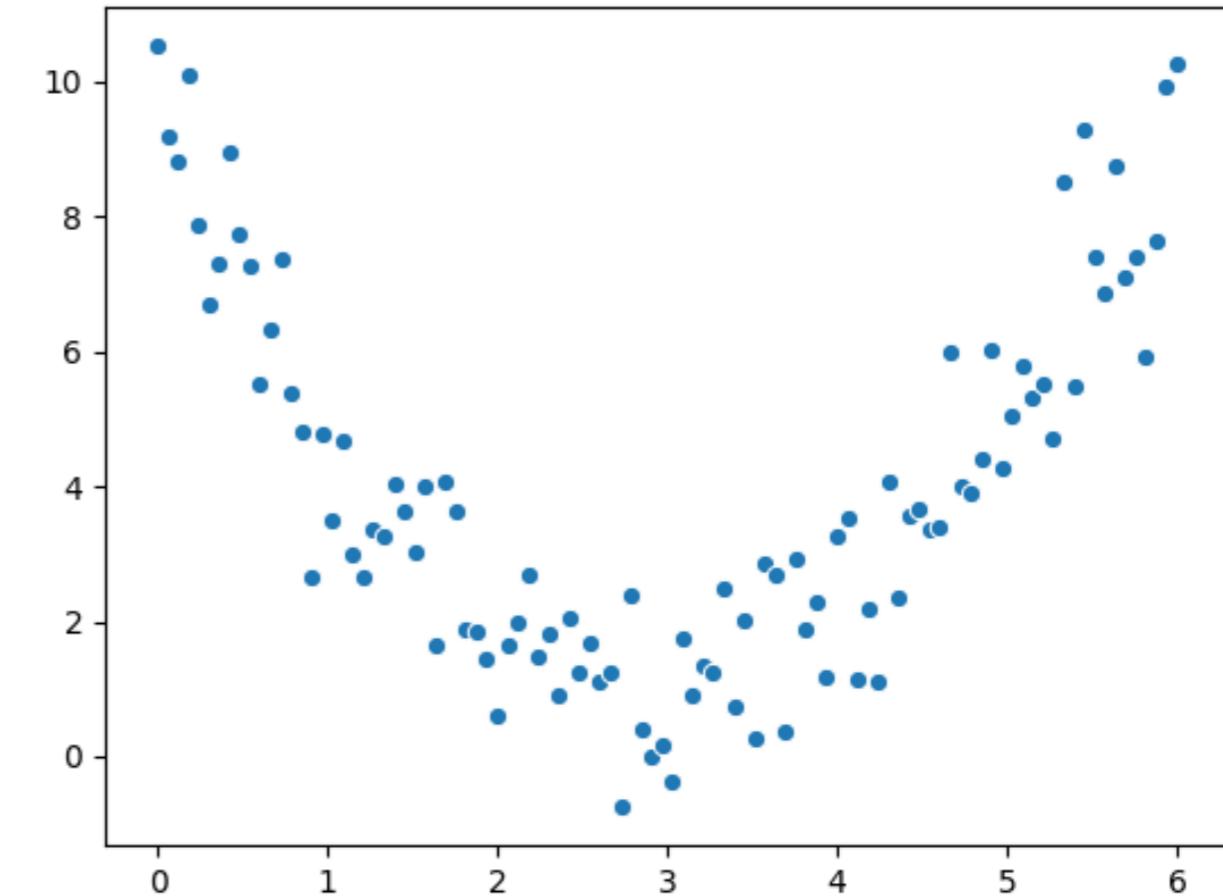
Correlation only accounts for linear relationships

Correlation shouldn't be used blindly

```
df['x'].corr(df['y'])
```

```
0.081094
```

Always visualize your data

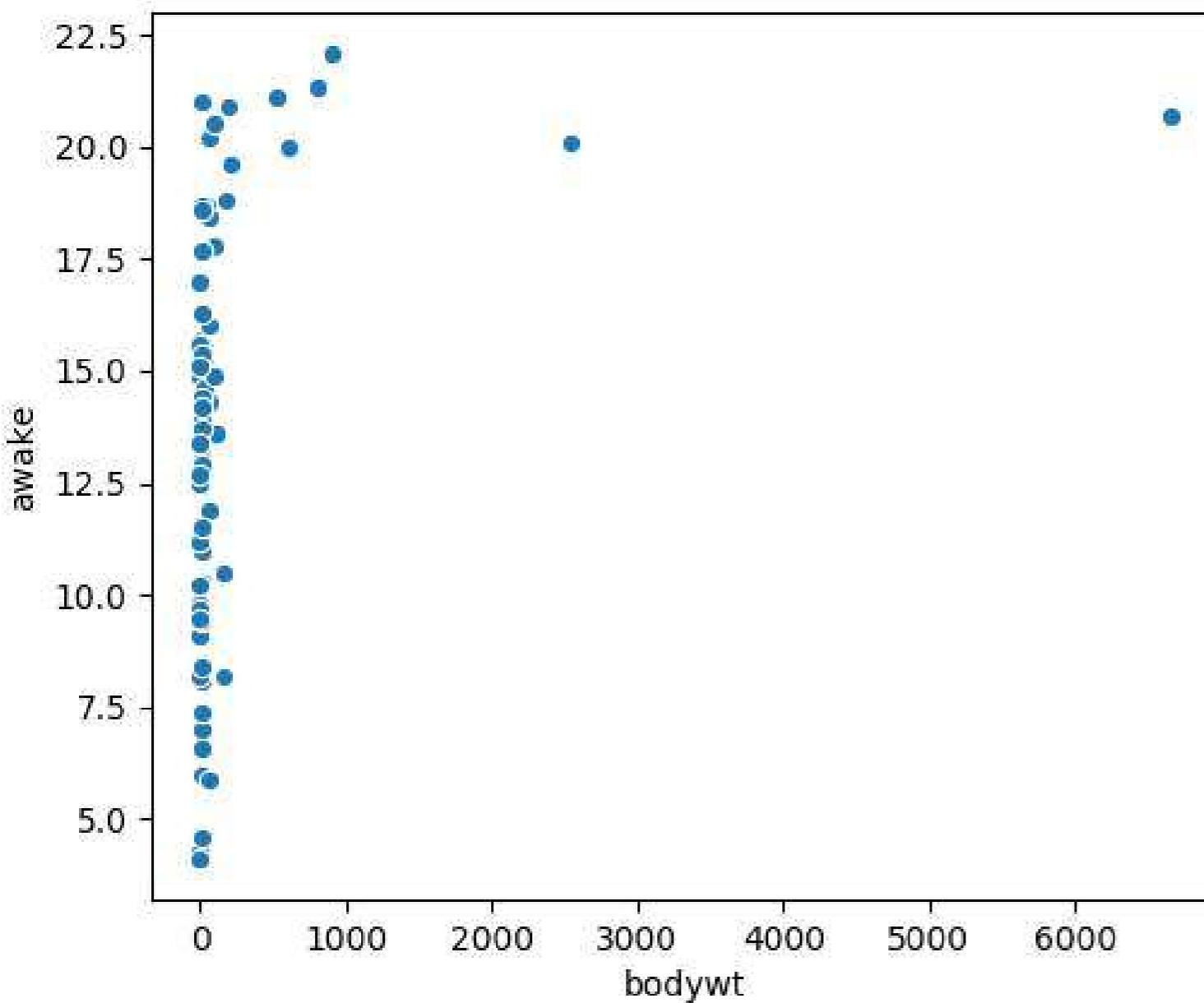


Mammal sleep data

```
print(msleep)
```

	name	genus	vore	order	...	sleep_cycle	awake	brainwt	bodywt
1	Cheetah	Acinonyx	carni	Carnivora	...	NaN	11.9	NaN	50.000
2	Owl monkey	Aotus	omni	Primates	...	NaN	7.0	0.01550	0.480
3	Mountain beaver	Apodemus	herbi	Rodentia	...	NaN	9.6	NaN	1.350
4	Greater short-ta...	Blarina	omni	Soricomorpha	...	0.133333	9.1	0.00029	0.019
5	Cow	Bos	herbi	Artiodactyla	...	0.666667	20.0	0.42300	600.000
...
79	Tree shrew	Tupaia	omni	Scandentia	...	0.233333	15.1	0.00250	0.104
80	Bottle-nosed do...	Tursiops	carni	Cetacea	...	NaN	18.8	NaN	173.330
81	Genet	Genetta	carni	Carnivora	...	NaN	17.7	0.01750	2.000
82	Arctic fox	Vulpes	carni	Carnivora	...	NaN	11.5	0.04450	3.380
83	Red fox	Vulpes	carni	Carnivora	...	0.350000	14.2	0.05040	4.230

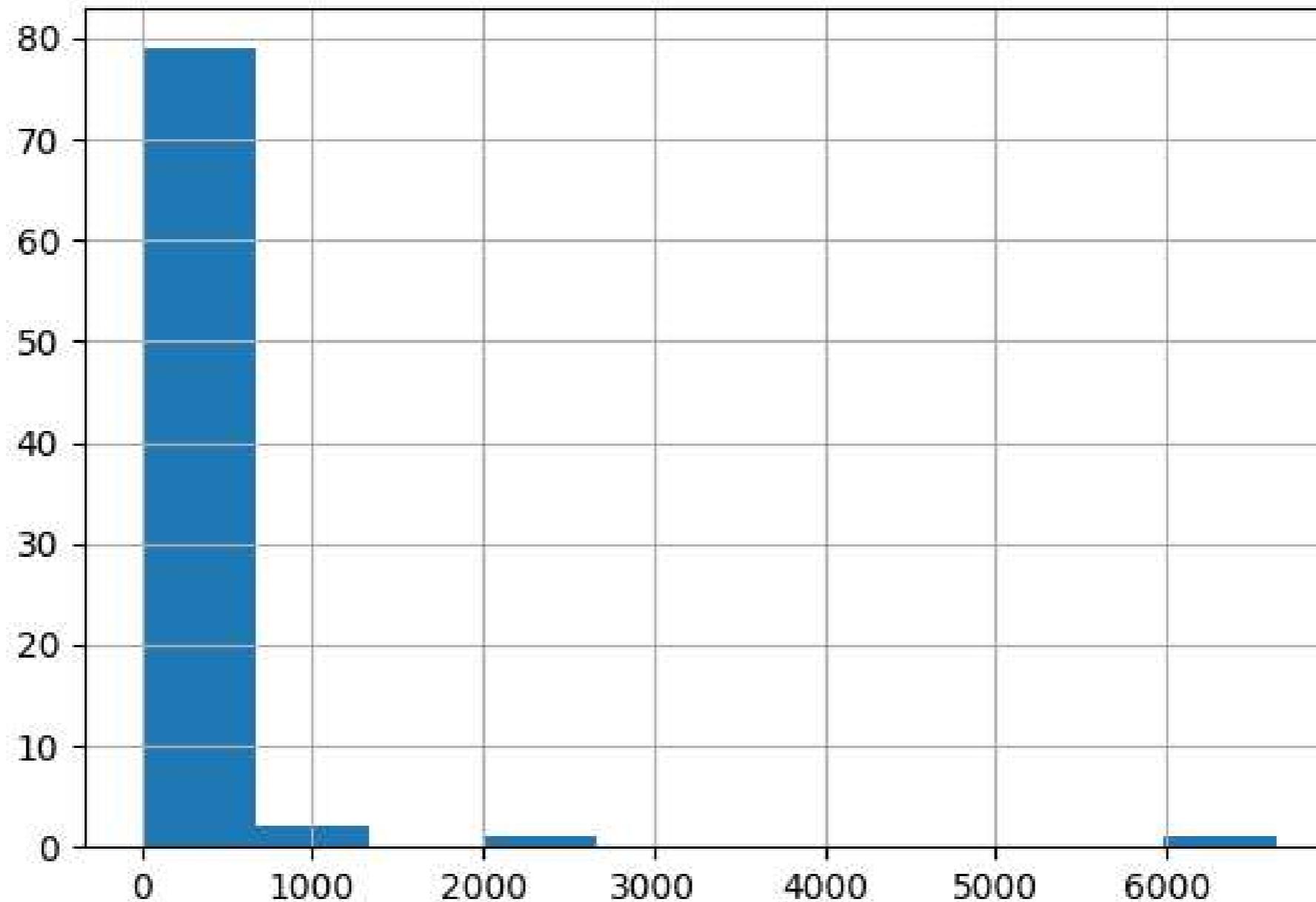
Body weight vs. awake time



```
msleep['bodywt'].corr(msleep['awake'])
```

```
0.3119801
```

Distribution of body weight

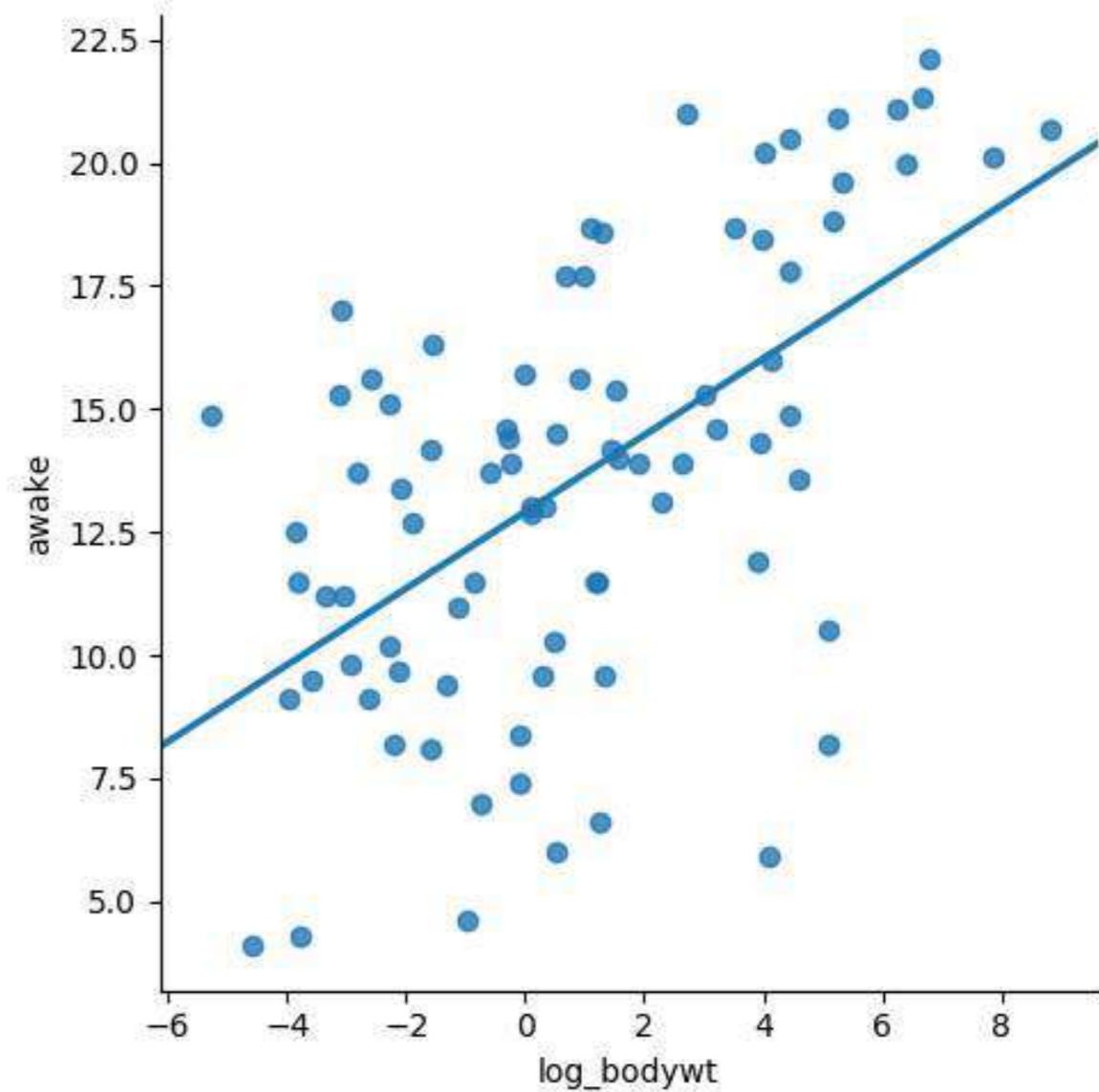


Log transformation

```
msleep['log_bodywt'] = np.log(msleep['bodywt'])  
  
sns.lmplot(x='log_bodywt',  
            y='awake',  
            data=msleep,  
            ci=None)  
plt.show()
```

```
msleep['log_bodywt'].corr(msleep['awake'])
```

0.5687943



Other transformations

- Log transformation (`log(x)`)
- Square root transformation (`sqrt(x)`)
- Reciprocal transformation (`1 / x`)
- Combinations of these, e.g.:
 - `log(x)` and `log(y)`
 - `sqrt(x)` and `1 / y`

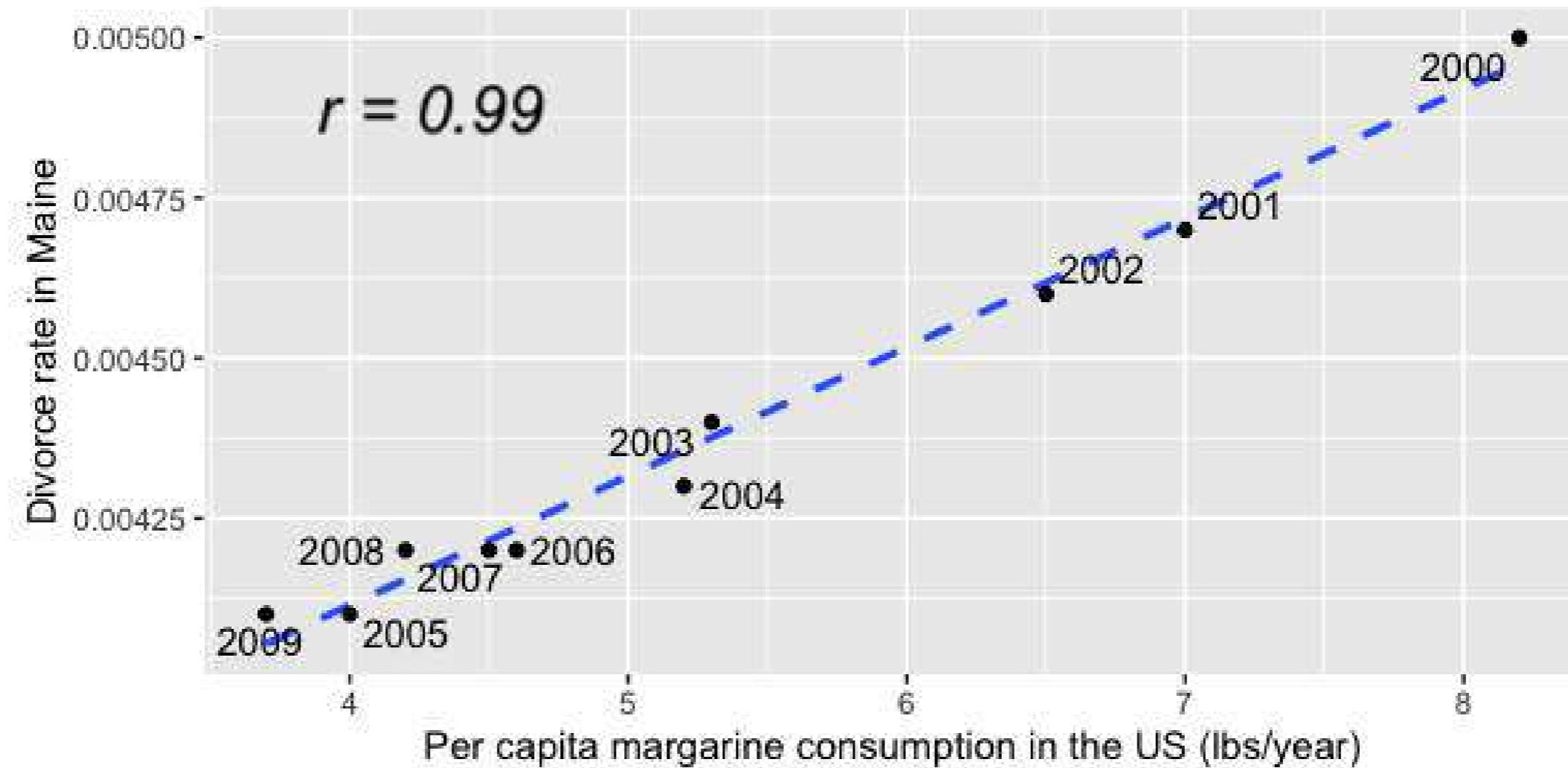
Why use a transformation?

- Certain statistical methods rely on variables having a linear relationship
 - Correlation coefficient
 - Linear regression

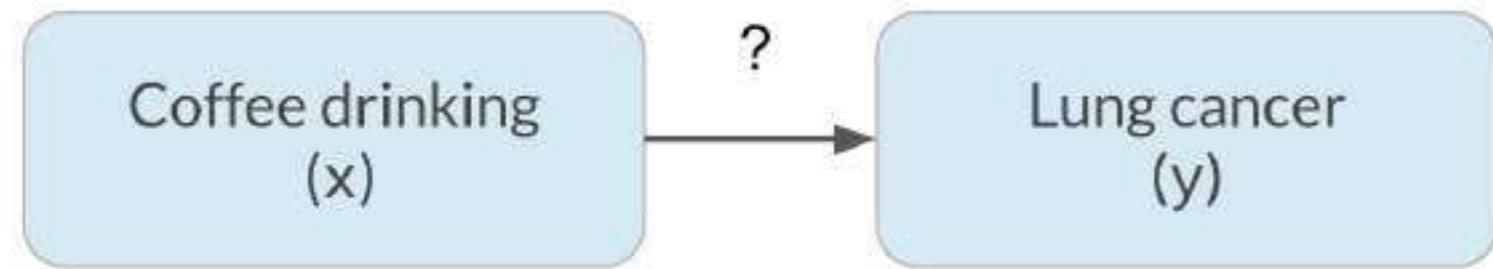
[Introduction to Linear Modeling in Python](#)

Correlation does not imply causation

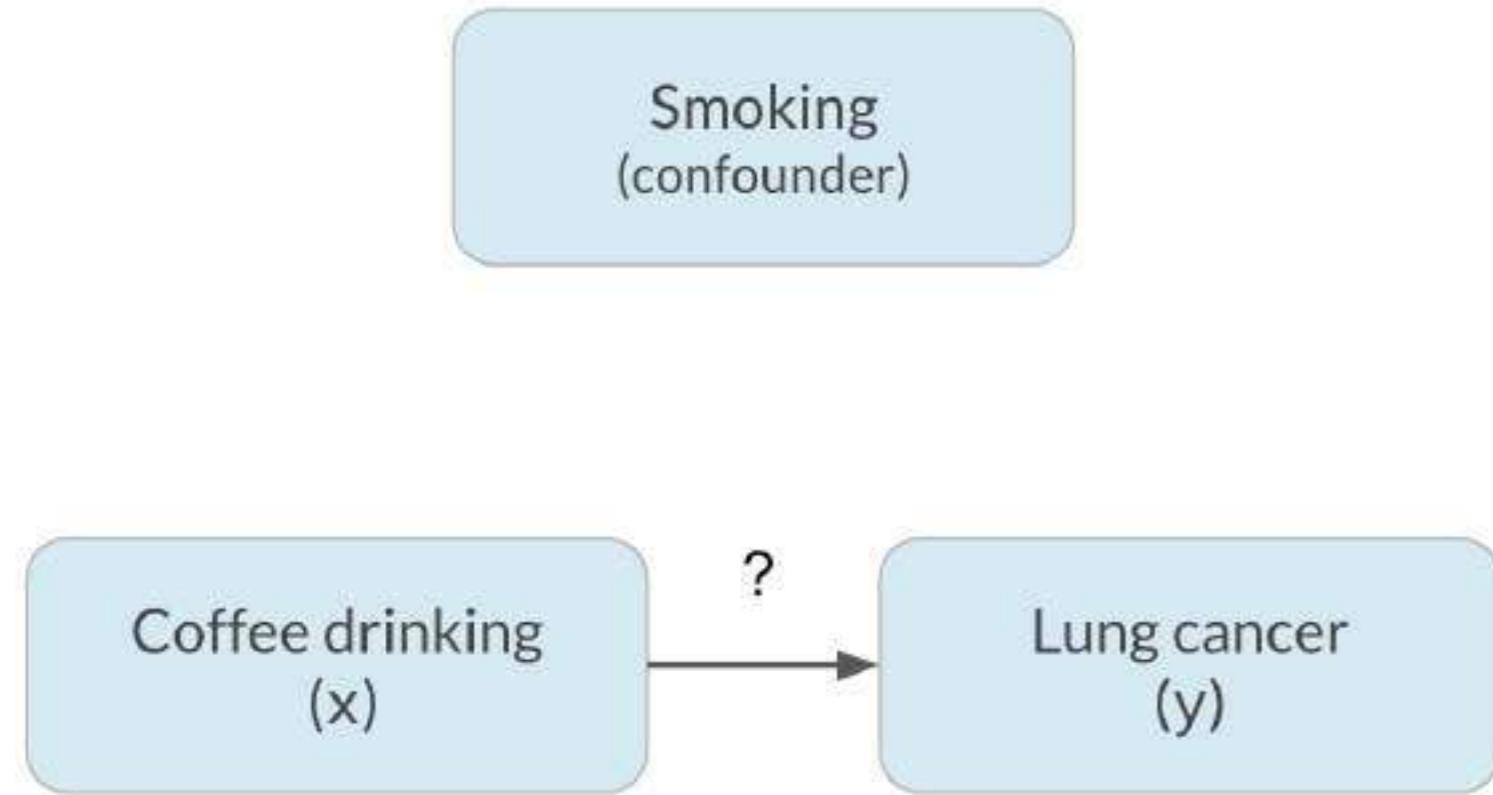
x is correlated with y does not mean x causes y



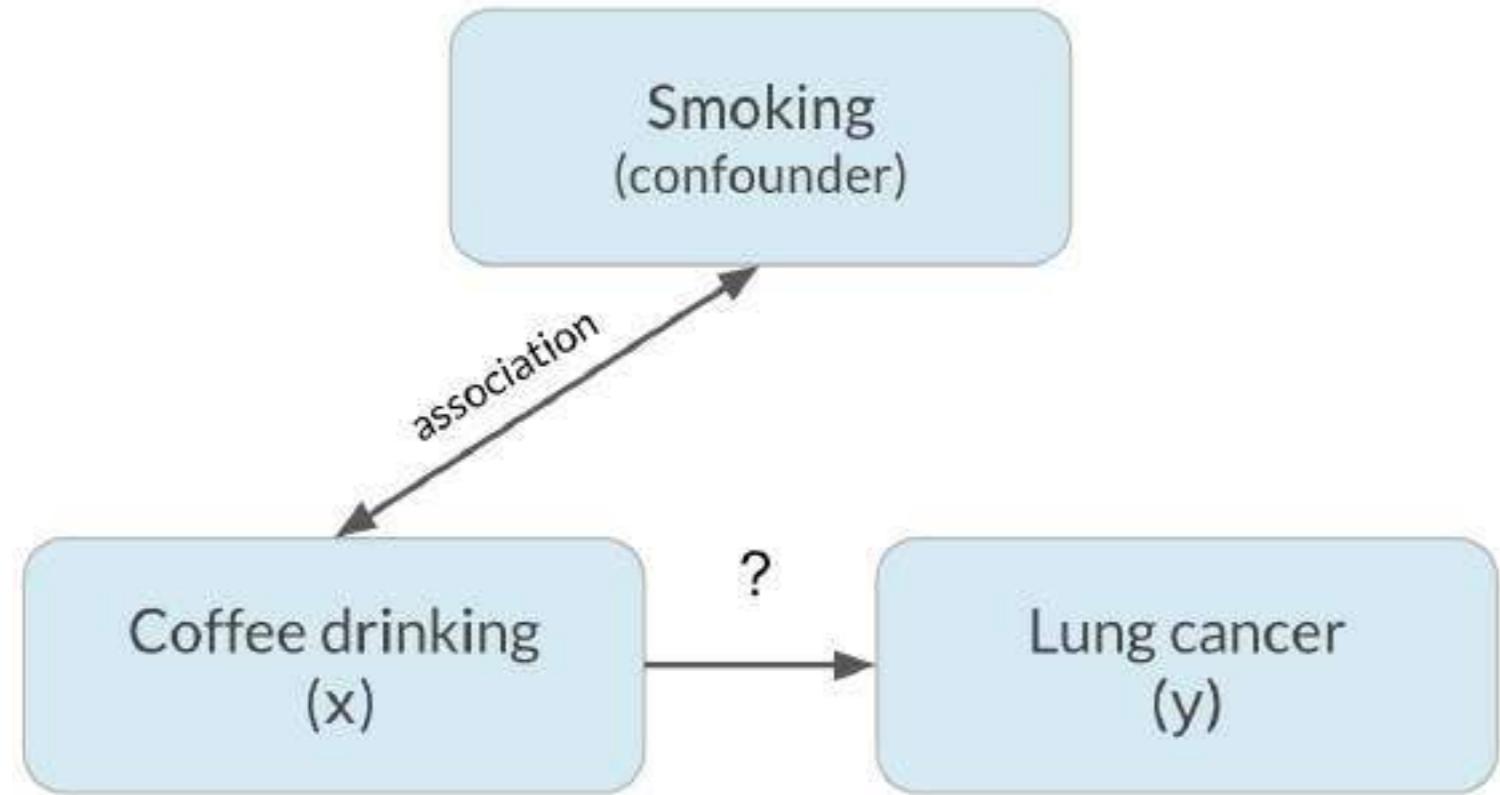
Confounding



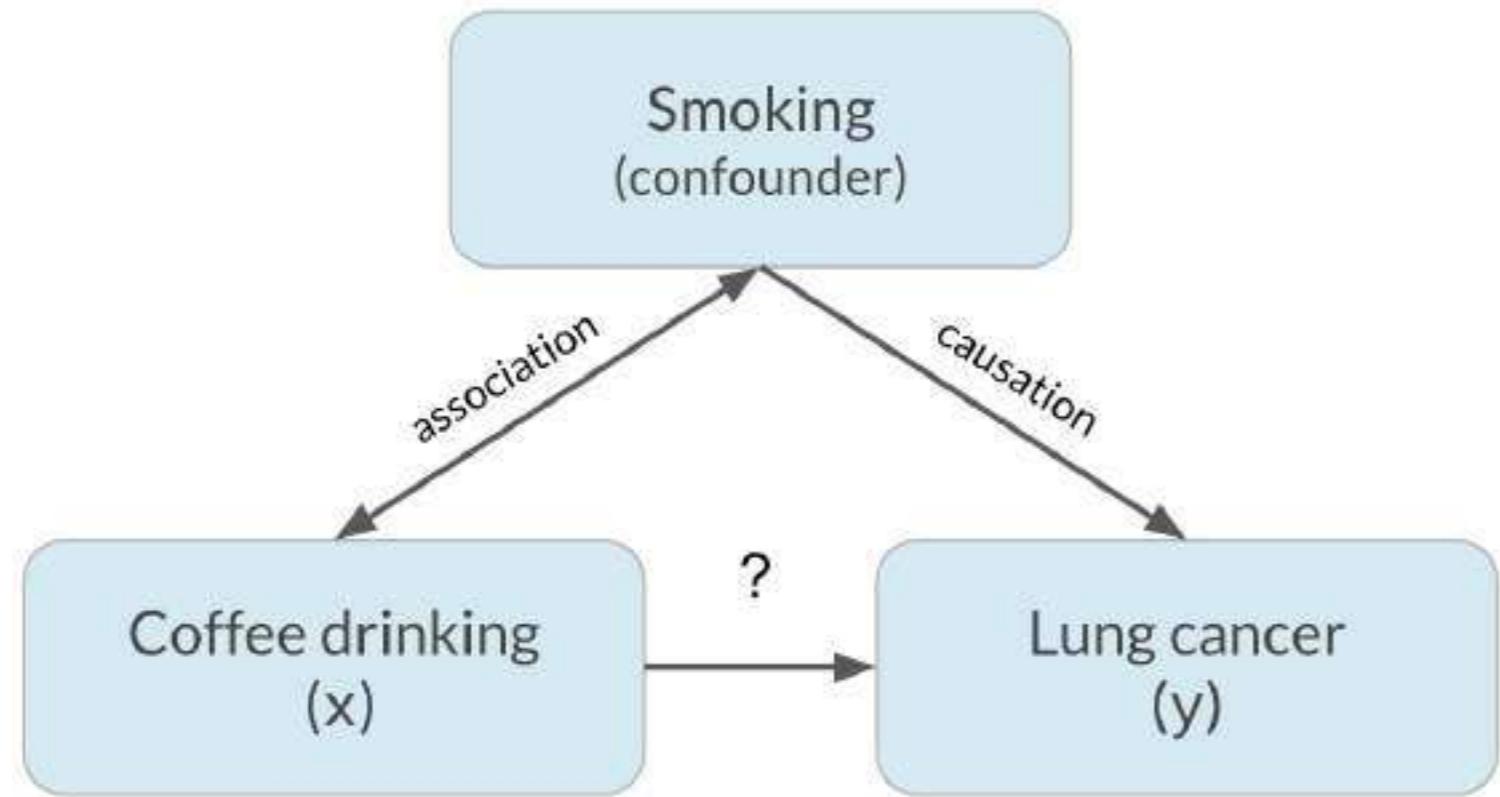
Confounding



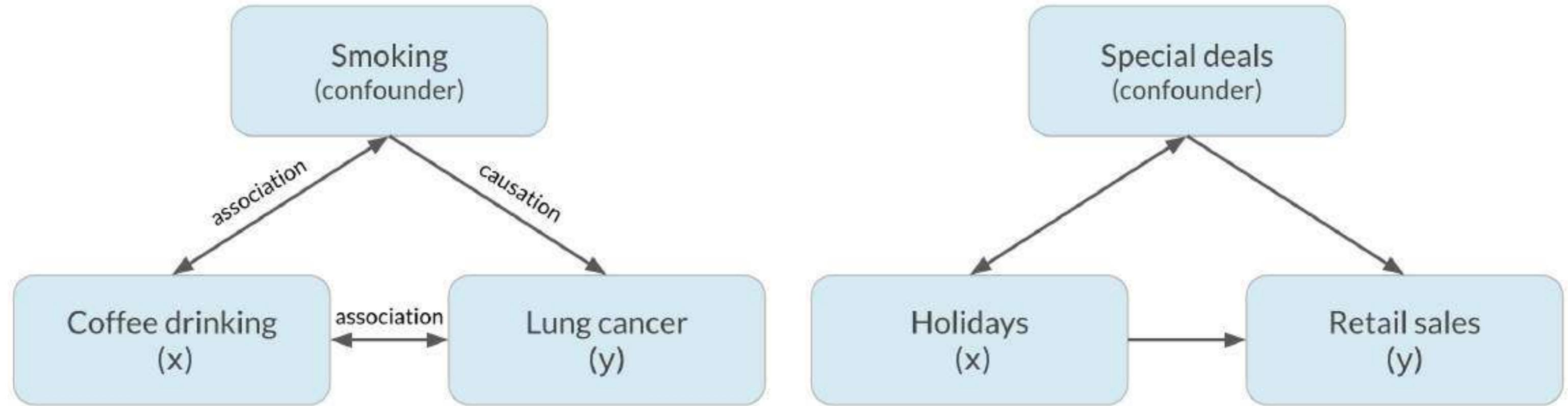
Confounding



Confounding



Confounding

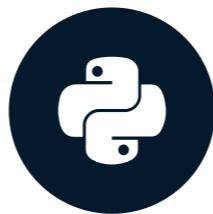


Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

Design of experiments

INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

Content Developer, DataCamp

Vocabulary

Experiment aims to answer: *What is the effect of the treatment on the response?*

- Treatment: explanatory/independent variable
- Response: response/dependent variable

E.g.: *What is the effect of an advertisement on the number of products purchased?*

- Treatment: advertisement
- Response: number of products purchased

Controlled experiments

- Participants are assigned by researchers to either treatment group or control group
 - Treatment group sees advertisement
 - Control group does not
- Groups should be comparable so that causation can be inferred
- If groups are not comparable, this could lead to confounding (bias)
 - Treatment group average age: 25
 - Control group average age: 50
 - Age is a potential confounder

The gold standard of experiments will use...

- Randomized controlled trial
 - Participants are assigned to treatment/control *randomly*, not based on any other characteristics
 - Choosing randomly helps ensure that groups are comparable
- Placebo
 - Resembles treatment, but has no effect
 - Participants will not know which group they're in
 - In clinical trials, a sugar pill ensures that the effect of the drug is actually due to the drug itself and not the idea of receiving the drug

The gold standard of experiments will use...

- Double-blind trial
 - Person administering the treatment/running the study doesn't know whether the treatment is real or a placebo
 - Prevents bias in the response and/or analysis of results

Fewer opportunities for bias = more reliable conclusion about causation

Observational studies

- Participants are not assigned randomly to groups
 - Participants assign themselves, usually based on pre-existing characteristics
- Many research questions are not conducive to a controlled experiment
 - You can't force someone to smoke or have a disease
 - You can't make someone have certain past behavior
- Establish association, not causation
 - Effects can be confounded by factors that got certain people into the control or treatment group
 - There are ways to control for confounders to get more reliable conclusions about association

Longitudinal vs. cross-sectional studies

Longitudinal study

- Participants are followed over a period of time to examine effect of treatment on response
- Effect of age on height is not confounded by generation
- More expensive, results take longer

Cross-sectional study

- Data on participants is collected from a single snapshot in time
- Effect of age on height is confounded by generation
- Cheaper, faster, more convenient

Let's practice!

INTRODUCTION TO STATISTICS IN PYTHON

Congratulations!

INTRODUCTION TO STATISTICS IN PYTHON



Maggie Matsui

Content Developer, DataCamp

Overview

Chapter 1

- What is statistics?
- Measures of center
- Measures of spread

Chapter 3

- Normal distribution
- Central limit theorem
- Poisson distribution

Chapter 2

- Measuring chance
- Probability distributions
- Binomial distribution

Chapter 4

- Correlation
- Controlled experiments
- Observational studies

Build on your skills

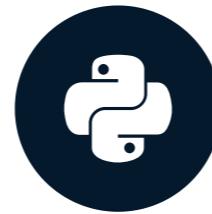
- [Introduction to Linear Modeling in Python](#)

Congratulations!

INTRODUCTION TO STATISTICS IN PYTHON

How to use dates & times with pandas

MANIPULATING TIME SERIES DATA IN PYTHON



Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence

Date & time series functionality

- At the root: data types for date & time information
 - Objects for points in time and periods
 - Attributes & methods reflect time-related details
- Sequences of dates & periods:
 - Series or DataFrame columns
 - Index: convert object into Time Series
- Many Series/DataFrame methods rely on time information in the index to provide time-series functionality

Basic building block: pd.Timestamp

```
import pandas as pd # assumed imported going forward
from datetime import datetime # To manually create dates
time_stamp = pd.Timestamp(datetime(2017, 1, 1))
pd.Timestamp('2017-01-01') == time_stamp
```

```
True # Understands dates as strings
```

```
time_stamp # type: pandas.tslib.Timestamp
```

```
Timestamp('2017-01-01 00:00:00')
```

Basic building block: pd.Timestamp

- Timestamp object has many attributes to store time-specific information

```
time_stamp.year
```

```
2017
```

```
time_stamp.day_name()
```

```
'Sunday'
```

More building blocks: pd.Period & freq

```
period = pd.Period('2017-01')  
period # default: month-end
```

```
Period('2017-01', 'M')
```

```
period.asfreq('D') # convert to daily
```

```
Period('2017-01-31', 'D')
```

```
period.to_timestamp().to_period('M')
```

```
Period('2017-01', 'M')
```

- Period object has freq attribute to store frequency info
- Convert pd.Period() to pd.Timestamp() and back

More building blocks: pd.Period & freq

```
period + 2
```

- Frequency info enables basic date arithmetic

```
Period('2017-03', 'M')
```

```
pd.Timestamp('2017-01-31', 'M') + 1
```

```
Timestamp('2017-02-28 00:00:00', freq='M')
```

Sequences of dates & times

- `pd.date_range` : `start` , `end` , `periods` , `freq`

```
index = pd.date_range(start='2017-1-1', periods=12, freq='M')
```

```
index
```

```
DatetimeIndex(['2017-01-31', '2017-02-28', '2017-03-31', ...,
                 '2017-09-30', '2017-10-31', '2017-11-30', '2017-12-31'],
                dtype='datetime64[ns]', freq='M')
```

- `pd.DateTimeIndex` : sequence of `Timestamp` objects with frequency info

Sequences of dates & times

```
index[0]
```

```
Timestamp('2017-01-31 00:00:00', freq='M')
```

```
index.to_period()
```

```
PeriodIndex(['2017-01', '2017-02', '2017-03', '2017-04', ...,
             '2017-11', '2017-12'], dtype='period[M]', freq='M')
```

Create a time series: pd.DateTimelIndex

```
pd.DataFrame({'data': index}).info()
```

```
RangeIndex: 12 entries, 0 to 11  
Data columns (total 1 columns):  
data    12 non-null datetime64[ns]  
dtypes: datetime64[ns](1)
```

Create a time series: pd.DatetimeIndex

- np.random.random :
 - Random numbers: [0,1]
 - 12 rows, 2 columns

```
data = np.random.random((size=12,2))  
pd.DataFrame(data=data, index=index).info()
```

```
DatetimeIndex: 12 entries, 2017-01-31 to 2017-12-31  
Freq: M  
Data columns (total 2 columns):  
 0    12 non-null float64  
 1    12 non-null float64  
dtypes: float64(2)
```

Frequency aliases & time info

There are many frequency aliases besides 'M' and 'D':

Period	Alias
Hour	H
Day	D
Week	W
Month	M
Quarter	Q
Year	A

These may be further differentiated by beginning/end of period, or business-specific definition

You can also access these pd.Timestamp() attributes:

attribute
.second, .minute, .hour,
.day, .month, .quarter, .year
.weekday
dayofweek
.weekofyear
.dayofyear

Let's practice!

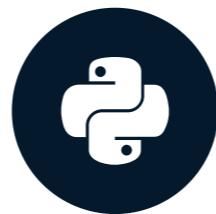
MANIPULATING TIME SERIES DATA IN PYTHON

Indexing & resampling time series

MANIPULATING TIME SERIES DATA IN PYTHON

Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence



Time series transformation

Basic time series transformations include:

- Parsing string dates and convert to `datetime64`
- Selecting & slicing for specific subperiods
- Setting & changing `DatetimeIndex` frequency
 - Upsampling vs Downsampling

Getting GOOG stock prices

```
google = pd.read_csv('google.csv') # import pandas as pd  
google.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 504 entries, 0 to 503  
Data columns (total 2 columns):  
date      504 non-null object  
price     504 non-null float64  
dtypes: float64(1), object(1)
```

```
google.head()
```

```
       date    price  
0  2015-01-02  524.81  
1  2015-01-05  513.87  
2  2015-01-06  501.96  
3  2015-01-07  501.10  
4  2015-01-08  502.68
```

Converting string dates to datetime64

- `pd.to_datetime()` :
 - Parse date string
 - Convert to `datetime64`

```
google.date = pd.to_datetime(google.date)
google.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 504 entries, 0 to 503
Data columns (total 2 columns):
date      504 non-null datetime64[ns]
price     504 non-null float64
dtypes: datetime64[ns](1), float64(1)
```

Converting string dates to datetime64

- `.set_index()` :
 - Date into index
 - `inplace` :
 - don't create copy

```
google.set_index('date', inplace=True)  
google.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 504 entries, 2015-01-02 to 2016-12-30  
Data columns (total 1 columns):  
price    504 non-null float64  
dtypes: float64(1)
```

Plotting the Google stock time series

```
google.price.plot(title='Google Stock Price')  
plt.tight_layout(); plt.show()
```



Partial string indexing

- Selecting/indexing using strings that parse to dates

```
google['2015'].info() # Pass string for part of date
```

```
DatetimeIndex: 252 entries, 2015-01-02 to 2015-12-31  
Data columns (total 1 columns):  
price    252 non-null float64  
dtypes: float64(1)
```

```
google['2015-3': '2016-2'].info() # Slice includes last month
```

```
DatetimeIndex: 252 entries, 2015-03-02 to 2016-02-29  
Data columns (total 1 columns):  
price    252 non-null float64  
dtypes: float64(1)  
memory usage: 3.9 KB
```

Partial string indexing

```
google.loc['2016-6-1', 'price'] # Use full date with .loc[]
```

```
734.15
```

.asfreq(): set frequency

- `.asfreq('D')` :
 - Convert `DateTimeIndex` to calendar day frequency

```
google.asfreq('D').info() # set calendar day frequency
```

```
DatetimeIndex: 729 entries, 2015-01-02 to 2016-12-30
Freq: D
Data columns (total 1 columns):
price    504 non-null float64
dtypes: float64(1)
```

.asfreq(): set frequency

- Upsampling:
 - Higher frequency implies new dates => missing data

```
google.asfreq('D').head()
```

```
price  
date  
2015-01-02  524.81  
2015-01-03    NaN  
2015-01-04    NaN  
2015-01-05  513.87  
2015-01-06  501.96
```

.asfreq(): reset frequency

- `.asfreq('B')` :
 - Convert `DateTimeIndex` to business day frequency

```
google = google.asfreq('B') # Change to calendar day frequency
google.info()
```

```
DatetimeIndex: 521 entries, 2015-01-02 to 2016-12-30
```

```
Freq: B
```

```
Data columns (total 1 columns):
```

```
price    504 non-null float64
```

```
dtypes: float64(1)
```

.asfreq(): reset frequency

```
google[google.price.isnull()] # Select missing 'price' values
```

```
      price  
date  
2015-01-19    NaN  
2015-02-16    NaN  
...  
2016-11-24    NaN  
2016-12-26    NaN
```

- Business days that were not trading days

Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Lags, changes, and returns for stock price series

MANIPULATING TIME SERIES DATA IN PYTHON

Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence



Basic time series calculations

- Typical Time Series manipulations include:
 - Shift or lag values back or forward back in time
 - Get the difference in value for a given time period
 - Compute the percent change over any number of periods
- `pandas` built-in methods rely on `pd.DateTimeIndex`

Getting GOOG stock prices

- Let `pd.read_csv()` do the parsing for you!

```
google = pd.read_csv('google.csv', parse_dates=['date'], index_col='date')
```

```
google.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 504 entries, 2015-01-02 to 2016-12-30
Data columns (total 1 columns):
price    504 non-null float64
dtypes: float64(1)
```

Getting GOOG stock prices

```
google.head()
```

```
      price  
date  
2015-01-02  524.81  
2015-01-05  513.87  
2015-01-06  501.96  
2015-01-07  501.10  
2015-01-08  502.68
```

.shift(): Moving data between past & future

- `.shift()` :
 - defaults to `periods=1`
 - 1 period into future

```
google['shifted'] = google.price.shift() # default: periods=1  
google.head(3)
```

```
      price  shifted  
date  
2015-01-02  542.81      NaN  
2015-01-05  513.87  542.81  
2015-01-06  501.96  513.87
```

.shift(): Moving data between past & future

- `.shift(periods=-1)` :
 - lagged data
 - 1 period back in time

```
google['lagged'] = google.price.shift(periods=-1)  
google[['price', 'lagged', 'shifted']].tail(3)
```

```
      price  lagged  shifted  
date  
2016-12-28  785.05  782.79  791.55  
2016-12-29  782.79  771.82  785.05  
2016-12-30  771.82       NaN  782.79
```

Calculate one-period percent change

- x_t / x_{t-1}

```
google['change'] = google.price.div(google.shifted)
google[['price', 'shifted', 'change']].head(3)
```

```
      price    shifted     change
Date
2017-01-03  786.14        NaN        NaN
2017-01-04  786.90  786.14  1.0000967
2017-01-05  794.02  786.90  1.009048
```

Calculate one-period percent change

```
google['return'] = google.change.sub(1).mul(100)  
google[['price', 'shifted', 'change', 'return']].head(3)
```

```
   price  shifted  change  return  
date  
2015-01-02  524.81      NaN      NaN      NaN  
2015-01-05  513.87  524.81    0.98   -2.08  
2015-01-06  501.96  513.87    0.98   -2.32
```

.diff(): built-in time-series change

- Difference in value for two adjacent periods
- $x_t - x_{t-1}$

```
google['diff'] = google.price.diff()  
google[['price', 'diff']].head(3)
```

```
      price      diff  
date  
2015-01-02  524.81      NaN  
2015-01-05  513.87    -10.94  
2015-01-06  501.96    -11.91
```

.pct_change(): built-in time-series % change

- Percent change for two adjacent periods
- $\frac{x_t}{x_{t-1}}$

```
google['pct_change'] = google.price.pct_change().mul(100)  
google[['price', 'return', 'pct_change']].head(3)
```

	price	return	pct_change
date			
2015-01-02	524.81	NaN	NaN
2015-01-05	513.87	-2.08	-2.08
2015-01-06	501.96	-2.32	-2.32

Looking ahead: Get multi-period returns

```
google['return_3d'] = google.price.pct_change(periods=3).mul(100)  
google[['price', 'return_3d']].head()
```

```
      price  return_3d  
date  
2015-01-02  524.81       NaN  
2015-01-05  513.87       NaN  
2015-01-06  501.96       NaN  
2015-01-07  501.10 -4.517825  
2015-01-08  502.68 -2.177594
```

- Percent change for two periods, 3 trading days apart

Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Compare time series growth rates

MANIPULATING TIME SERIES DATA IN PYTHON



Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence

Comparing stock performance

- Stock price series: hard to compare at different levels
- Simple solution: normalize price series to start at 100
- Divide all prices by first in series, multiply by 100
 - Same starting point
 - All prices relative to starting point
 - Difference to starting point in percentage points

Normalizing a single series (1)

```
google = pd.read_csv('google.csv', parse_dates=['date'], index_col='date')
google.head(3)
```

```
      price
date
2010-01-04  313.06
2010-01-05  311.68
2010-01-06  303.83
```

```
first_price = google.price.iloc[0] # int-based selection
first_price
```

```
313.06
```

```
first_price == google.loc['2010-01-04', 'price']
```

```
True
```

Normalizing a single series (2)

```
normalized = google.price.div(first_price).mul(100)  
normalized.plot(title='Google Normalized Series')
```



Normalizing multiple series (1)

```
prices = pd.read_csv('stock_prices.csv',  
                     parse_dates=['date'],  
                     index_col='date')  
  
prices.info()
```

```
DatetimeIndex: 1761 entries, 2010-01-04 to 2016-12-30  
Data columns (total 3 columns):  
AAPL    1761 non-null float64  
GOOG    1761 non-null float64  
YHOO    1761 non-null float64  
dtypes: float64(3)
```

```
prices.head(2)
```

```
          AAPL    GOOG    YHOO  
Date  
2010-01-04  30.57  313.06  17.10  
2010-01-05  30.63  311.68  17.23
```

Normalizing multiple series (2)

```
prices.iloc[0]
```

```
AAPL      30.57  
GOOG     313.06  
YHOO     17.10  
Name: 2010-01-04 00:00:00, dtype: float64
```

```
normalized = prices.div(prices.iloc[0])  
normalized.head(3)
```

```
          AAPL      GOOG      YHOO  
Date  
2010-01-04  1.000000  1.000000  1.000000  
2010-01-05  1.001963  0.995592  1.007602  
2010-01-06  0.985934  0.970517  1.004094
```

- `.div()` : automatic alignment of Series index & DataFrame columns

Comparing with a benchmark (1)

```
index = pd.read_csv('benchmark.csv', parse_dates=['date'], index_col='date')
index.info()
```

```
DatetimeIndex: 1826 entries, 2010-01-01 to 2016-12-30
Data columns (total 1 columns):
SP500    1762 non-null float64
dtypes: float64(1)
```

```
prices = pd.concat([prices, index], axis=1).dropna()
prices.info()
```

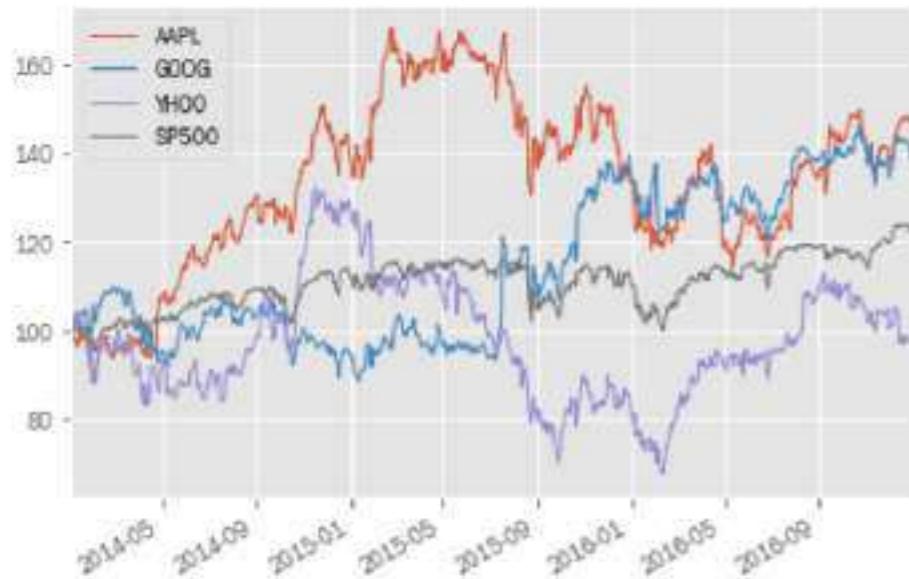
```
DatetimeIndex: 1761 entries, 2010-01-04 to 2016-12-30
Data columns (total 4 columns):
AAPL     1761 non-null float64
GOOG     1761 non-null float64
YHOO     1761 non-null float64
SP500    1761 non-null float64
dtypes: float64(4)
```

Comparing with a benchmark (2)

```
prices.head(1)
```

```
          AAPL      GOOG      YHOO      SP500  
2010-01-04  30.57  313.06  17.10  1132.99
```

```
normalized = prices.div(prices.iloc[0]).mul(100)  
normalized.plot()
```



Plotting performance difference

```
diff = normalized[tickers].sub(normalized['SP500'], axis=0)
```

```
GOOG      YHOO      AAPL  
2010-01-04  0.000000  0.000000  0.000000  
2010-01-05 -0.752375  0.448669 -0.115294  
2010-01-06 -3.314604  0.043069 -1.772895
```

- `.sub(..., axis=0)` : Subtract a Series from each DataFrame column by aligning indexes

Plotting performance difference

```
diff.plot()
```



Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Changing the time series frequency: resampling

MANIPULATING TIME SERIES DATA IN PYTHON

Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence



Changing the frequency: resampling

- `DateTimeIndex` : set & change freq using `.asfreq()`
- But frequency conversion affects the data
 - Upsampling: fill or interpolate missing data
 - Downsampling: aggregate existing data
- `pandas` API:
 - `.asfreq()` , `.reindex()`
 - `.resample()` + transformation method

Getting started: quarterly data

```
dates = pd.date_range(start='2016', periods=4, freq='Q')
data = range(1, 5)
quarterly = pd.Series(data=data, index=dates)
quarterly
```

```
2016-03-31    1
2016-06-30    2
2016-09-30    3
2016-12-31    4
Freq: Q-DEC, dtype: int64 # Default: year-end quarters
```

Upsampling: quarter => month

```
monthly = quarterly.asfreq('M') # to month-end frequency
```

```
2016-03-31    1.0
2016-04-30    NaN
2016-05-31    NaN
2016-06-30    2.0
2016-07-31    NaN
2016-08-31    NaN
2016-09-30    3.0
2016-10-31    NaN
2016-11-30    NaN
2016-12-31    4.0
Freq: M, dtype: float64
```

- Upsampling creates missing values

```
monthly = monthly.to_frame('baseline') # to DataFrame
```

Upsampling: fill methods

```
monthly['ffill'] = quarterly.asfreq('M', method='ffill')
monthly['bfill'] = quarterly.asfreq('M', method='bfill')
monthly['value'] = quarterly.asfreq('M', fill_value=0)
```

Upsampling: fill methods

- `bfill` : backfill
- `ffill` : forward fill

	baseline	ffill	bfill	value
2016-03-31	1.0	1	1	1
2016-04-30	NaN	1	2	0
2016-05-31	NaN	1	2	0
2016-06-30	2.0	2	2	2
2016-07-31	NaN	2	3	0
2016-08-31	NaN	2	3	0
2016-09-30	3.0	3	3	3
2016-10-31	NaN	3	4	0
2016-11-30	NaN	3	4	0
2016-12-31	4.0	4	4	4

Add missing months: .reindex()

```
dates = pd.date_range(start='2016',  
                      periods=12,  
                      freq='M')
```

```
DatetimeIndex(['2016-01-31',  
                '2016-02-29',  
                ...,  
                '2016-11-30',  
                '2016-12-31'],  
               dtype='datetime64[ns]', freq='M')
```

```
quarterly.reindex(dates)
```

```
2016-01-31      NaN  
2016-02-29      NaN  
2016-03-31      1.0  
2016-04-30      NaN  
2016-05-31      NaN  
2016-06-30      2.0  
2016-07-31      NaN  
2016-08-31      NaN  
2016-09-30      3.0  
2016-10-31      NaN  
2016-11-30      NaN  
2016-12-31      4.0
```

- `.reindex()` :
 - conform DataFrame to new index
 - same filling logic as `.asfreq()`

Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Upsampling & interpolation with .resample()

MANIPULATING TIME SERIES DATA IN PYTHON

Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence



Frequency conversion & transformation methods

- `.resample()` : similar to `.groupby()`
- Groups data within resampling period and applies one or several methods to each group
- New date determined by offset - start, end, etc
- Upsampling: fill from existing or interpolate values
- Downsampling: apply aggregation to existing data

Getting started: monthly unemployment rate

```
unrate = pd.read_csv('unrate.csv', parse_dates['Date'], index_col='Date')  
unrate.info()
```

```
DatetimeIndex: 208 entries, 2000-01-01 to 2017-04-01  
Data columns (total 1 columns):  
UNRATE    208 non-null float64 # no frequency information  
dtypes: float64(1)
```

```
unrate.head()
```

```
UNRATE  
DATE  
2000-01-01      4.0  
2000-02-01      4.1  
2000-03-01      4.0  
2000-04-01      3.8  
2000-05-01      4.0
```

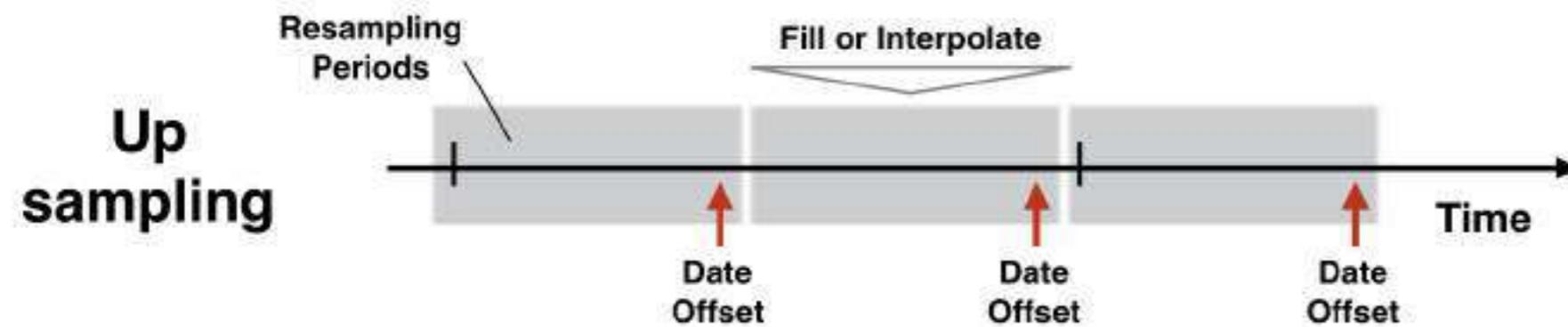
- Reporting date: 1st day of month

Resampling Period & Frequency Offsets

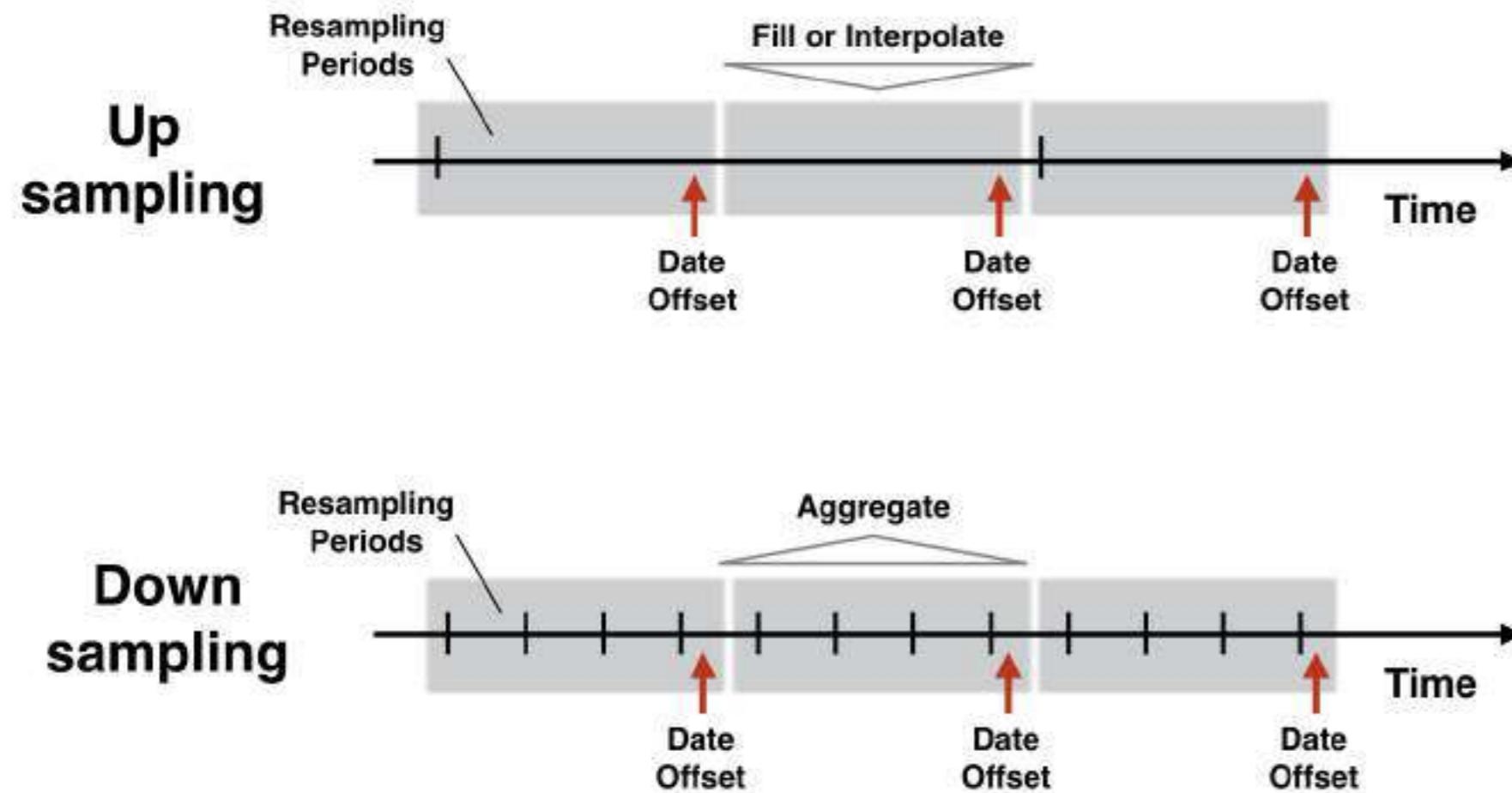
- Resample creates new date for frequency offset
- Several alternatives to calendar month end

Frequency	Alias	Sample Date
Calendar Month End	M	2017-04-30
Calendar Month Start	MS	2017-04-01
Business Month End	BM	2017-04-28
Business Month Start	BMS	2017-04-03

Resampling logic



Resampling logic



Assign frequency with .resample()

```
unrate.asfreq('MS').info()
```

```
DatetimeIndex: 208 entries, 2000-01-01 to 2017-04-01  
Freq: MS  
Data columns (total 1 columns):  
UNRATE    208 non-null float64  
dtypes: float64(1)
```

```
unrate.resample('MS') # creates Resampler object
```

```
DatetimeIndexResampler [freq=<MonthBegin>, axis=0, closed=left,  
label=left, convention=start, base=0]
```

Assign frequency with .resample()

```
unrate.asfreq('MS').equals(unrate.resample('MS').asfreq())
```

```
True
```

- `.resample()` : returns data only when calling another method

Quarterly real GDP growth

```
gdp = pd.read_csv('gdp.csv')  
gdp.info()
```

```
DatetimeIndex: 69 entries, 2000-01-01 to 2017-01-01  
Data columns (total 1 columns):  
gdp    69 non-null float64 # no frequency info  
dtypes: float64(1)
```

```
gdp.head(2)
```

```
gpd  
DATE  
2000-01-01  1.2  
2000-04-01  7.8
```

Interpolate monthly real GDP growth

```
gdp_1 = gdp.resample('MS').ffill().add_suffix('_ffill')
```

```
gpd_ffill  
DATE  
2000-01-01 1.2  
2000-02-01 1.2  
2000-03-01 1.2  
2000-04-01 7.8
```

Interpolate monthly real GDP growth

```
gdp_2 = gdp.resample('MS').interpolate().add_suffix('_inter')
```

```
gpd_inter
```

```
DATE
```

```
2000-01-01 1.200000
```

```
2000-02-01 3.400000
```

```
2000-03-01 5.600000
```

```
2000-04-01 7.800000
```

- `.interpolate()` : finds points on straight line between existing data

Concatenating two DataFrames

```
df1 = pd.DataFrame([1, 2, 3], columns=['df1'])  
df2 = pd.DataFrame([4, 5, 6], columns=['df2'])  
pd.concat([df1, df2])
```

```
df1  df2  
0   1.0  NaN  
1   2.0  NaN  
2   3.0  NaN  
0   NaN  4.0  
1   NaN  5.0  
2   NaN  6.0
```

Concatenating two DataFrames

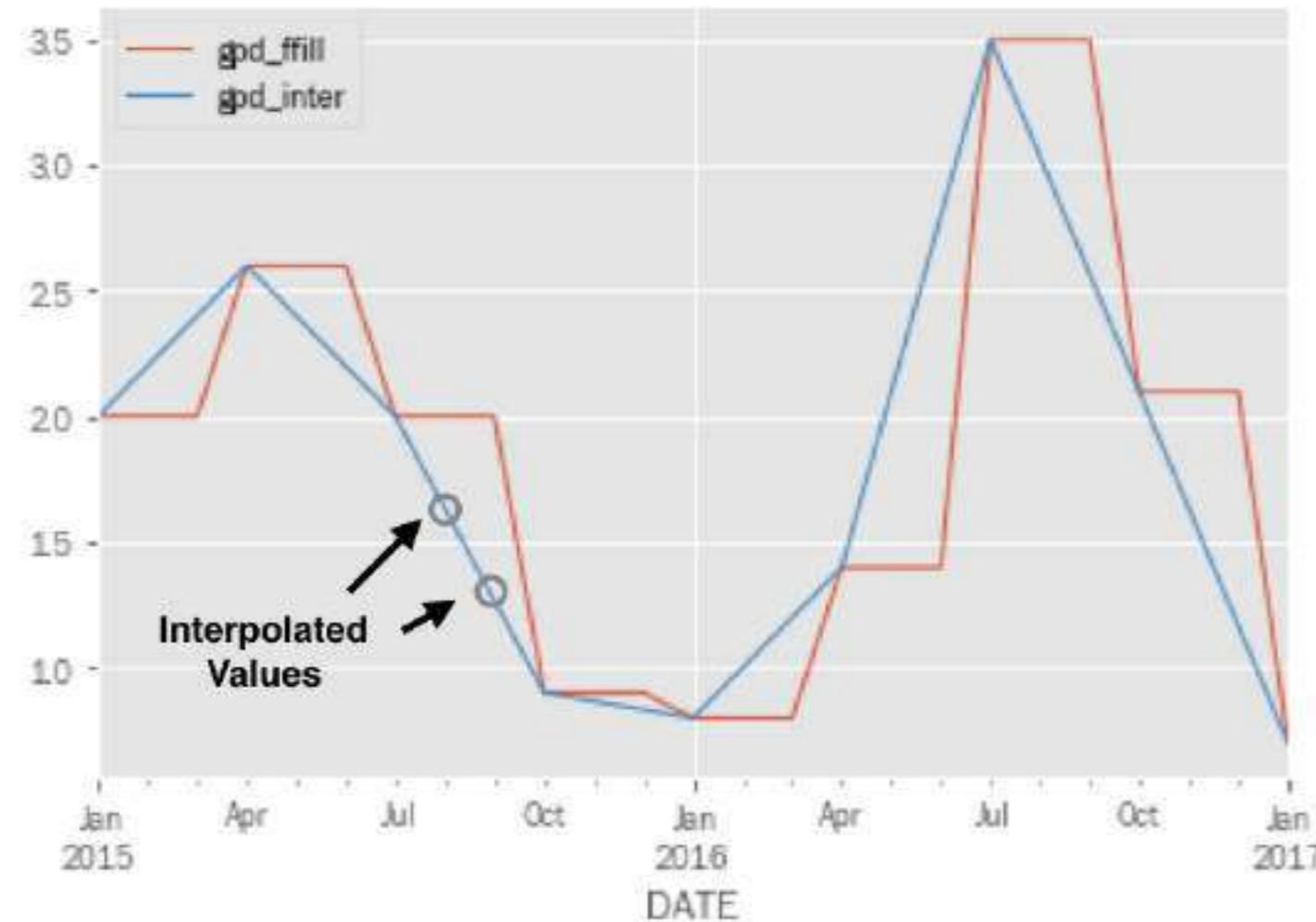
```
pd.concat([df1, df2], axis=1)
```

```
df1   df2  
0     1     4  
1     2     5  
2     3     6
```

- `axis=1` : concatenate horizontally

Plot interpolated real GDP growth

```
pd.concat([gdp_1, gdp_2], axis=1).loc['2015':].plot()
```



Combine GDP growth & unemployment

```
pd.concat([unrate, gdp_inter], axis=1).plot();
```



Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Downsampling & aggregation

MANIPULATING TIME SERIES DATA IN PYTHON



Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence

Downsampling & aggregation methods

- So far: upsampling, fill logic & interpolation
- Now: downsampling
 - hour to day
 - day to month, etc
- How to represent the existing values at the new date?
 - Mean, median, last value?

Air quality: daily ozone levels

```
ozone = pd.read_csv('ozone.csv',  
                     parse_dates=['date'],  
                     index_col='date')  
  
ozone.info()
```

```
DatetimeIndex: 6291 entries, 2000-01-01 to 2017-03-31  
Data columns (total 1 columns):  
Ozone    6167 non-null float64  
dtypes: float64(1)
```

```
ozone = ozone.resample('D').asfreq()  
ozone.info()
```

```
DatetimeIndex: 6300 entries, 1998-01-05 to 2017-03-31  
Freq: D  
Data columns (total 1 columns):  
Ozone    6167 non-null float64  
dtypes: float64(1)
```

Creating monthly ozone data

```
ozone.resample('M').mean().head()
```

```
Ozone  
date  
2000-01-31 0.010443  
2000-02-29 0.011817  
2000-03-31 0.016810  
2000-04-30 0.019413  
2000-05-31 0.026535
```

```
ozone.resample('M').median().head()
```

```
Ozone  
date  
2000-01-31 0.009486  
2000-02-29 0.010726  
2000-03-31 0.017004  
2000-04-30 0.019866  
2000-05-31 0.026018
```

.resample().mean() : Monthly average, assigned to end of calendar month

Creating monthly ozone data

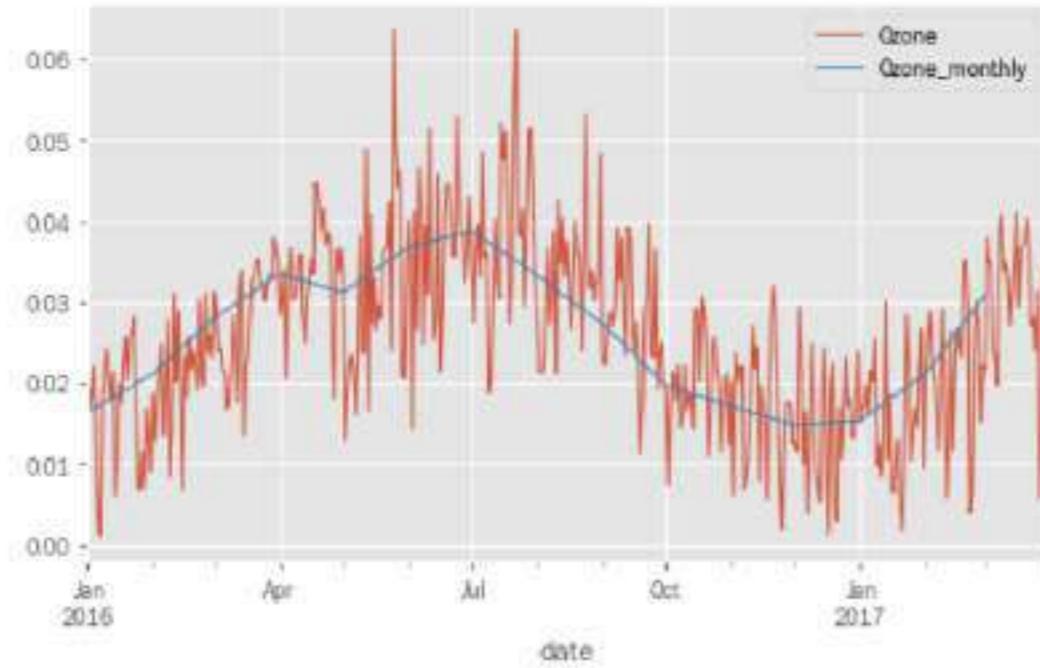
```
ozone.resample('M').agg(['mean', 'std']).head()
```

```
Ozone
      mean      std
date
2000-01-31  0.010443  0.004755
2000-02-29  0.011817  0.004072
2000-03-31  0.016810  0.004977
2000-04-30  0.019413  0.006574
2000-05-31  0.026535  0.008409
```

- `.resample().agg()` : List of aggregation functions like `groupby`

Plotting resampled ozone data

```
ozone = ozone.loc['2016':]  
ax = ozone.plot()  
monthly = ozone.resample('M').mean()  
monthly.add_suffix('_monthly').plot(ax=ax)
```



ax=ax:

Matplotlib lets you plot again on the axes object returned by the first plot

Resampling multiple time series

```
data = pd.read_csv('ozone_pm25.csv',  
                   parse_dates=['date'],  
                   index_col='date')  
  
data = data.resample('D').asfreq()  
  
data.info()
```

```
DatetimeIndex: 6300 entries, 2000-01-01 to 2017-03-31  
Freq: D  
Data columns (total 2 columns):  
Ozone      6167 non-null float64  
PM25       6167 non-null float64  
dtypes: float64(2)
```

Resampling multiple time series

```
data = data.resample('BM').mean()  
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 207 entries, 2000-01-31 to 2017-03-31  
Freq: BM  
Data columns (total 2 columns):  
 ozone      207 non-null float64  
 pm25       207 non-null float64  
 dtypes: float64(2)
```

Resampling multiple time series

```
df.resample('M').first().head(4)
```

```
Ozone      PM25  
date  
2000-01-31  0.005545  20.800000  
2000-02-29  0.016139  6.500000  
2000-03-31  0.017004  8.493333  
2000-04-30  0.031354  6.889474
```

```
df.resample('MS').first().head()
```

```
Ozone      PM25  
date  
2000-01-01  0.004032  37.320000  
2000-02-01  0.010583  24.800000  
2000-03-01  0.007418  11.106667  
2000-04-01  0.017631  11.700000  
2000-05-01  0.022628  9.700000
```

Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Rolling window functions with pandas

MANIPULATING TIME SERIES DATA IN PYTHON

Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence



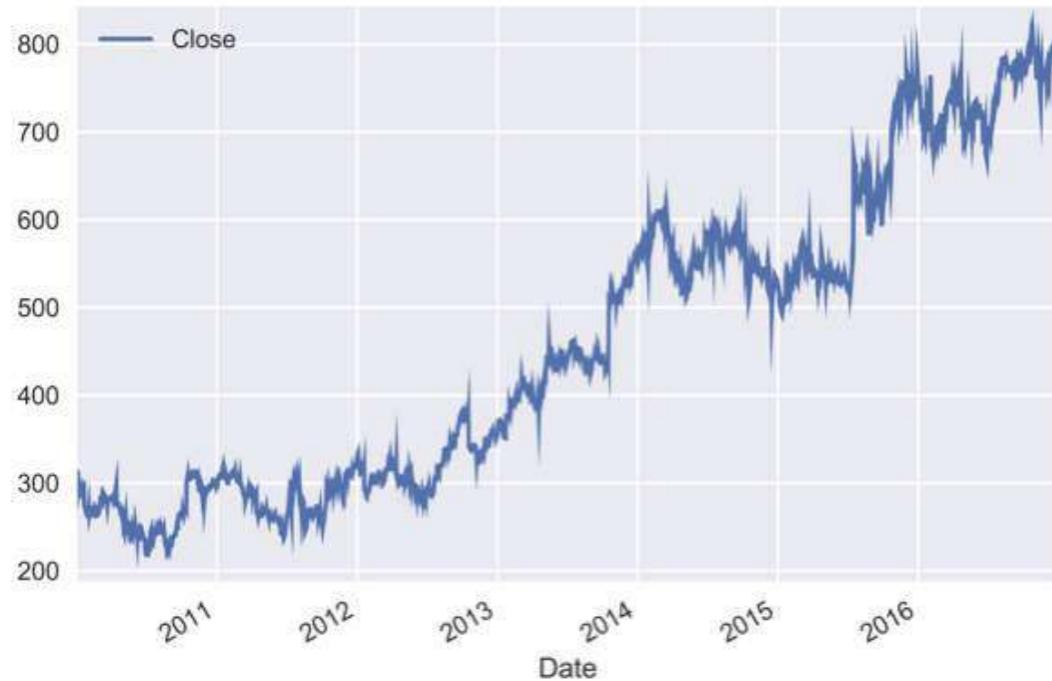
Window functions in pandas

- Windows identify sub periods of your time series
- Calculate metrics for sub periods inside the window
- Create a new time series of metrics
- Two types of windows:
 - Rolling: same size, sliding (this video)
 - Expanding: contain all prior values (next video)

Calculating a rolling average

```
data = pd.read_csv('google.csv', parse_dates=['date'], index_col='date')
```

```
DatetimeIndex: 1761 entries, 2010-01-04 to 2016-12-30  
Data columns (total 1 columns):  
price    1761 non-null float64  
dtypes: float64(1)
```



Calculating a rolling average

```
# Integer-based window size  
data.rolling(window=30).mean() # fixed # observations
```

```
DatetimeIndex: 1761 entries, 2010-01-04 to 2017-05-24  
Data columns (total 1 columns):  
 price    1732 non-null float64  
 dtypes: float64(1)
```

- `window=30` : # business days
- `min_periods` : choose value < 30 to get results for first days

Calculating a rolling average

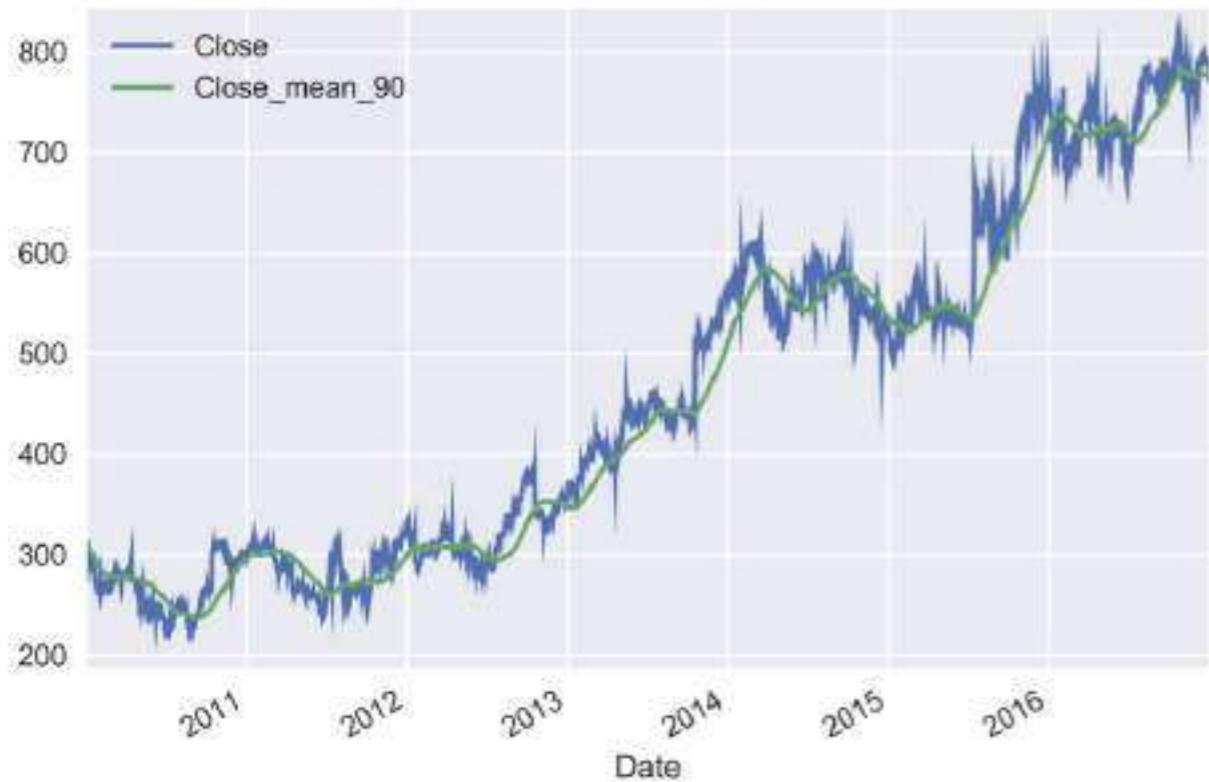
```
# Offset-based window size  
data.rolling(window='30D').mean() # fixed period length
```

```
DatetimeIndex: 1761 entries, 2010-01-04 to 2017-05-24  
Data columns (total 1 columns):  
 price    1761 non-null float64  
 dtypes: float64(1)
```

- 30D : # calendar days

90 day rolling mean

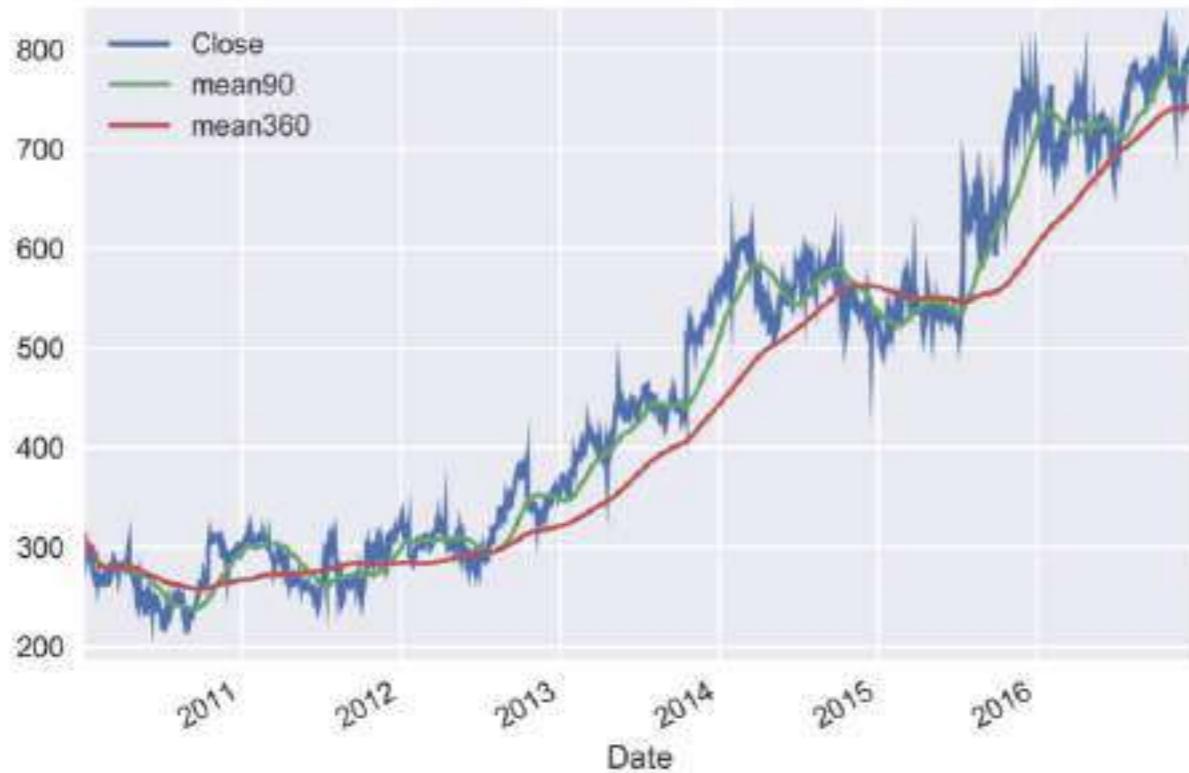
```
r90 = data.rolling(window='90D').mean()  
google.join(r90.add_suffix('_mean_90')).plot()
```



.join:
**concatenate Series or
DataFrame along
axis=1**

90 & 360 day rolling means

```
data['mean90'] = r90  
r360 = data['price'].rolling(window='360D').mean()  
data['mean360'] = r360; data.plot()
```



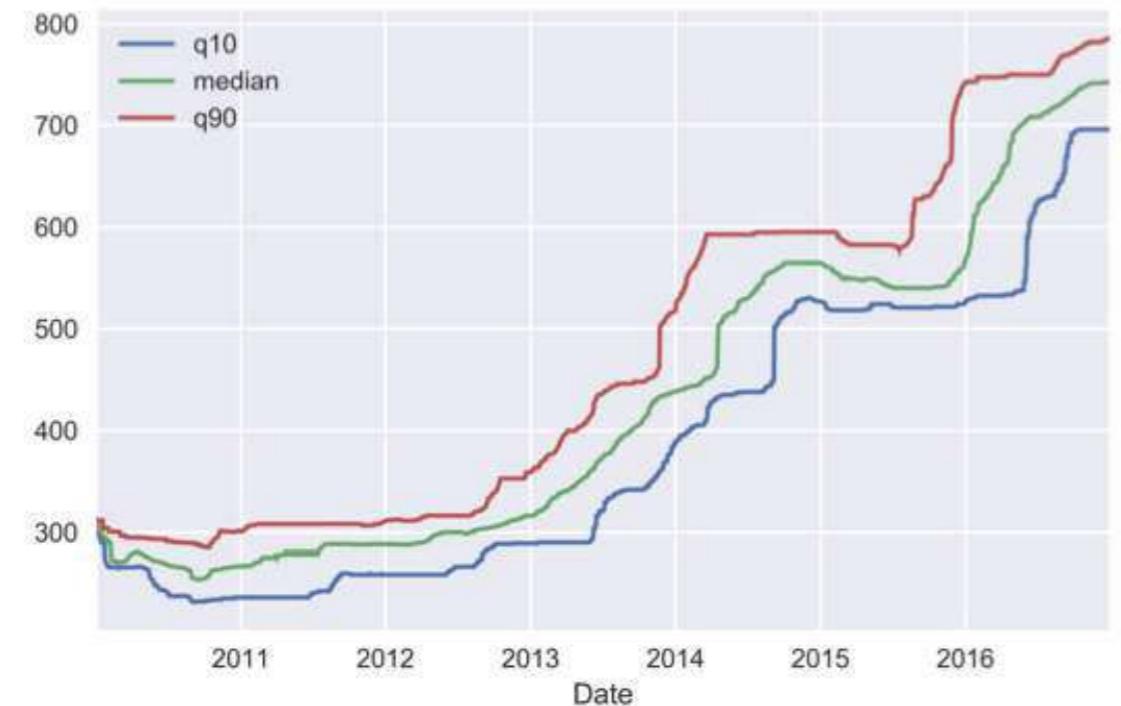
Multiple rolling metrics (1)

```
r = data.price.rolling('90D').agg(['mean', 'std'])  
r.plot(subplots = True)
```



Multiple rolling metrics (2)

```
rolling = data.google.rolling('360D')
q10 = rolling.quantile(0.1).to_frame('q10')
median = rolling.median().to_frame('median')
q90 = rolling.quantile(0.9).to_frame('q90')
pd.concat([q10, median, q90], axis=1).plot()
```



Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Expanding window functions with pandas

MANIPULATING TIME SERIES DATA IN PYTHON

Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence



Expanding windows in pandas

- From rolling to expanding windows
- Calculate metrics for periods up to current date
- New time series reflects all historical values
- Useful for running rate of return, running min/max
- Two options with pandas:
 - `.expanding()` - just like `.rolling()`
 - `.cumsum()` , `.cumprod()` , `cummin()` / `max()`

The basic idea

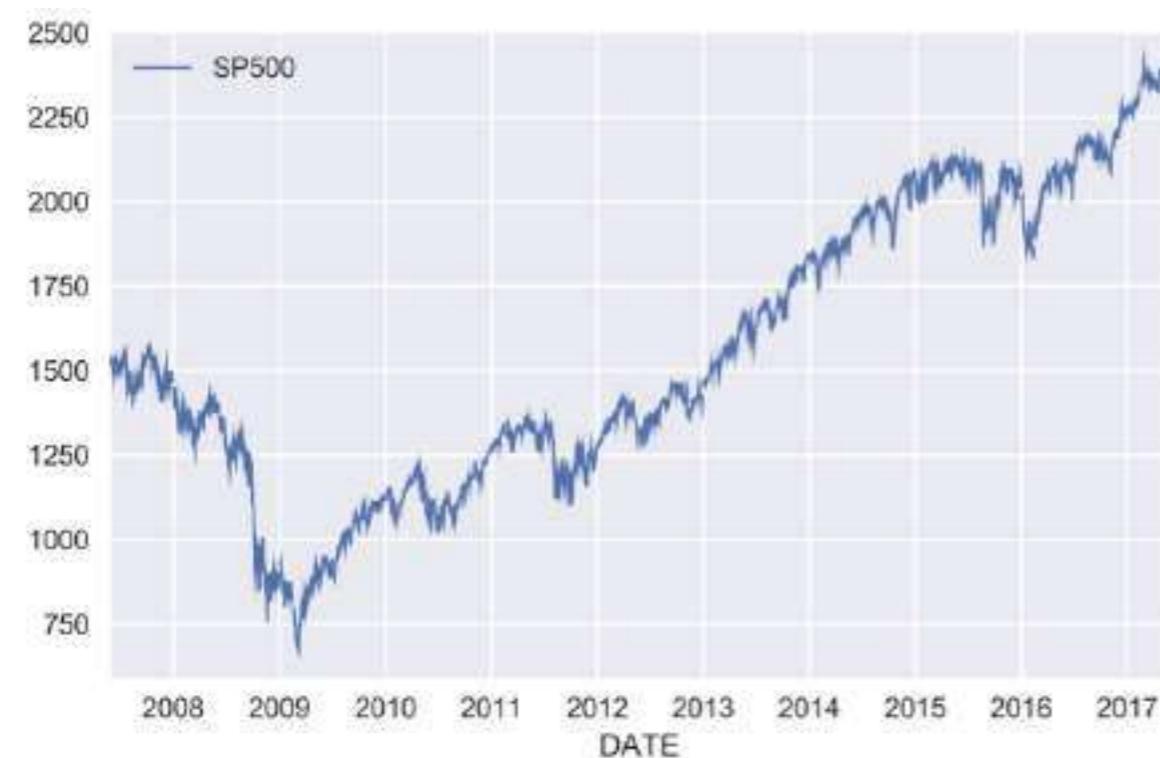
```
df = pd.DataFrame({'data': range(5)})  
df['expanding sum'] = df.data.expanding().sum()  
df['cumulative sum'] = df.data.cumsum()  
df
```

	data	expanding sum	cumulative sum
0	0	0.0	0
1	1	1.0	1
2	2	3.0	3
3	3	6.0	6
4	4	10.0	10

Get data for the S&P 500

```
data = pd.read_csv('sp500.csv', parse_dates=['date'], index_col='date')
```

```
DatetimeIndex: 2519 entries, 2007-05-24 to 2017-05-24  
Data columns (total 1 columns):  
SP500    2519 non-null float64
```



How to calculate a running return

- Single period return r_t : current price over last price minus 1:

$$r_t = \frac{P_t}{P_{t-1}} - 1$$

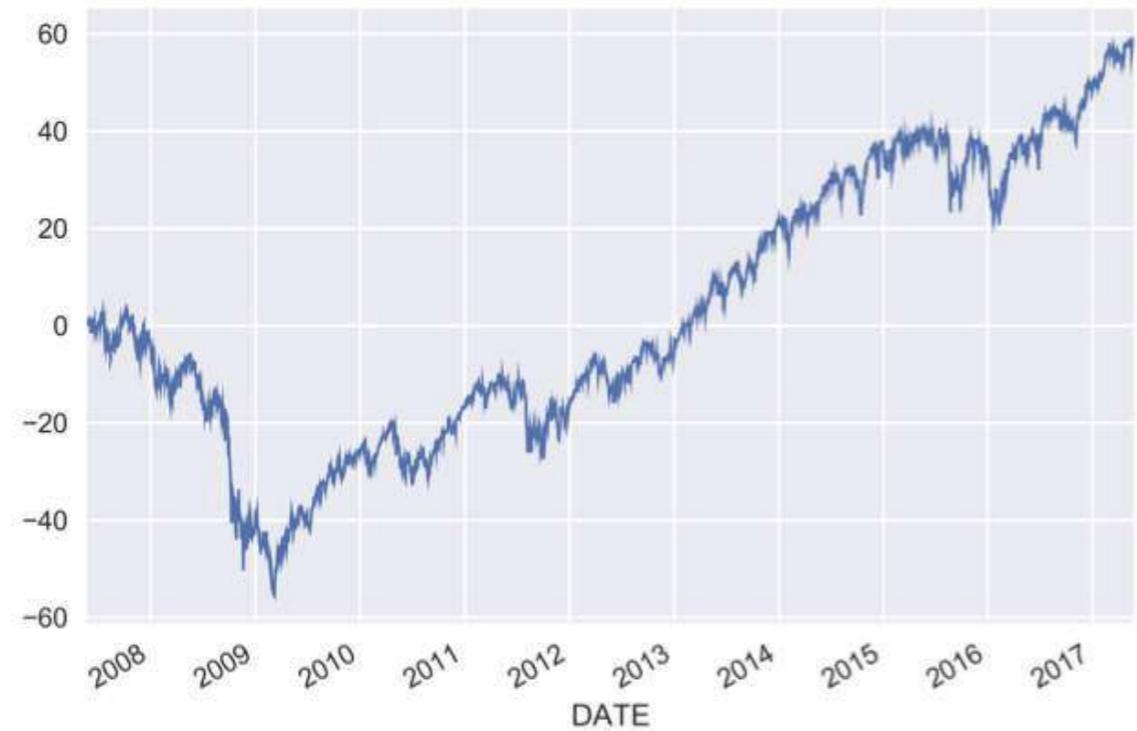
- Multi-period return: product of $(1 + r_t)$ for all periods, minus 1:

$$R_T = (1 + r_1)(1 + r_2)\dots(1 + r_T) - 1$$

- For the period return: `.pct_change()`
- For basic math `.add()`, `.sub()`, `.mul()`, `.div()`
- For cumulative product: `.cumprod()`

Running rate of return in practice

```
pr = data.SP500.pct_change() # period return  
pr_plus_one = pr.add(1)  
cumulative_return = pr_plus_one.cumprod().sub(1)  
cumulative_return.mul(100).plot()
```



Getting the running min & max

```
data['running_min'] = data.SP500.expanding().min()  
data['running_max'] = data.SP500.expanding().max()  
data.plot()
```



Rolling annual rate of return

```
def multi_period_return(period_returns):  
    return np.prod(period_returns + 1) - 1  
pr = data.SP500.pct_change() # period return  
r = pr.rolling('360D').apply(multi_period_return)  
data['Rolling 1yr Return'] = r.mul(100)  
data.plot(subplots=True)
```

Rolling annual rate of return

```
data['Rolling 1yr Return'] = r.mul(100)  
data.plot(subplots=True)
```

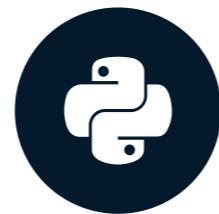


Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Case study: S&P500 price simulation

MANIPULATING TIME SERIES DATA IN PYTHON



Stefan Jansen

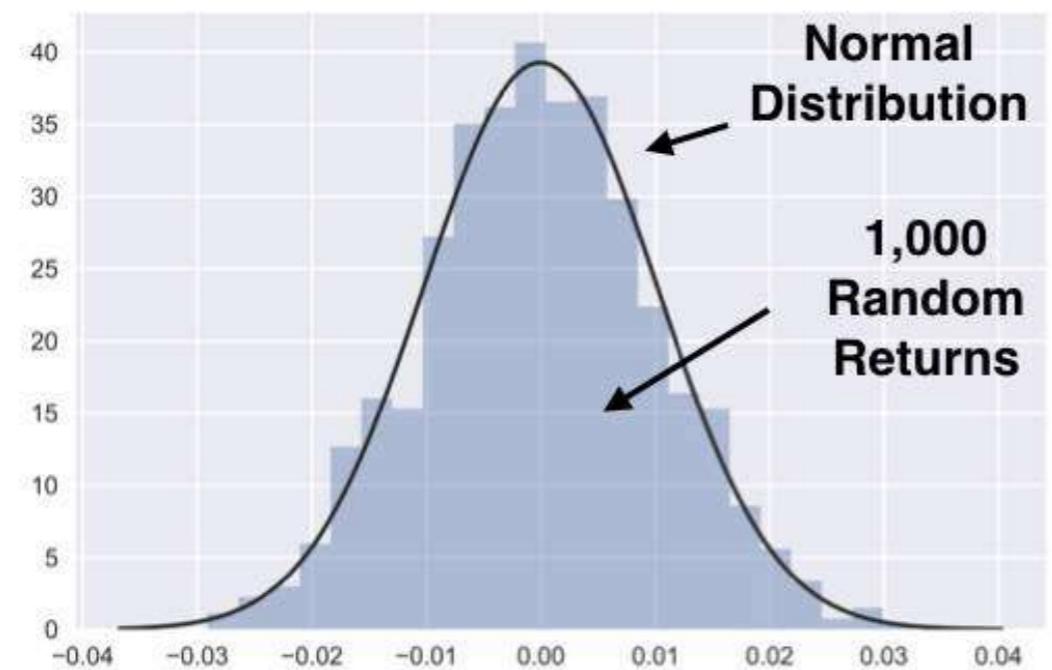
Founder & Lead Data Scientist at
Applied Artificial Intelligence

Random walks & simulations

- Daily stock returns are hard to predict
- Models often assume they are random in nature
- Numpy allows you to generate random numbers
- From random returns to prices: use `.cumprod()`
- Two examples:
 - Generate random returns
 - Randomly selected actual SP500 returns

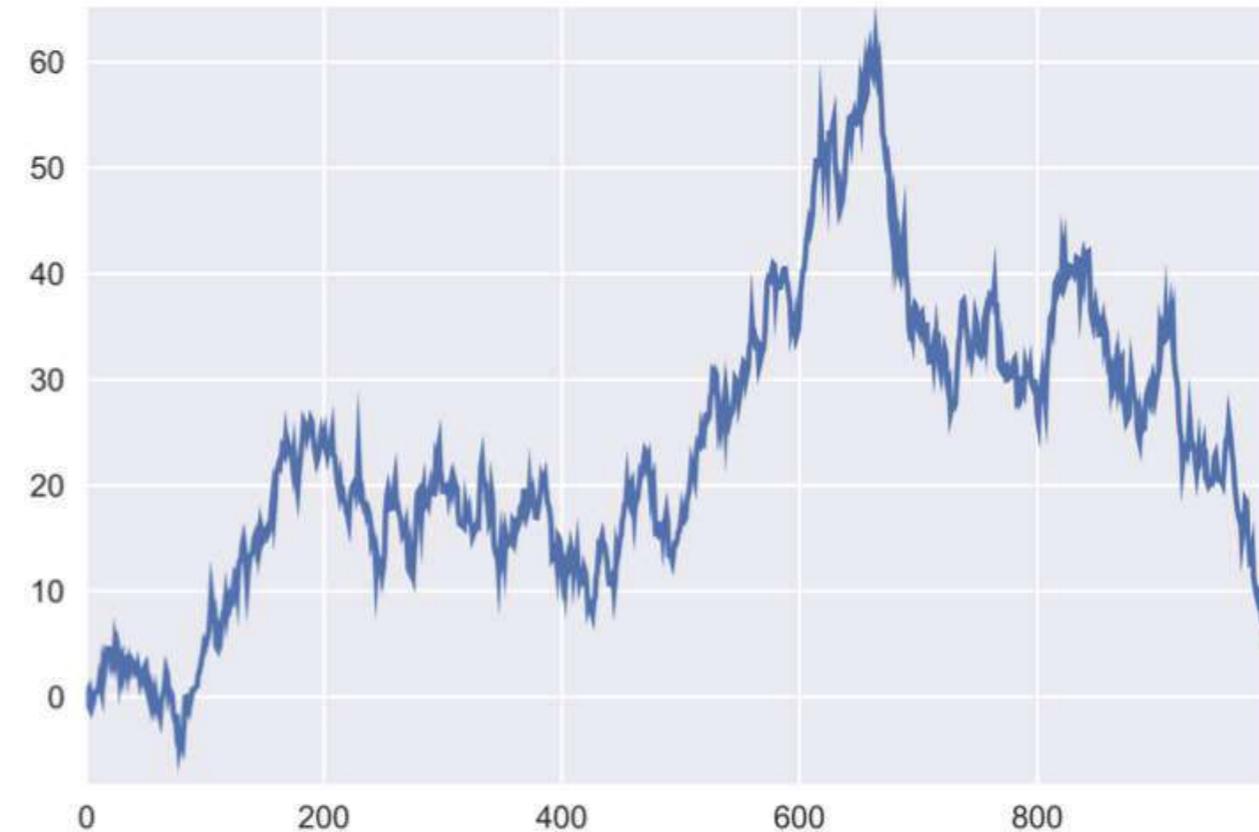
Generate random numbers

```
from numpy.random import normal, seed  
from scipy.stats import norm  
seed(42)  
random_returns = normal(loc=0, scale=0.01, size=1000)  
sns.distplot(random_returns, fit=norm, kde=False)
```



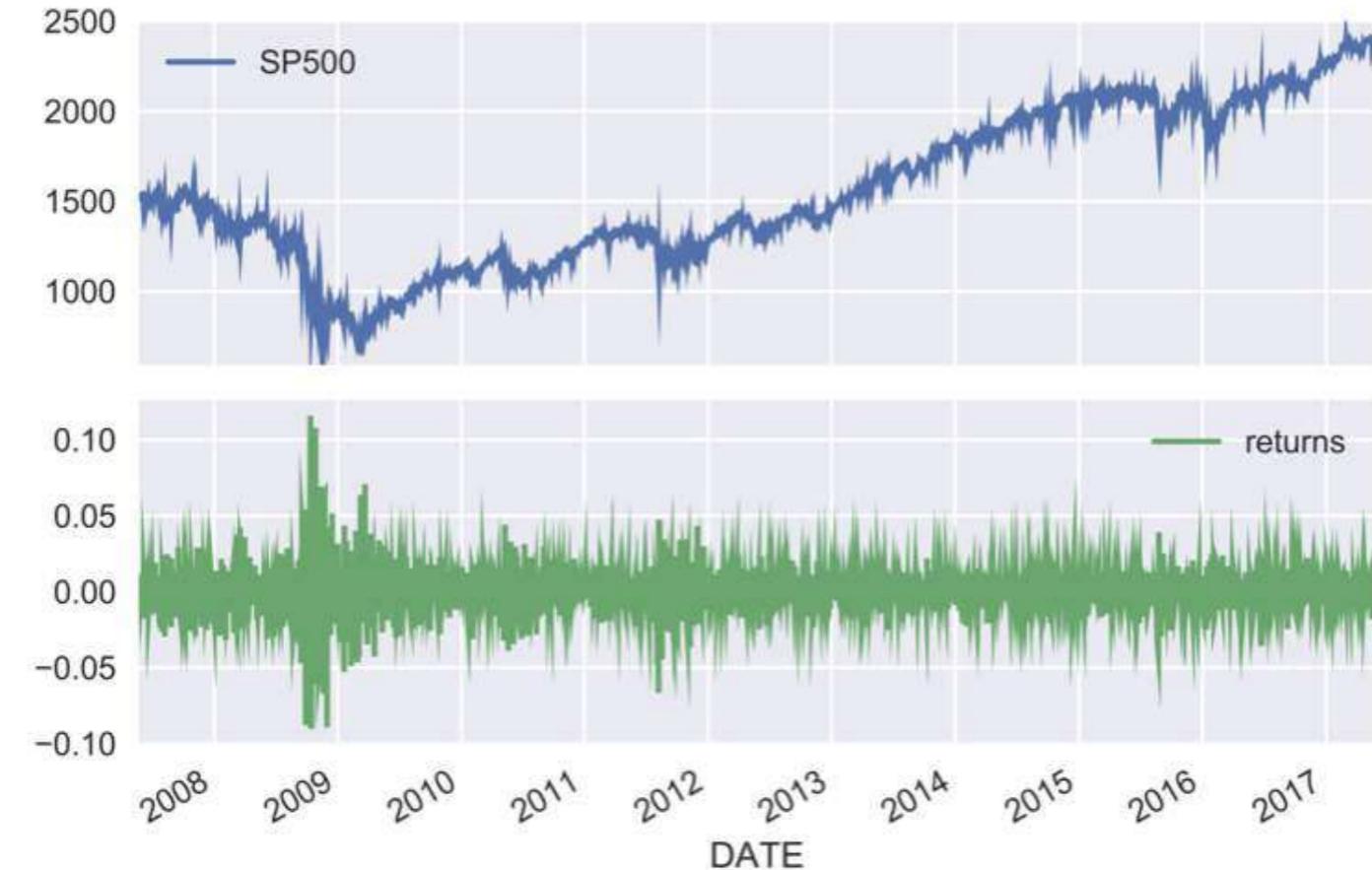
Create a random price path

```
return_series = pd.Series(random_returns)  
random_prices = return_series.add(1).cumprod().sub(1)  
random_prices.mul(100).plot()
```



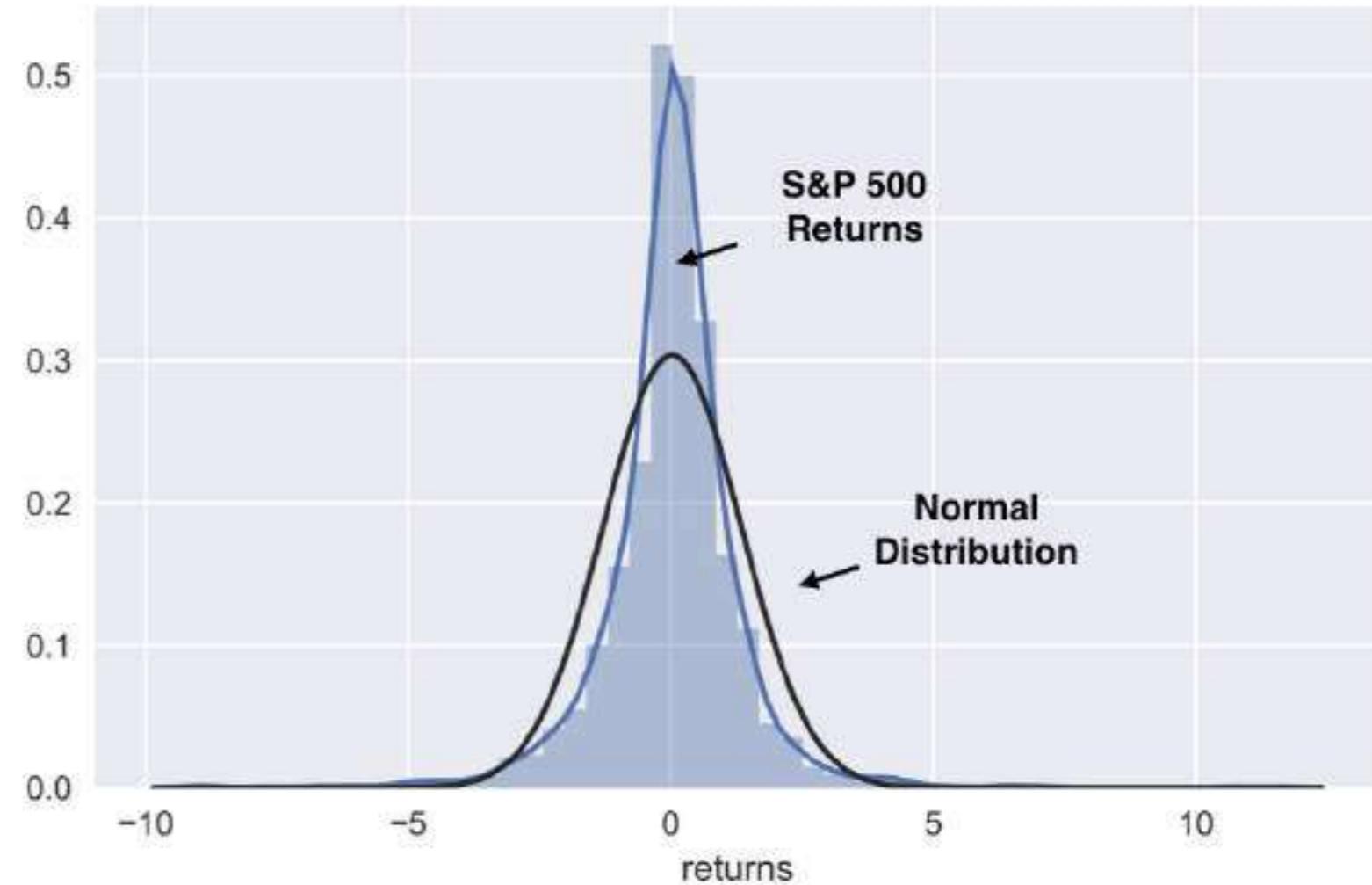
S&P 500 prices & returns

```
data = pd.read_csv('sp500.csv', parse_dates=['date'], index_col='date')
data['returns'] = data.SP500.pct_change()
data.plot(subplots=True)
```



S&P return distribution

```
sns.distplot(data.returns.dropna().mul(100), fit=norm)
```



Generate random S&P 500 returns

```
from numpy.random import choice  
sample = data.returns.dropna()  
n_obs = data.returns.count()  
random_walk = choice(sample, size=n_obs)  
random_walk = pd.Series(random_walk, index=sample.index)  
random_walk.head()
```

DATE	
2007-05-29	-0.008357
2007-05-30	0.003702
2007-05-31	-0.013990
2007-06-01	0.008096
2007-06-04	0.013120

Random S&P 500 prices (1)

```
start = data.SP500.first('D')
```

```
DATE  
2007-05-25    1515.73  
Name: SP500, dtype: float64
```

```
sp500_random = start.append(random_walk.add(1))  
sp500_random.head()
```

```
DATE  
2007-05-25    1515.730000  
2007-05-29    0.998290  
2007-05-30    0.995190  
2007-05-31    0.997787  
2007-06-01    0.983853  
dtype: float64
```

Random S&P 500 prices (2)

```
data['SP500_random'] = sp500_random.cumprod()  
data[['SP500', 'SP500_random']].plot()
```



Let's practice!

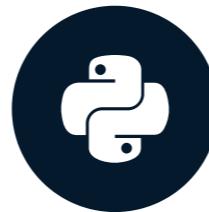
MANIPULATING TIME SERIES DATA IN PYTHON

Relationships between time series: correlation

MANIPULATING TIME SERIES DATA IN PYTHON

Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence



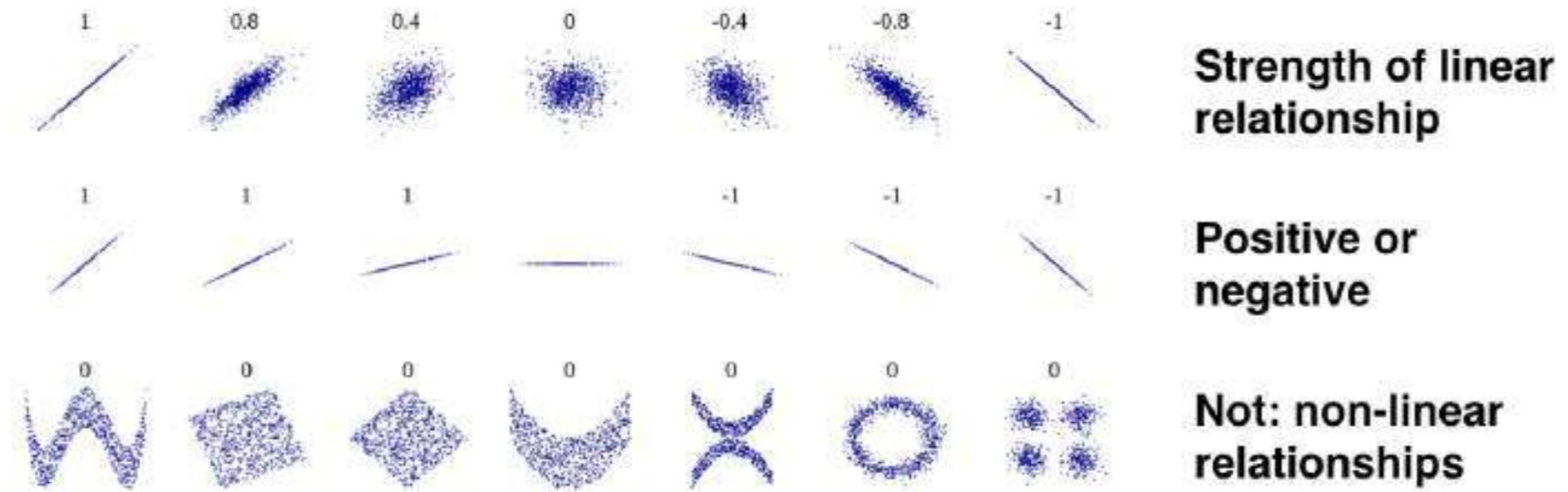
Correlation & relations between series

- So far, focus on characteristics of individual variables
- Now: characteristic of relations between variables
- Correlation: measures linear relationships
- Financial markets: important for prediction and risk management
- `pandas` & `seaborn` have tools to compute & visualize

Correlation & linear relationships

- Correlation coefficient: how similar is the pairwise movement of two variables around their averages?
- Varies between **-1** and **+1**

$$r = \frac{\sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})}{s_x s_y}$$



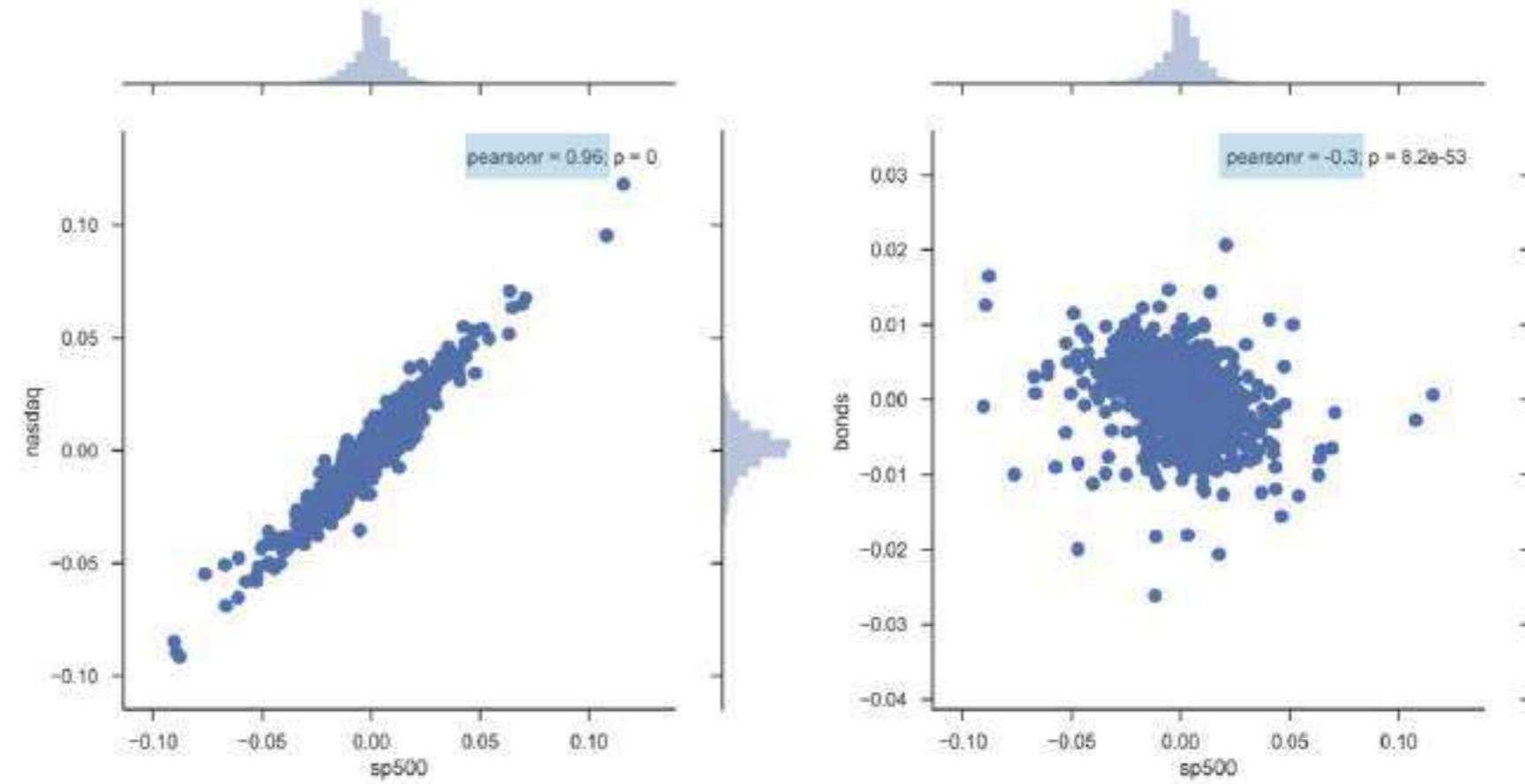
Importing five price time series

```
data = pd.read_csv('assets.csv', parse_dates=['date'],
                    index_col='date')
data = data.dropna().info()
```

```
DatetimeIndex: 2469 entries, 2007-05-25 to 2017-05-22
Data columns (total 5 columns):
sp500      2469 non-null float64
nasdaq     2469 non-null float64
bonds      2469 non-null float64
gold       2469 non-null float64
oil        2469 non-null float64
```

Visualize pairwise linear relationships

```
daily_returns = data.pct_change()  
sns.jointplot(x='sp500', y='nasdaq', data=data_returns);
```



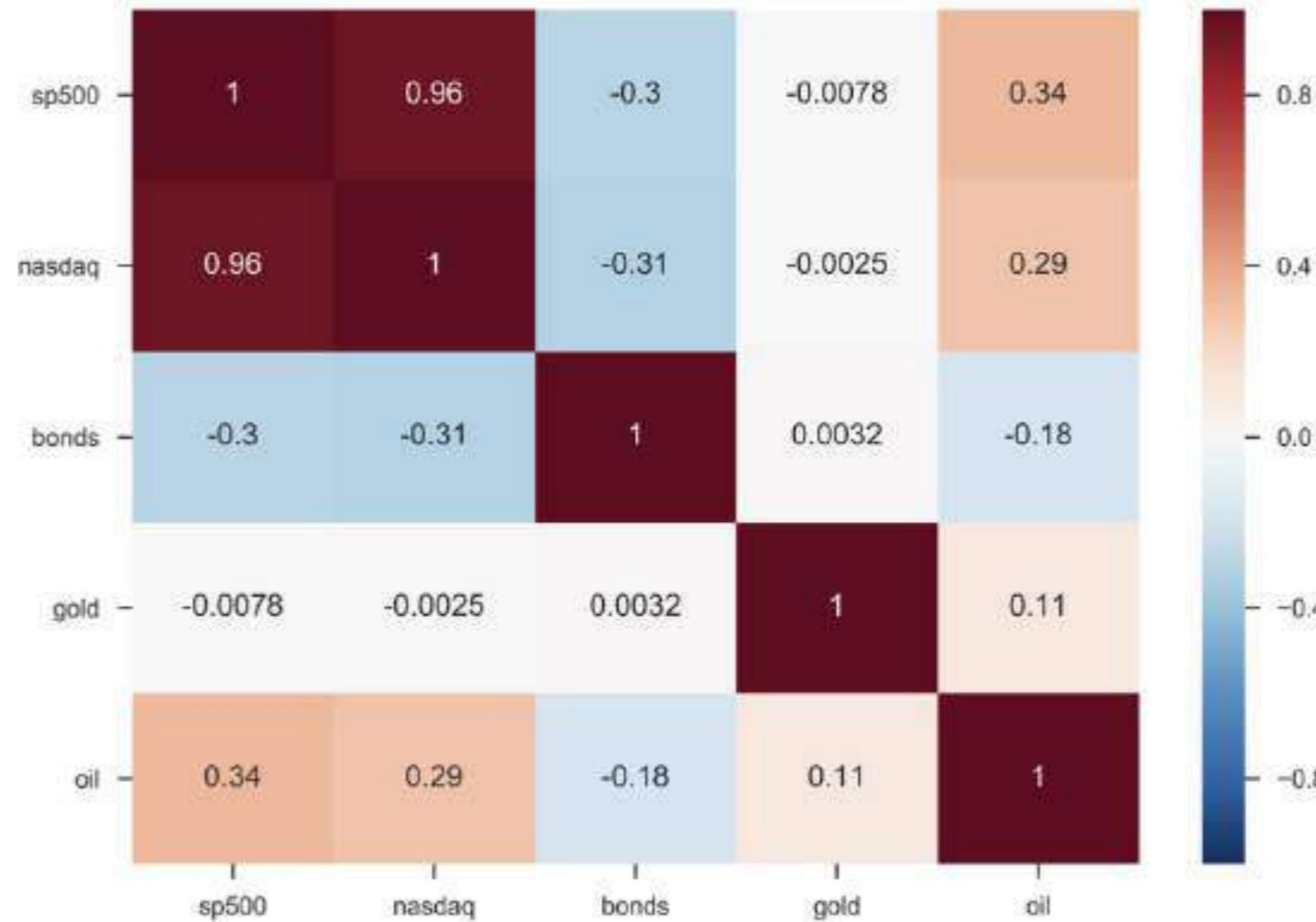
Calculate all correlations

```
correlations = returns.corr()  
correlations
```

```
bonds          oil         gold        sp500      nasdaq  
bonds  1.000000 -0.183755  0.003167 -0.300877 -0.306437  
oil    -0.183755  1.000000  0.105930  0.335578  0.289590  
gold    0.003167  0.105930  1.000000 -0.007786 -0.002544  
sp500   -0.300877  0.335578 -0.007786  1.000000  0.959990  
nasdaq -0.306437  0.289590 -0.002544  0.959990  1.000000
```

Visualize all correlations

```
sns.heatmap(correlations, annot=True)
```



Let's practice!

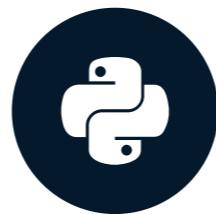
MANIPULATING TIME SERIES DATA IN PYTHON

Select index components & import data

MANIPULATING TIME SERIES DATA IN PYTHON

Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence



Market value-weighted index

- Composite performance of various stocks
- Components weighted by market capitalization
 - Share Price \times Number of Shares \Rightarrow Market Value
- Larger components get higher percentage weightings
- Key market indexes are value-weighted:
 - S&P 500 , NASDAQ , Wilshire 5000 , Hang Seng

Build a cap-weighted Index

- Apply new skills to construct value-weighted index
 - Select components from exchange listing data
 - Get component number of shares and stock prices
 - Calculate component weights
 - Calculate index
 - Evaluate performance of components and index

Load stock listing data

```
nyse = pd.read_excel('listings.xlsx', sheet_name='nyse',  
                     na_values='n/a')  
  
nyse.info()
```

```
RangeIndex: 3147 entries, 0 to 3146  
Data columns (total 7 columns):  
 Stock Symbol      3147 non-null object # Stock Ticker  
 Company Name     3147 non-null object  
 Last Sale        3079 non-null float64 # Latest Stock Price  
 Market Capitalization  3147 non-null float64  
 IPO Year         1361 non-null float64 # Year of listing  
 Sector            2177 non-null object  
 Industry           2177 non-null object  
 dtypes: float64(3), object(4)
```

Load & prepare listing data

```
nyse.set_index('Stock Symbol', inplace=True)  
nyse.dropna(subset=['Sector'], inplace=True)  
nyse['Market Capitalization'] /= 1e6 # in Million USD
```

```
Index: 2177 entries, DDD to ZTO  
Data columns (total 6 columns):  
Company Name          2177 non-null object  
Last Sale              2175 non-null float64  
Market Capitalization  2177 non-null float64  
IPO Year               967 non-null float64  
Sector                 2177 non-null object  
Industry                2177 non-null object  
dtypes: float64(3), object(3)
```

Select index components

```
components = nyse.groupby(['Sector'])['Market Capitalization'].nlargest(1)  
components.sort_values(ascending=False)
```

```
Sector           Stock Symbol  
Health Care     JNJ          338834.390080  
Energy          XOM          338728.713874  
Finance         JPM          300283.250479  
Miscellaneous   BABA         275525.000000  
Public Utilities T            247339.517272  
Basic Industries PG           230159.644117  
Consumer Services WMT          221864.614129  
Consumer Non-Durables KO           183655.305119  
Technology      ORCL          181046.096000  
Capital Goods   TM            155660.252483  
Transportation  UPS           90180.886756  
Consumer Durables ABB          48398.935676  
Name: Market Capitalization, dtype: float64
```

Import & prepare listing data

```
tickers = components.index.get_level_values('Stock Symbol')  
tickers
```

```
Index(['PG', 'TM', 'ABB', 'KO', 'WMT', 'XOM', 'JPM', 'JNJ', 'BABA', 'T',  
       'ORCL', 'UPS'], dtype='object', name='Stock Symbol')
```

```
tickers.tolist()
```

```
['PG',  
 'TM',  
 'ABB',  
 'KO',  
 'WMT',  
 ...  
 'T',  
 'ORCL',  
 'UPS']
```

Stock index components

```
columns = ['Company Name', 'Market Capitalization', 'Last Sale']
component_info = nyse.loc[tickers, columns]
pd.options.display.float_format = '{:.2f}'.format
```

Stock Symbol	Company Name	Market Capitalization	Last Sale
PG	Procter & Gamble Company (The)	230,159.64	90.03
TM	Toyota Motor Corp Ltd Ord	155,660.25	104.18
ABB	ABB Ltd	48,398.94	22.63
KO	Coca-Cola Company (The)	183,655.31	42.79
WMT	Wal-Mart Stores, Inc.	221,864.61	73.15
XOM	Exxon Mobil Corporation	338,728.71	81.69
JPM	J P Morgan Chase & Co	300,283.25	84.40
JNJ	Johnson & Johnson	338,834.39	124.99
BABA	Alibaba Group Holding Limited	275,525.00	110.21
T	AT&T Inc.	247,339.52	40.28
ORCL	Oracle Corporation	181,046.10	44.00
UPS	United Parcel Service, Inc.	90,180.89	103.74

Import & prepare listing data

```
data = pd.read_csv('stocks.csv', parse_dates=['Date'],
                   index_col='Date').loc[:, tickers.tolist()]

data.info()
```

```
DatetimeIndex: 252 entries, 2016-01-04 to 2016-12-30
```

```
Data columns (total 12 columns):
```

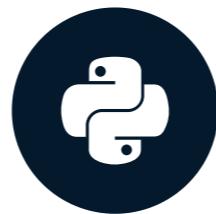
```
ABB    252 non-null float64
BABA   252 non-null float64
JNJ    252 non-null float64
JPM    252 non-null float64
KO     252 non-null float64
ORCL   252 non-null float64
PG     252 non-null float64
T      252 non-null float64
TM     252 non-null float64
UPS    252 non-null float64
WMT    252 non-null float64
XOM    252 non-null float64
dtypes: float64(12)
```

Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Build a market-cap weighted index

MANIPULATING TIME SERIES DATA IN PYTHON



Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence

Build your value-weighted index

- Key inputs:
 - number of shares
 - stock price series

Build your value-weighted index

- Key inputs:
 - number of shares
 - stock price series
 - Normalize index to start at 100



Aggregate
Market Value
per Period

Stock index components

components

Stock Symbol	Company Name	Market Capitalization	Last Sale
PG	Procter & Gamble Company (The)	230,159.64	90.03
TM	Toyota Motor Corp Ltd Ord	155,660.25	104.18
ABB	ABB Ltd	48,398.94	22.63
KO	Coca-Cola Company (The)	183,655.31	42.79
WMT	Wal-Mart Stores, Inc.	221,864.61	73.15
XOM	Exxon Mobil Corporation	338,728.71	81.69
JPM	J P Morgan Chase & Co	300,283.25	84.40
JNJ	Johnson & Johnson	338,834.39	124.99
BABA	Alibaba Group Holding Limited	275,525.00	110.21
T	AT&T Inc.	247,339.52	40.28
ORCL	Oracle Corporation	181,046.10	44.00
UPS	United Parcel Service, Inc.	90,180.89	103.74

Number of shares outstanding

```
shares = components['Market Capitalization'].div(components['Last Sale'])
```

```
Stock Symbol
PG      2,556.48 # Outstanding shares in million
TM      1,494.15
ABB     2,138.71
KO      4,292.01
WMT     3,033.01
XOM     4,146.51
JPM     3,557.86
JNJ     2,710.89
BABA    2,500.00
T       6,140.50
ORCL    4,114.68
UPS     869.30
dtype: float64
```

- Market Capitalization = Number of Shares x Share Price

Historical stock prices

```
data = pd.read_csv('stocks.csv', parse_dates=['Date'],
                   index_col='Date').loc[:, tickers.tolist()]
market_cap_series = data.mul(no_shares)
market_series.info()
```

```
DatetimeIndex: 252 entries, 2016-01-04 to 2016-12-30
Data columns (total 12 columns):
ABB      252 non-null float64
BABA     252 non-null float64
JNJ      252 non-null float64
JPM      252 non-null float64
...
TM       252 non-null float64
UPS      252 non-null float64
WMT      252 non-null float64
XOM      252 non-null float64
dtypes: float64(12)
```

From stock prices to market value

```
market_cap_series.first('D').append(market_cap_series.last('D'))
```

	ABB	BABA	JNJ	JPM	KO	ORCL	\\"
Date							
2016-01-04	37,470.14	191,725.00	272,390.43	226,350.95	181,981.42	147,099.95	
2016-12-30	45,062.55	219,525.00	312,321.87	307,007.60	177,946.93	158,209.60	
	PG	T	TM	UPS	WMT	XOM	
Date							
2016-01-04	200,351.12	210,926.33	181,479.12	82,444.14	186,408.74	321,188.96	
2016-12-30	214,948.60	261,155.65	175,114.05	99,656.23	209,641.59	374,264.34	

Aggregate market value per period

```
agg_mcap = market_cap_series.sum(axis=1) # Total market cap  
agg_mcap(title='Aggregate Market Cap')
```



Value-based index

```
index = agg_mcap.div(agg_mcap.iloc[0]).mul(100) # Divide by 1st value  
index.plot(title='Market-Cap Weighted Index')
```

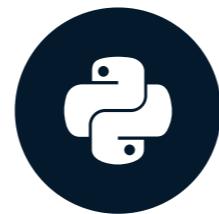


Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Evaluate index performance

MANIPULATING TIME SERIES DATA IN PYTHON



Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence

Evaluate your value-weighted index

- Index return:
 - Total index return
 - Contribution by component
- Performance vs Benchmark
 - Total period return
 - Rolling returns for sub periods

Value-based index - recap

```
agg_market_cap = market_cap_series.sum(axis=1)  
index = agg_market_cap.div(agg_market_cap.iloc[0]).mul(100)  
index.plot(title='Market-Cap Weighted Index')
```



Value contribution by stock

```
agg_market_cap.iloc[-1] - agg_market_cap.iloc[0]
```

315,037.71

Value contribution by stock

```
change = market_cap_series.first('D').append(market_cap_series.last('D'))  
change.diff().iloc[-1].sort_values() # or: .loc['2016-12-30']
```

```
TM      -6,365.07  
KO      -4,034.49  
ABB      7,592.41  
ORCL     11,109.65  
PG      14,597.48  
UPS     17,212.08  
WMT     23,232.85  
BABA    27,800.00  
JNJ     39,931.44  
T       50,229.33  
XOM     53,075.38  
JPM     80,656.65  
  
Name: 2016-12-30 00:00:00, dtype: float64
```

Market-cap based weights

```
market_cap = components['Market Capitalization']
weights = market_cap.div(market_cap.sum())
weights.sort_values().mul(100)
```

```
Stock Symbol
ABB      1.85
UPS      3.45
TM       5.96
ORCL     6.93
KO       7.03
WMT     8.50
PG       8.81
T        9.47
BABA    10.55
JPM     11.50
XOM     12.97
JNJ     12.97
Name: Market Capitalization, dtype: float64
```

Value-weighted component returns

```
index_return = (index.iloc[-1] / index.iloc[0] - 1) * 100
```

```
14.06
```

```
weighted_returns = weights.mul(index_return)  
weighted_returns.sort_values().plot(kind='barh')
```



Performance vs benchmark

```
data = index.to_frame('Index') # Convert pd.Series to pd.DataFrame  
data['SP500'] = pd.read_csv('sp500.csv', parse_dates=['Date'],  
                           index_col='Date')  
data.SP500 = data.SP500.div(data.SP500.iloc[0], axis=0).mul(100)
```



Performance vs benchmark: 30D rolling return

```
def multi_period_return(r):  
    return (np.prod(r + 1) - 1) * 100  
  
data.pct_change().rolling('30D').apply(multi_period_return).plot()
```

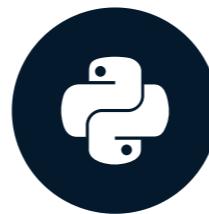


Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Index correlation & exporting to Excel

MANIPULATING TIME SERIES DATA IN PYTHON



Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence

Some additional analysis of your index

- Daily return correlations:
- Calculate among all components
- Visualize the result as heatmap
- Write results to excel using `.xls` and `.xlsx` formats:
 - Single worksheet
 - Multiple worksheets

Index components - price data

```
data = DataReader(tickers, 'google', start='2016', end='2017')['Close']  
data.info()
```

```
DatetimeIndex: 252 entries, 2016-01-04 to 2016-12-30
```

```
Data columns (total 12 columns):
```

```
ABB    252 non-null float64  
BABA   252 non-null float64  
JNJ    252 non-null float64  
JPM    252 non-null float64  
KO     252 non-null float64  
ORCL   252 non-null float64  
PG     252 non-null float64  
T      252 non-null float64  
TM     252 non-null float64  
UPS    252 non-null float64  
WMT    252 non-null float64  
XOM    252 non-null float64
```

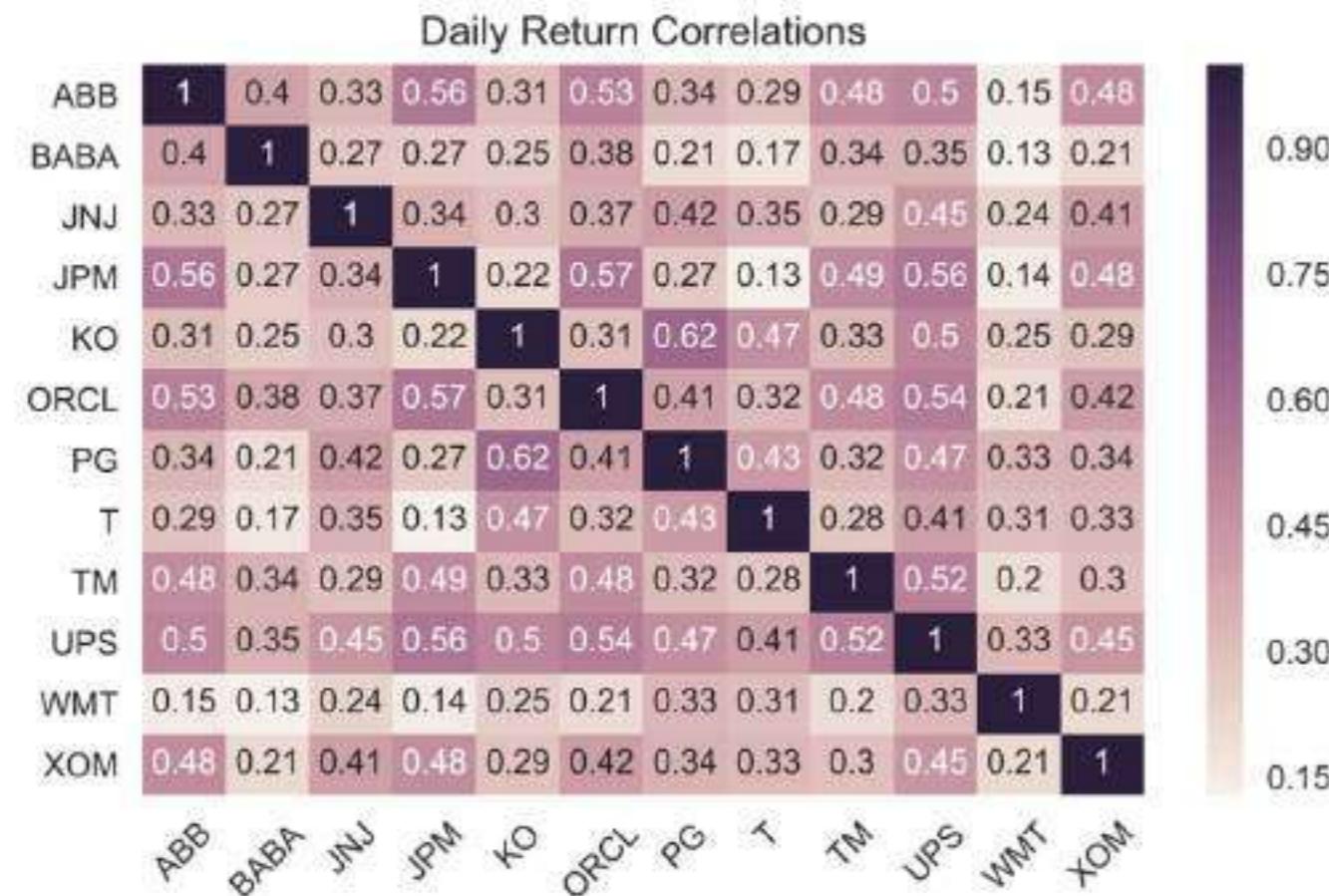
Index components: return correlations

```
daily_returns = data.pct_change()  
correlations = daily_returns.corr()
```

	ABB	BABA	JNJ	JPM	KO	ORCL	PG	T	TM	UPS	WMT	XOM
ABB	1.00	0.40	0.33	0.56	0.31	0.53	0.34	0.29	0.48	0.50	0.15	0.48
BABA	0.40	1.00	0.27	0.27	0.25	0.38	0.21	0.17	0.34	0.35	0.13	0.21
JNJ	0.33	0.27	1.00	0.34	0.30	0.37	0.42	0.35	0.29	0.45	0.24	0.41
JPM	0.56	0.27	0.34	1.00	0.22	0.57	0.27	0.13	0.49	0.56	0.14	0.48
KO	0.31	0.25	0.30	0.22	1.00	0.31	0.62	0.47	0.33	0.50	0.25	0.29
ORCL	0.53	0.38	0.37	0.57	0.31	1.00	0.41	0.32	0.48	0.54	0.21	0.42
PG	0.34	0.21	0.42	0.27	0.62	0.41	1.00	0.43	0.32	0.47	0.33	0.34
T	0.29	0.17	0.35	0.13	0.47	0.32	0.43	1.00	0.28	0.41	0.31	0.33
TM	0.48	0.34	0.29	0.49	0.33	0.48	0.32	0.28	1.00	0.52	0.20	0.30
UPS	0.50	0.35	0.45	0.56	0.50	0.54	0.47	0.41	0.52	1.00	0.33	0.45
WMT	0.15	0.13	0.24	0.14	0.25	0.21	0.33	0.31	0.20	0.33	1.00	0.21
XOM	0.48	0.21	0.41	0.48	0.29	0.42	0.34	0.33	0.30	0.45	0.21	1.00

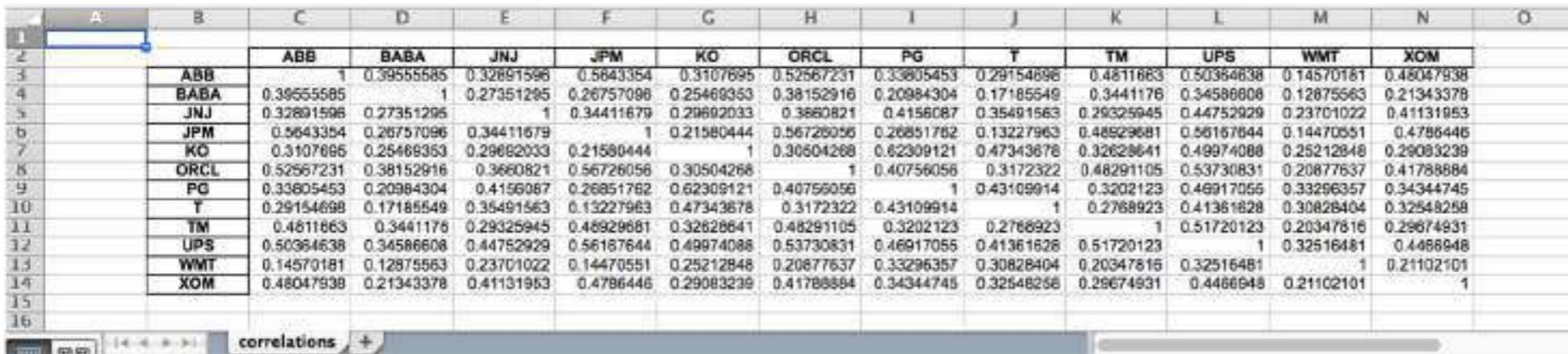
Index components: return correlations

```
sns.heatmap(correlations, annot=True)  
plt.xticks(rotation=45)  
plt.title('Daily Return Correlations')
```



Saving to a single Excel worksheet

```
correlations.to_excel(excel_writer= 'correlations.xls',  
                     sheet_name='correlations',  
                     startrow=1,  
                     startcol=1)
```



The screenshot shows a Microsoft Excel spreadsheet with the title bar 'correlations'. The spreadsheet contains a correlation matrix for twelve stocks. The columns and rows are labeled with stock tickers: ABB, BABA, JNJ, JPM, KO, ORCL, PG, T, TM, UPS, WMT, and XOM. The matrix is symmetric, with 1s on the diagonal and correlation values ranging from approximately -0.2 to 1.0 in the off-diagonal cells. The cells are colored based on their value, with darker shades indicating higher correlations.

	ABB	BABA	JNJ	JPM	KO	ORCL	PG	T	TM	UPS	WMT	XOM
ABB	1	0.39555585	0.32891598	0.5843354	0.3107695	0.52567231	0.33805453	0.29154696	0.4811663	0.50364638	0.14570181	0.48047938
BABA		1	0.27351295	0.26757098	0.25460353	0.38152916	0.20984304	0.17185549	0.3441176	0.34586608	0.12875563	0.21343378
JNJ			1	0.34411679	0.29602033	0.3860821	0.4156087	0.35491563	0.29325945	0.44752929	0.23701022	0.41131953
JPM				1	0.21580444	0.56726056	0.26851762	0.13227963	0.48929681	0.58167844	0.14470551	0.4786446
KO					1	0.30504268	0.62309121	0.47343678	0.32628641	0.49974088	0.25212848	0.29083239
ORCL						1	0.40756056	0.3172322	0.48291105	0.53730831	0.20877637	0.41788884
PG							1	0.43109914	0.3202123	0.46917056	0.33296357	0.34344745
T								1	0.2768923	0.41361828	0.30828404	0.32548258
TM									1	0.51720123	0.20347616	0.29674931
UPS										1	0.32516481	0.4468948
WMT											1	0.21102101
XOM												1

Saving to multiple Excel worksheets

```
data.index = data.index.date # Keep only date component  
with pd.ExcelWriter('stock_data.xlsx') as writer:  
    corr.to_excel(excel_writer=writer, sheet_name='correlations')  
    data.to_excel(excel_writer=writer, sheet_name='prices')  
    data.pct_change().to_excel(writer, sheet_name='returns')
```



A screenshot of Microsoft Excel displaying three tabs at the bottom: "correlations", "prices", and "returns". The "correlations" tab is active. The main area shows a data table with 12 rows (labeled 1 to 12) and 13 columns (labeled A to M). The columns represent stock symbols: ABB, BABA, JNJ, JPM, KO, ORCL, PG, T, TM, UPS, WMT, and XOM. The first row contains the column headers. The data shows price values for each stock on specific dates.

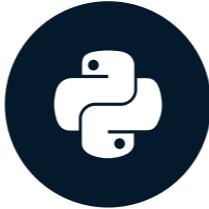
	A	B	C	D	E	F	G	H	I	J	K	L	M
1		ABB	BABA	JNJ	JPM	KO	ORCL	PG	T	TM	UPS	WMT	XOM
2	2016-01-04	17.52	76.69	100.48	63.62	42.4	35.75	78.37	34.35	121.46	94.84	61.46	77.46
3	2016-01-05	17.21	78.63	100.9	63.73	42.55	35.64	78.62	34.59	121.14	95.78	62.92	78.12
4	2016-01-06	16.92	77.33	100.39	62.81	42.32	35.82	77.86	34.06	118.38	94.42	63.55	77.47
5	2016-01-07	16.5	72.72	99.22	60.27	41.62	35.04	77.18	33.51	115.57	92.6	65.03	76.23
6	2016-01-08	16.31	70.8	98.16	58.92	41.51	34.65	75.97	33.54	113.06	91.39	63.54	74.69
7	2016-01-11	16.31	69.92	97.57	58.83	41.58	34.94	76.67	33.95	114.81	91.66	64.22	73.69
8	2016-01-12	16.66	72.68	98.24	58.96	42.12	35.37	76.51	33.9	115.83	93	63.62	75.2
9	2016-01-13	16.3	70.29	97.02	57.34	41.85	34.08	75.85	33.74	114.99	90.61	61.92	75.65
10	2016-01-14	16.73	72.25	98.89	58.2	41.88	34.79	76.15	34.3	116.29	91.15	63.06	79.12
11	2016-01-15	16.06	69.59	97	57.04	41.5	34.12	74.98	33.99	112.6	90.04	61.93	77.58
12	2016-01-19	16.26	70.13	97.5	57.01	41.92	34.55	76.73	34.51	115.02	90.35	62.56	76.4

Let's practice!

MANIPULATING TIME SERIES DATA IN PYTHON

Congratulations!

MANIPULATING TIME SERIES DATA IN PYTHON



Stefan Jansen

Founder & Lead Data Scientist at
Applied Artificial Intelligence

Congratulations!

MANIPULATING TIME SERIES DATA IN PYTHON

Introduction to Data Visualization with Matplotlib

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem

Data Scientist



Data visualization

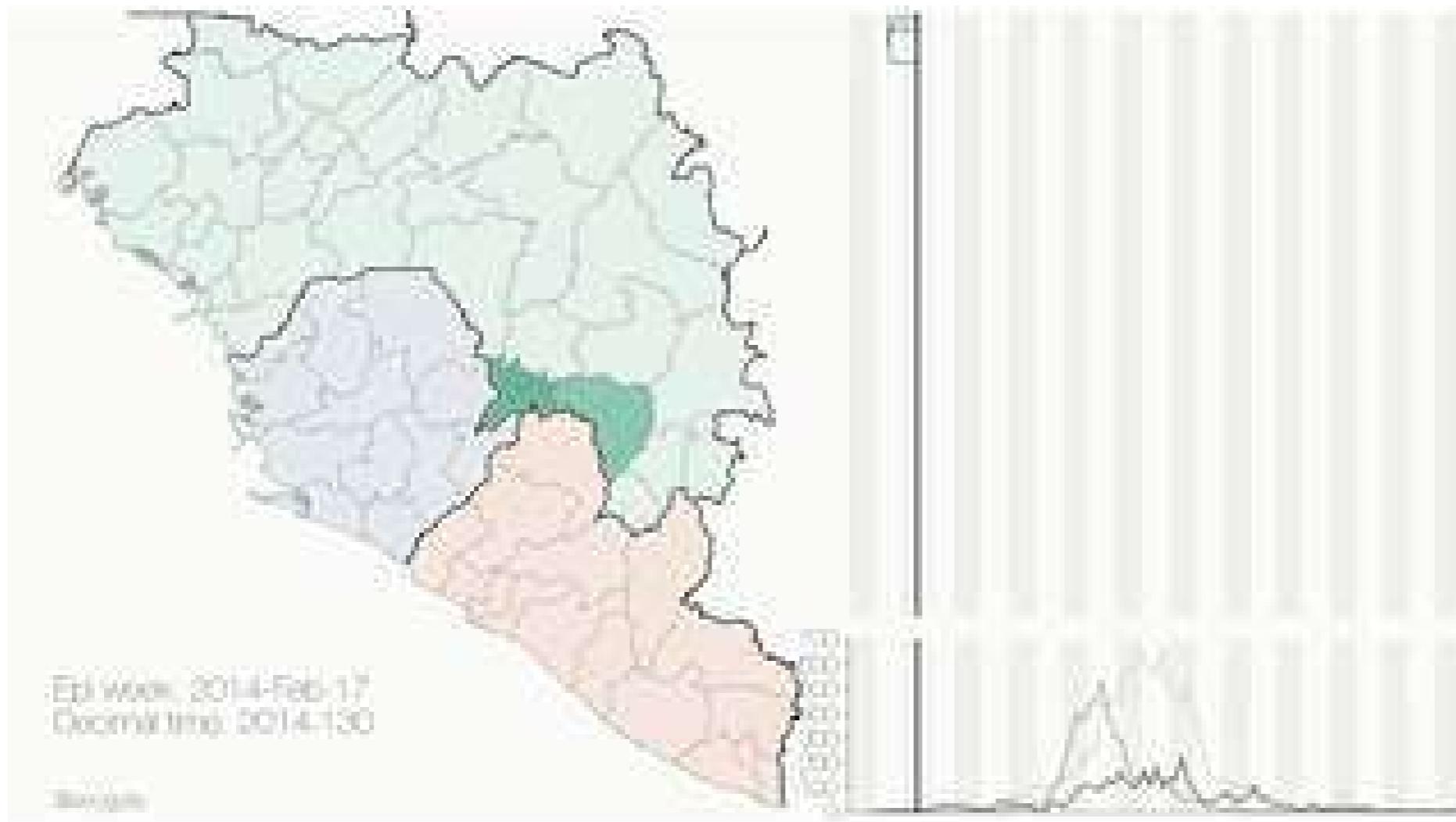
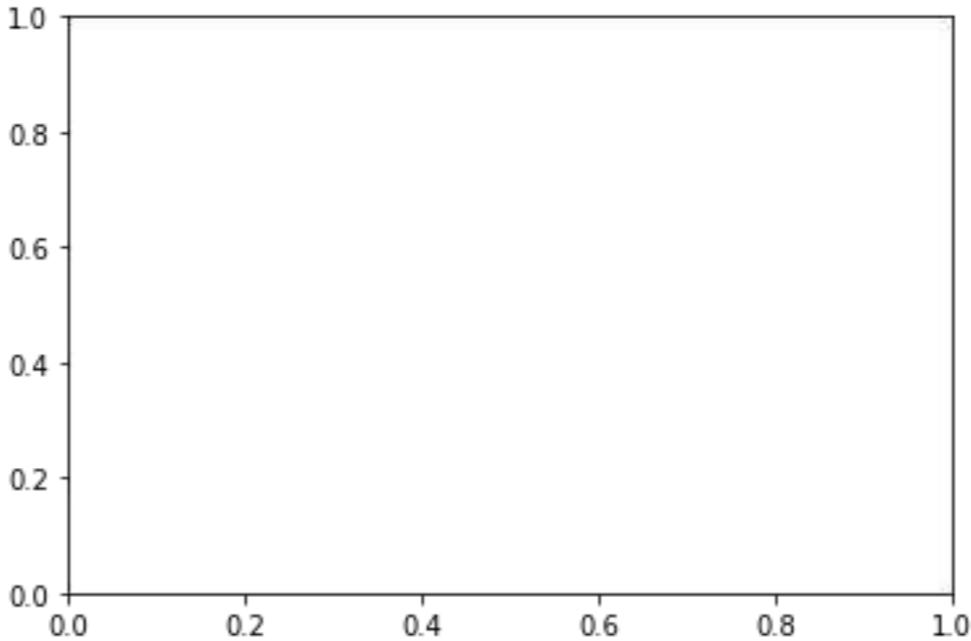


Image credit: [Gytis Dudas](#) and [Andrew Rambaut](#)

Introducing the pyplot interface

```
import matplotlib.pyplot as plt  
fig, ax = plt.subplots()  
plt.show()
```



Adding data to axes

```
seattle_weather["MONTH"]
```

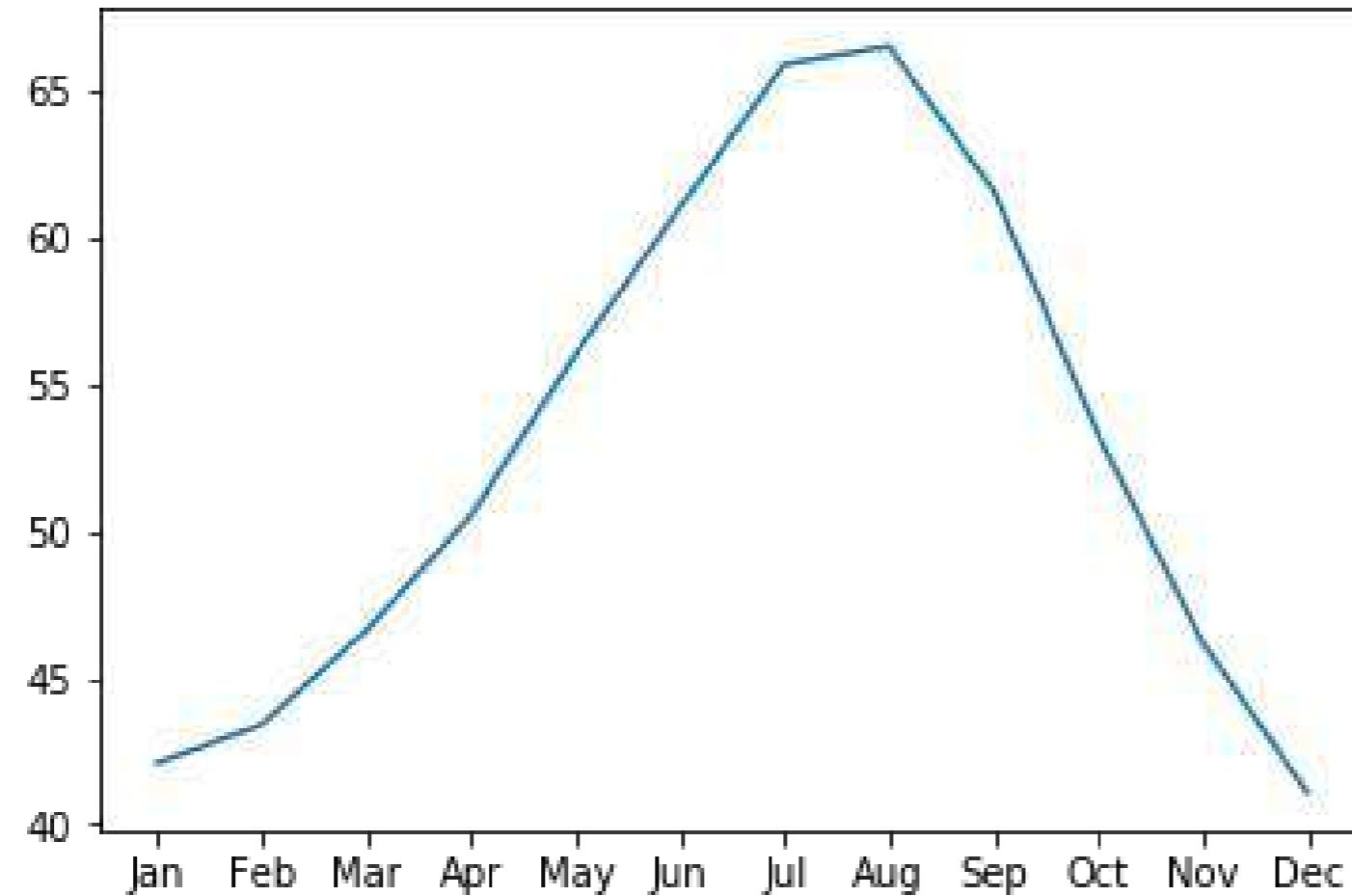
```
DATE
1 Jan
2 Feb
3 Mar
4 Apr
5 May
6 Jun
7 Jul
8 Aug
9 Sep
10 Oct
11 Nov
12 Dec
Name: MONTH, dtype: object
```

```
seattle_weather["MLY-TAVG-NORMAL"]
```

```
1 42.1
2 43.4
3 46.6
4 50.5
5 56.0
6 61.0
7 65.9
8 66.5
9 61.6
10 53.3
11 46.2
12 41.1
Name: MLY-TAVG-NORMAL, dtype: float64
```

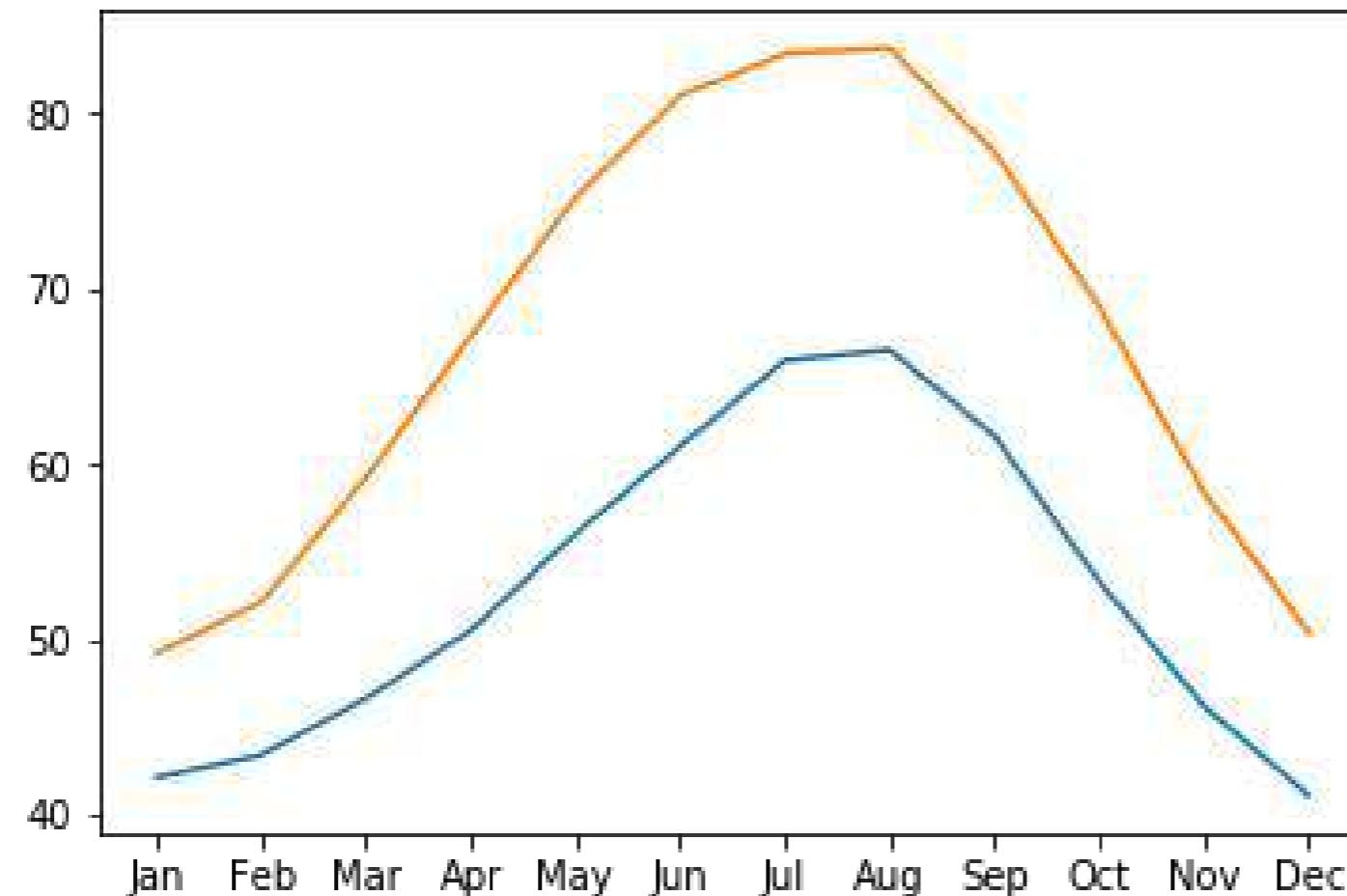
Adding data to axes

```
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-TAVG-NORMAL"])
plt.show()
```



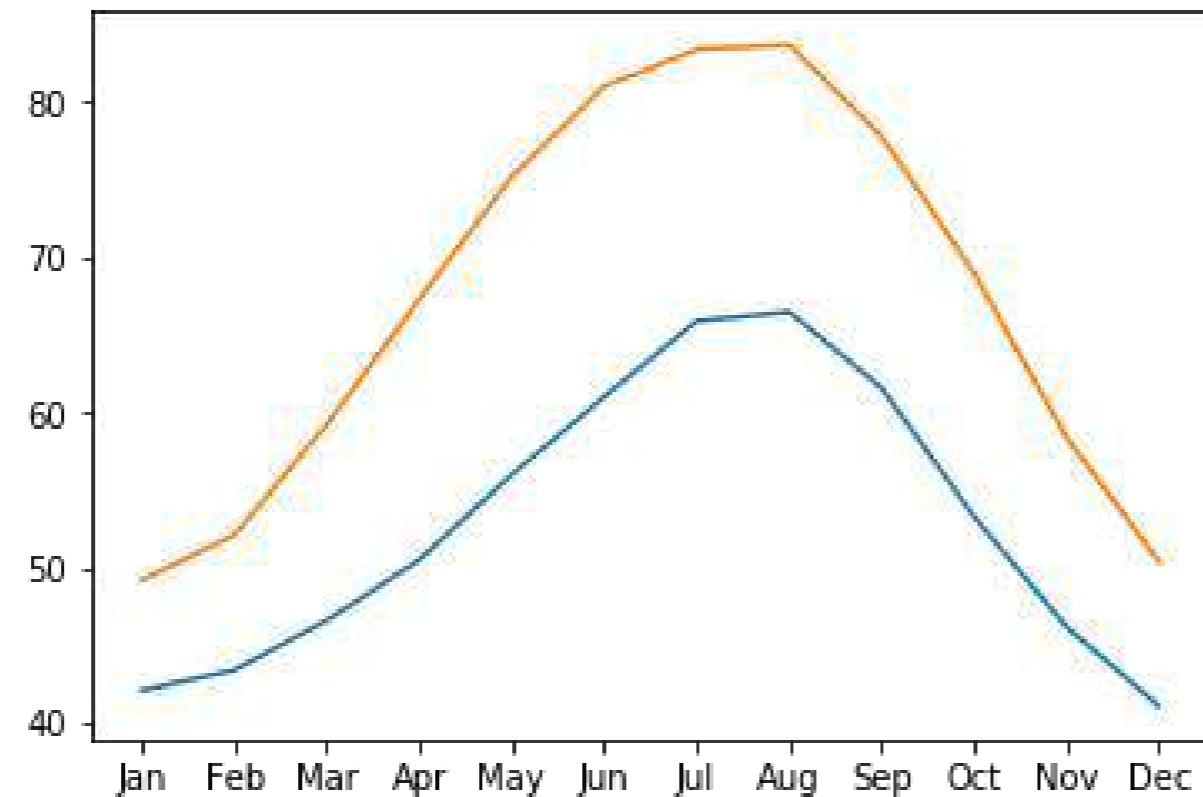
Adding more data

```
ax.plot(austin_weather["MONTH"], austin_weather["MLY-TAVG-NORMAL"])
plt.show()
```



Putting it all together

```
fig, ax = plt.subplots()  
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-TAVG-NORMAL"])  
ax.plot(austin_weather["MONTH"], austin_weather["MLY-TAVG-NORMAL"])  
plt.show()
```

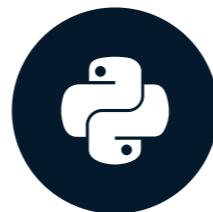


Practice making a figure!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Customizing your plots

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

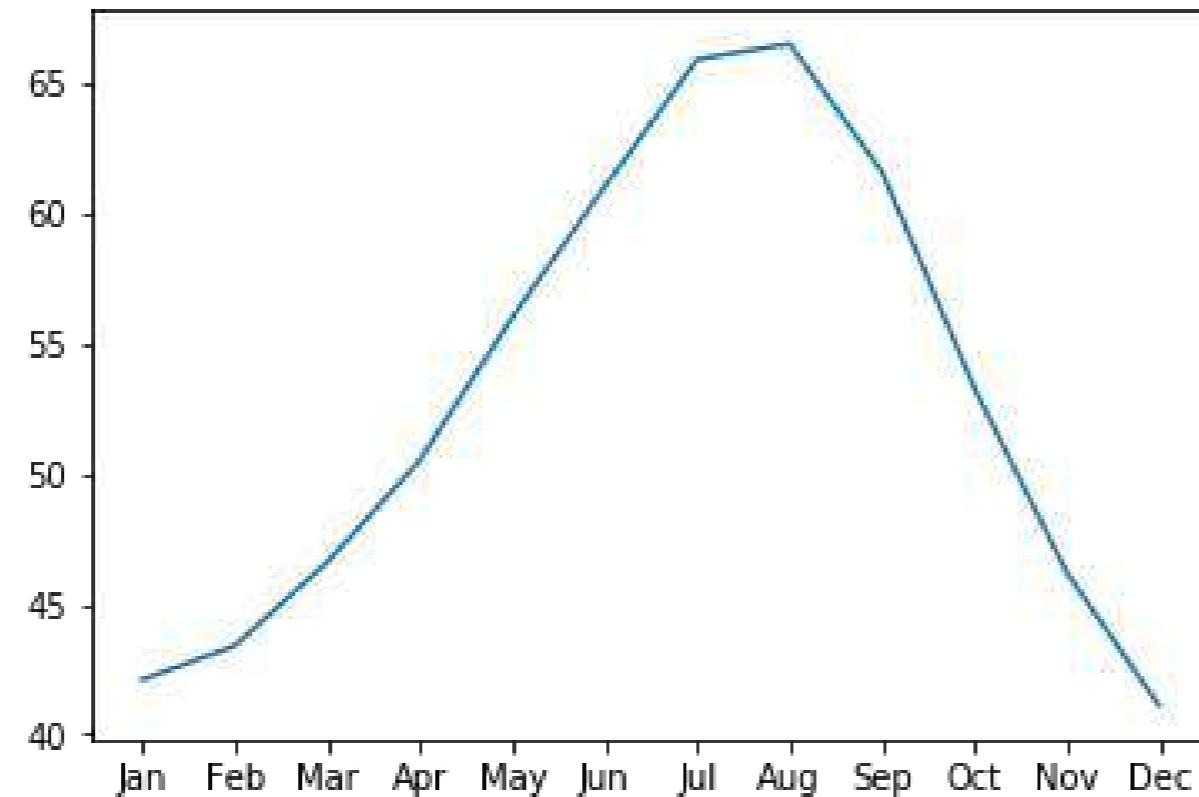


Ariel Rokem

Data Scientist

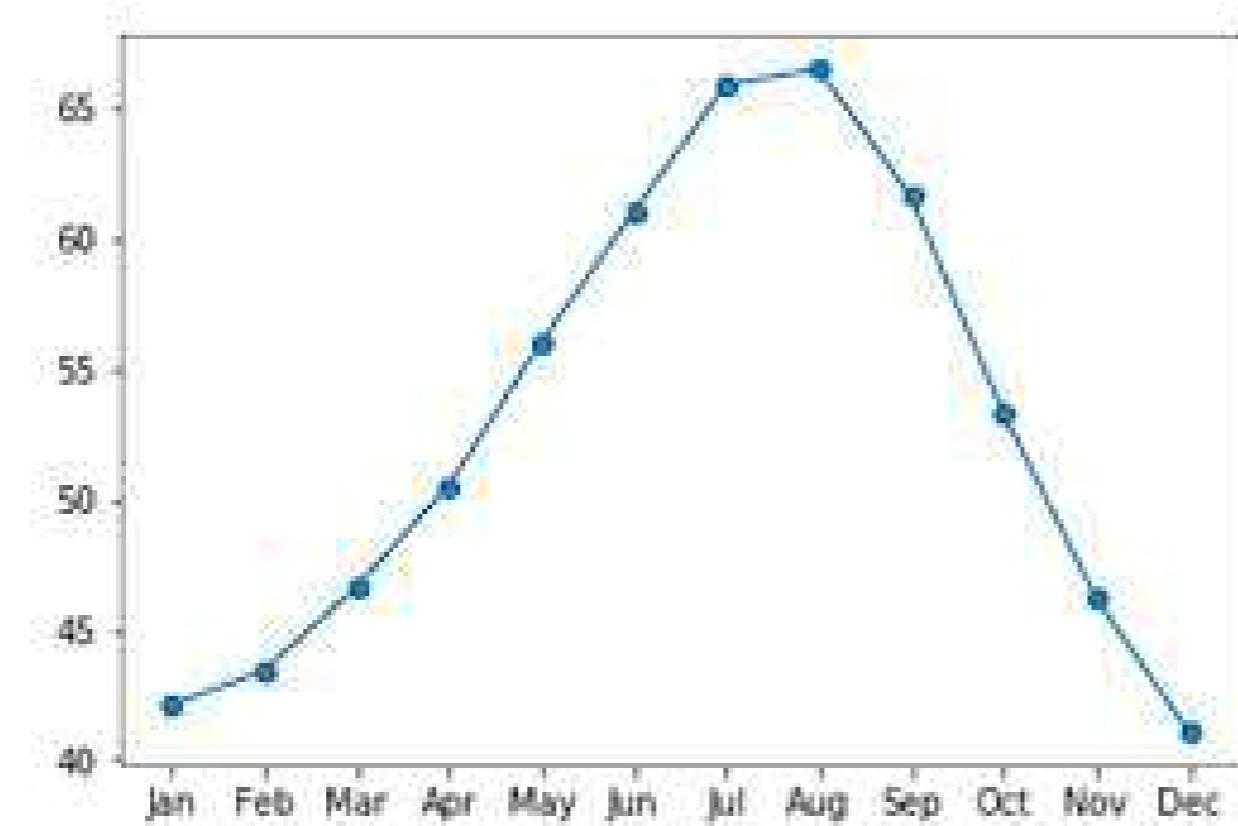
Customizing data appearance

```
ax.plot(seattle_weather["MONTH"],  
        seattle_weather["MLY-PRCP-NORMAL"])  
plt.show()
```



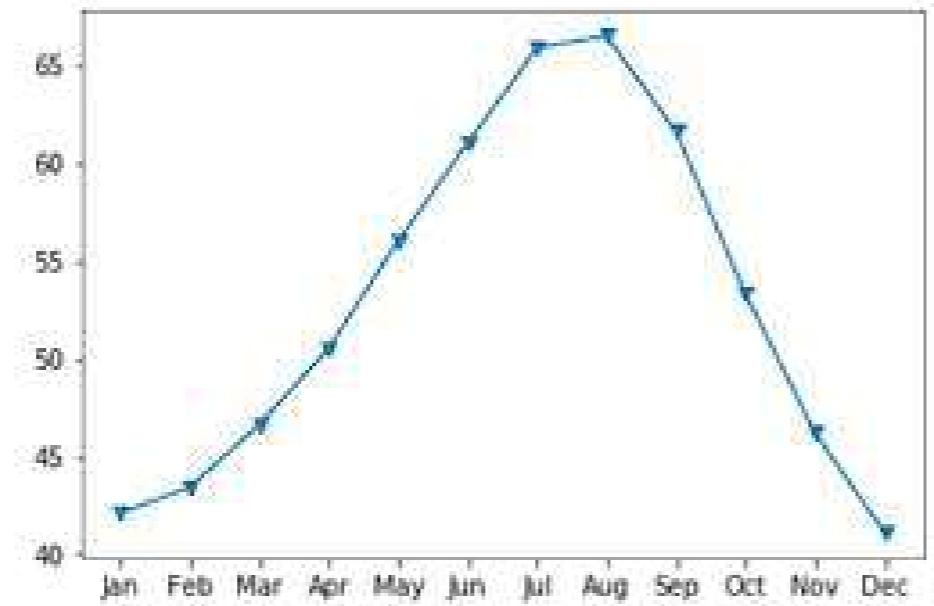
Adding markers

```
ax.plot(seattle_weather["MONTH"],  
        seattle_weather["MLY-PRCP-NORMAL"],  
        marker="o")  
  
plt.show()
```



Choosing markers

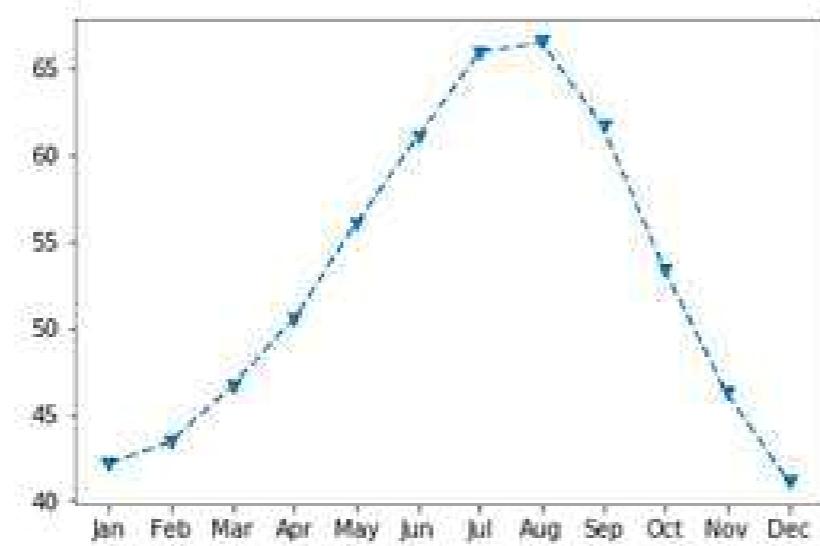
```
ax.plot(seattle_weather["MONTH"],  
        seattle_weather["MLY-PRCP-NORMAL"],  
        marker="v")  
  
plt.show()
```



https://matplotlib.org/api/markers_api.html

Setting the linestyle

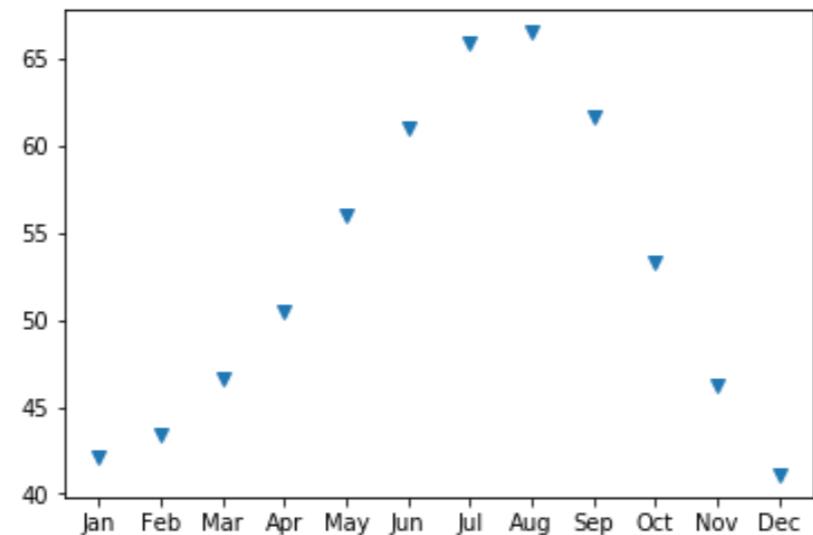
```
fig, ax = plt.subplots()  
ax.plot(seattle_weather["MONTH"],  
        seattle_weather["MLY-TAVG-NORMAL"],  
        marker="v", linestyle="--")  
plt.show()
```



https://matplotlib.org/gallery/lines_bars_and_markers/line_styles_reference.html

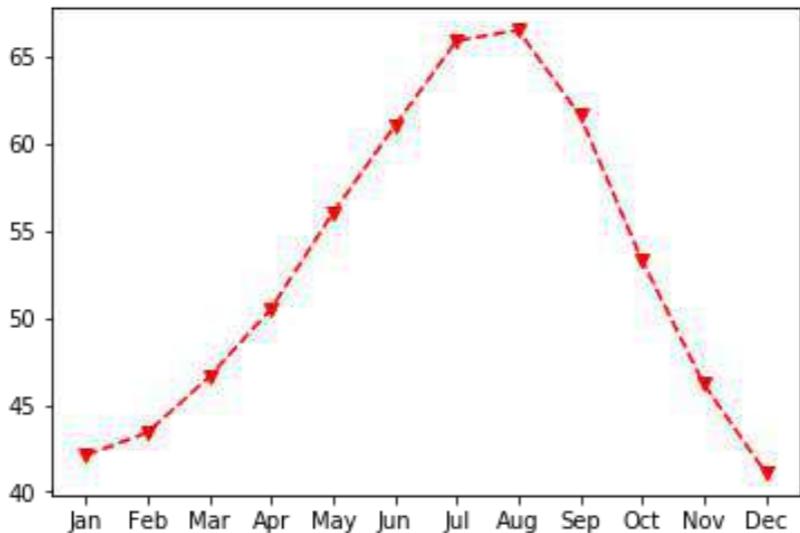
Eliminating lines with linestyle

```
fig, ax = plt.subplots()  
ax.plot(seattle_weather["MONTH"],  
        seattle_weather["MLY-TAVG-NORMAL"],  
        marker="v", linestyle="None")  
plt.show()
```



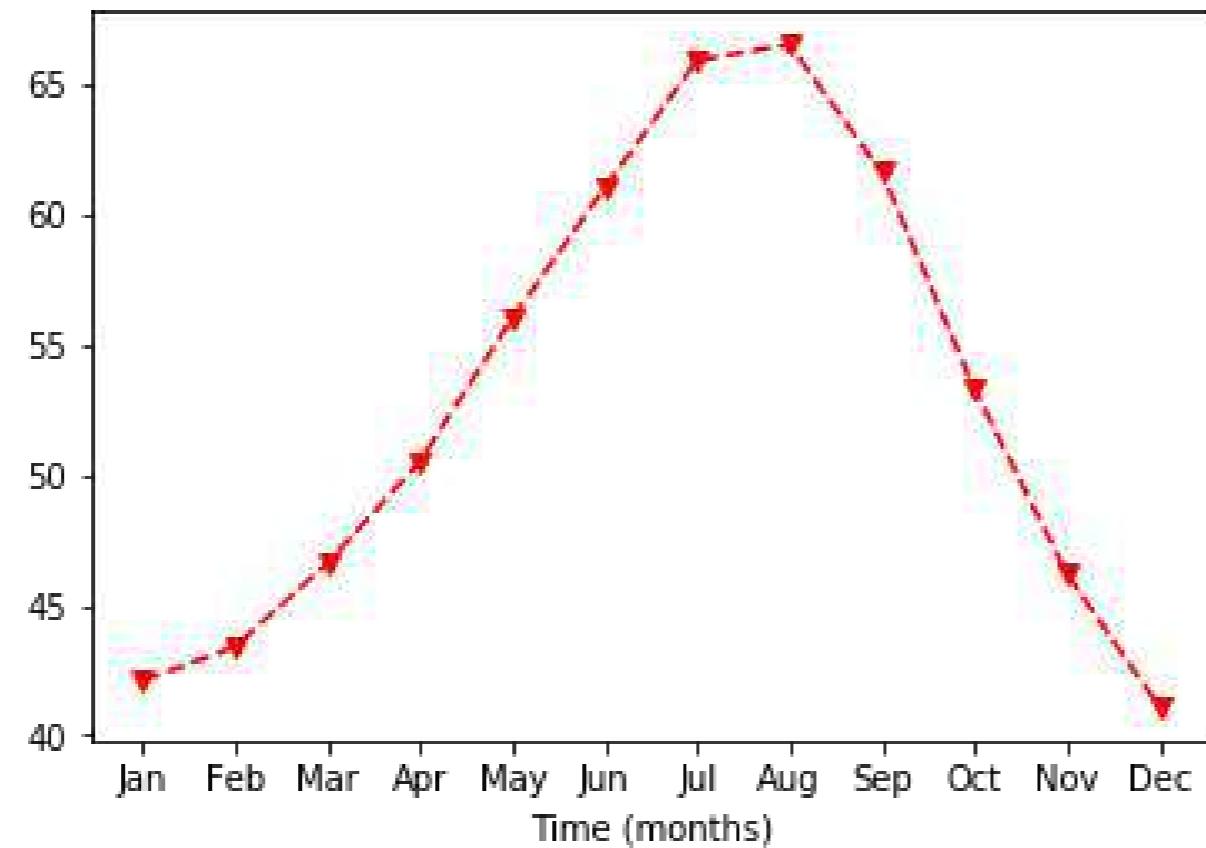
Choosing color

```
fig, ax = plt.subplots()  
ax.plot(seattle_weather["MONTH"],  
        seattle_weather["MLY-TAVG-NORMAL"],  
        marker="v", linestyle="--", color="r")  
plt.show()
```



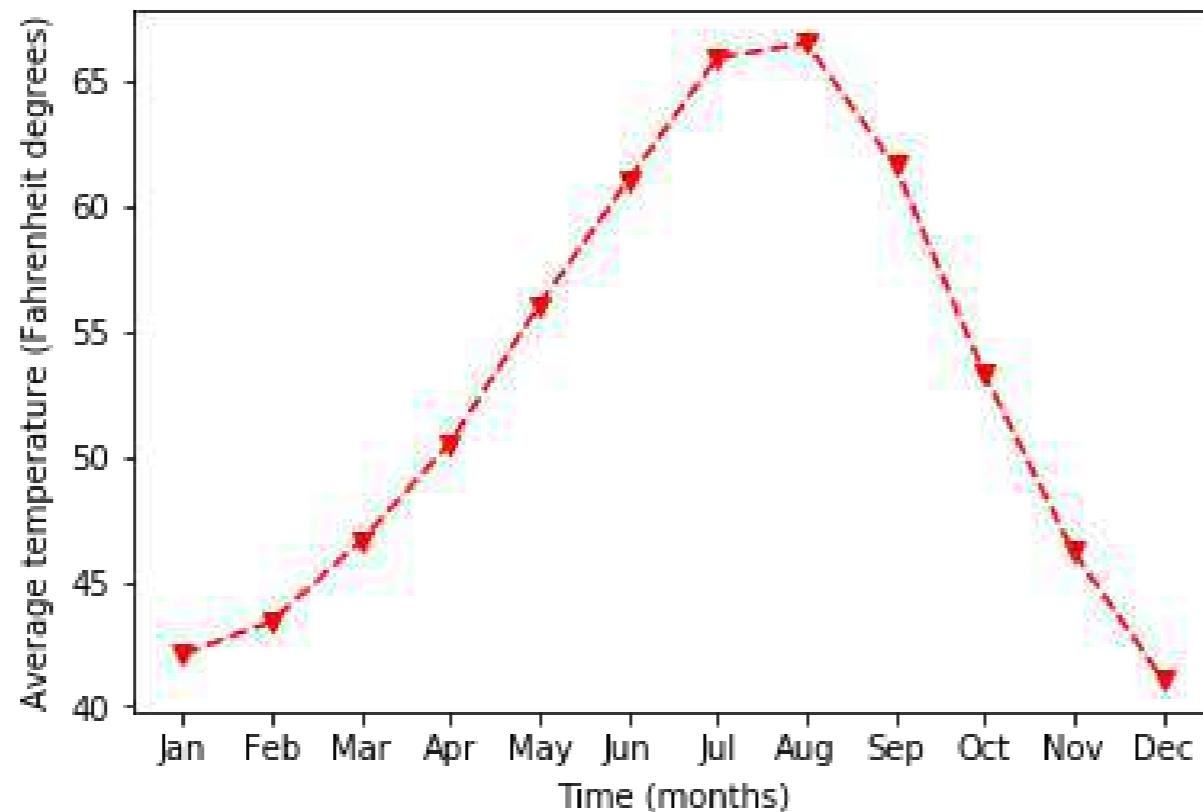
Customizing the axes labels

```
ax.set_xlabel("Time (months)")  
plt.show()
```



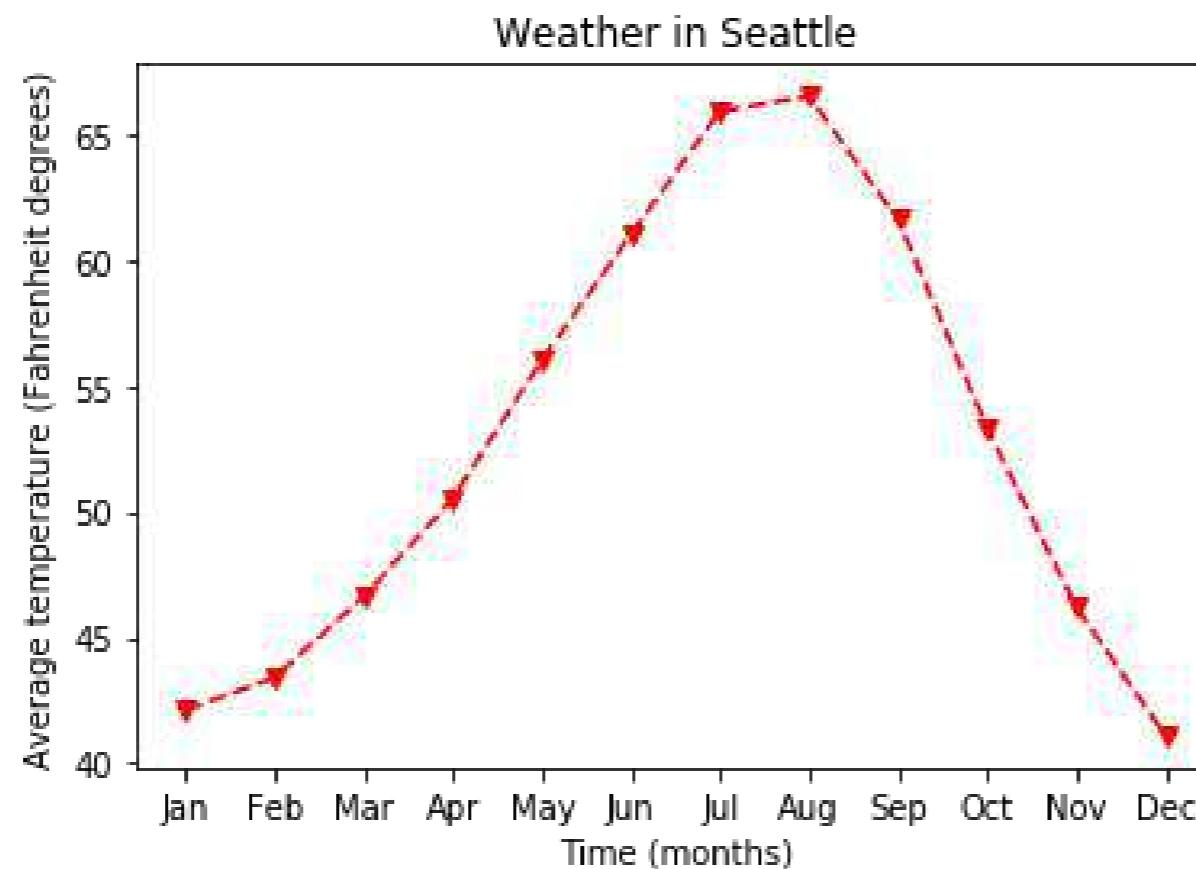
Setting the y axis label

```
ax.set_xlabel("Time (months)")  
ax.set_ylabel("Average temperature (Fahrenheit degrees)")  
plt.show()
```



Adding a title

```
ax.set_title("Weather in Seattle")  
plt.show()
```

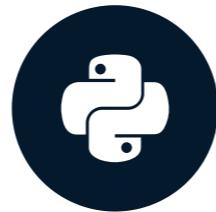


Practice customizing your plots!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Small multiples

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

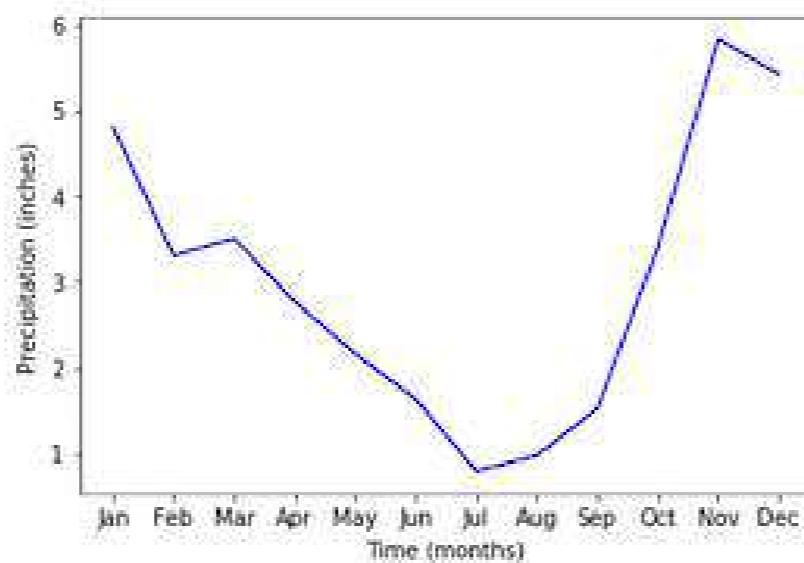


Ariel Rokem

Data Scientist

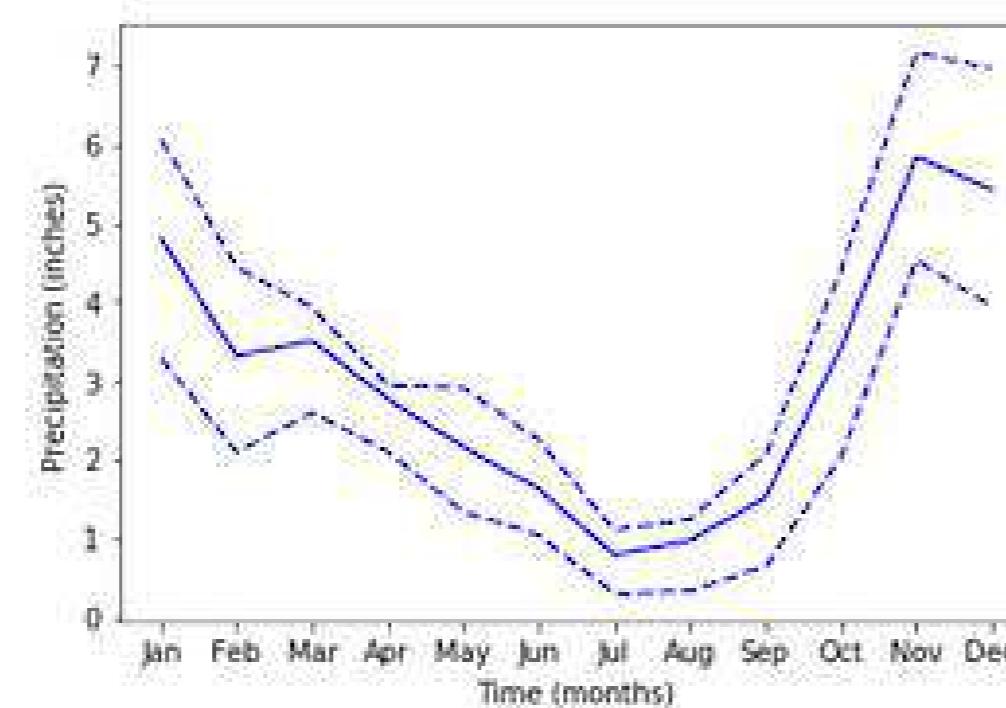
Adding data

```
ax.plot(seattle_weather["MONTH"],  
        seattle_weather["MLY-PRCP-NORMAL"],  
        color='b')  
  
ax.set_xlabel("Time (months)")  
ax.set_ylabel("Precipitation (inches)")  
plt.show()
```



Adding more data

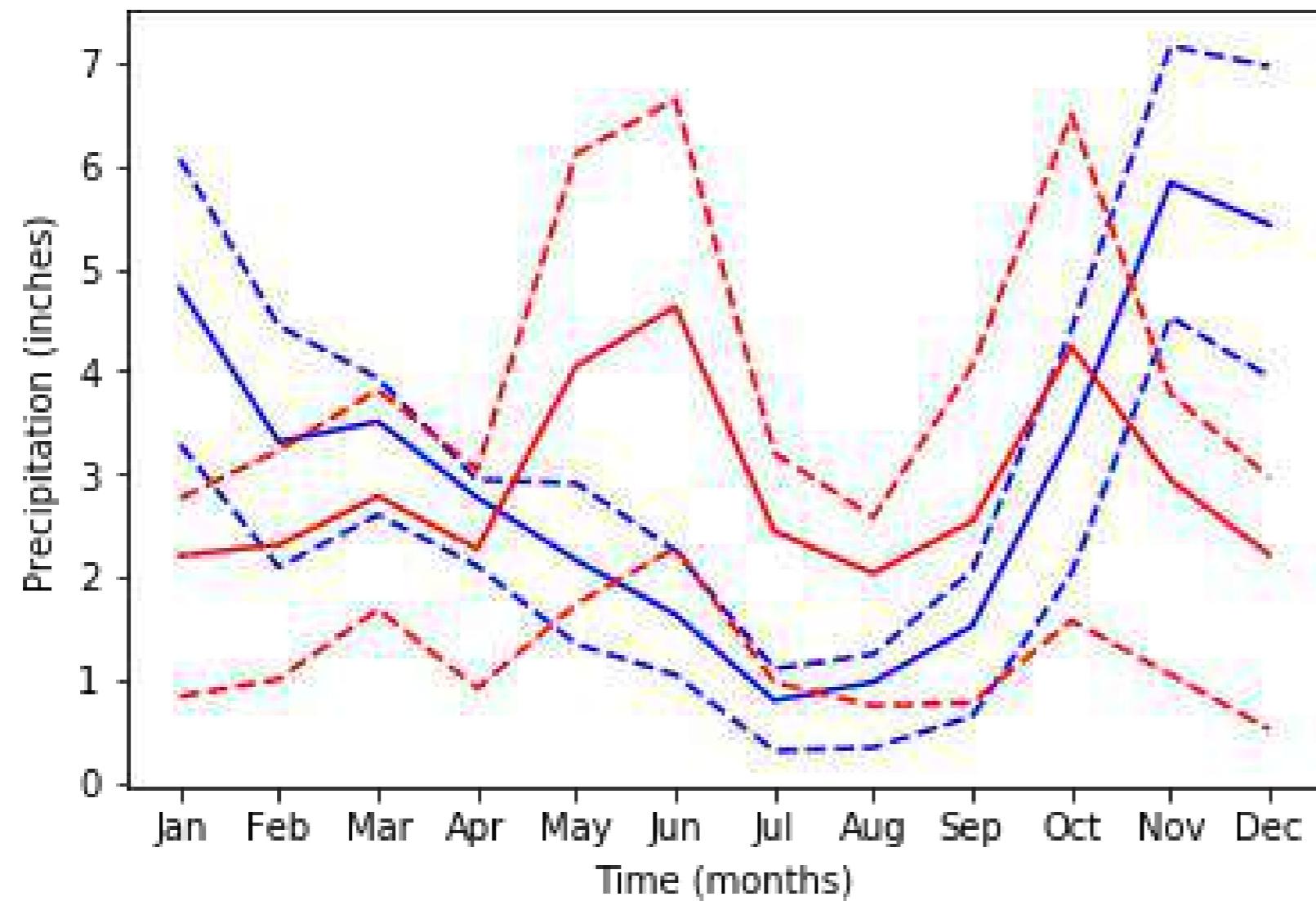
```
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-PRCP-25PCTL"],  
        linestyle='--', color='b')  
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-PRCP-75PCTL"],  
        linestyle='--', color=color)  
plt.show()
```



And more data

```
ax.plot(austin_weather["MONTH"], austin_weather["MLY-PRCP-NORMAL"],
        color='r')
ax.plot(austin_weather["MONTH"], austin_weather["MLY-PRCP-25PCTL"],
        linestyle='--', color='r')
ax.plot(austin_weather["MONTH"], austin_weather["MLY-PRCP-75PCTL"],
        linestyle='--', color='r')
plt.show()
```

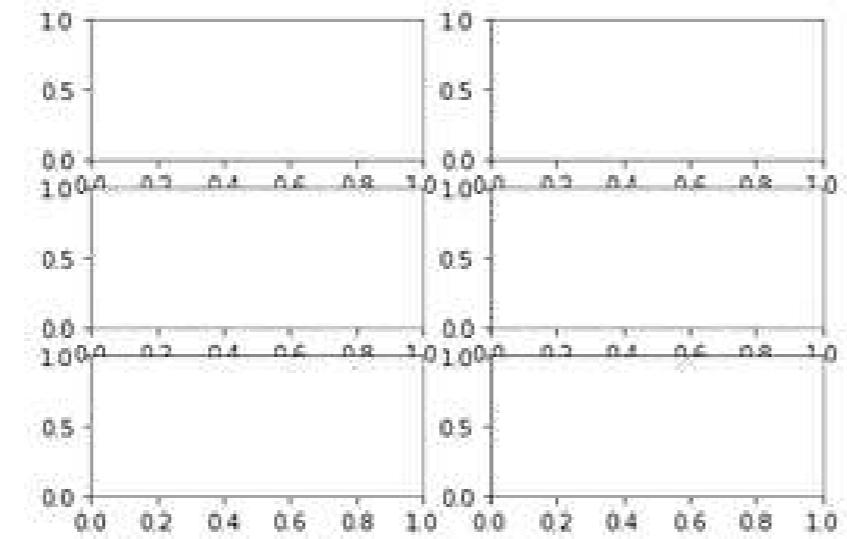
Too much data!



Small multiples with plt.subplots

```
fig, ax = plt.subplots()
```

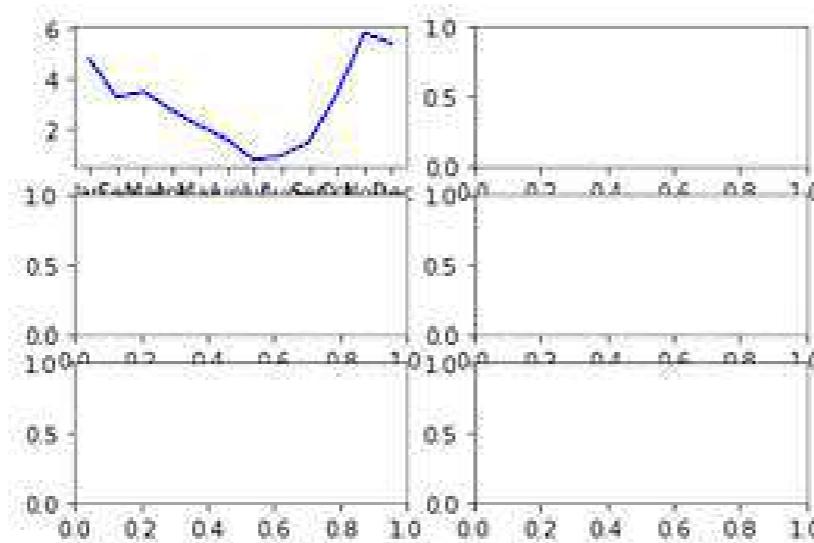
```
fig, ax = plt.subplots(3, 2)  
plt.show()
```



Adding data to subplots

```
ax.shape  
(3, 2)
```

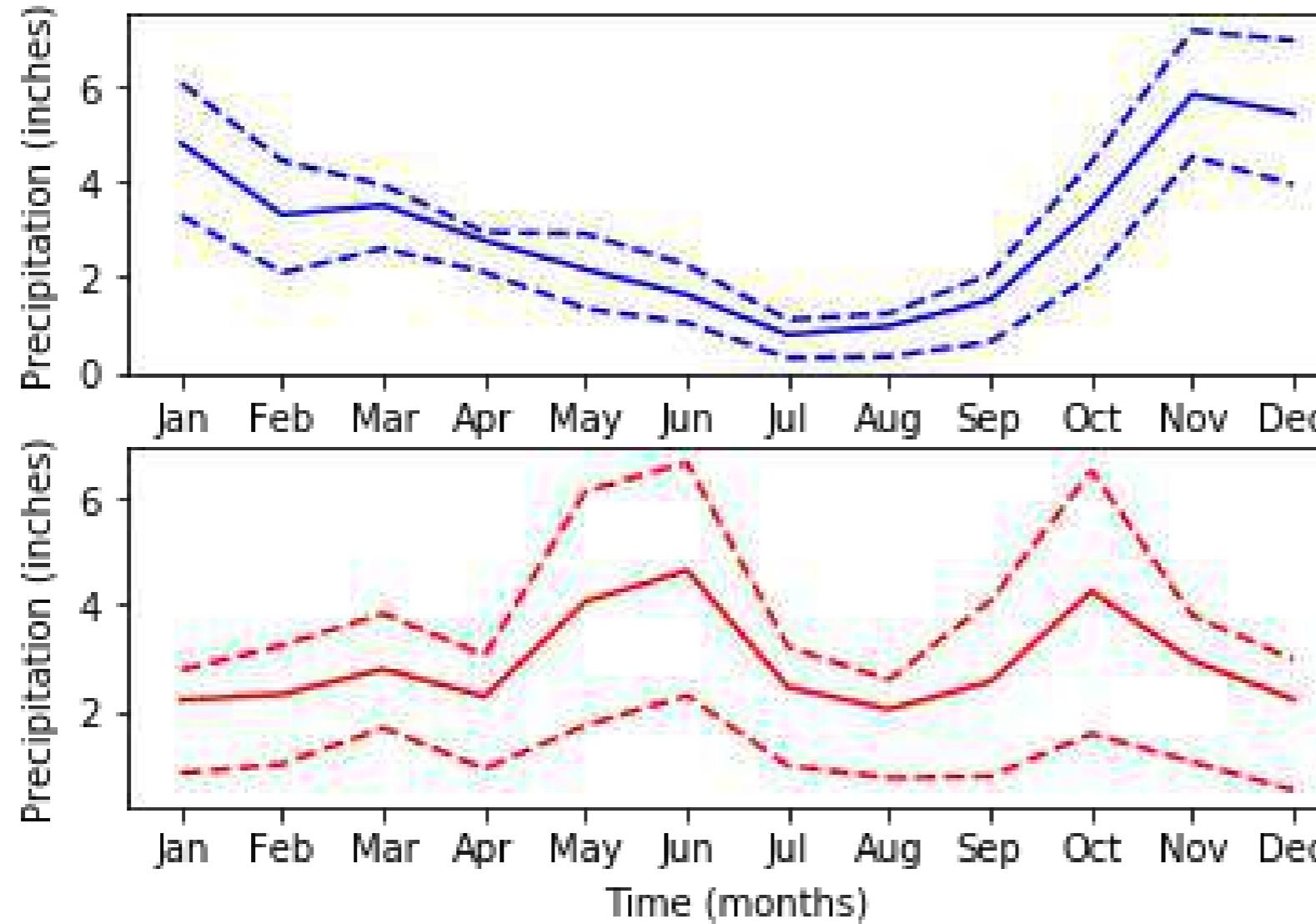
```
ax[0, 0].plot(seattle_weather["MONTH"],  
               seattle_weather["MLY-PRCP-NORMAL"],  
               color='b')  
  
plt.show()
```



Subplots with data

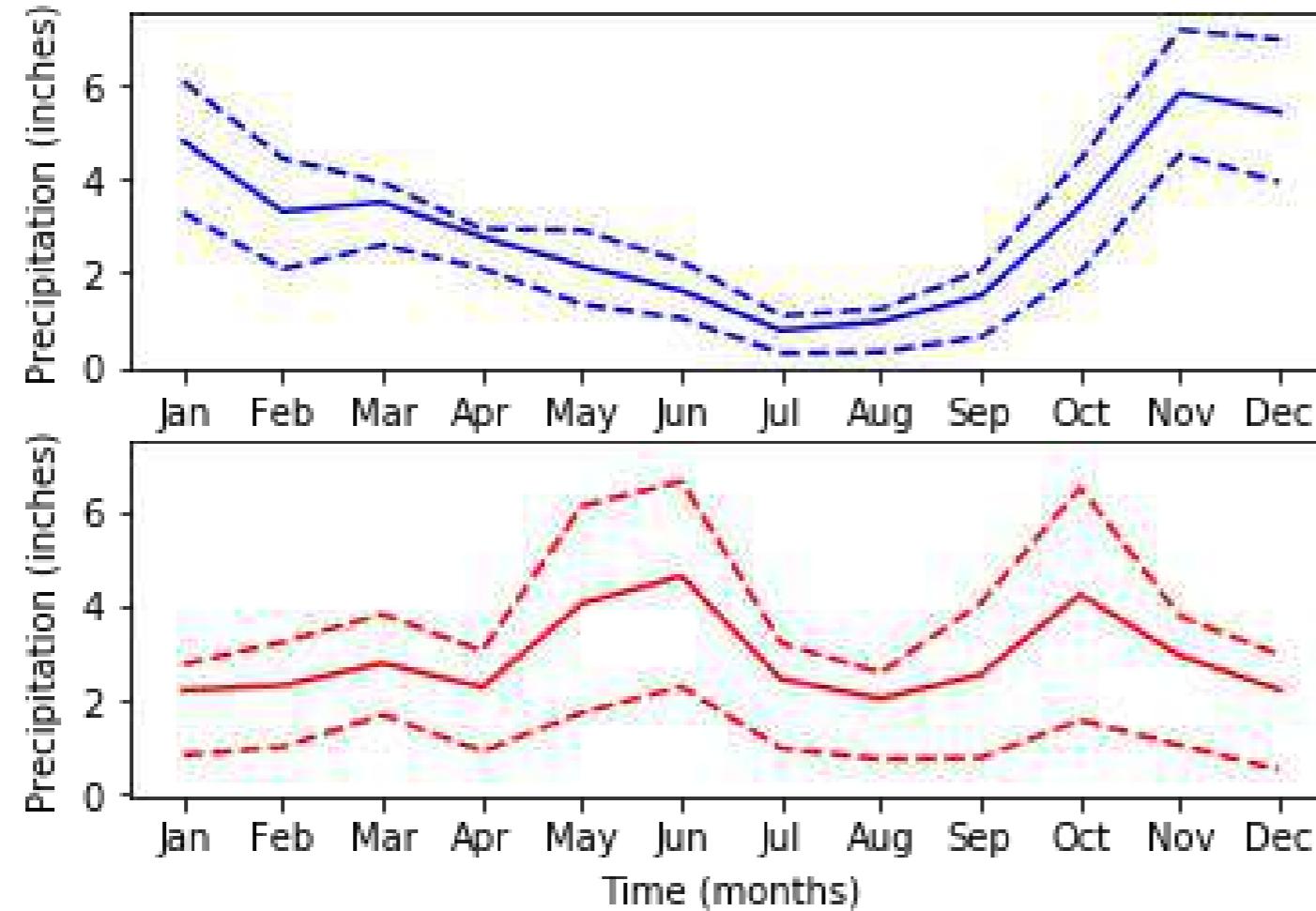
```
fig, ax = plt.subplots(2, 1)
ax[0].plot(seattle_weather["MONTH"], seattle_weather["MLY-PRCP-NORMAL"],
            color='b')
ax[0].plot(seattle_weather["MONTH"], seattle_weather["MLY-PRCP-25PCTL"],
            linestyle='--', color='b')
ax[0].plot(seattle_weather["MONTH"], seattle_weather["MLY-PRCP-75PCTL"],
            linestyle='--', color='b')
ax[1].plot(austin_weather["MONTH"], austin_weather["MLY-PRCP-NORMAL"],
            color='r')
ax[1].plot(austin_weather["MONTH"], austin_weather["MLY-PRCP-25PCTL"],
            linestyle='--', color='r')
ax[1].plot(austin_weather["MONTH"], austin_weather["MLY-PRCP-75PCTL"],
            linestyle='--', color='r')
ax[0].set_ylabel("Precipitation (inches)")
ax[1].set_ylabel("Precipitation (inches)")
ax[1].set_xlabel("Time (months)")
plt.show()
```

Subplots with data



Sharing the y-axis range

```
fig, ax = plt.subplots(2, 1, sharey=True)
```

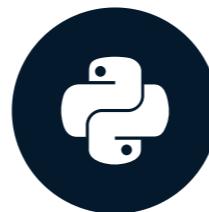


Practice making subplots!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Plotting time-series data

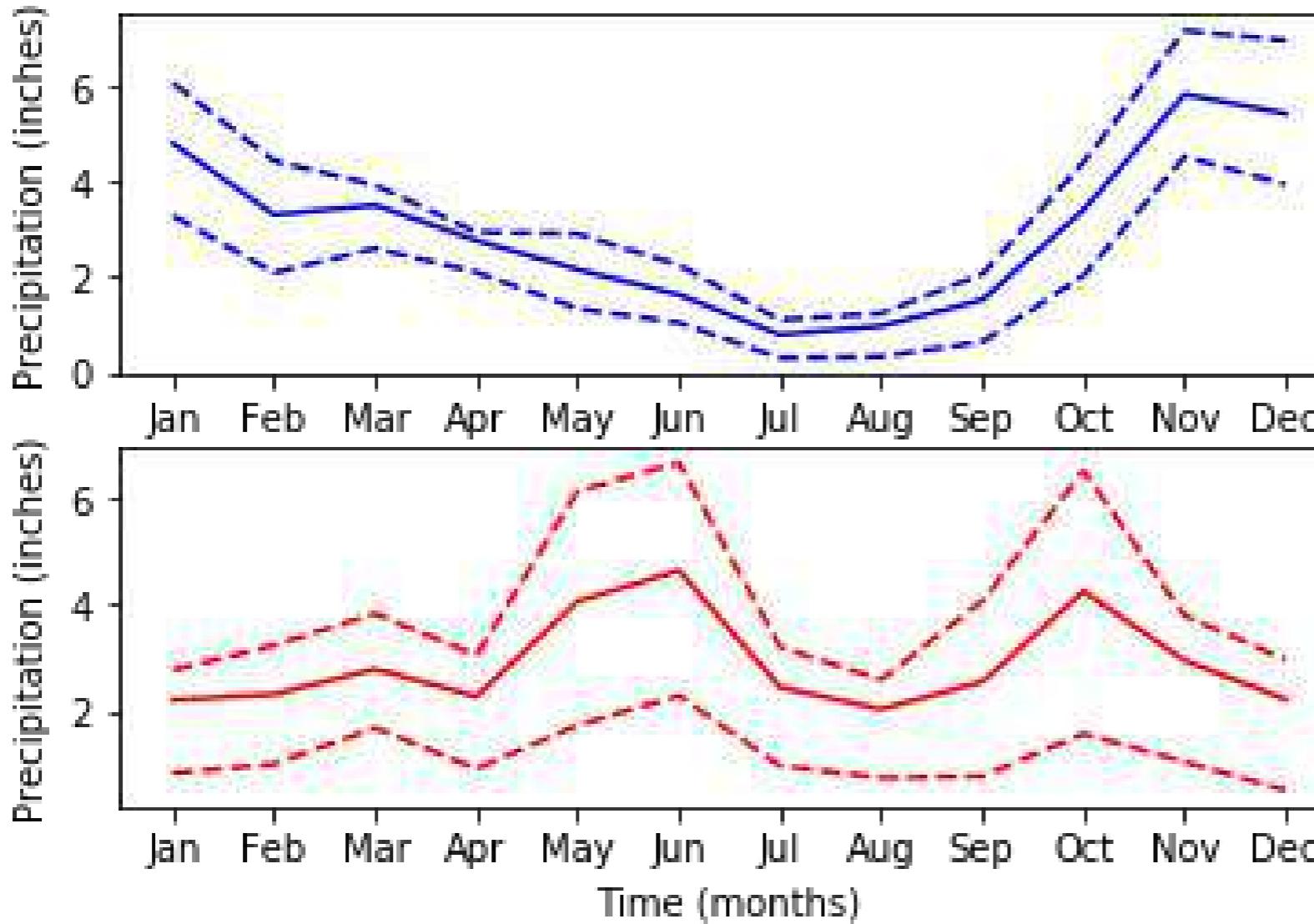
INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB



Ariel Rokem

Data Scientist

Time-series data



Climate change time-series

```
date,co2,relative_temp  
1958-03-06,315.71,0.1  
1958-04-06,317.45,0.01  
1958-05-06,317.5,0.08  
1958-06-06,-99.99,-0.05  
1958-07-06,315.86,0.06  
1958-08-06,314.93,-0.06  
...  
2016-08-06,402.27,0.98  
2016-09-06,401.05,0.87  
2016-10-06,401.59,0.89  
2016-11-06,403.55,0.93  
2016-12-06,404.45,0.81
```

DatetimeIndex

climate_change.index

```
DatetimeIndex(['1958-03-06', '1958-04-06', '1958-05-06', '1958-06-06',
                 '1958-07-06', '1958-08-06', '1958-09-06', '1958-10-06',
                 '1958-11-06', '1958-12-06',
                 ...
                 '2016-03-06', '2016-04-06', '2016-05-06', '2016-06-06',
                 '2016-07-06', '2016-08-06', '2016-09-06', '2016-10-06',
                 '2016-11-06', '2016-12-06'],
                dtype='datetime64[ns]', name='date', length=706, freq=None)
```

Time-series data

```
climate_change['relative_temp']
```

```
0      0.10  
1      0.01  
2      0.08  
3     -0.05  
4      0.06  
5     -0.06  
6     -0.03  
7      0.04  
...  
701    0.98  
702    0.87  
703    0.89  
704    0.93  
705    0.81  
Name:co2, Length: 706, dtype: float64
```

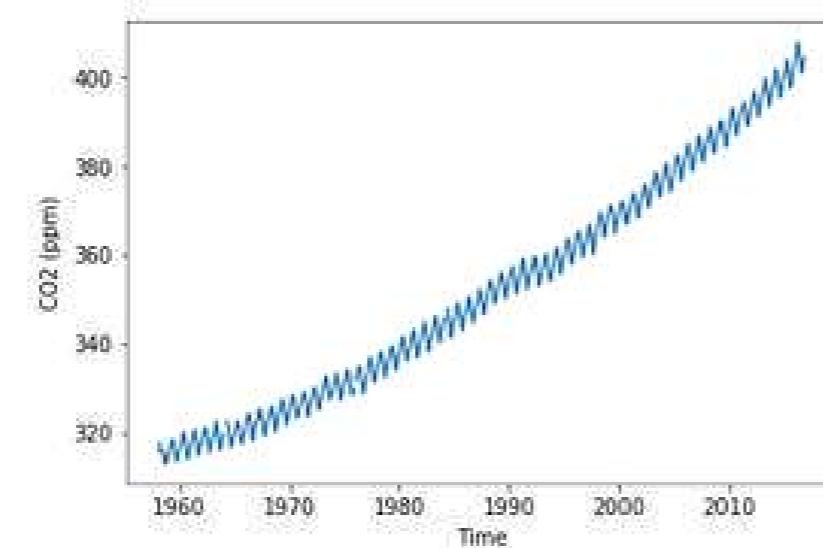
```
climate_change['co2']
```

```
0      315.71  
1      317.45  
2      317.50  
3        NaN  
4      315.86  
5      314.93  
6      313.20  
7        NaN  
...  
701    402.27  
702    401.05  
703    401.59  
704    403.55  
705    404.45  
Name:co2, Length: 706, dtype: float64
```

Plotting time-series data

```
import matplotlib.pyplot as plt  
fig, ax = plt.subplots()
```

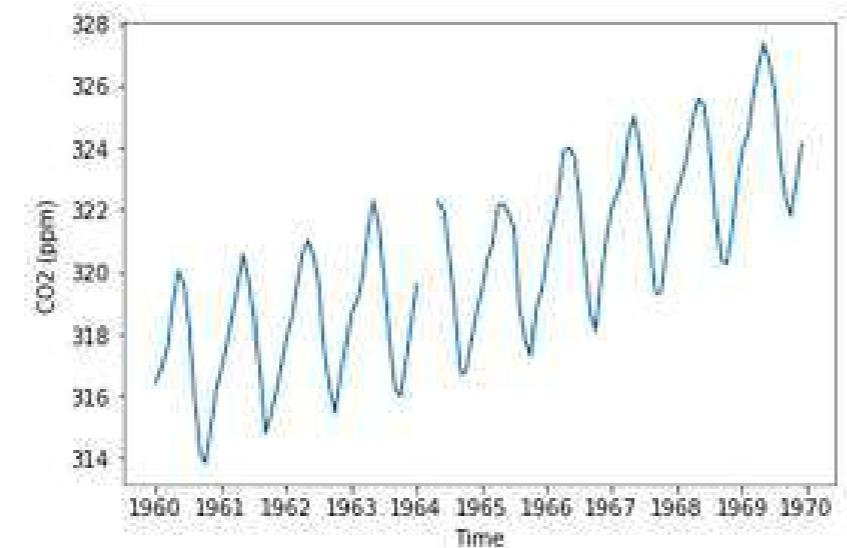
```
ax.plot(climate_change.index, climate_change['co2'])  
ax.set_xlabel('Time')  
ax.set_ylabel('CO2 (ppm)')  
plt.show()
```



Zooming in on a decade

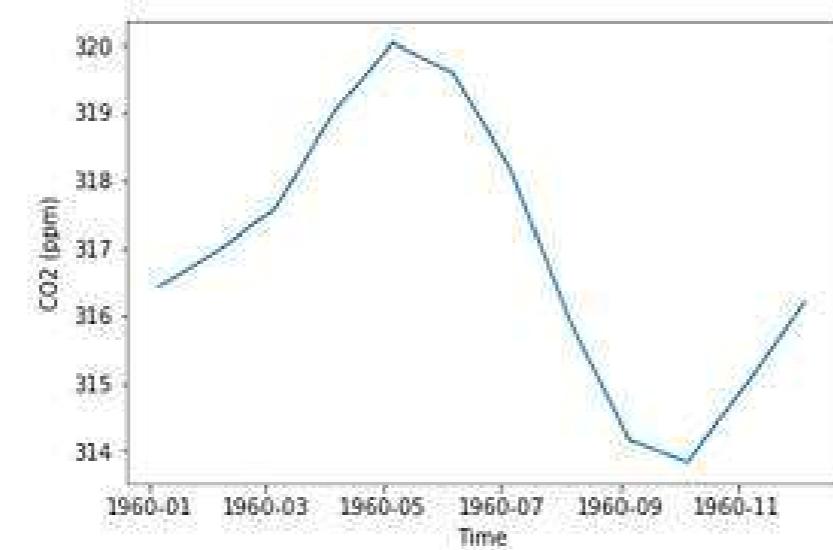
```
sixties = climate_change["1960-01-01":"1969-12-31"]
```

```
fig, ax = plt.subplots()  
ax.plot(sixties.index, sixties['co2'])  
ax.set_xlabel('Time')  
ax.set_ylabel('CO2 (ppm)')  
plt.show()
```



Zooming in on one year

```
sixty_nine = climate_change["1969-01-01":"1969-12-31"]
fig, ax = plt.subplots()
ax.plot(sixty_nine.index, sixty_nine['co2'])
ax.set_xlabel('Time')
ax.set_ylabel('CO2 (ppm)')
plt.show()
```



Let's practice time-series plotting!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Plotting time-series with different variables

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem

Data Scientist



Plotting two time-series together

```
import pandas as pd  
climate_change = pd.read_csv('climate_change.csv',  
                             parse_dates=["date"],  
                             index_col="date")
```

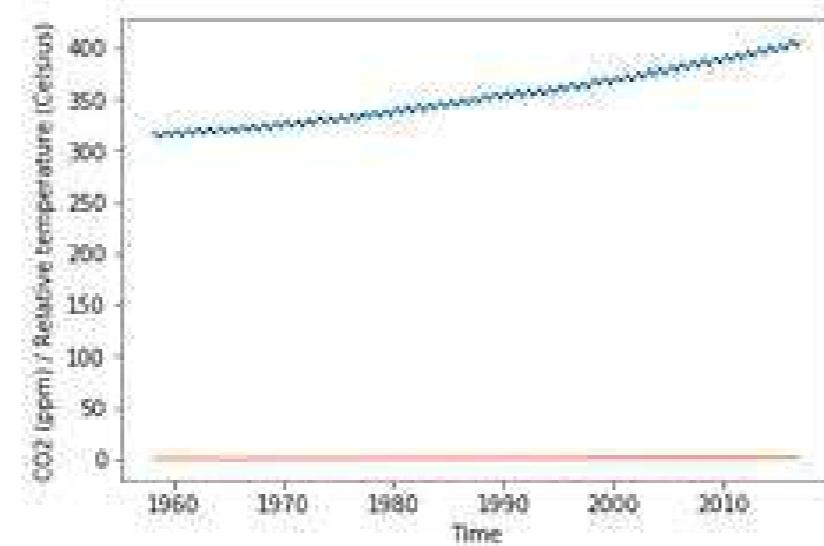
```
climate_change
```

```
      co2  relative_temp  
date  
1958-03-06  315.71        0.10  
1958-04-06  317.45        0.01  
1958-07-06  315.86        0.06  
...          ...         ...  
2016-11-06  403.55        0.93  
2016-12-06  404.45        0.81
```

[706 rows x 2 columns]

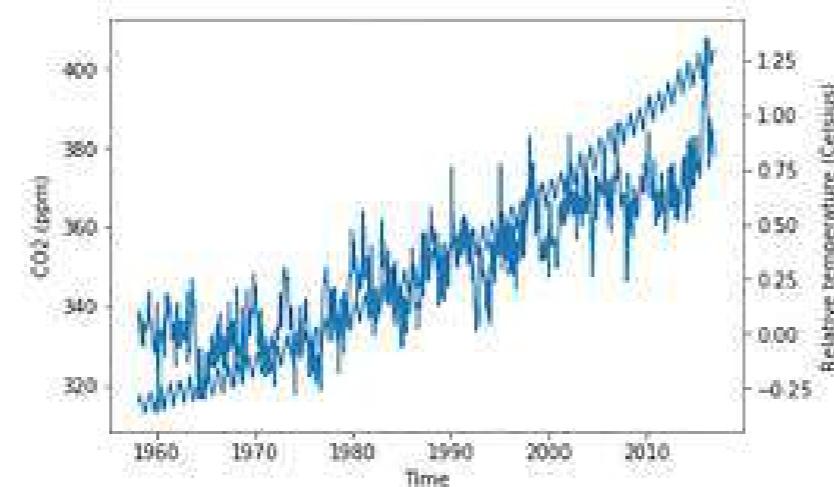
Plotting two time-series together

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(climate_change.index, climate_change["co2"])
ax.plot(climate_change.index, climate_change["relative_temp"])
ax.set_xlabel('Time')
ax.set_ylabel('CO2 (ppm) / Relative temperature')
plt.show()
```



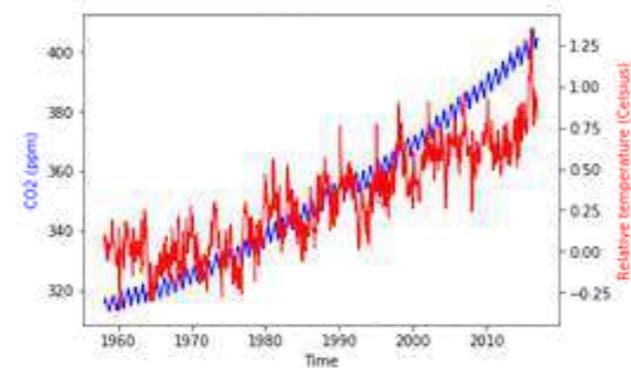
Using twin axes

```
fig, ax = plt.subplots()  
ax.plot(climate_change.index, climate_change["co2"])  
ax.set_xlabel('Time')  
ax.set_ylabel('CO2 (ppm)')  
ax2 = ax.twinx()  
ax2.plot(climate_change.index, climate_change["relative_temp"])  
ax2.set_ylabel('Relative temperature (Celsius)')  
plt.show()
```



Separating variables by color

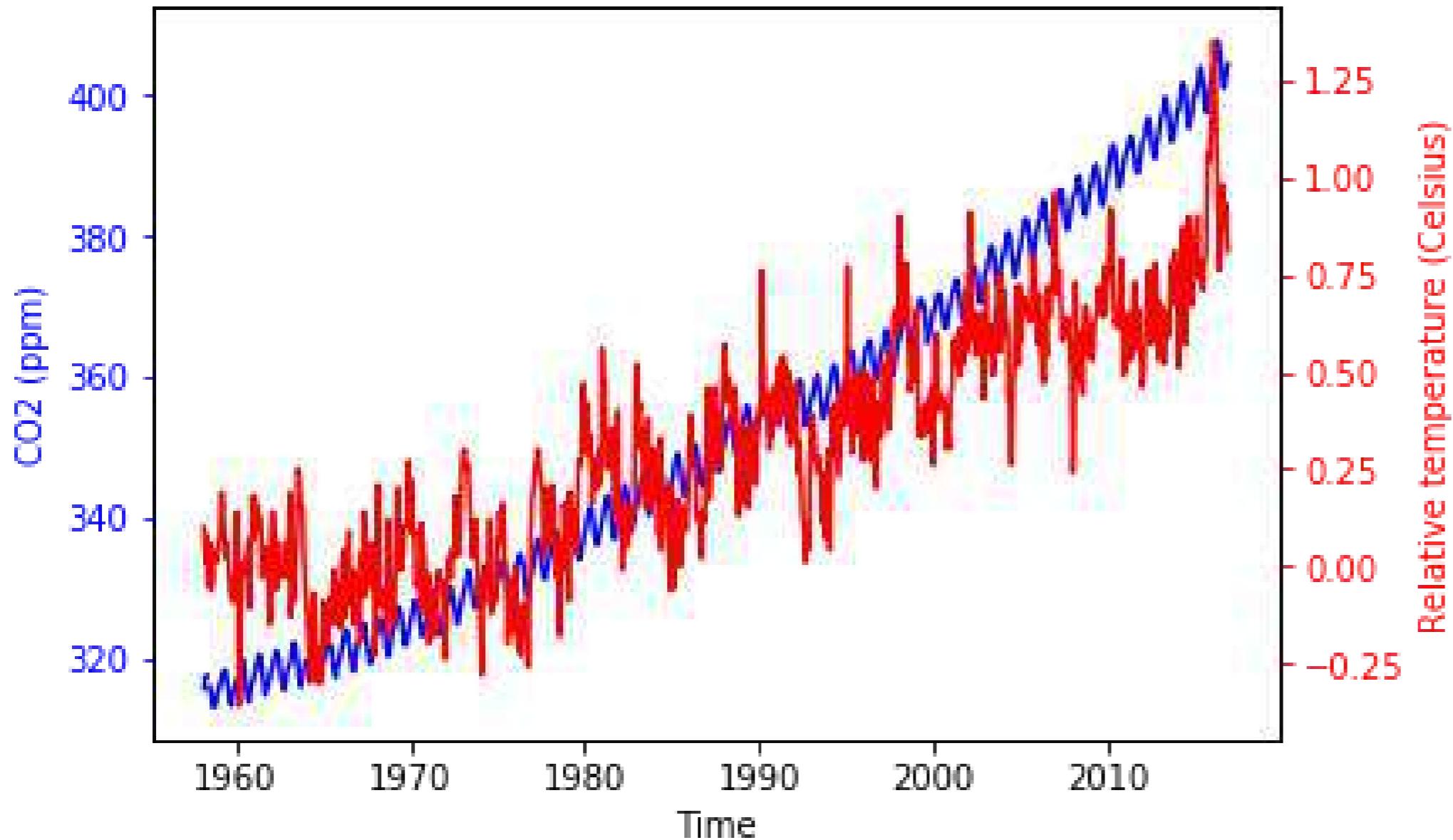
```
fig, ax = plt.subplots()
ax.plot(climate_change.index, climate_change["co2"], color='blue')
ax.set_xlabel('Time')
ax.set_ylabel('CO2 (ppm)', color='blue')
ax2 = ax.twinx()
ax2.plot(climate_change.index, climate_change["relative_temp"],
          color='red')
ax2.set_ylabel('Relative temperature (Celsius)', color='red')
plt.show()
```



Coloring the ticks

```
fig, ax = plt.subplots()
ax.plot(climate_change.index, climate_change["co2"],
         color='blue')
ax.set_xlabel('Time')
ax.set_ylabel('CO2 (ppm)', color='blue')
ax.tick_params('y', colors='blue')
ax2 = ax.twinx()
ax2.plot(climate_change.index,
          climate_change["relative_temp"],
          color='red')
ax2.set_ylabel('Relative temperature (Celsius)',
color='red')
ax2.tick_params('y', colors='red')
plt.show()
```

Coloring the ticks

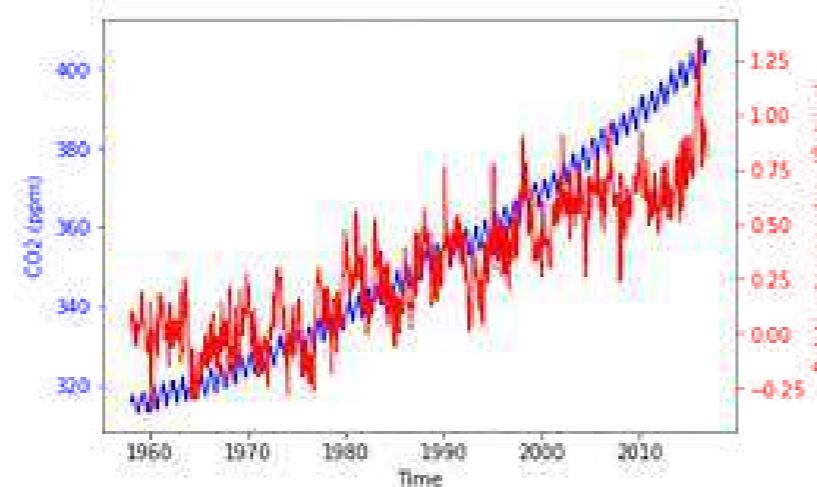


A function that plots time-series

```
def plot_timeseries(axes, x, y, color, xlabel, ylabel):  
    axes.plot(x, y, color=color)  
    axes.set_xlabel(xlabel)  
    axes.set_ylabel(ylabel, color=color)  
    axes.tick_params('y', colors=color)
```

Using our function

```
fig, ax = plt.subplots()  
plot_timeseries(ax, climate_change.index, climate_change['co2'],  
                 'blue', 'Time', 'CO2 (ppm)')  
  
ax2 = ax.twinx()  
plot_timeseries(ax, climate_change.index,  
                climate_change['relative_temp'],  
                'red', 'Time', 'Relative temperature (Celsius)')  
  
plt.show()
```

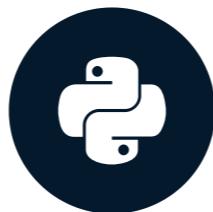


Create your own function!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Annotating time-series data

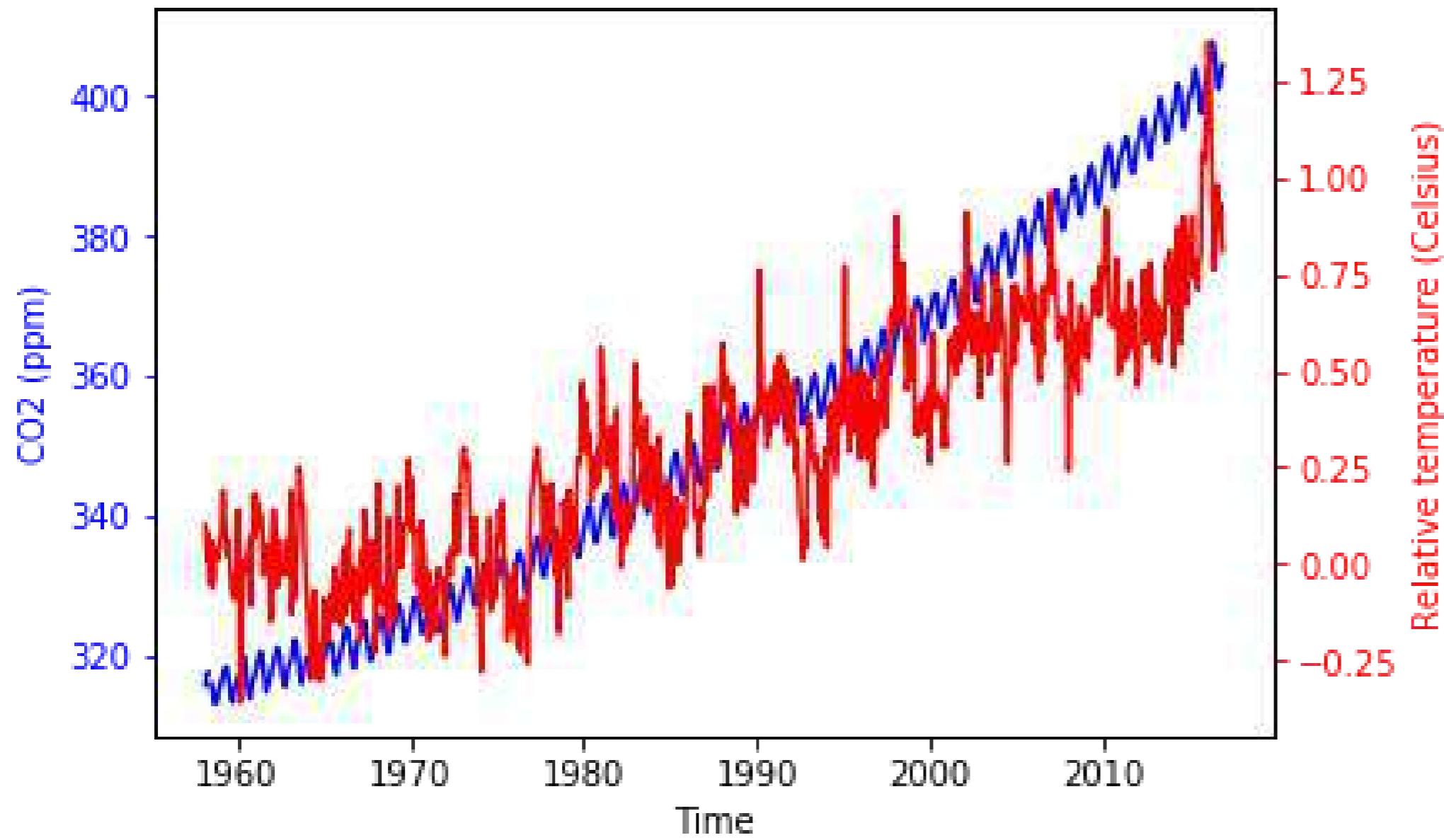
INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB



Ariel Rokem

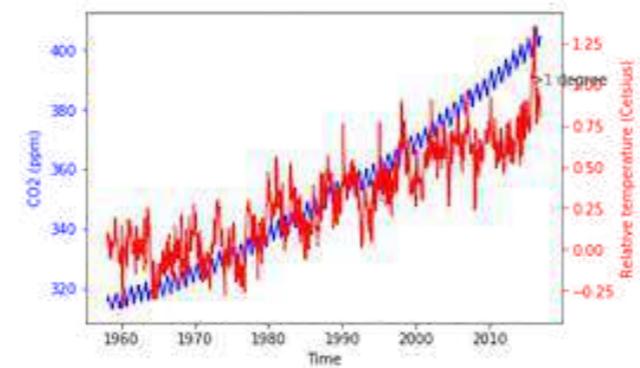
Data Scientist

Time-series data



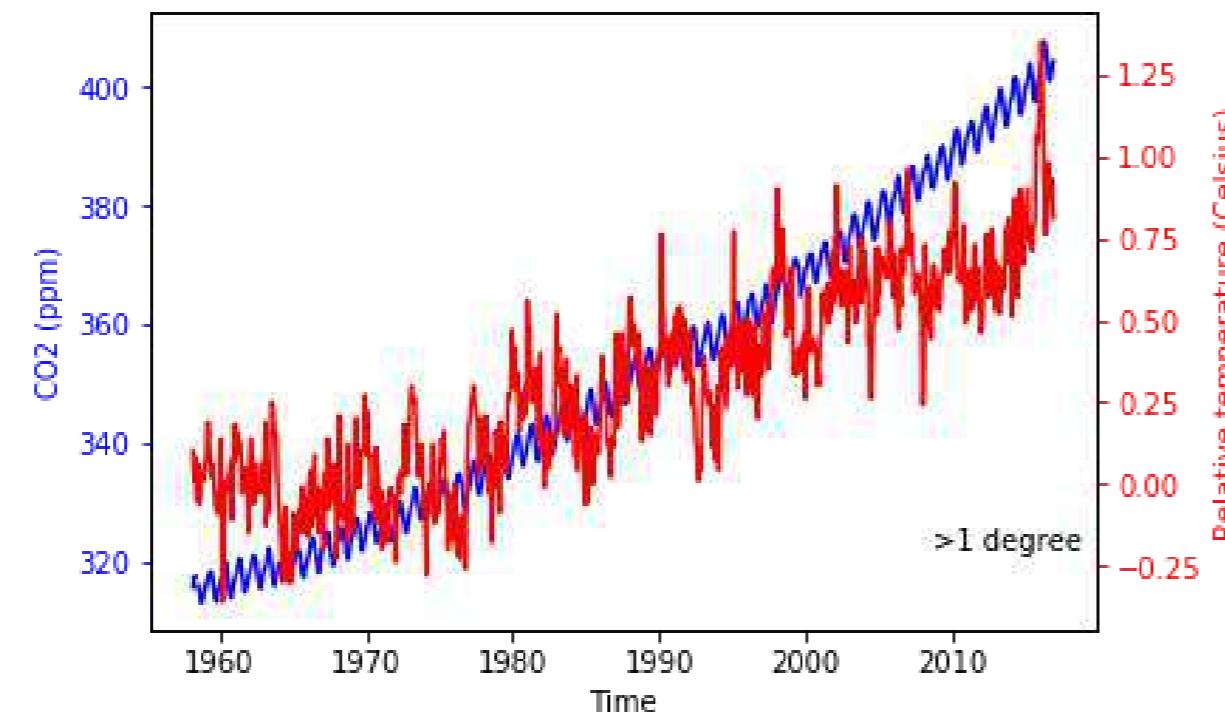
Annotation

```
fig, ax = plt.subplots()
plot_timeseries(ax, climate_change.index, climate_change['co2'],
                 'blue', 'Time', 'CO2 (ppm)')
ax2 = ax.twinx()
plot_timeseries(ax2, climate_change.index,
                 climate_change['relative_temp'],
                 'red', 'Time', 'Relative temperature (Celsius)')
ax2.annotate(">1 degree", xy=[pd.Timestamp("2015-10-06"), 1])
plt.show()
```



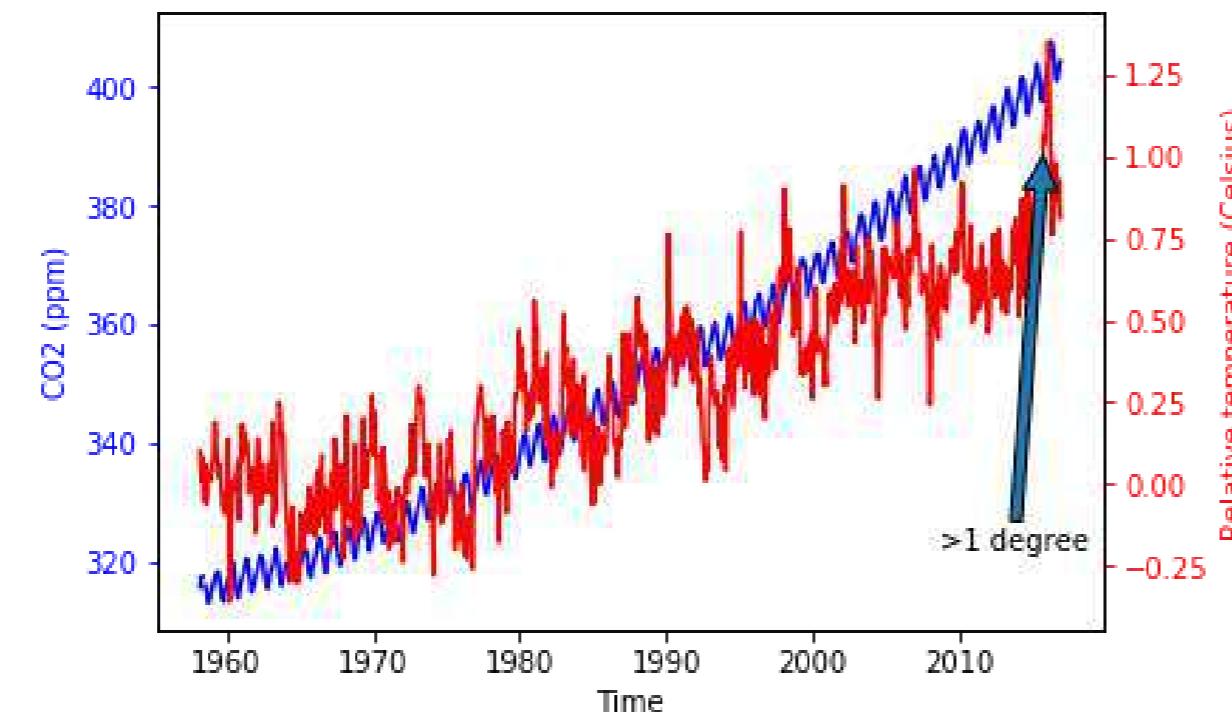
Positioning the text

```
ax2.annotate(">1 degree",  
            xy=(pd.Timestamp('2015-10-06'), 1),  
            xytext=(pd.Timestamp('2008-10-06'), -0.2))
```



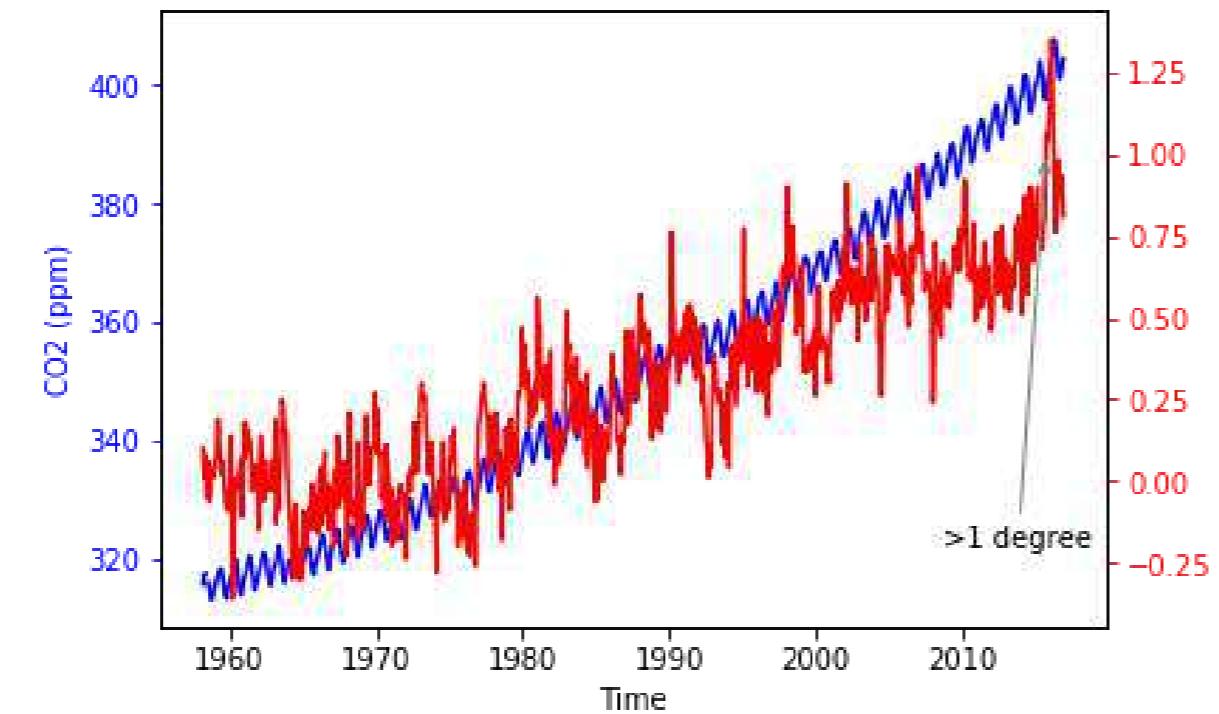
Adding arrows to annotation

```
ax2.annotate(">1 degree",  
            xy=(pd.Timestamp('2015-10-06'), 1),  
            xytext=(pd.Timestamp('2008-10-06'), -0.2),  
            arrowprops={})
```



Customizing arrow properties

```
ax2.annotate(">1 degree",
    xy=(pd.Timestamp('2015-10-06'), 1),
    xytext=(pd.Timestamp('2008-10-06'), -0.2),
    arrowprops={"arrowstyle": "->", "color": "gray"})
```



Customizing annotations

<https://matplotlib.org/users/annotations.html>

Practice annotating plots!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Quantitative comparisons: bar- charts

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem

Data Scientist



Olympic medals

, Gold, Silver, Bronze

United States, 137, 52, 67

Germany, 47, 43, 67

Great Britain, 64, 55, 26

Russia, 50, 28, 35

China, 44, 30, 35

France, 20, 55, 21

Australia, 23, 34, 25

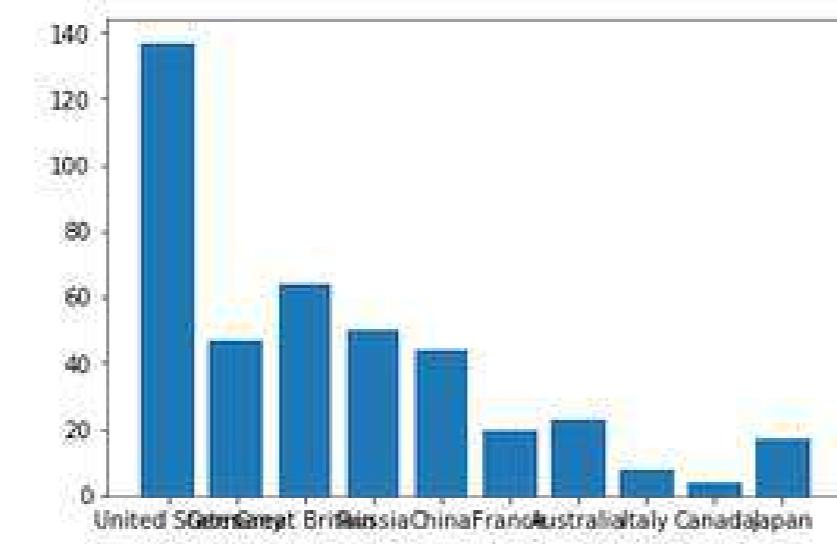
Italy, 8, 38, 24

Canada, 4, 4, 61

Japan, 17, 13, 34

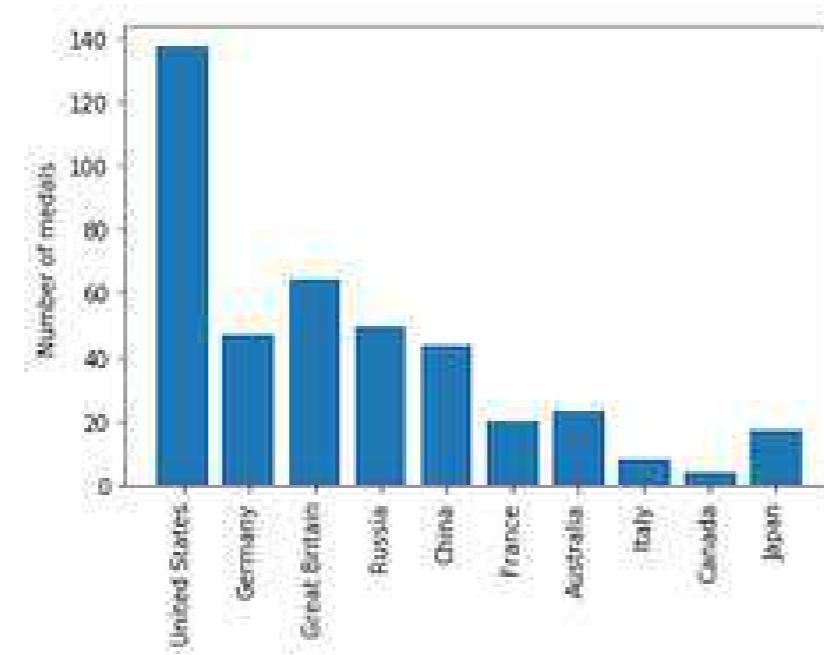
Olympic medals: visualizing the data

```
medals = pd.read_csv('medals_by_country_2016.csv', index_col=0)
fig, ax = plt.subplots()
ax.bar(medals.index, medals["Gold"])
plt.show()
```



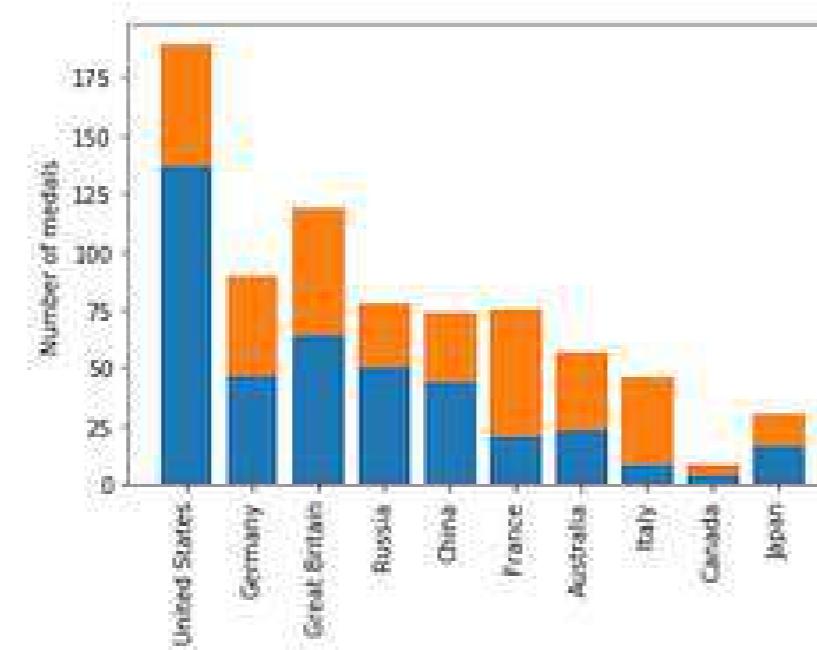
Interlude: rotate the tick labels

```
fig, ax = plt.subplots()  
ax.bar(medals.index, medals["Gold"])  
ax.set_xticklabels(medals.index, rotation=90)  
ax.set_ylabel("Number of medals")  
plt.show()
```



Olympic medals: visualizing the other medals

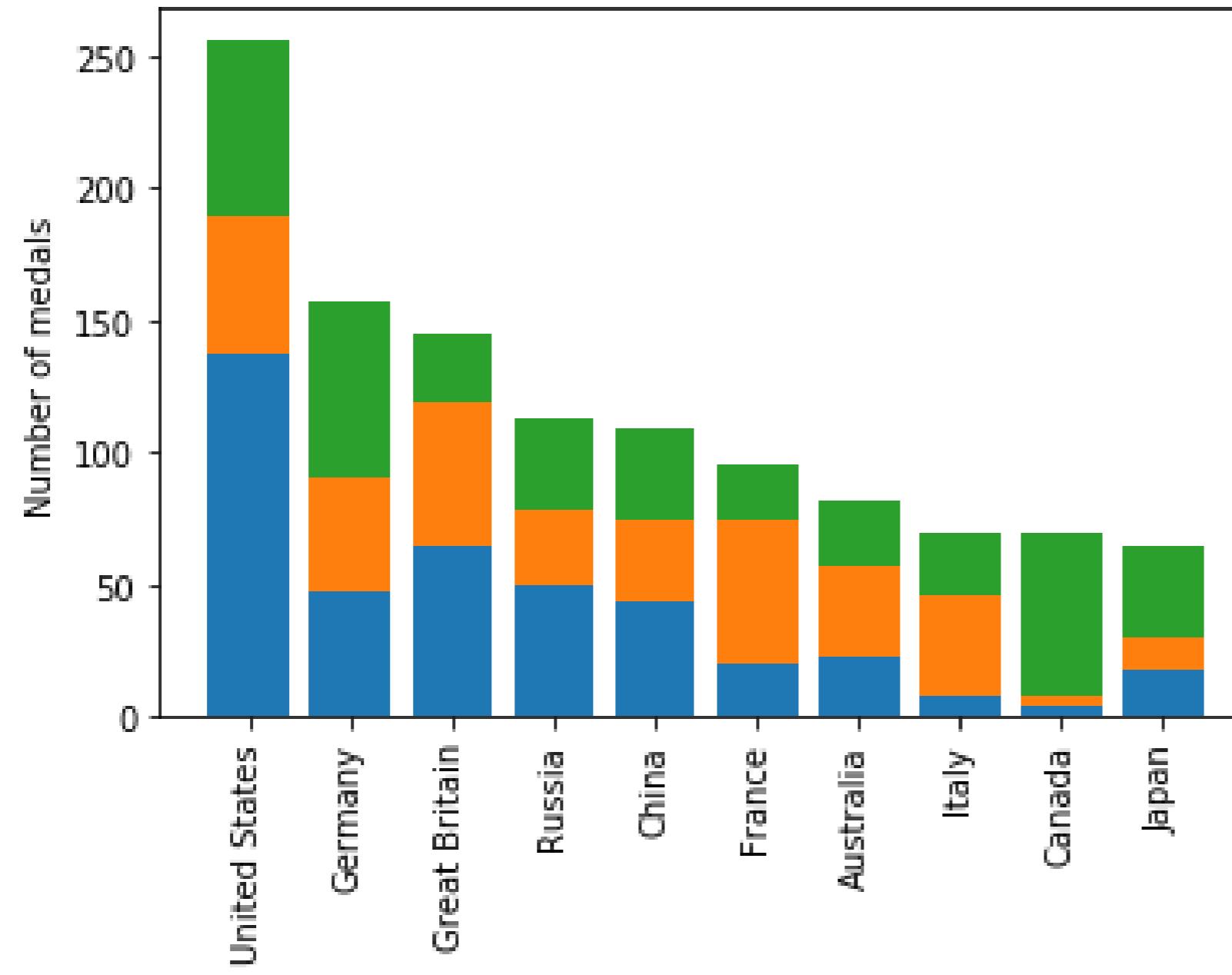
```
fig, ax = plt.subplots  
ax.bar(medals.index, medals["Gold"])  
ax.bar(medals.index, medals["Silver"], bottom=medals["Gold"])  
ax.set_xticklabels(medals.index, rotation=90)  
ax.set_ylabel("Number of medals")  
plt.show()
```



Olympic medals: visualizing all three

```
fig, ax = plt.subplots  
ax.bar(medals.index, medals["Gold"])  
  
ax.bar(medals.index, medals["Silver"], bottom=medals["Gold"])  
ax.bar(medals.index, medals["Bronze"],  
       bottom=medals["Gold"] + medals["Silver"])  
ax.set_xticklabels(medals.index, rotation=90)  
ax.set_ylabel("Number of medals")  
plt.show()
```

Stacked bar chart



Adding a legend

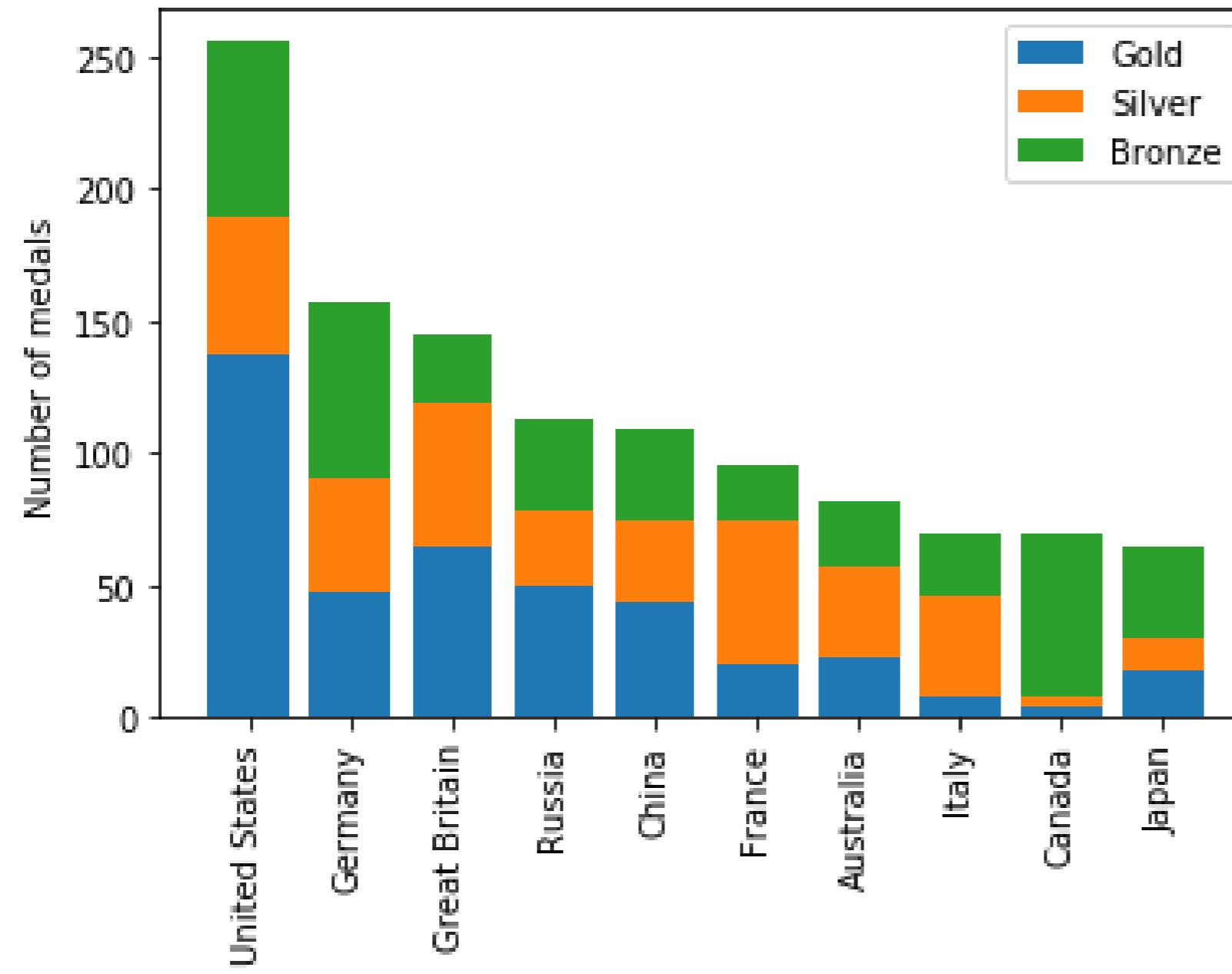
```
fig, ax = plt.subplots  
ax.bar(medals.index, medals["Gold"])  
ax.bar(medals.index, medals["Silver"], bottom=medals["Gold"])  
ax.bar(medals.index, medals["Bronze"],  
       bottom=medals["Gold"] + medals["Silver"])  
  
ax.set_xticklabels(medals.index, rotation=90)  
ax.set_ylabel("Number of medals")
```

Adding a legend

```
fig, ax = plt.subplots
ax.bar(medals.index, medals["Gold"], label="Gold")
ax.bar(medals.index, medals["Silver"], bottom=medals["Gold"],
       label="Silver")
ax.bar(medals.index, medals["Bronze"],
       bottom=medals["Gold"] + medals["Silver"],
       label="Bronze")

ax.set_xticklabels(medals.index, rotation=90)
ax.set_ylabel("Number of medals")
ax.legend()
plt.show()
```

Stacked bar chart with legend



Create a bar chart!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Quantitative comparisons: histograms

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB



Ariel Rokem

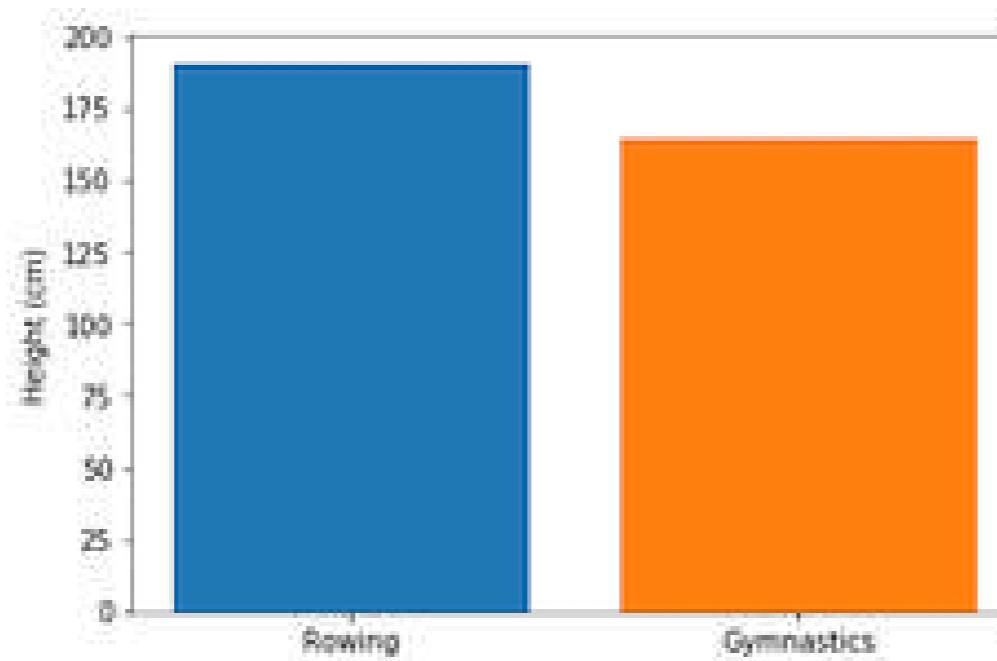
Data Scientist

Histograms

ID		Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
158	62	Giovanni Abagnale	M	21.0	198.0	90.0	Italy	ITA	2016 Summer	2016	Summer	Rio de Janeiro	Rowing	Rowing Men's Coxless Pairs	Bronze
11648	6346	Jrmie Azou	M	27.0	178.0	71.0	France	FRA	2016 Summer	2016	Summer	Rio de Janeiro	Rowing	Rowing Men's Lightweight Double Sculls	Gold
14871	8025	Thomas Gabriel Jrmie Baroukh	M	28.0	183.0	70.0	France	FRA	2016 Summer	2016	Summer	Rio de Janeiro	Rowing	Rowing Men's Lightweight Coxless Fours	Bronze
15215	8214	Jacob Jepsen Barse	M	27.0	188.0	73.0	Denmark	DEN	2016 Summer	2016	Summer	Rio de Janeiro	Rowing	Rowing Men's Lightweight Coxless Fours	Silver
18441	9764	Alexander Belonogoff	M	26.0	187.0	90.0	Australia	AUS	2016 Summer	2016	Summer	Rio de Janeiro	Rowing	Rowing Men's Quadruple Sculls	Silver

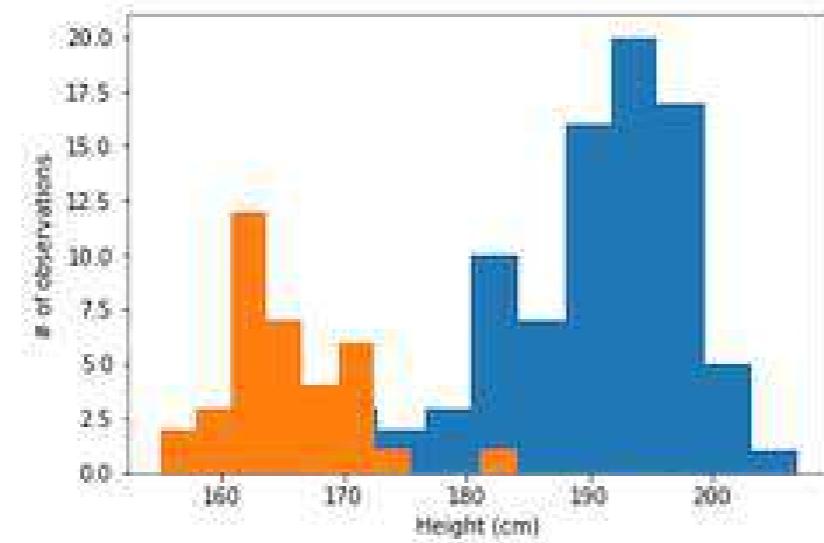
A bar chart again

```
fig, ax = plt.subplots()  
ax.bar("Rowing", mens_rowing["Height"].mean())  
ax.bar("Gymnastics", mens_gymnastics["Height"].mean())  
ax.set_ylabel("Height (cm)")  
plt.show()
```



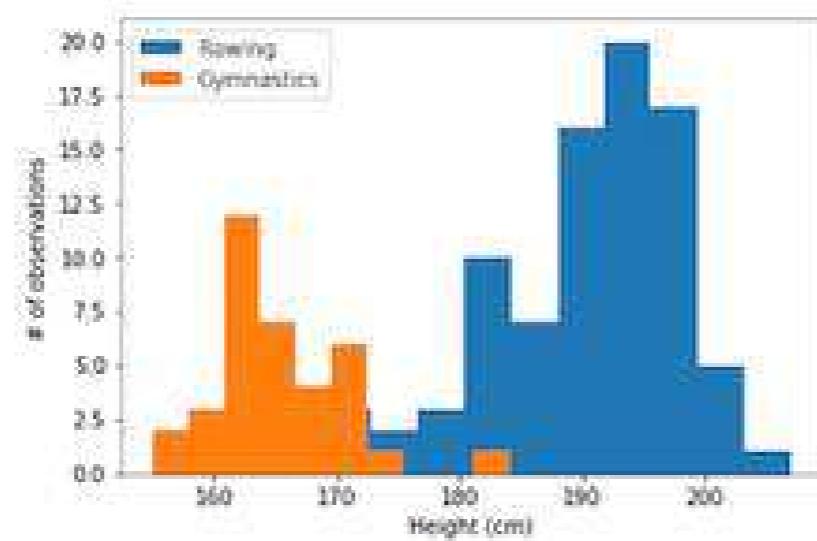
Introducing histograms

```
fig, ax = plt.subplots()  
ax.hist(mens_rowing["Height"])  
ax.hist(mens_gymnastic["Height"])  
ax.set_xlabel("Height (cm)")  
ax.set_ylabel("# of observations")  
plt.show()
```



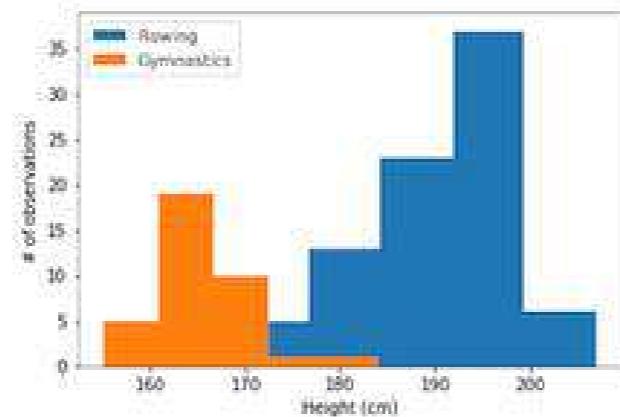
Labels are needed

```
ax.hist(mens_rowing["Height"], label="Rowing")
ax.hist(mens_gymnastic["Height"], label="Gymnastics")
ax.set_xlabel("Height (cm)")
ax.set_ylabel("# of observations")
ax.legend()
plt.show()
```



Customizing histograms: setting the number of bins

```
ax.hist(mens_rowing["Height"], label="Rowing", bins=5)
ax.hist(mens_gymnastic["Height"], label="Gymnastics", bins=5)
ax.set_xlabel("Height (cm)")
ax.set_ylabel("# of observations")
ax.legend()
plt.show()
```

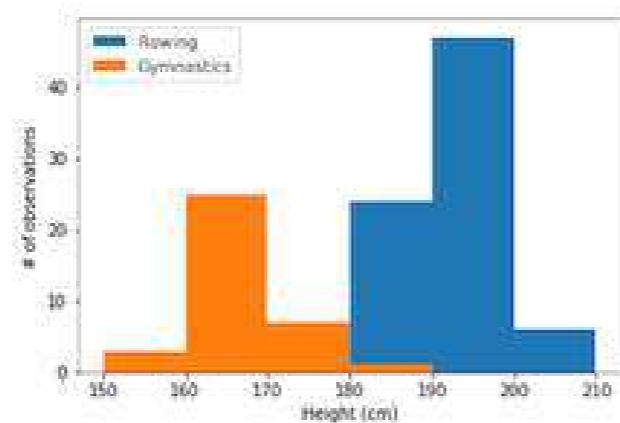


Customizing histograms: setting bin boundaries

```
ax.hist(mens_rowing["Height"], label="Rowing",
        bins=[150, 160, 170, 180, 190, 200, 210])

ax.hist(mens_gymnastic["Height"], label="Gymnastics",
        bins=[150, 160, 170, 180, 190, 200, 210])

ax.set_xlabel("Height (cm)")
ax.set_ylabel("# of observations")
ax.legend()
plt.show()
```



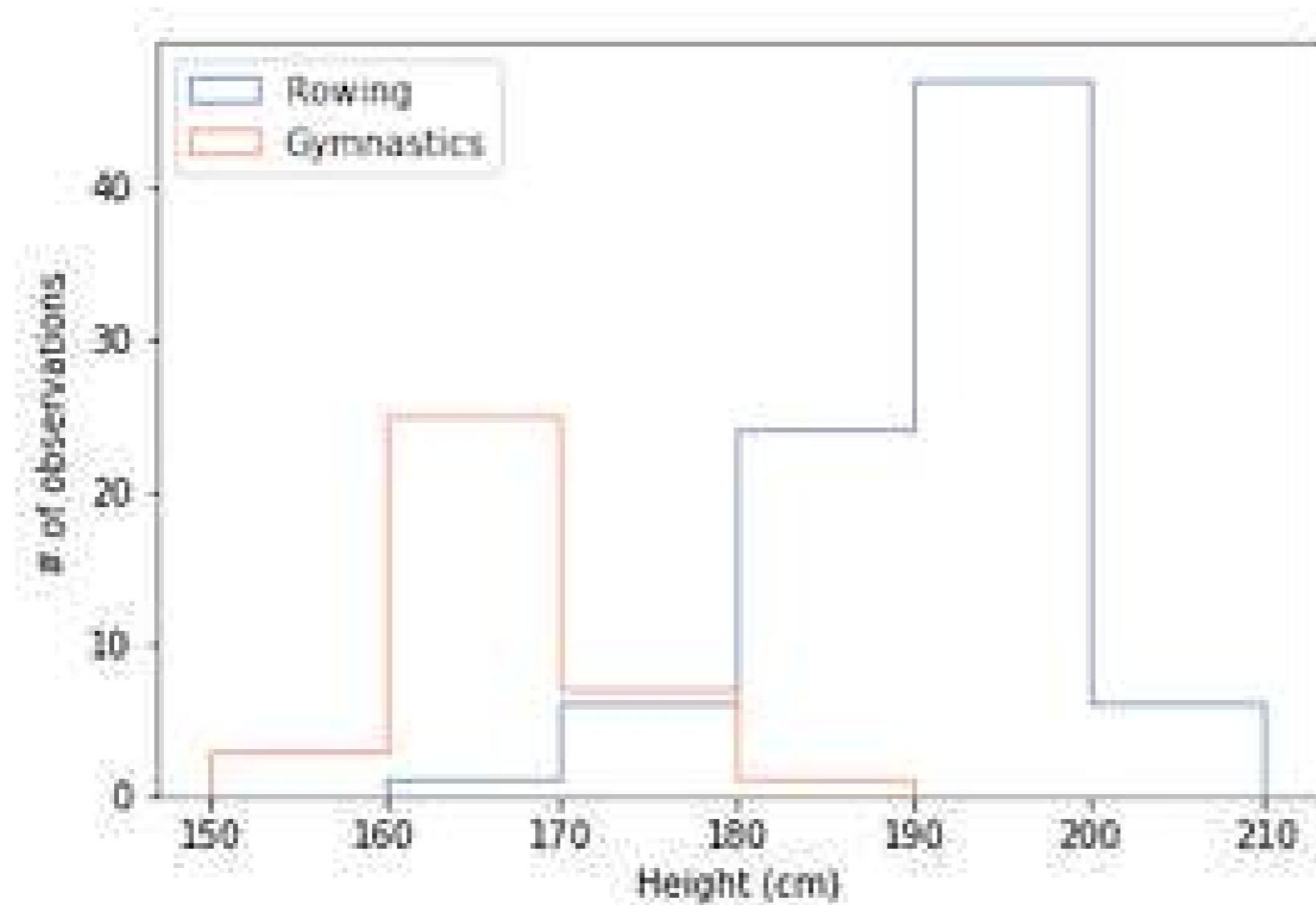
Customizing histograms: transparency

```
ax.hist(mens_rowing["Height"], label="Rowing",
        bins=[150, 160, 170, 180, 190, 200, 210],
        histtype="step")

ax.hist(mens_gymnastic["Height"], label="Gymnastics",
        bins=[150, 160, 170, 180, 190, 200, 210],
        histtype="step")

ax.set_xlabel("Height (cm)")
ax.set_ylabel("# of observations")
ax.legend()
plt.show()
```

Histogram with a histtype of step

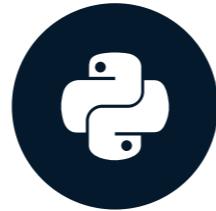


Create your own histogram!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Statistical plotting

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB



Ariel Rokem

Data Scientist

Adding error bars to bar charts

```
fig, ax = plt.subplots()

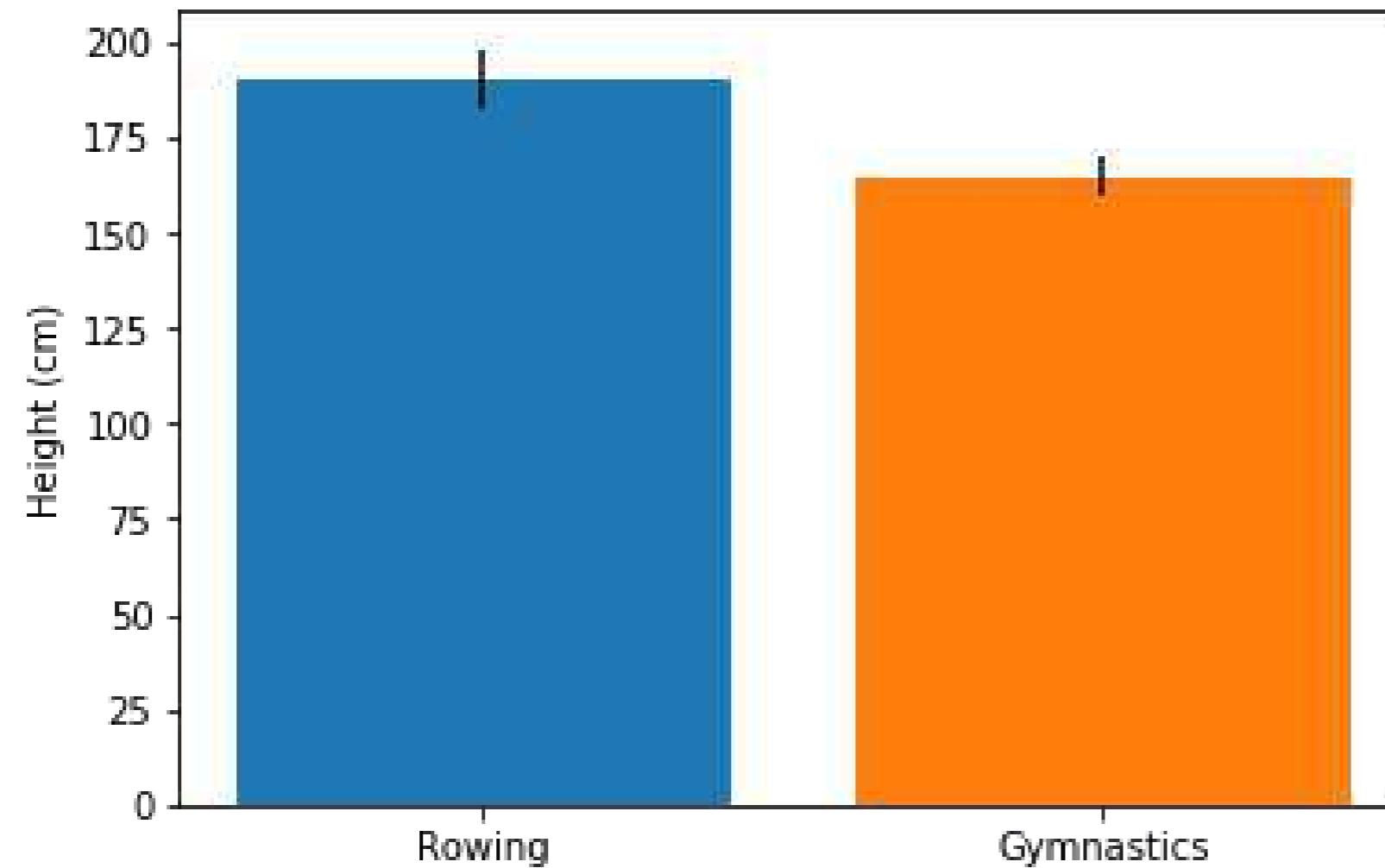
ax.bar("Rowing",
       mens_rowing["Height"].mean(),
       yerr=mens_rowing["Height"].std())

ax.bar("Gymnastics",
       mens_gymnastics["Height"].mean(),
       yerr=mens_gymnastics["Height"].std())

ax.set_ylabel("Height (cm)")

plt.show()
```

Error bars in a bar chart



Adding error bars to plots

```
fig, ax = plt.subplots()

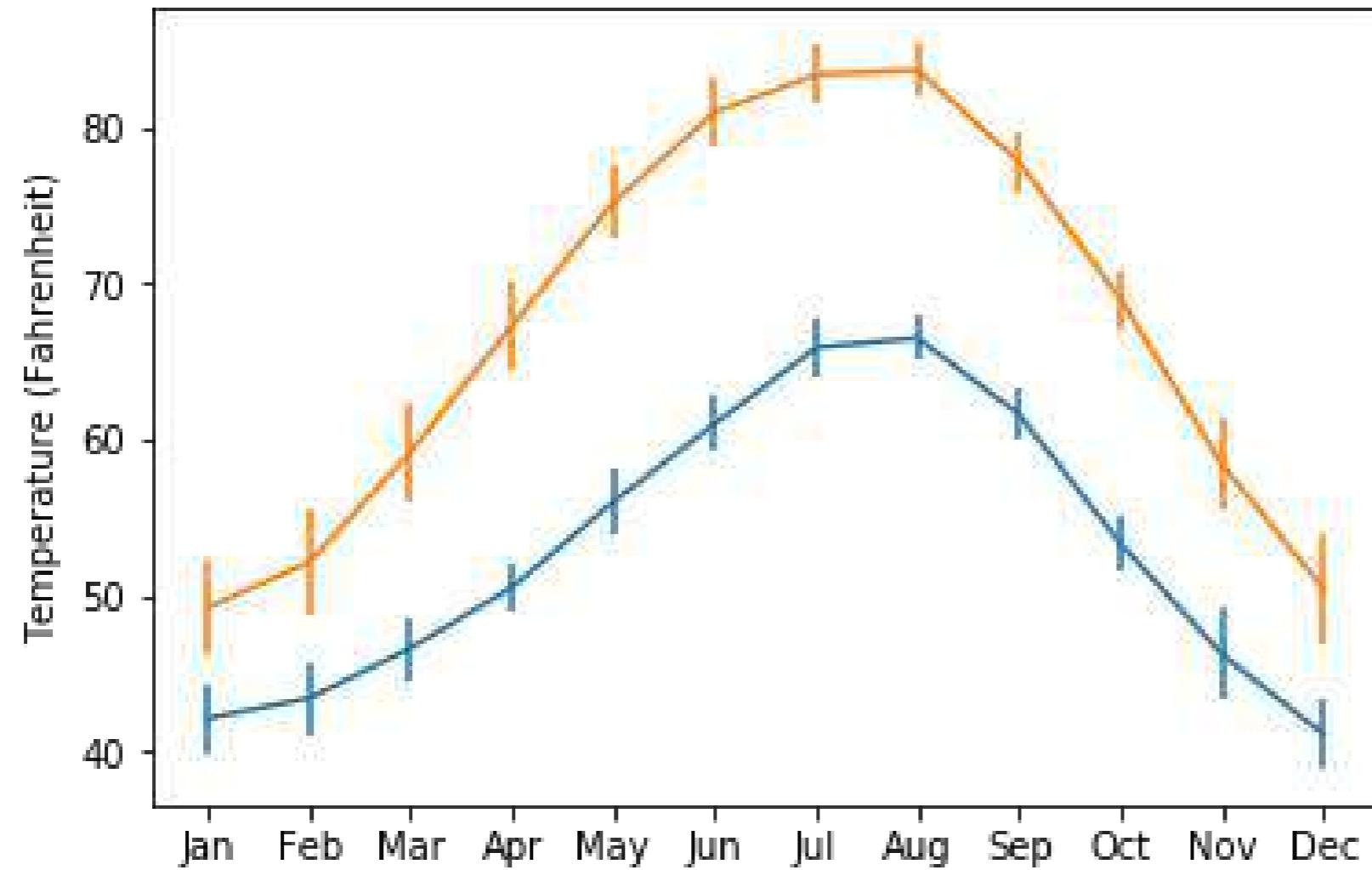
ax.errorbar(seattle_weather["MONTH"],
            seattle_weather["MLY-TAVG-NORMAL"],
            yerr=seattle_weather["MLY-TAVG-STDDEV"])

ax.errorbar(austin_weather["MONTH"],
            austin_weather["MLY-TAVG-NORMAL"],
            yerr=austin_weather["MLY-TAVG-STDDEV"])

ax.set_ylabel("Temperature (Fahrenheit)")

plt.show()
```

Error bars in plots

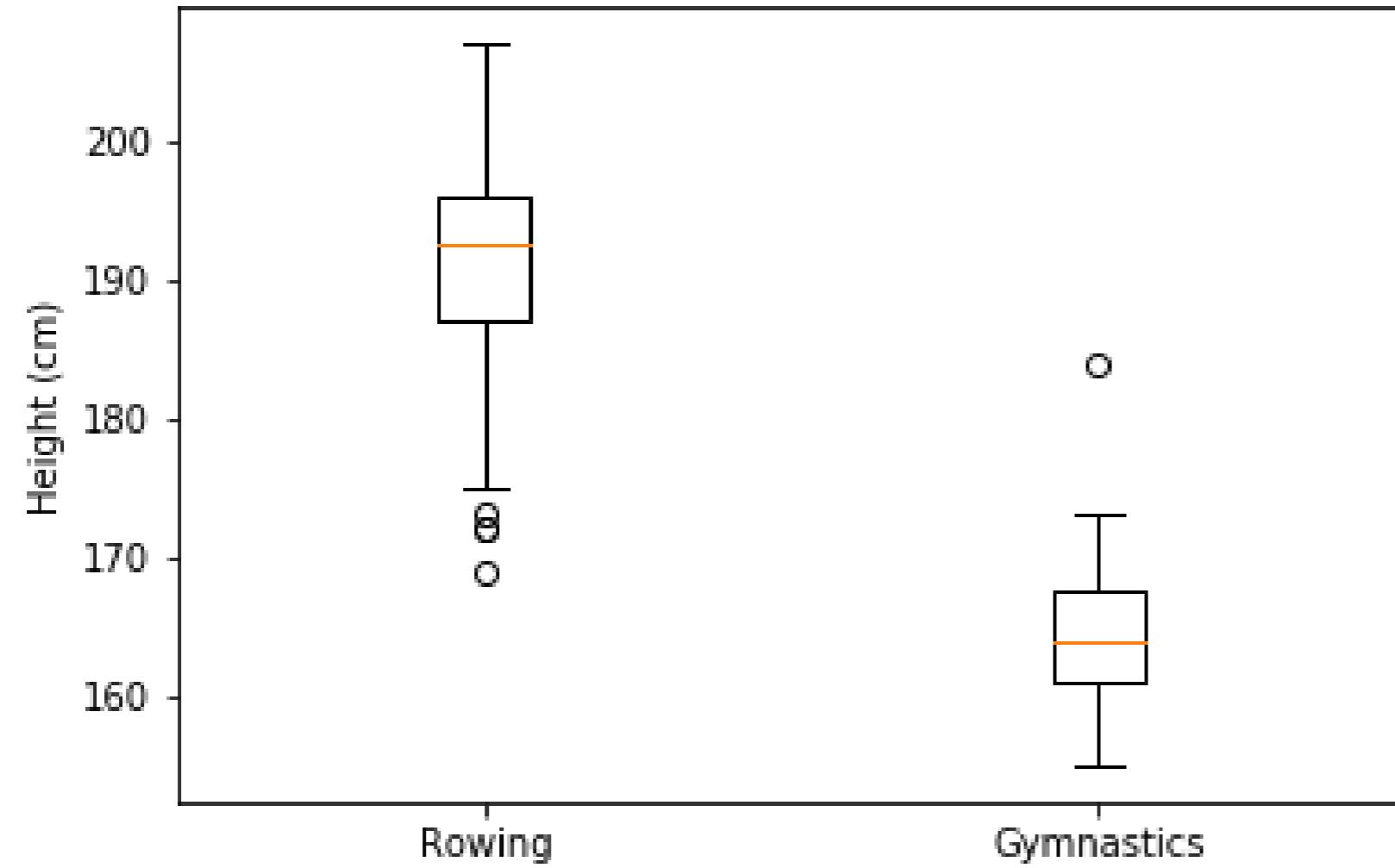


Adding boxplots

```
fig, ax = plt.subplots()
ax.boxplot([mens_rowing["Height"],
            mens_gymnastics["Height"]])
ax.set_xticklabels(["Rowing", "Gymnastics"])
ax.set_ylabel("Height (cm)")

plt.show()
```

Interpreting boxplots



Try it yourself!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Quantitative comparisons: scatter plots

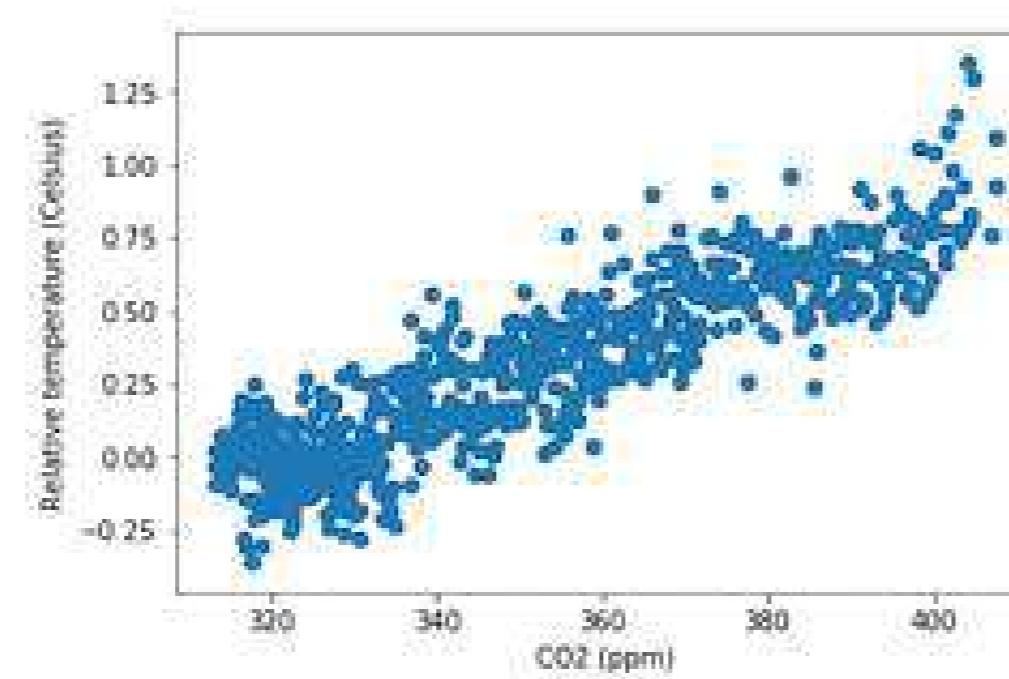
INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem
Data Scientist



Introducing scatter plots

```
fig, ax = plt.subplots()  
ax.scatter(climate_change["co2"], climate_change["relative_temp"])  
ax.set_xlabel("CO2 (ppm)")  
ax.set_ylabel("Relative temperature (Celsius)")  
plt.show()
```



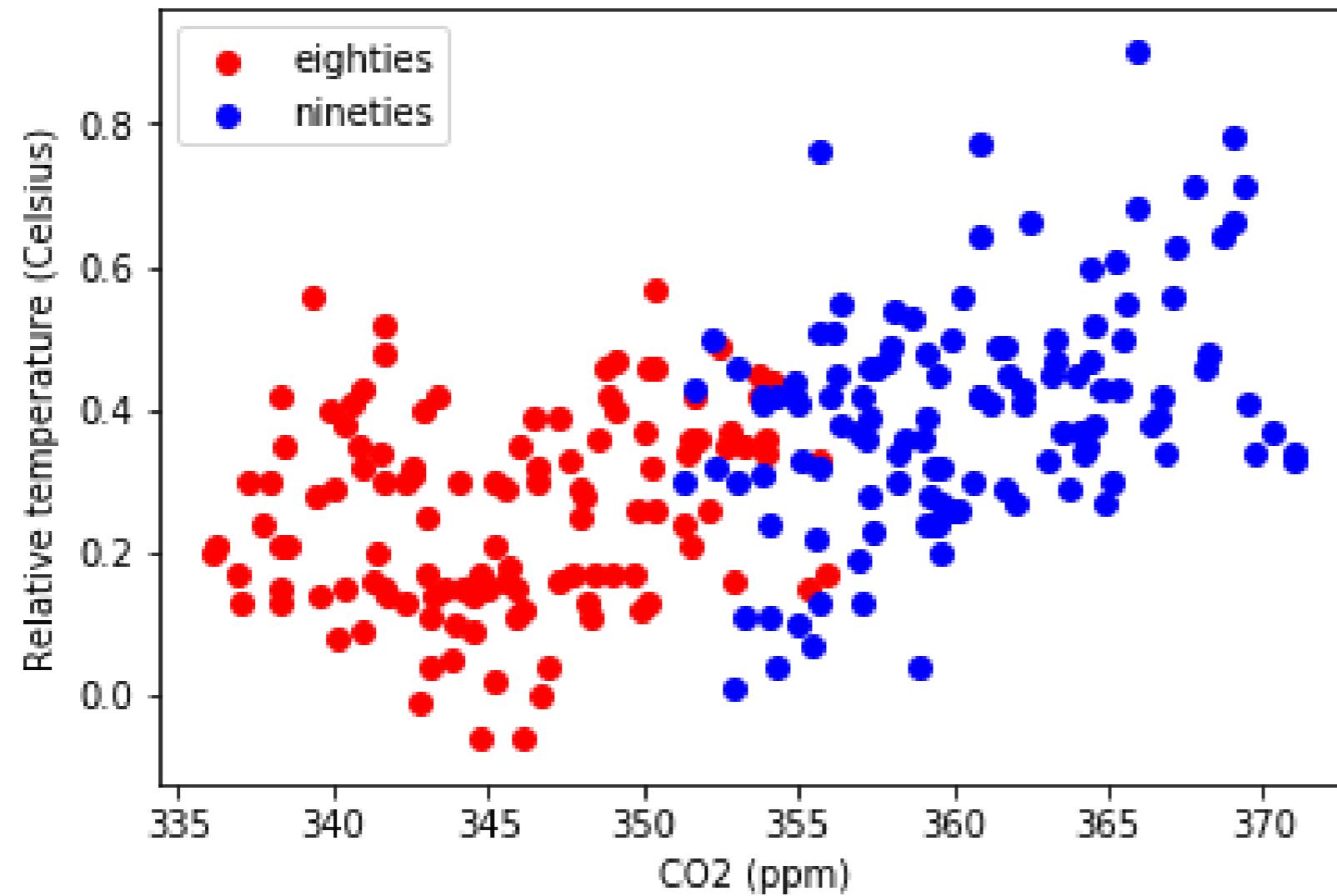
Customizing scatter plots

```
eighties = climate_change["1980-01-01":"1989-12-31"]
nineties = climate_change["1990-01-01":"1999-12-31"]
fig, ax = plt.subplots()
ax.scatter(eighties["co2"], eighties["relative_temp"],
           color="red", label="eighties")
ax.scatter(nineties["co2"], nineties["relative_temp"],
           color="blue", label="nineties")
ax.legend()

ax.set_xlabel("CO2 (ppm)")
ax.set_ylabel("Relative temperature (Celsius)")

plt.show()
```

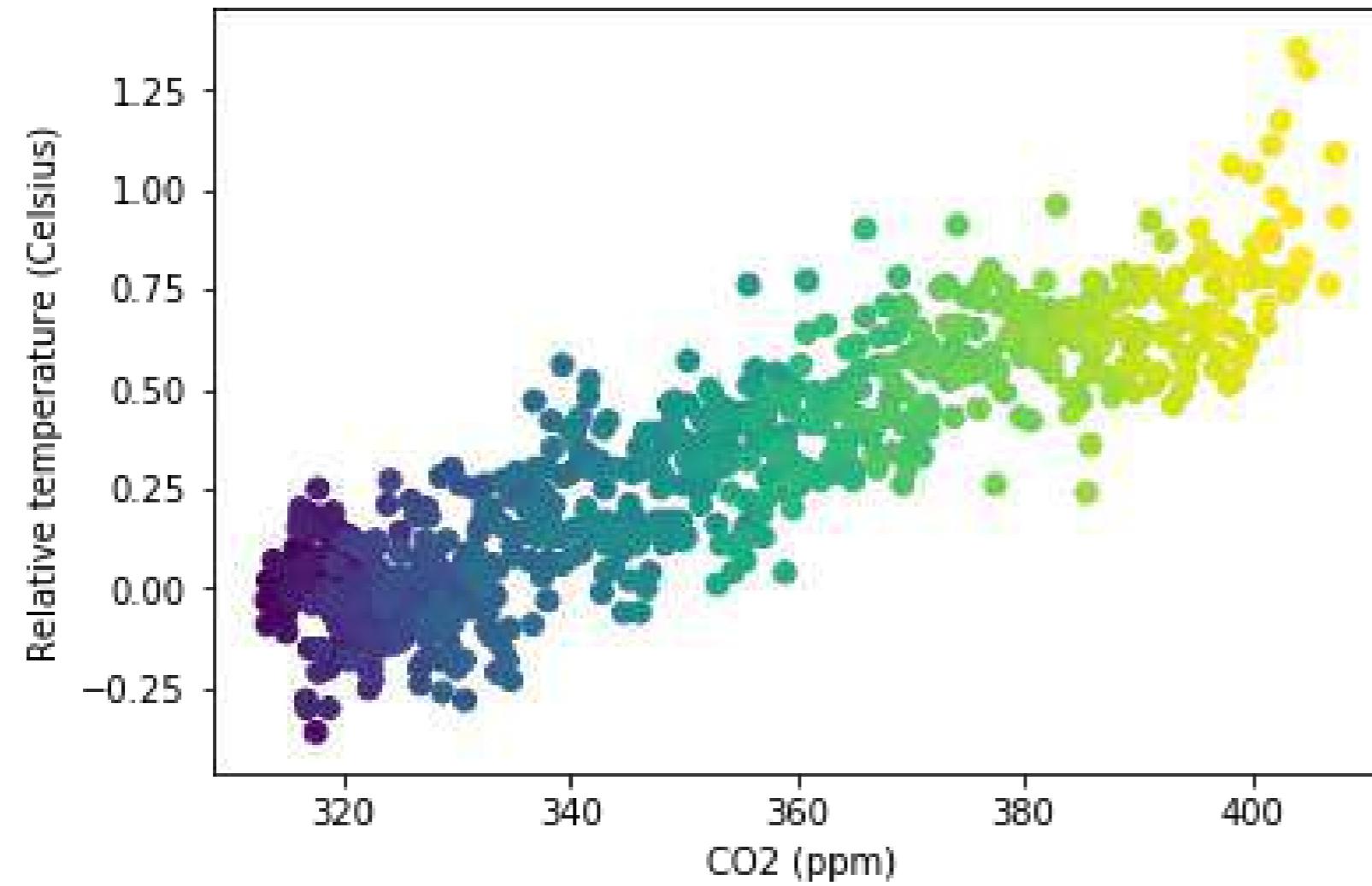
Encoding a comparison by color



Encoding a third variable by color

```
fig, ax = plt.subplots()  
ax.scatter(climate_change["co2"], climate_change["relative_temp"],  
           c=climate_change.index)  
ax.set_xlabel("CO2 (ppm)")  
ax.set_ylabel("Relative temperature (Celsius)")  
plt.show()
```

Encoding time in color



**Practice making
your own scatter
plots!**

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Preparing your figures to share with others

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

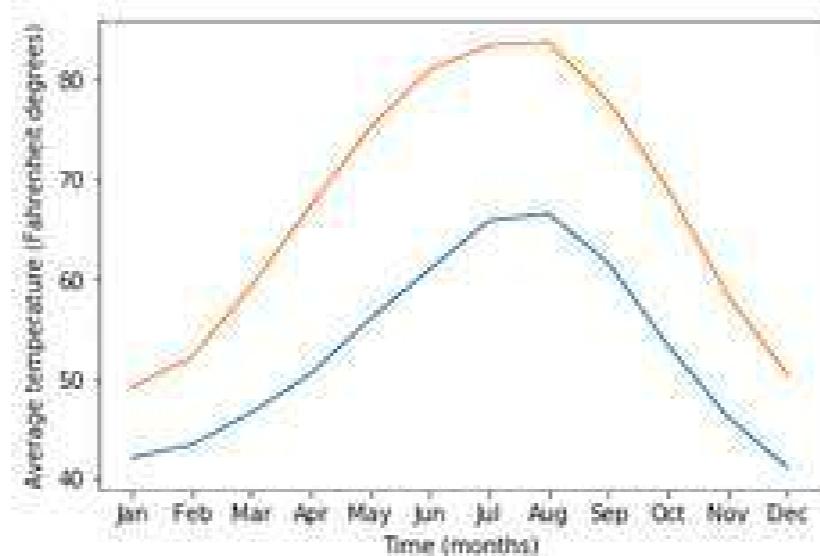
Ariel Rokem

Data Scientist



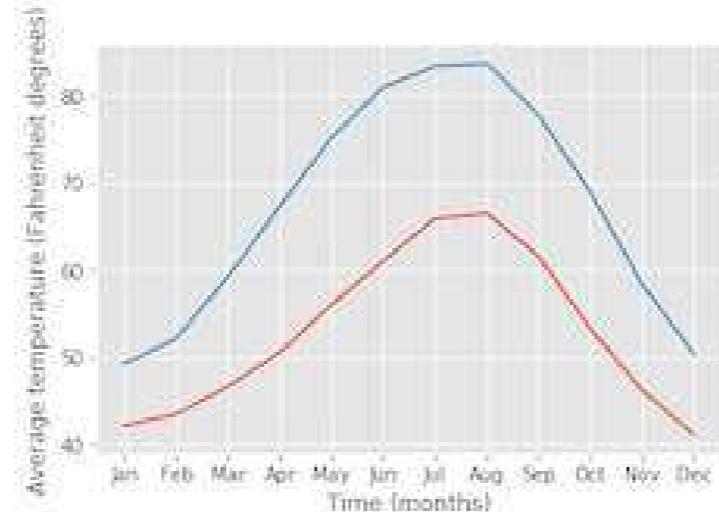
Changing plot style

```
import matplotlib.pyplot as plt  
fig, ax = plt.subplots()  
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-TAVG-NORMAL"])  
ax.plot(austin_weather["MONTH"], austin_weather["MLY-TAVG-NORMAL"])  
ax.set_xlabel("Time (months)")  
ax.set_ylabel("Average temperature (Fahrenheit degrees)")  
plt.show()
```



Choosing a style

```
plt.style.use("ggplot")
fig, ax = plt.subplots()
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-TAVG-NORMAL"])
ax.plot(austin_weather["MONTH"], austin_weather["MLY-TAVG-NORMAL"])
ax.set_xlabel("Time (months)")
ax.set_ylabel("Average temperature (Fahrenheit degrees)")
plt.show()
```



Back to the default

```
plt.style.use("default")
```

The available styles

https://matplotlib.org/gallery/style_sheets/style_sheets_reference.html

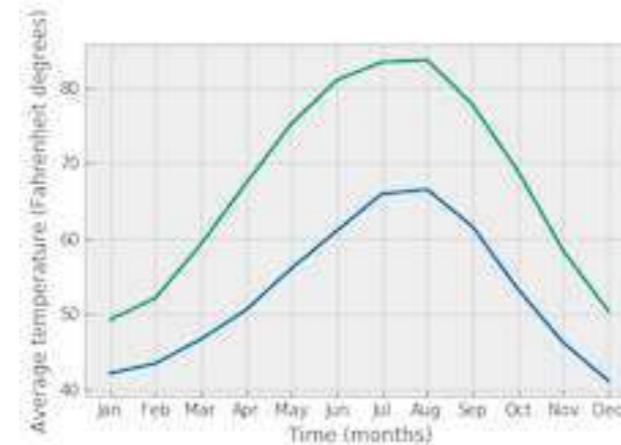
The "bmh" style

```
plt.style.use("bmh")
fig, ax = plt.subplots()
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-TAVG-NORMAL"])
ax.plot(austin_weather["MONTH"], austin_weather["MLY-TAVG-NORMAL"])
ax.set_xlabel("Time (months)")
ax.set_ylabel("Average temperature (Fahrenheit degrees)")
plt.show()
```



Seaborn styles

```
plt.style.use("seaborn-colorblind")
fig, ax = plt.subplots()
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-TAVG-NORMAL"])
ax.plot(austin_weather["MONTH"], austin_weather["MLY-TAVG-NORMAL"])
ax.set_xlabel("Time (months)")
ax.set_ylabel("Average temperature (Fahrenheit degrees)")
plt.show()
```



Guidelines for choosing plotting style

- Dark backgrounds are usually less visible
- If color is important, consider choosing colorblind-friendly options
 - "seaborn-colorblind" or "tableau-colorblind10"
- If you think that someone will want to print your figure, use less ink
- If it will be printed in black-and-white, use the "grayscale" style

**Practice choosing
the right style for
you!**

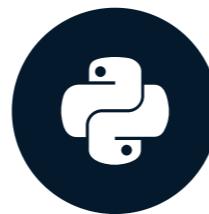
INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Sharing your visualizations with others

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem

Data Scientist

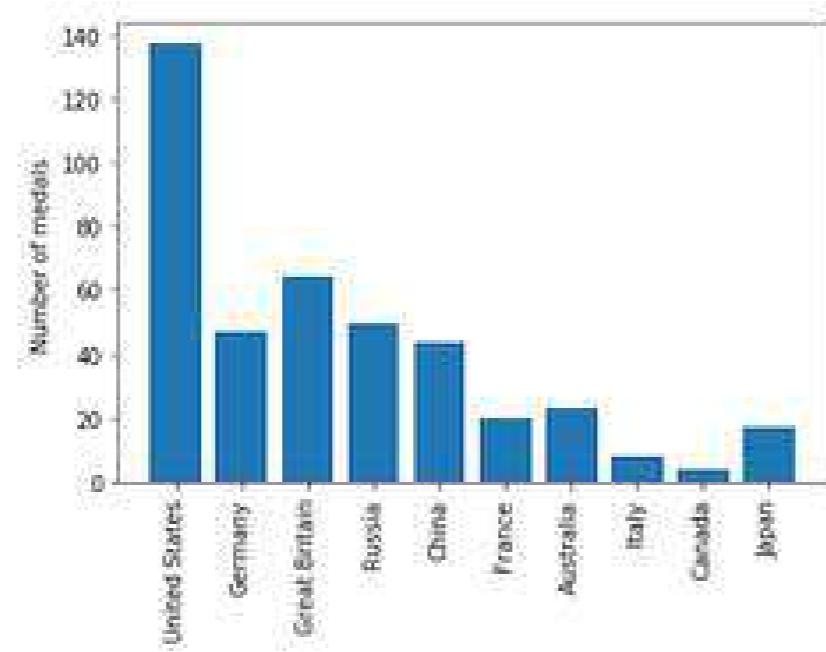


A figure to share

```
fig, ax = plt.subplots()

ax.bar(medals.index, medals["Gold"])
ax.set_xticklabels(medals.index, rotation=90)
ax.set_ylabel("Number of medals")

plt.show()
```



Saving the figure to file

```
fig, ax = plt.subplots()

ax.bar(medals.index, medals["Gold"])
ax.set_xticklabels(medals.index, rotation=90)
ax.set_ylabel("Number of medals")

fig.savefig("gold_medals.png")
```

```
ls
```

```
gold_medals.png
```

Different file formats

```
fig.savefig("gold_medals.jpg")
```

```
fig.savefig("gold_medals.jpg", quality=50)
```

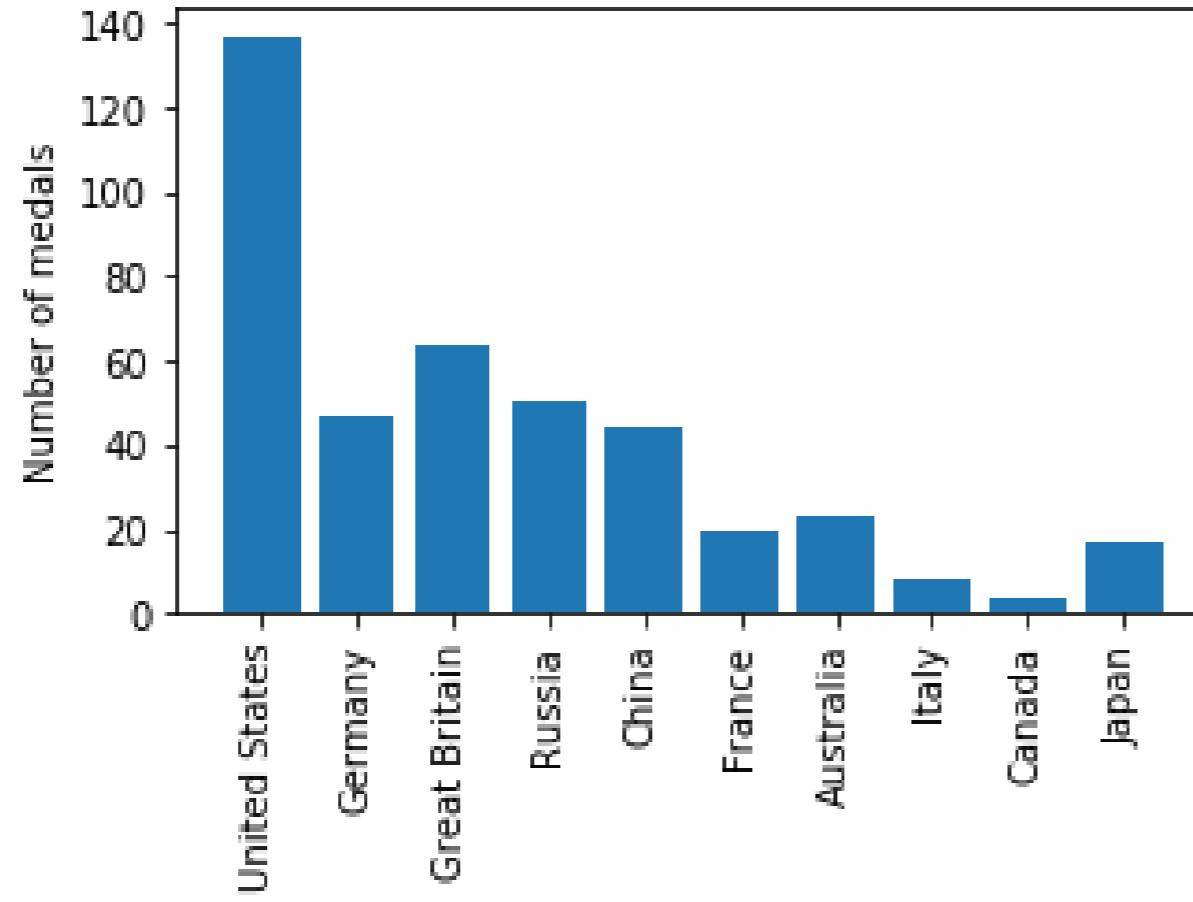
```
fig.savefig("gold_medals.svg")
```

Resolution

```
fig.savefig("gold_medals.png", dpi=300)
```

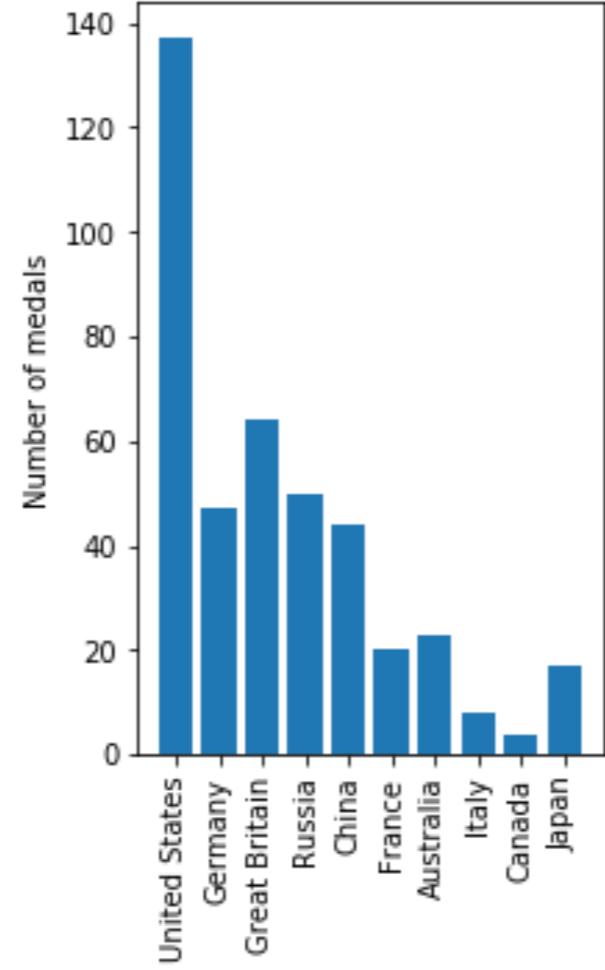
Size

```
fig.set_size_inches([5, 3])
```



Another aspect ratio

```
fig.set_size_inches([3, 5])
```

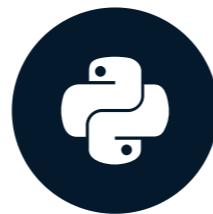


Practice saving your visualizations!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Automating figures from data

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB



Ariel Rokem

Data Scientist

Why automate?

- Ease and speed
- Flexibility
- Robustness
- Reproducibility

How many different kinds of data?

```
summer_2016_medals["Sport"]
```

```
ID  
62      Rowing  
65      Taekwondo  
73      Handball  
...  
134759   Handball  
135132   Volleyball  
135205   Boxing  
Name: Sport, Length: 976, dtype: object
```

Getting unique values of a column

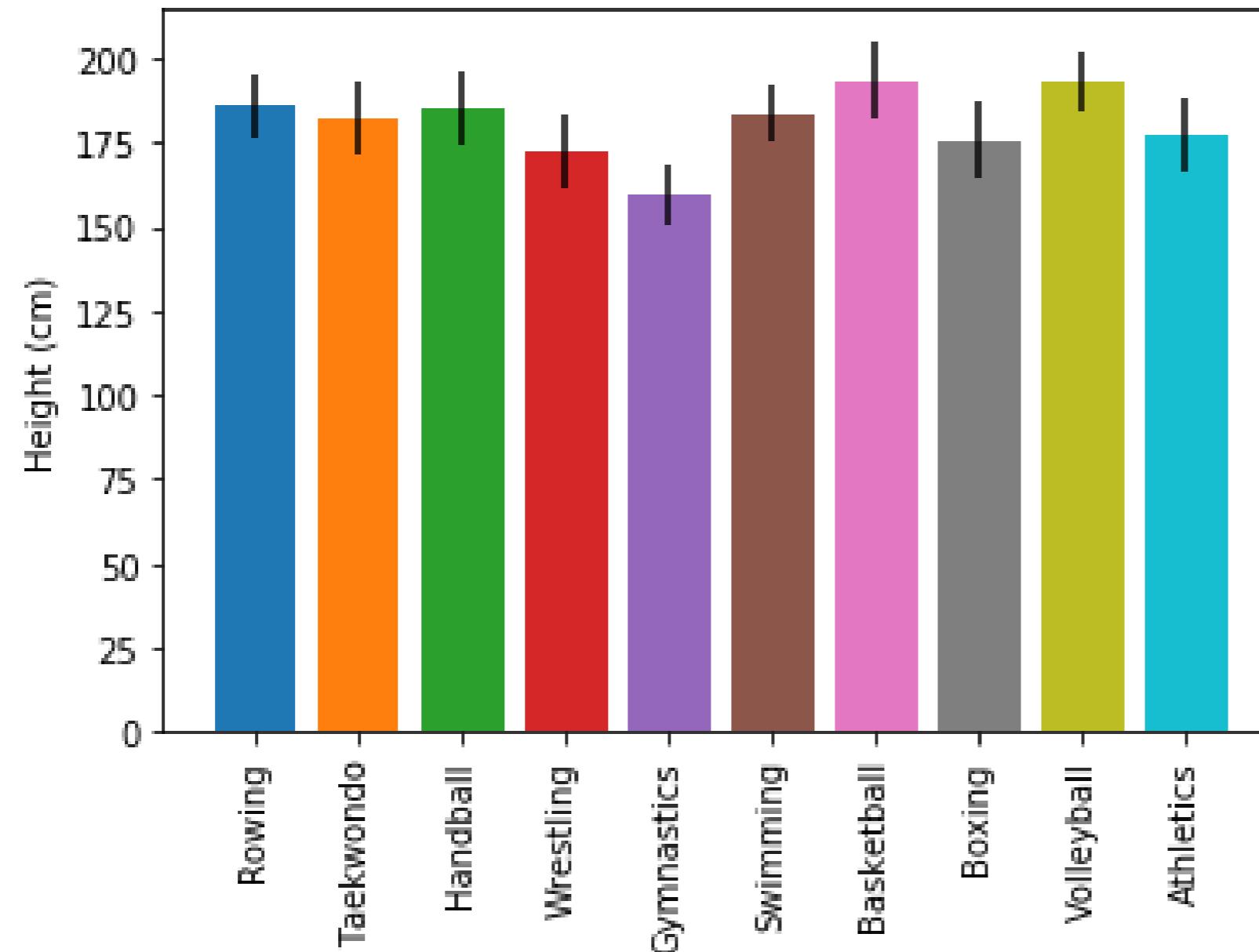
```
sports = summer_2016_medals["Sport"].unique()  
print(sports)  
['Rowing' 'Taekwondo' 'Handball' 'Wrestling'  
'Gymnastics' 'Swimming' 'Basketball' 'Boxing'  
'Volleyball' 'Athletics']
```

Bar-chart of heights for all sports

```
fig, ax = plt.subplots()

for sport in sports:
    sport_df = summer_2016_medals[summer_2016_medals["Sport"] == sport]
    ax.bar(sport, sport_df["Height"].mean(),
           yerr=sport_df["Height"].std())
ax.set_ylabel("Height (cm)")
ax.set_xticklabels(sports, rotation=90)
plt.show()
```

Figure derived automatically from the data

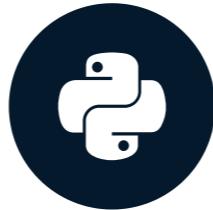


Practice automating visualizations!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Where to go next

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB



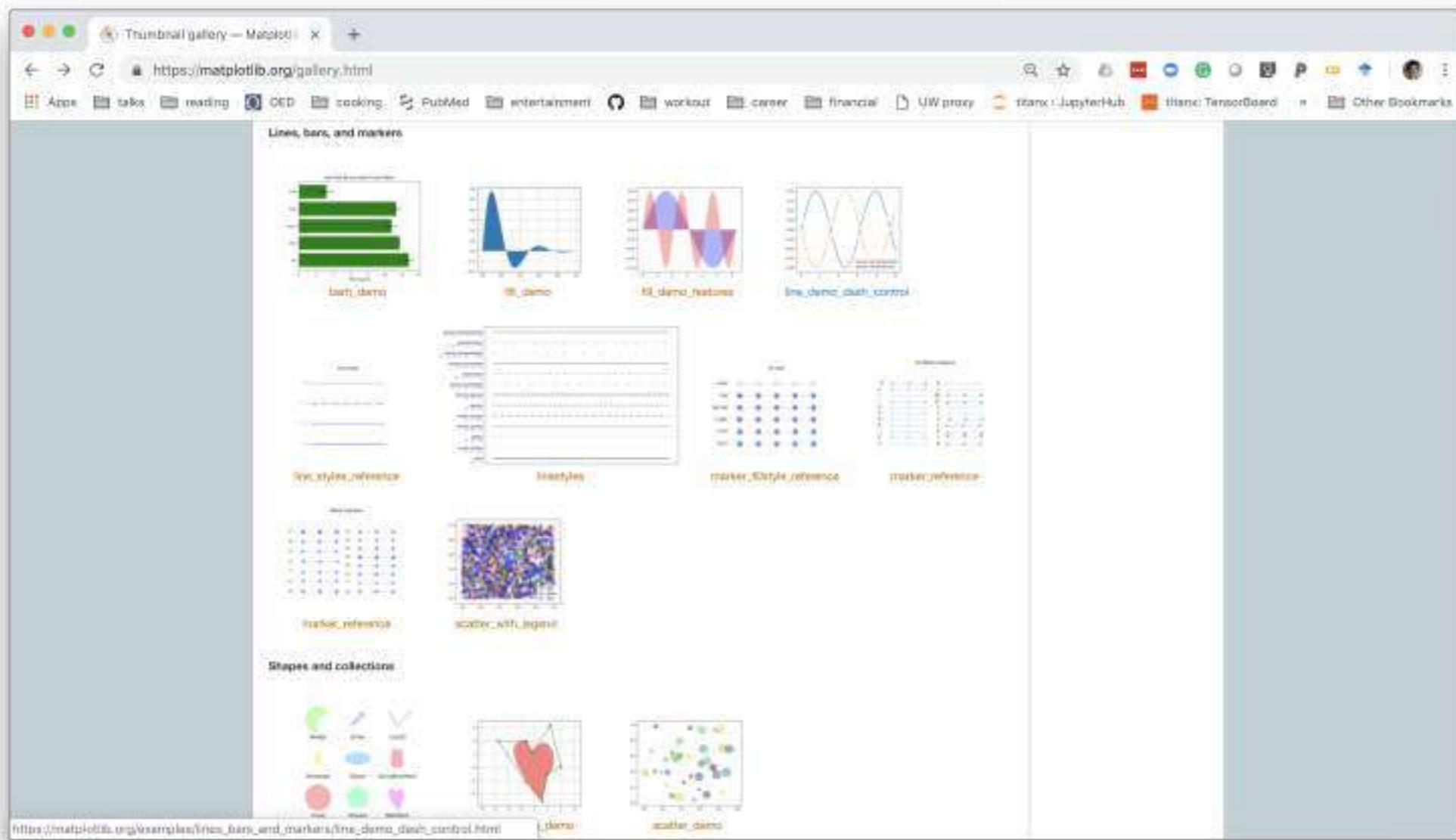
Ariel Rokem

Data Scientist

The Matplotlib gallery

<https://matplotlib.org/gallery.html>

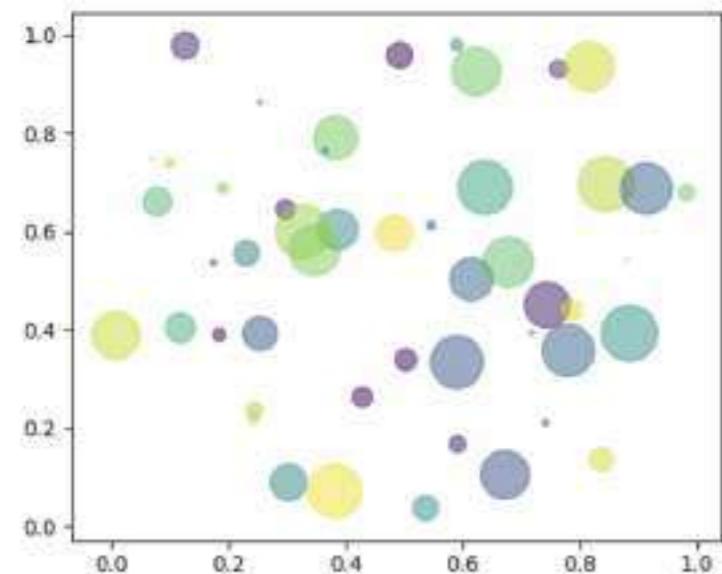
Gallery of examples



Example page with code

shapes_and_collections example code: scatter_demo.py

(Source code, png, pdf)



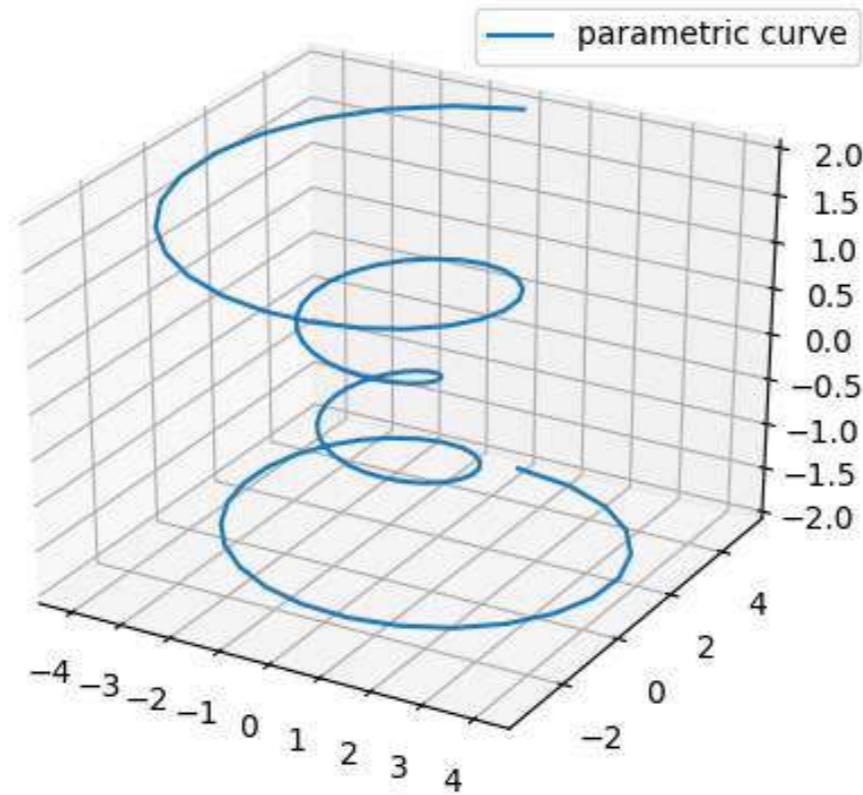
```
"""
Simple demo of a scatter plot.
"""

import numpy as np
import matplotlib.pyplot as plt

N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (15 * np.random.rand(N))**2 # 8 to 25 point radii

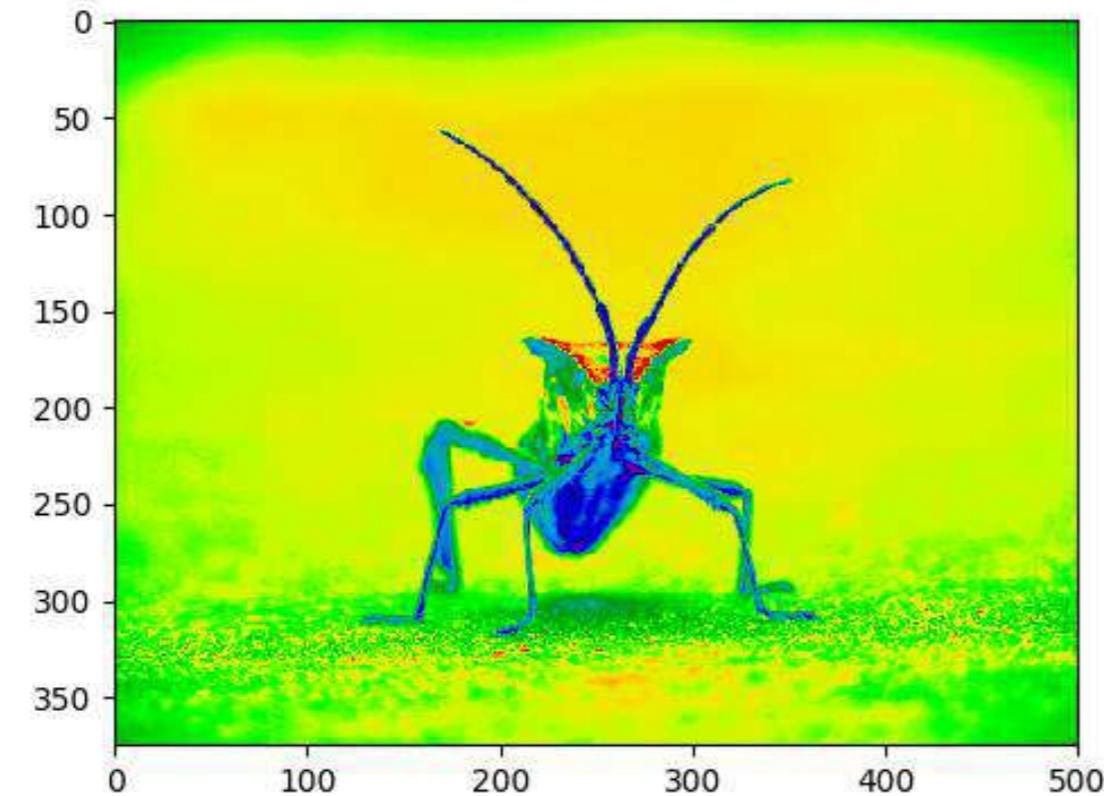
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```

Plotting data in 3D



https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html

Visualizing images with pseudo-color



https://matplotlib.org/users/image_tutorial.html

Animations

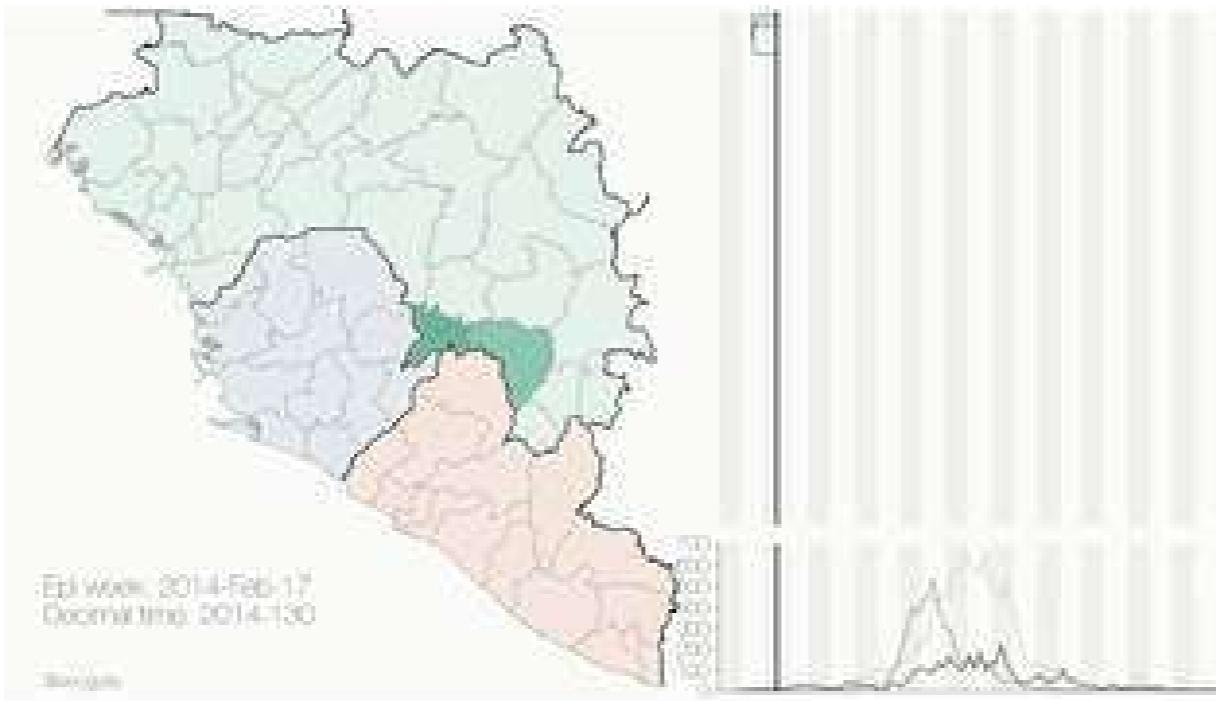


Image credit: [Gytis Dudas](#) and [Andrew Rambaut](#)

https://matplotlib.org/api/animation_api.html

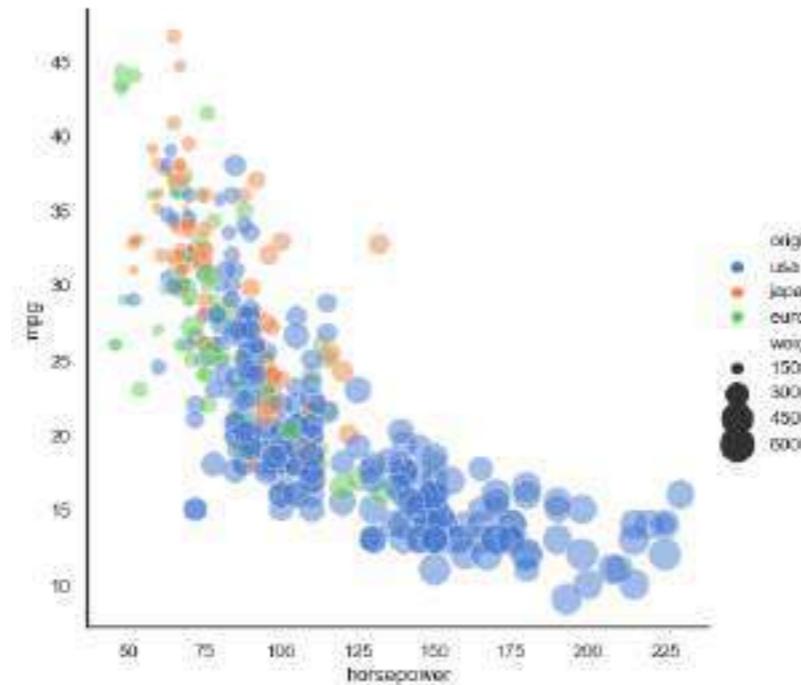
Using Matplotlib for geospatial data



<https://scitools.org.uk/cartopy/docs/latest/>

Pandas + Matplotlib = Seaborn

```
seaborn.relplot(x="horsepower", y="mpg", hue="origin", size="weight",
                 sizes=(40, 400), alpha=.5, palette="muted",
                 height=6, data=mpg)
```



Seaborn example gallery

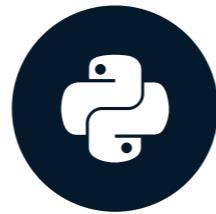
<https://seaborn.pydata.org/examples/index.html>

**Good luck
visualizing your
data!**

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Welcome to the course!

VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

Head of Data Science, Getty Images

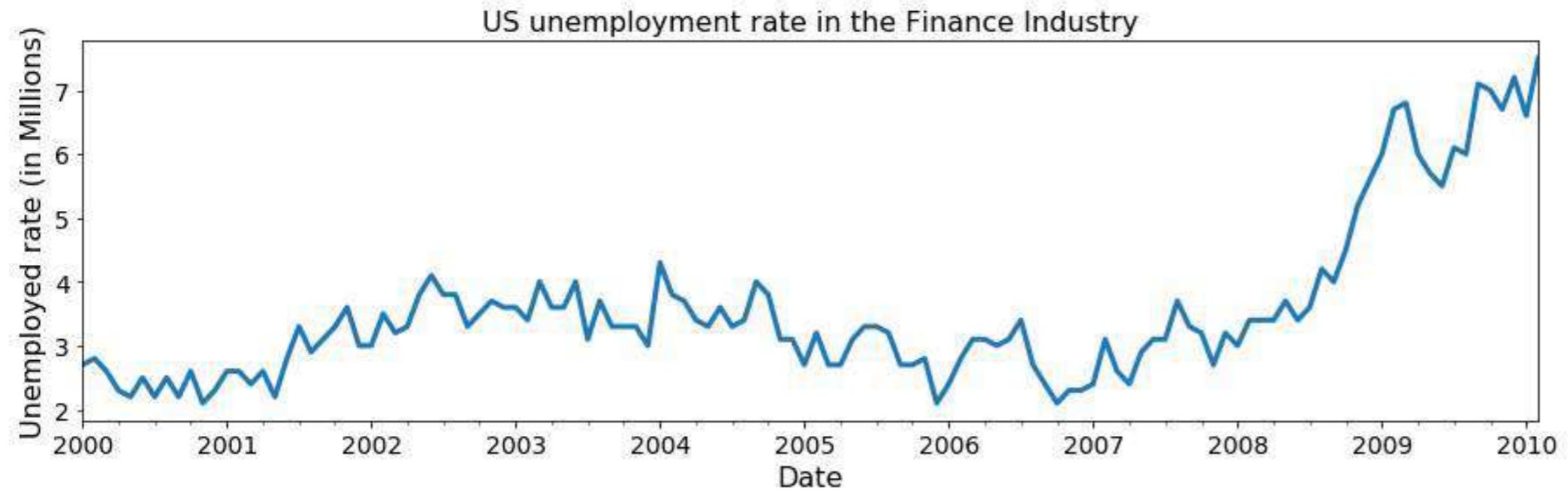
Prerequisites

- [Intro to Python for Data Science](#)
- [Intermediate Python for Data Science](#)

Time series in the field of Data Science

- Time series are a fundamental way to store and analyze many types of data
- Financial, weather and device data are all best handled as time series

Time series in the field of Data Science



Course overview

- Chapter 1: Getting started and personalizing your first time series plot
- Chapter 2: Summarizing and describing time series data
- Chapter 3: Advanced time series analysis
- Chapter 4: Working with multiple time series
- Chapter 5: Case Study

Reading data with Pandas

```
import pandas as pd  
df = pd.read_csv('ch2_co2_levels.csv')  
print(df)
```

```
      datestamp    co2  
0    1958-03-29  316.1  
1    1958-04-05  317.3  
2    1958-04-12  317.6  
...  
...  
...  
2281  2001-12-15  371.2  
2282  2001-12-22  371.3  
2283  2001-12-29  371.5
```

Preview data with Pandas

```
print(df.head(n=5))
```

```
    datestamp    co2
0  1958-03-29  316.1
1  1958-04-05  317.3
2  1958-04-12  317.6
3  1958-04-19  317.5
4  1958-04-26  316.4
```

```
print(df.tail(n=5))
```

```
    datestamp    co2
2279  2001-12-01  370.3
2280  2001-12-08  370.8
2281  2001-12-15  371.2
2282  2001-12-22  371.3
2283  2001-12-29  371.5
```

Check data types with Pandas

```
print(df.dtypes)
```

```
datestamp      object  
co2            float64  
dtype: object
```

Working with dates

To work with time series data in `pandas`, your date columns needs to be of the `datetime64` type.

```
pd.to_datetime(['2009/07/31', 'test'])
```

```
ValueError: Unknown string format
```

```
pd.to_datetime(['2009/07/31', 'test'], errors='coerce')
```

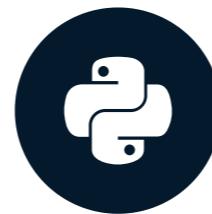
```
DatetimeIndex(['2009-07-31', 'NaT'],
                dtype='datetime64[ns]', freq=None)
```

Let's get started!

VISUALIZING TIME SERIES DATA IN PYTHON

Plot your first time series

VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

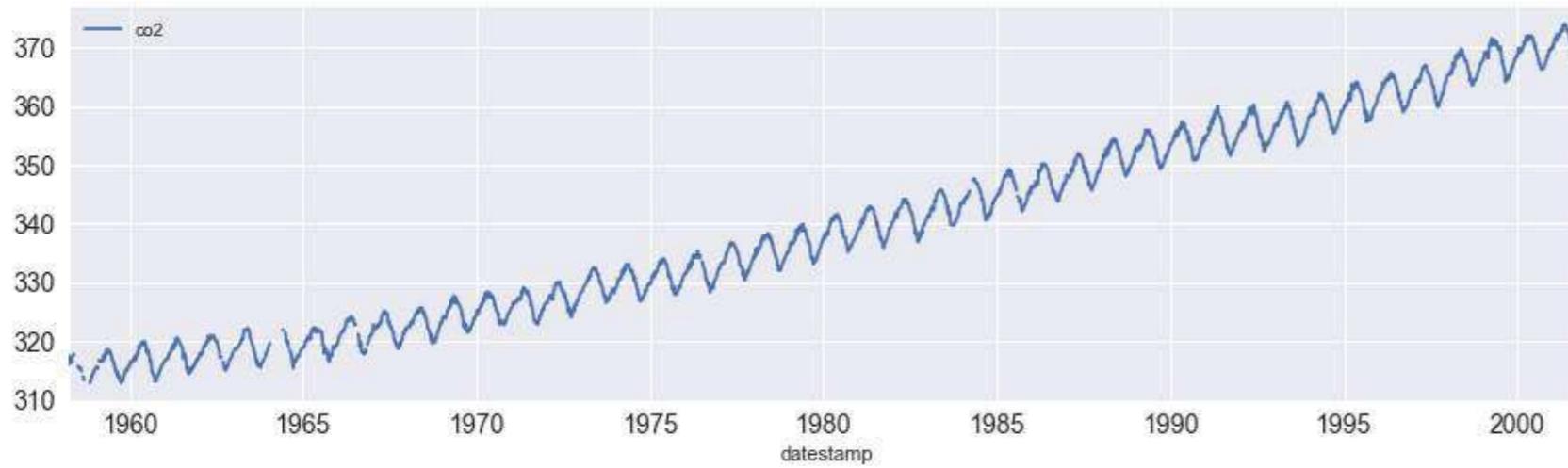
Head of Data Science, Getty Images

The Matplotlib library

- In Python, matplotlib is an extensive package used to plot data
- The pyplot submodule of matplotlib is traditionally imported using the `plt` alias

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

Plotting time series data



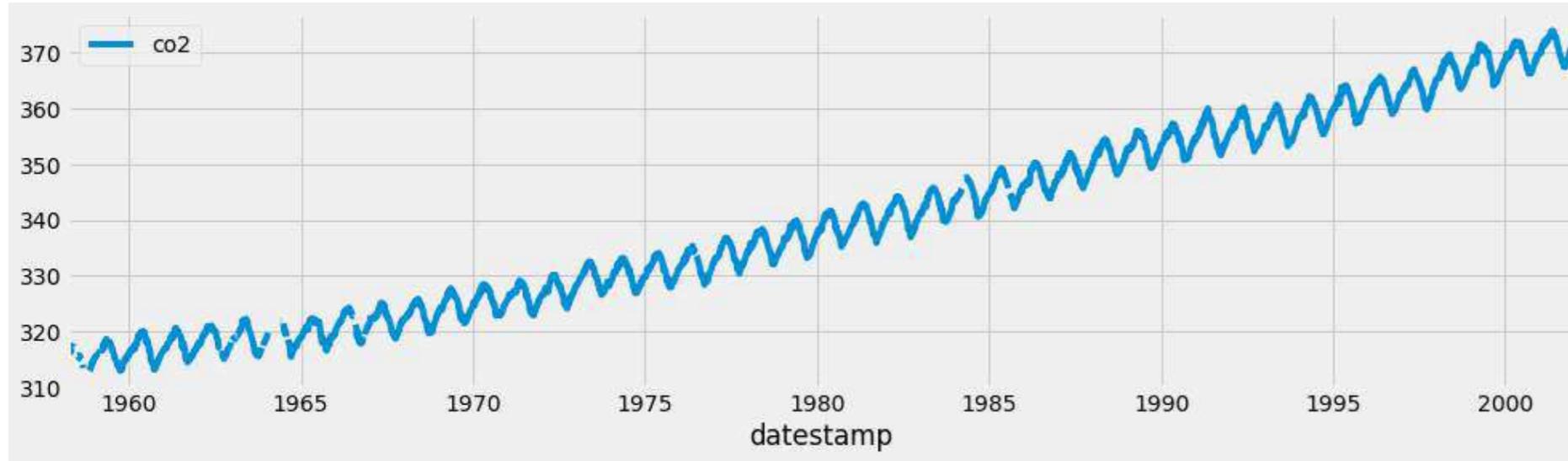
Plotting time series data

```
import matplotlib.pyplot as plt  
import pandas as pd  
  
df = df.set_index('date_column')  
df.plot()  
plt.show()
```

Adding style to your plots

```
plt.style.use('fivethirtyeight')  
df.plot()  
plt.show()
```

FiveThirtyEight style



Matplotlib style sheets

```
print(plt.style.available)
```

```
['seaborn-dark-palette', 'seaborn-darkgrid',
'seaborn-dark', 'seaborn-notebook',
'seaborn-pastel', 'seaborn-white',
'classic', 'ggplot', 'grayscale',
'dark_background', 'seaborn-poster',
'seaborn-muted', 'seaborn', 'bmh',
'seaborn-paper', 'seaborn-whitegrid',
'seaborn-bright', 'seaborn-talk',
'fivethirtyeight', 'seaborn-colorblind',
'seaborn-deep', 'seaborn-ticks']
```

Describing your graphs with labels

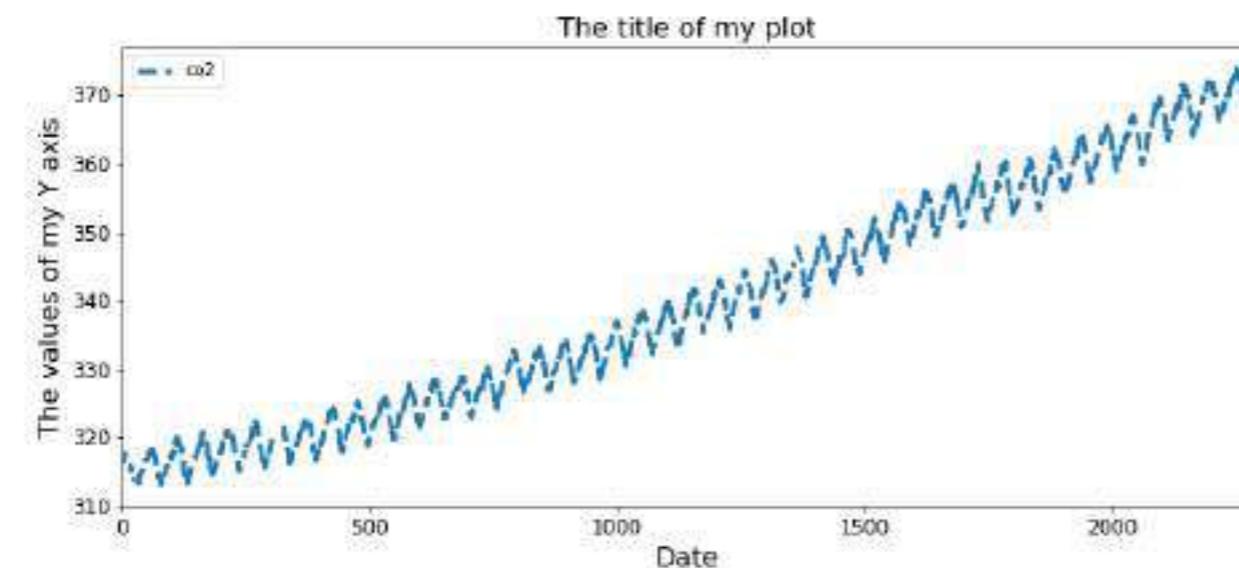
```
ax = df.plot(color='blue')
```

```
ax.set_xlabel('Date')
ax.set_ylabel('The values of my Y axis')
ax.set_title('The title of my plot')
plt.show()
```



Figure size, linewidth, linestyle and fontsize

```
ax = df.plot(figsize=(12, 5), fontsize=12,  
             linewidth=3, linestyle='--')  
  
ax.set_xlabel('Date', fontsize=16)  
ax.set_ylabel('The values of my Y axis', fontsize=16)  
ax.set_title('The title of my plot', fontsize=16)  
plt.show()
```

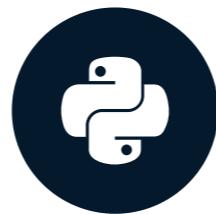


Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Customize your time series plot

VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

Head of Data Science, Getty Images

Slicing time series data

```
discoveries['1960':'1970']
```

```
discoveries['1950-01':'1950-12']
```

```
discoveries['1960-01-01':'1960-01-15']
```

Plotting subset of your time series data

```
import matplotlib.pyplot as plt  
plt.style.use('fivethirtyeight')  
df_subset = discoveries['1960':'1970']
```

```
ax = df_subset.plot(color='blue', fontsize=14)  
plt.show()
```



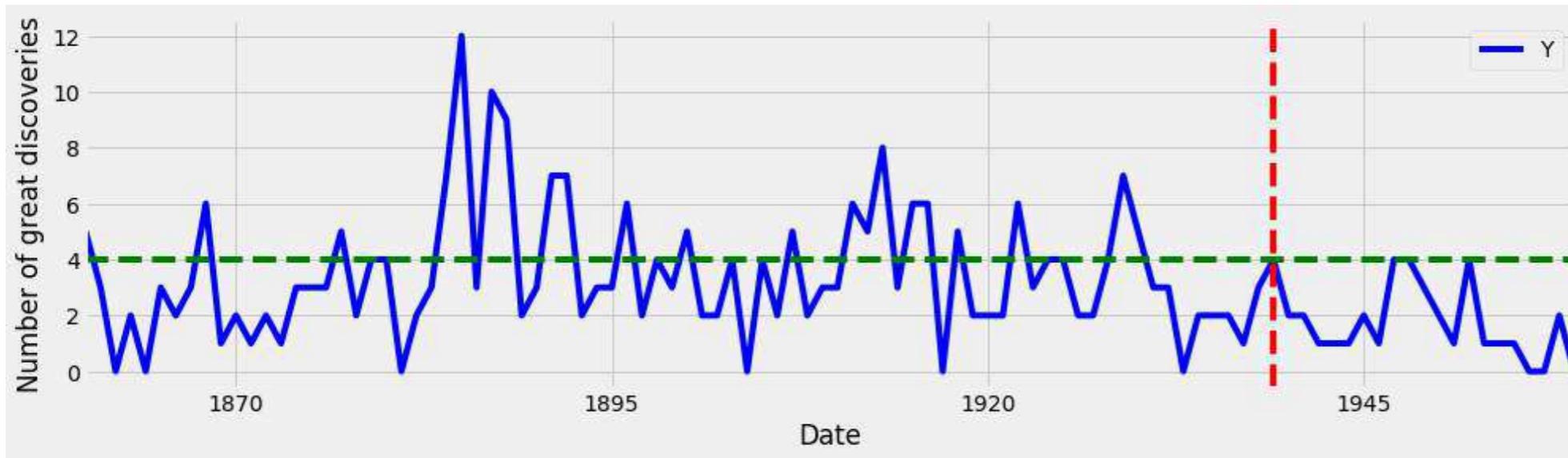
Adding markers

```
ax.axvline(x='1969-01-01',  
            color='red',  
            linestyle='--')
```

```
ax.axhline(y=100,  
            color='green',  
            linestyle='--')
```

Using markers: the full code

```
ax = discoveries.plot(color='blue')
ax.set_xlabel('Date')
ax.set_ylabel('Number of great discoveries')
ax.axvline('1969-01-01', color='red', linestyle='--')
ax.axhline(4, color='green', linestyle='--')
```



Highlighting regions of interest

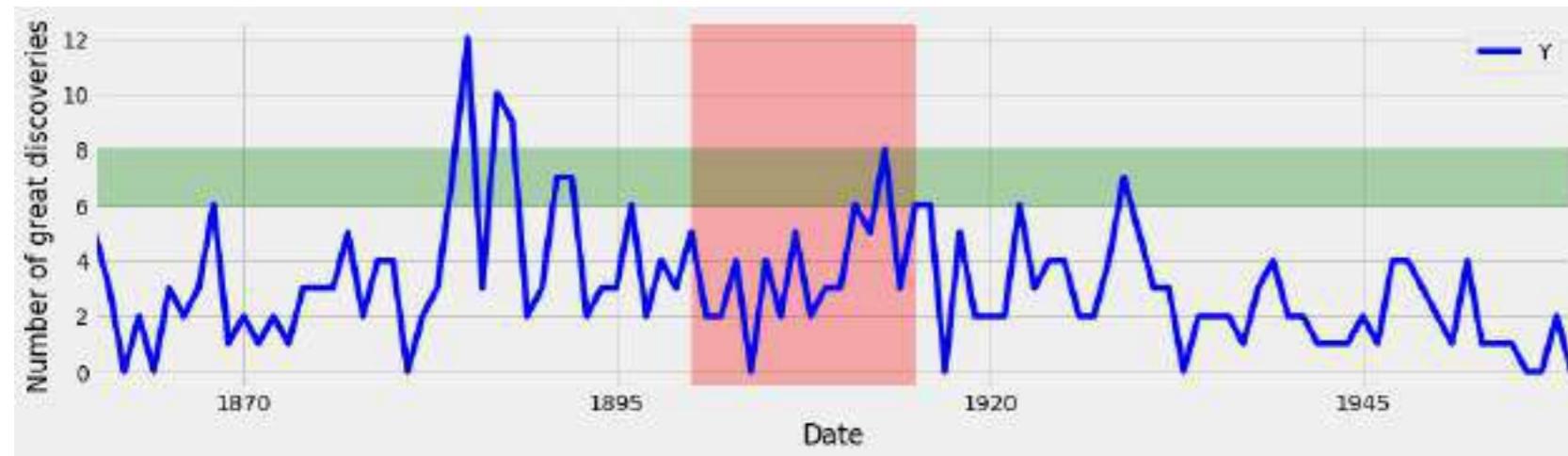
```
ax.axvspan('1964-01-01', '1968-01-01',  
           color='red', alpha=0.5)
```

```
ax.axhspan(8, 6, color='green',  
           alpha=0.2)
```

Highlighting regions of interest: the full code

```
ax = discoveries.plot(color='blue')
ax.set_xlabel('Date')
ax.set_ylabel('Number of great discoveries')
```

```
ax.axvspan('1964-01-01', '1968-01-01', color='red',
alpha=0.3)
ax.axhspan(8, 6, color='green', alpha=0.3)
```

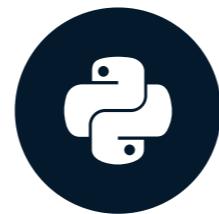


Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Clean your time series data

VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

Head of Data Science, Getty Images

The CO2 level time series

A snippet of the weekly measurements of CO2 levels at the Mauna Loa Observatory, Hawaii.

```
datastamp      co2
1958-03-29    316.1
1958-04-05    317.3
1958-04-12    317.6
...
...
2001-12-15    371.2
2001-12-22    371.3
2001-12-29    371.5
```

Finding missing values in a DataFrame

```
print(df.isnull())
```

```
datestamp    co2
1958-03-29   False
1958-04-05   False
1958-04-12   False
```

```
print(df.notnull())
```

```
datestamp    co2
1958-03-29   True
1958-04-05   True
1958-04-12   True
...
...
```

Counting missing values in a DataFrame

```
print(df.isnull().sum())
```

```
datestamp      0  
co2           59  
dtype: int64
```

Replacing missing values in a DataFrame

```
print(df)
```

```
...
5 1958-05-03 316.9
6 1958-05-10     NaN
7 1958-05-17 317.5
...
```

```
df = df.fillna(method='bfill')
print(df)
```

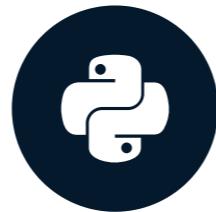
```
...
5 1958-05-03 316.9
6 1958-05-10 317.5
7 1958-05-17 317.5
...
```

Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Plot aggregates of your data

VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

Head of Data Science, Getty Images

Moving averages

- In the field of time series analysis, a moving average can be used for many different purposes:
 - smoothing out short-term fluctuations
 - removing outliers
 - highlighting long-term trends or cycles.

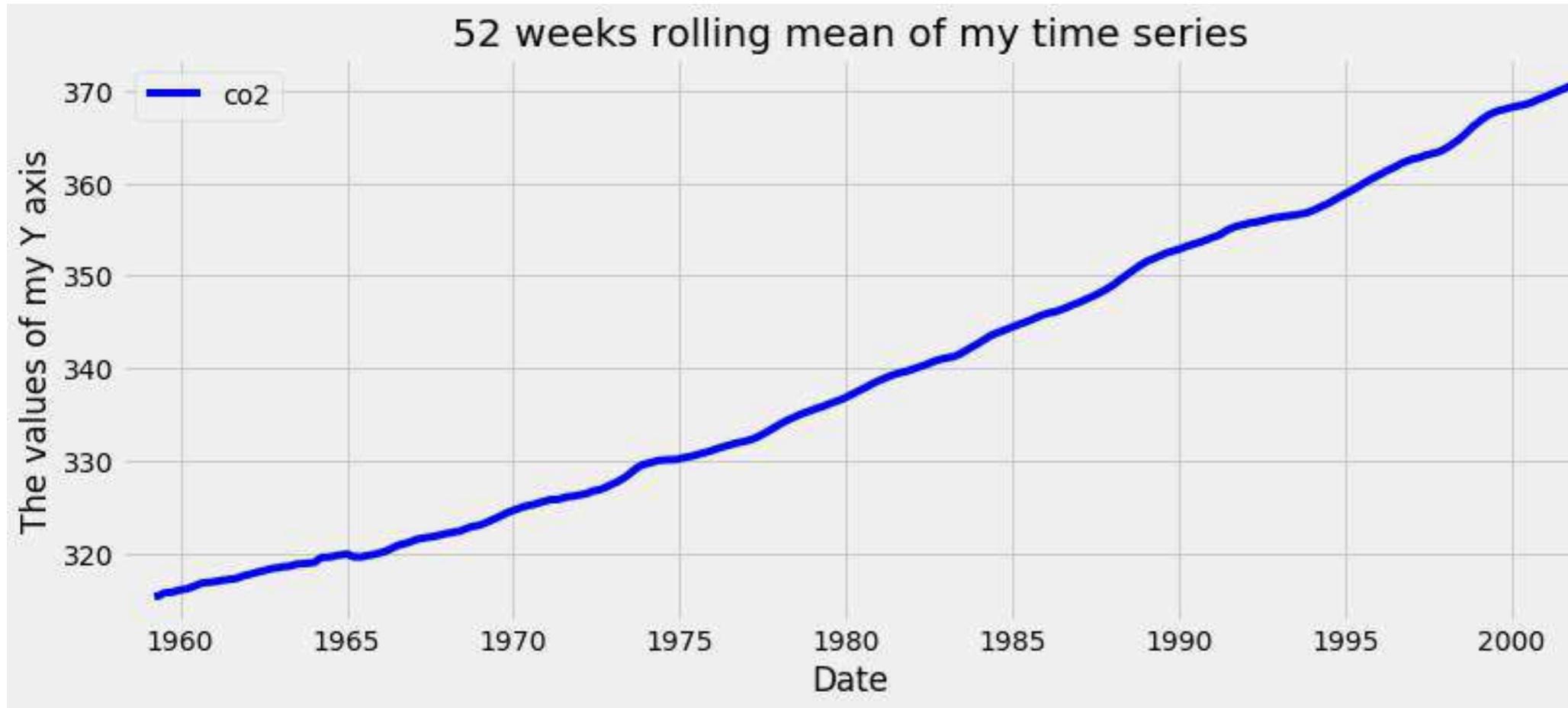
The moving average model

```
co2_levels_mean = co2_levels.rolling(window=52).mean()

ax = co2_levels_mean.plot()
ax.set_xlabel("Date")
ax.set_ylabel("The values of my Y axis")
ax.set_title("52 weeks rolling mean of my time series")

plt.show()
```

A plot of the moving average for the CO2 data



Computing aggregate values of your time series

```
co2_levels.index
```

```
DatetimeIndex(['1958-03-29', '1958-04-05', ...],  
               dtype='datetime64[ns]', name='datestamp',  
               length=2284, freq=None)
```

```
print(co2_levels.index.month)
```

```
array([ 3,  4,  4, ..., 12, 12, 12], dtype=int32)
```

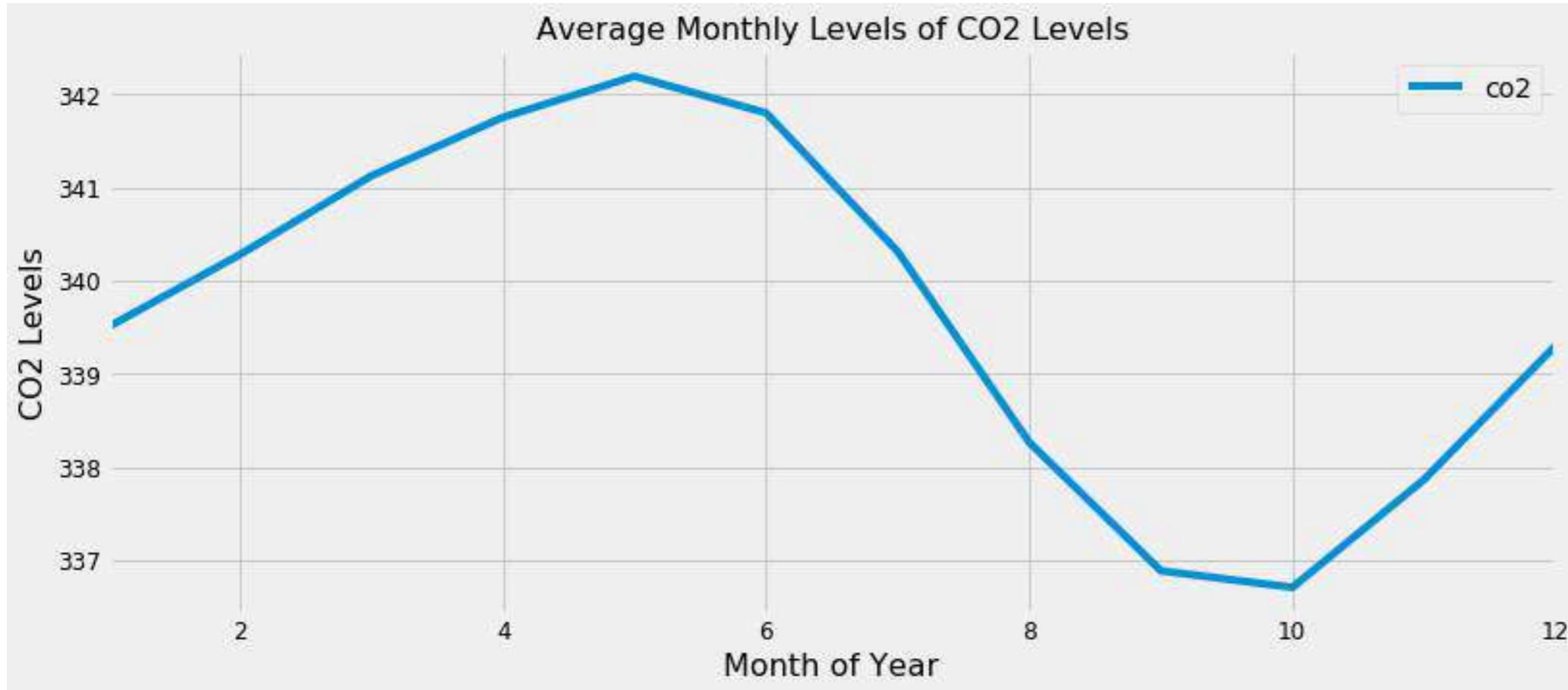
```
print(co2_levels.index.year)
```

```
array([1958, 1958, 1958, ..., 2001,  
      2001, 2001], dtype=int32)
```

Plotting aggregate values of your time series

```
index_month = co2_levels.index.month  
co2_levels_by_month = co2_levels.groupby(index_month).mean()  
co2_levels_by_month.plot()  
  
plt.show()
```

Plotting aggregate values of your time series



Let's practice!

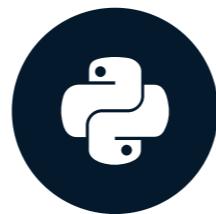
VISUALIZING TIME SERIES DATA IN PYTHON

Summarizing the values in your time series data

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent

Head of Data Science, Getty Images



Obtaining numerical summaries of your data

- What is the average value of this data?
- What is the maximum value observed in this time series?

The `.describe()` method automatically computes key statistics of all numeric columns in your DataFrame

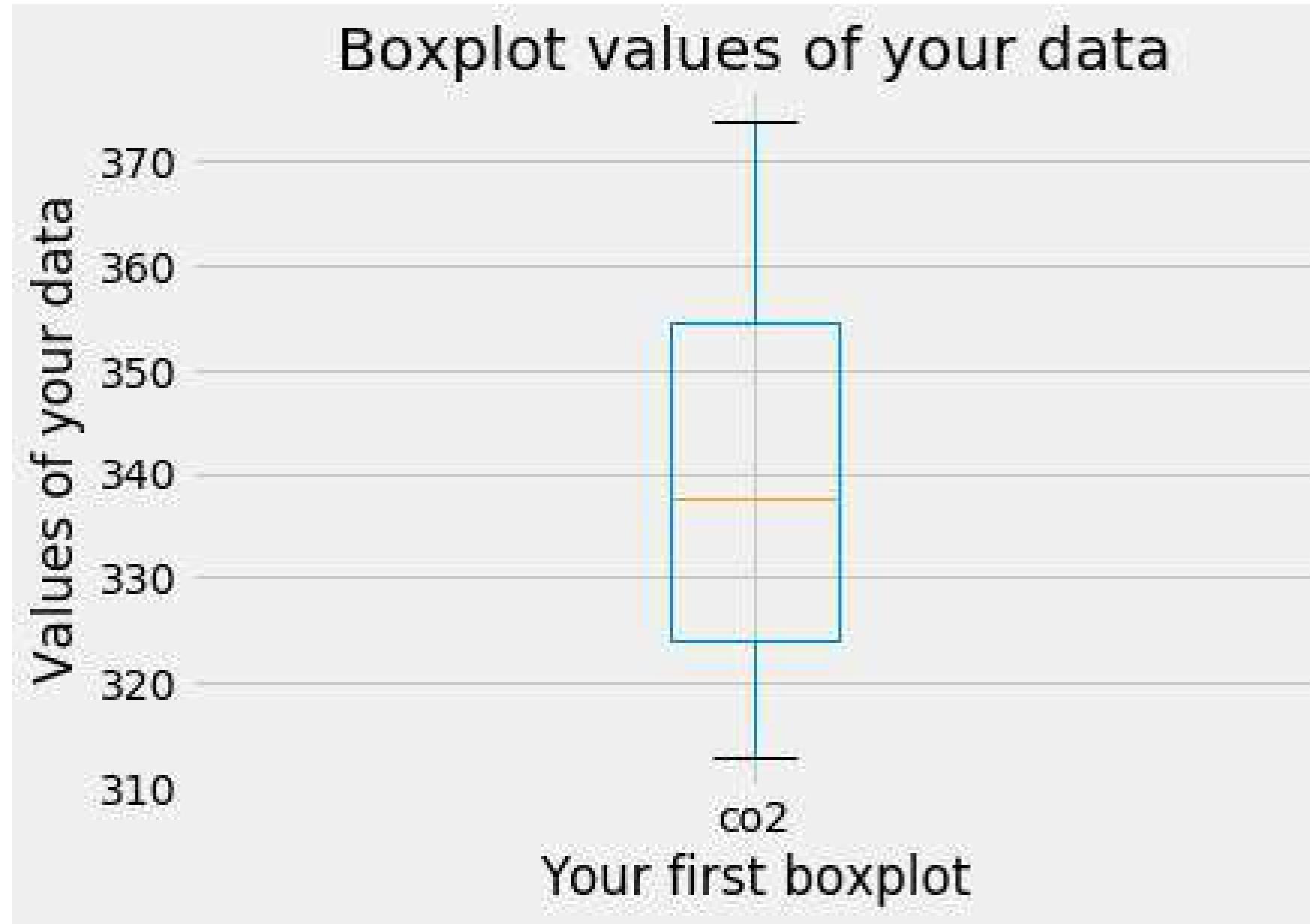
```
print(df.describe())
```

```
      co2
count  2284.000000
mean    339.657750
std     17.100899
min    313.000000
25%   323.975000
50%   337.700000
75%   354.500000
max   373.900000
```

Summarizing your data with boxplots

```
ax1 = df.boxplot()  
ax1.set_xlabel('Your first boxplot')  
ax1.set_ylabel('Values of your data')  
ax1.set_title('Boxplot values of your data')  
  
plt.show()
```

A boxplot of the values in the CO2 data

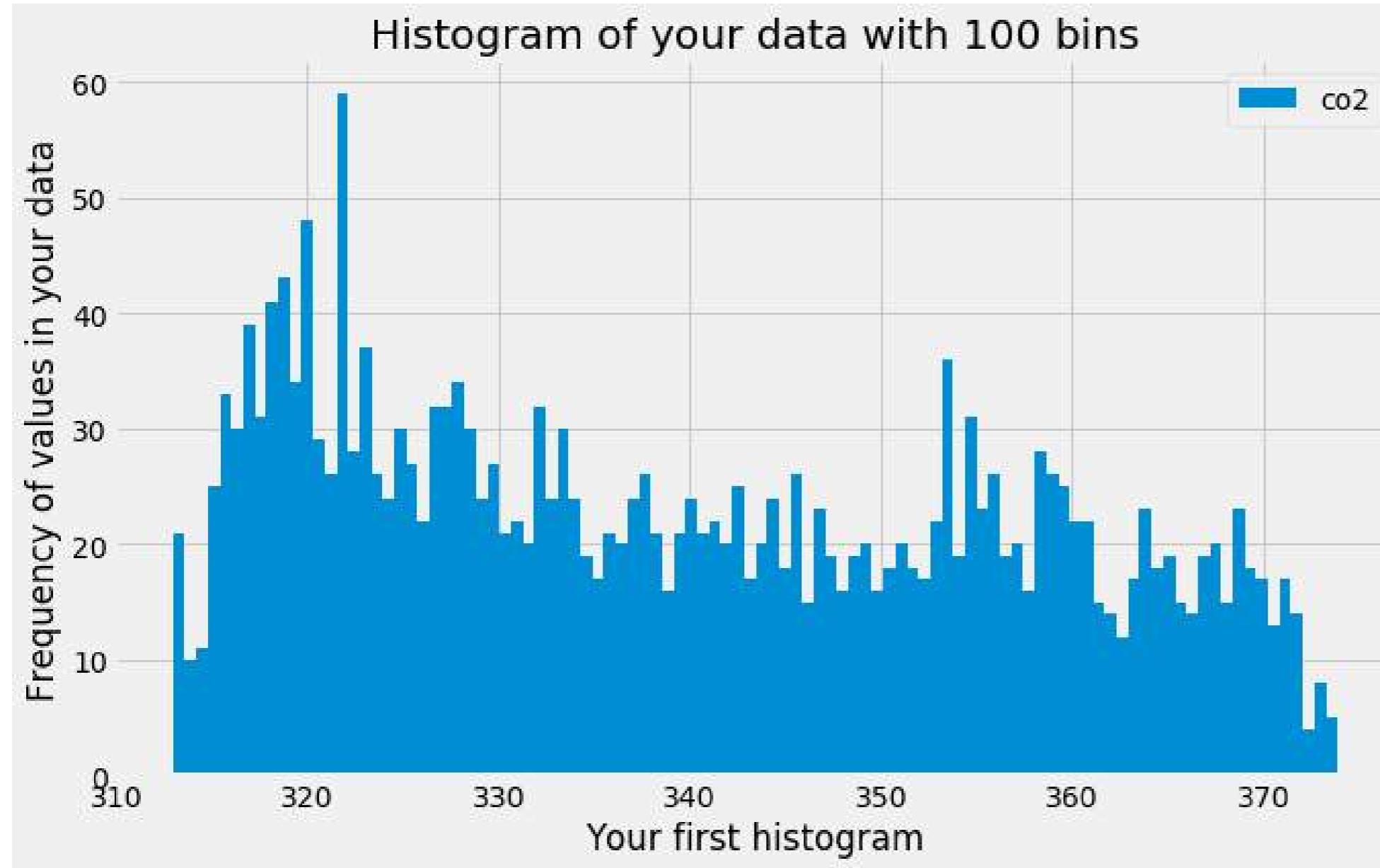


Summarizing your data with histograms

```
ax2 = df.plot(kind='hist', bins=100)
ax2.set_xlabel('Your first histogram')
ax2.set_ylabel('Frequency of values in your data')
ax2.set_title('Histogram of your data with 100 bins')

plt.show()
```

A histogram plot of the values in the CO2 data

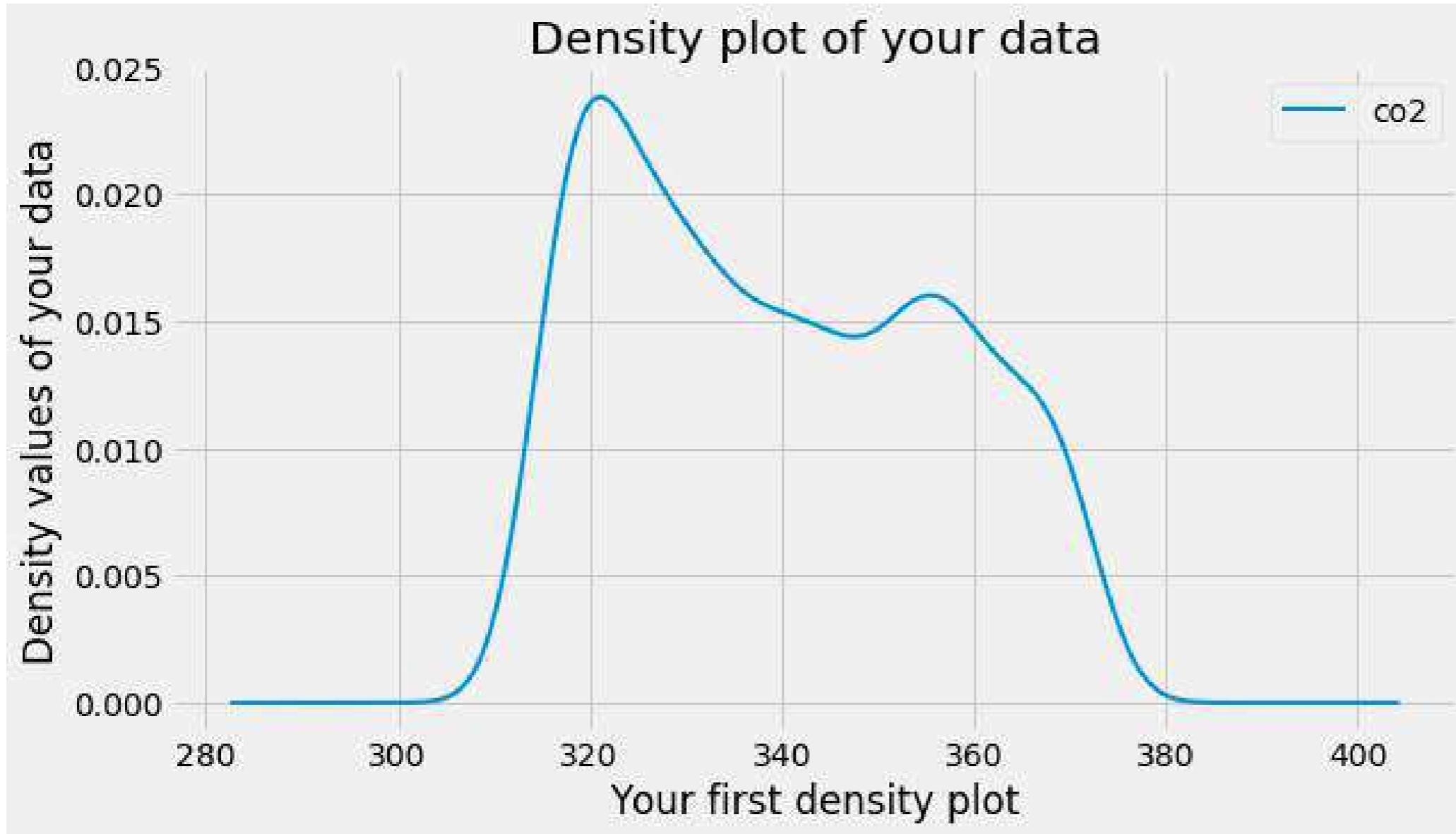


Summarizing your data with density plots

```
ax3 = df.plot(kind='density', linewidth=2)
ax3.set_xlabel('Your first density plot')
ax3.set_ylabel('Density values of your data')
ax3.set_title('Density plot of your data')

plt.show()
```

A density plot of the values in the CO2 data



Let's practice!

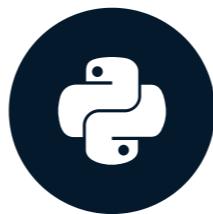
VISUALIZING TIME SERIES DATA IN PYTHON

Autocorrelation and Partial autocorrelation

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent

Head of Data Science, Getty Images



Autocorrelation in time series data

- Autocorrelation is measured as the correlation between a time series and a delayed copy of itself
- For example, an autocorrelation of order 3 returns the correlation between a time series at points (t_1 , t_2 , t_3 , ...) and its own values lagged by 3 time points, i.e. (t_4 , t_5 , t_6 , ...)
- It is used to find repetitive patterns or periodic signal in time series

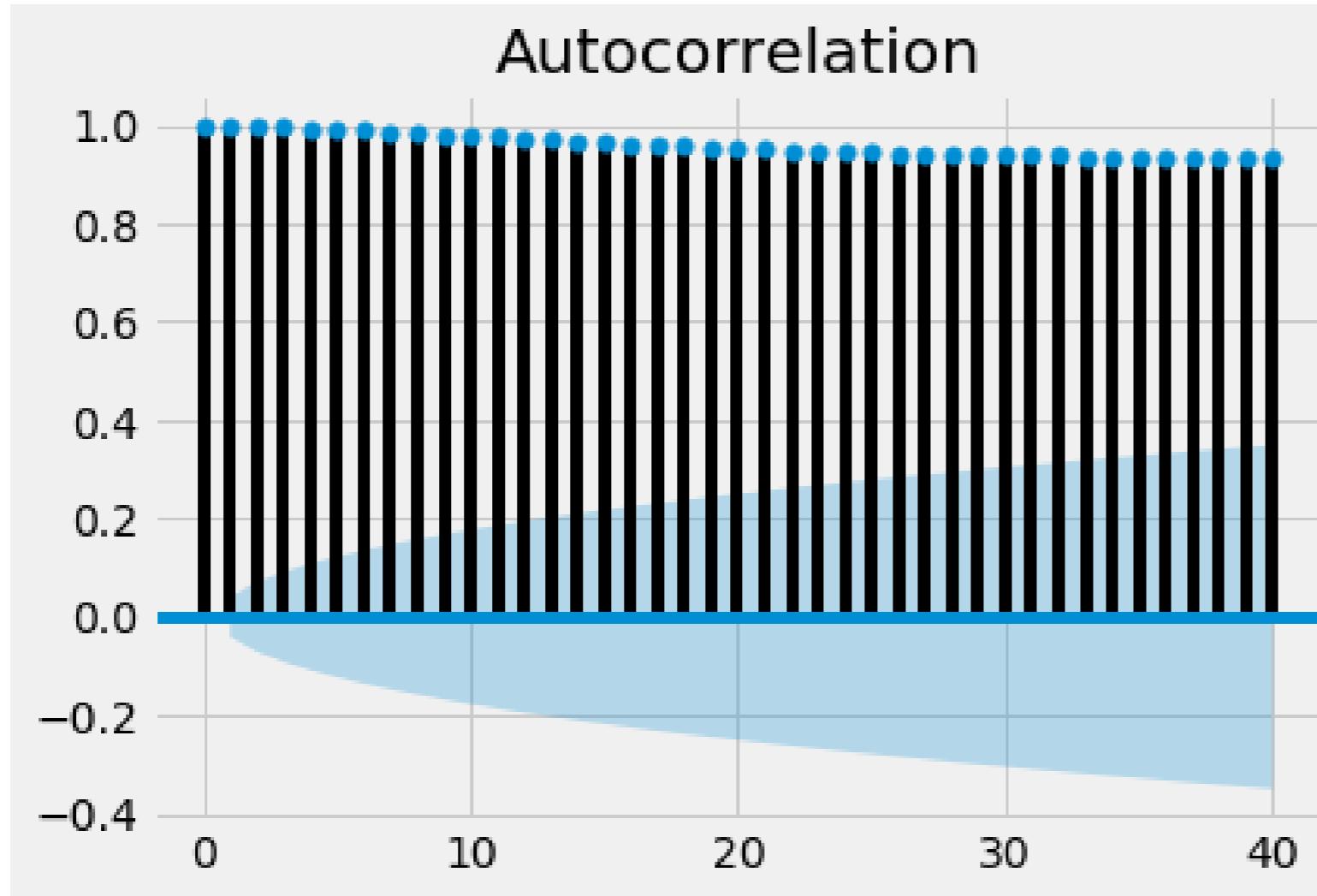
Statsmodels

`statsmodels` is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration.

Plotting autocorrelations

```
import matplotlib.pyplot as plt  
from statsmodels.graphics import tsaplots  
fig = tsaplots.plot_acf(co2_levels['co2'], lags=40)  
  
plt.show()
```

Interpreting autocorrelation plots



Partial autocorrelation in time series data

- Contrary to autocorrelation, partial autocorrelation removes the effect of previous time points
- For example, a partial autocorrelation function of `order 3` returns the correlation between our time series (`t1`, `t2`, `t3`, ...) and lagged values of itself by 3 time points (`t4`, `t5`, `t6`, ...), but only after removing all effects attributable to lags 1 and 2

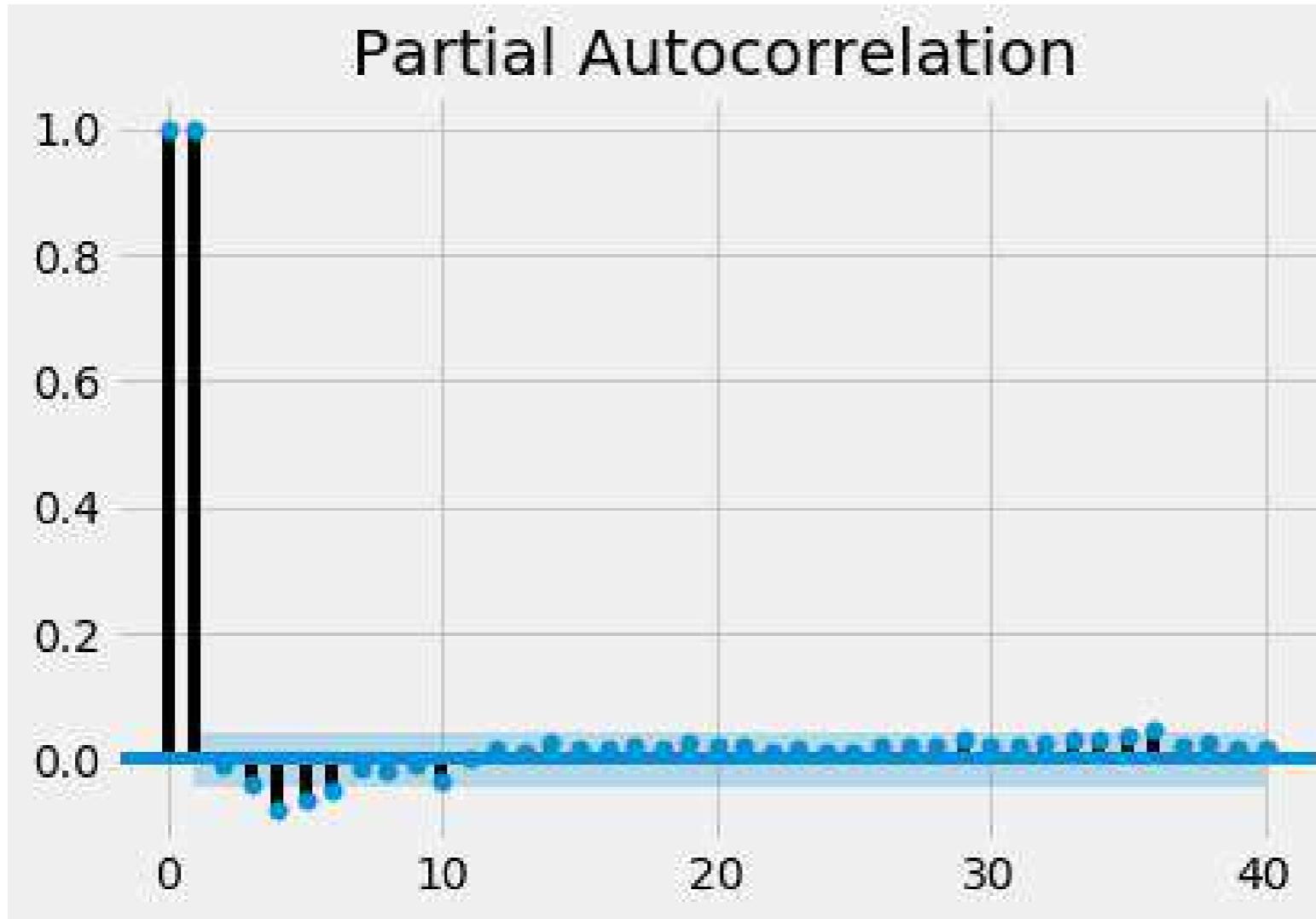
Plotting partial autocorrelations

```
import matplotlib.pyplot as plt

from statsmodels.graphics import tsaplots
fig = tsaplots.plot_pacf(co2_levels['co2'], lags=40)

plt.show()
```

Interpreting partial autocorrelations plot

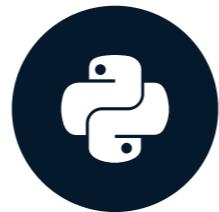


Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Seasonality, trend and noise in time series data

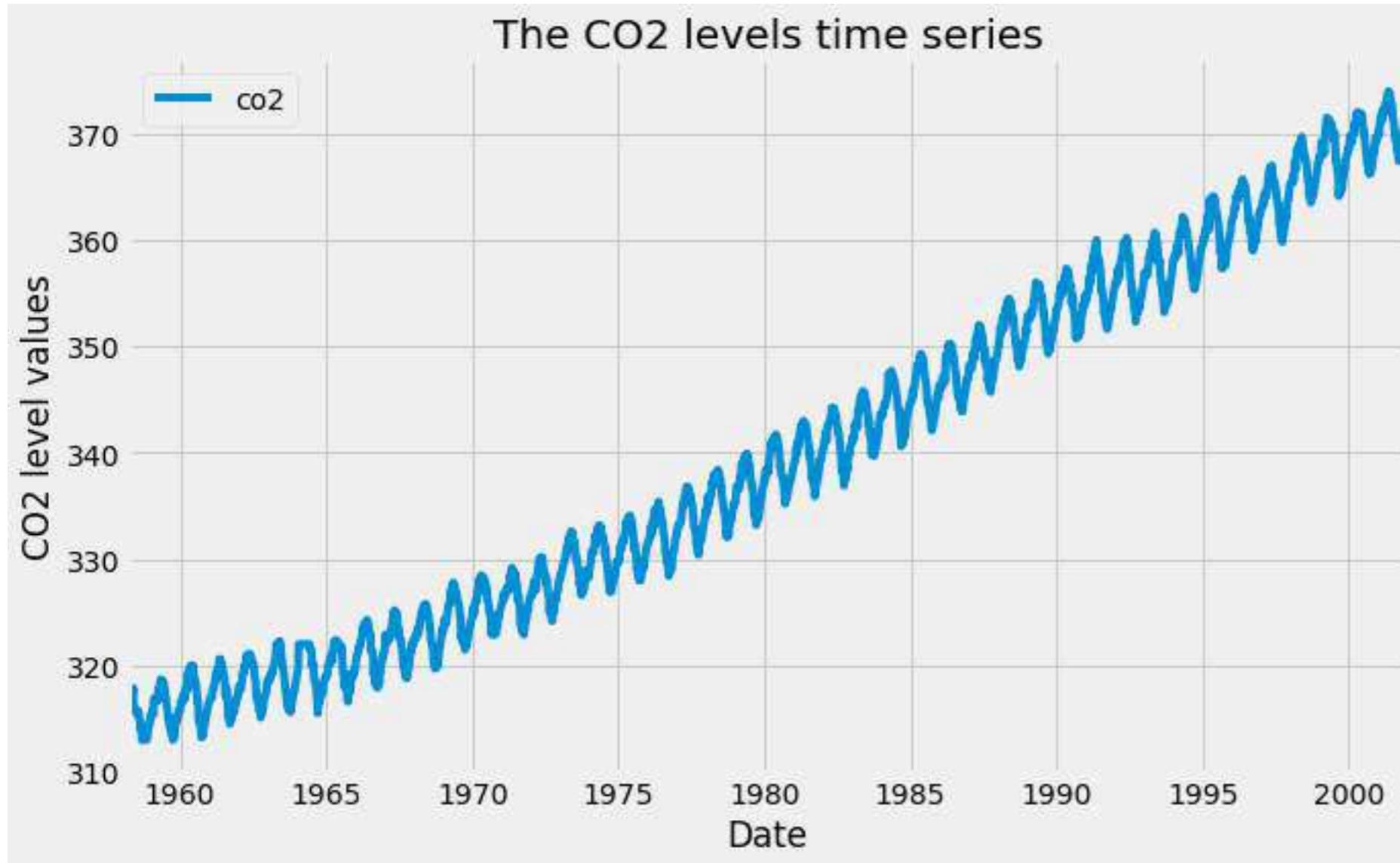
VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

Head of Data Science, Getty Images

Properties of time series



The properties of time series

- Seasonality: does the data display a clear periodic pattern?
- Trend: does the data follow a consistent upwards or downwards slope?
- Noise: are there any outlier points or missing values that are not consistent with the rest of the data?

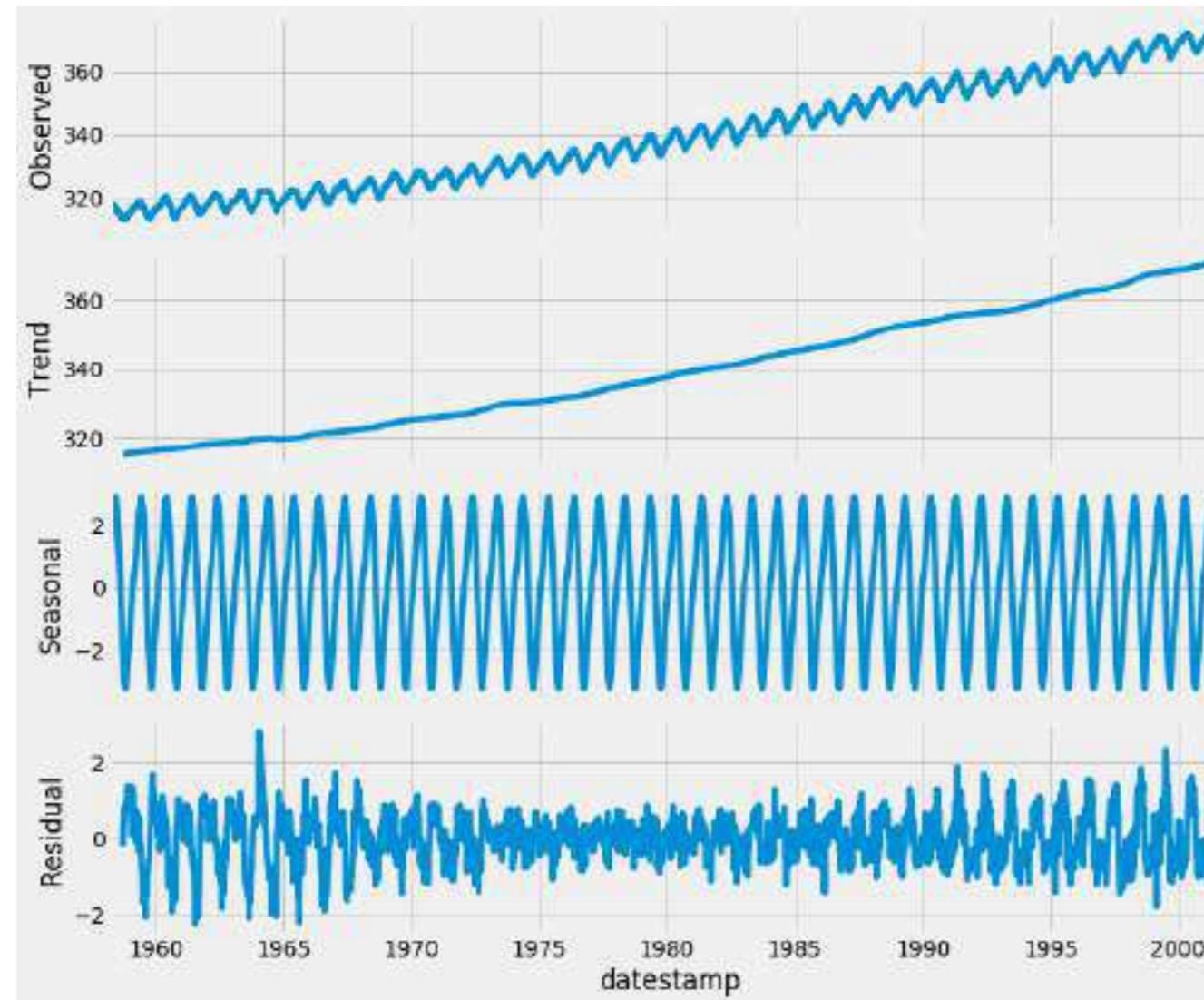
Time series decomposition

```
import statsmodels.api as sm
import matplotlib.pyplot as plt
from pylab import rcParams

rcParams['figure.figsize'] = 11, 9
decomposition = sm.tsa.seasonal_decompose(
    co2_levels['co2'])
fig = decomposition.plot()

plt.show()
```

A plot of time series decomposition on the CO2 data



Extracting components from time series decomposition

```
print(dir(decomposition))
```

```
['__class__', '__delattr__', '__dict__',
... 'plot', 'resid', 'seasonal', 'trend']
```

```
print(decomposition.seasonal)
```

```
datestamp
1958-03-29    1.028042
1958-04-05    1.235242
1958-04-12    1.412344
1958-04-19    1.701186
```

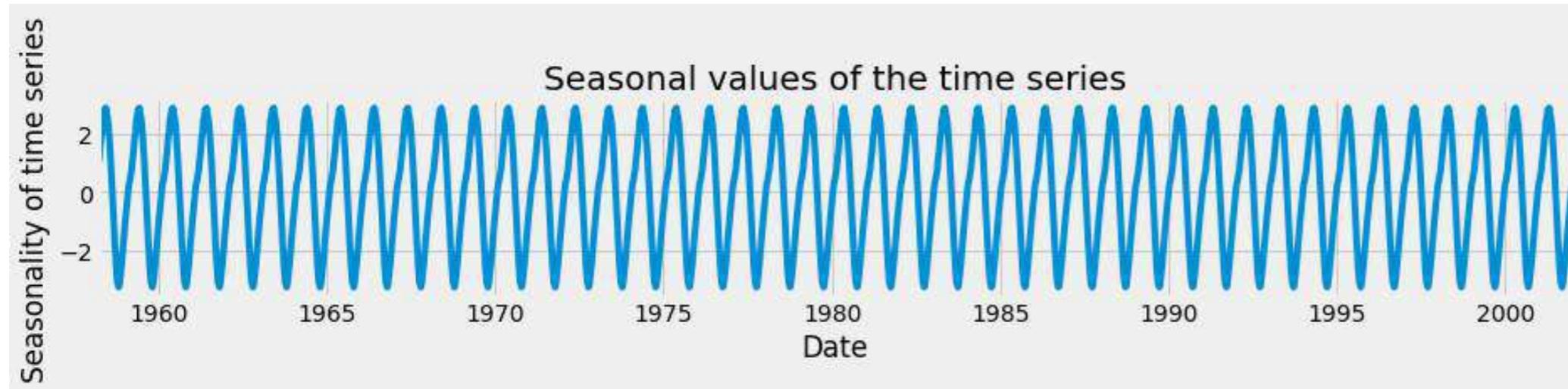
Seasonality component in time series

```
decomp_seasonal = decomposition.seasonal

ax = decomp_seasonal.plot(figsize=(14, 2))
ax.set_xlabel('Date')
ax.set_ylabel('Seasonality of time series')
ax.set_title('Seasonal values of the time series')

plt.show()
```

Seasonality component in time series



Trend component in time series

```
decomp_trend = decomposition.trend  
  
ax = decomp_trend.plot(figsize=(14, 2))  
ax.set_xlabel('Date')  
ax.set_ylabel('Trend of time series')  
ax.set_title('Trend values of the time series')  
  
plt.show()
```

Trend component in time series



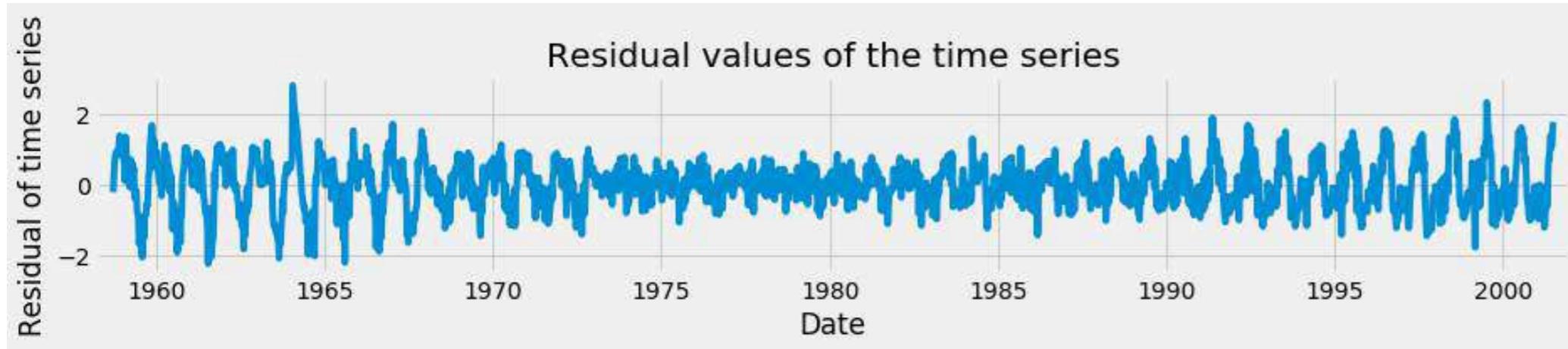
Noise component in time series

```
decomp_resid = decomp.resid

ax = decomp_resid.plot(figsize=(14, 2))
ax.set_xlabel('Date')
ax.set_ylabel('Residual of time series')
ax.set_title('Residual values of the time series')

plt.show()
```

Noise component in time series



Let's practice!

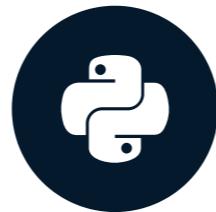
VISUALIZING TIME SERIES DATA IN PYTHON

A review on what you have learned so far

VISUALIZING TIME SERIES DATA IN PYTHON

Thomas Vincent

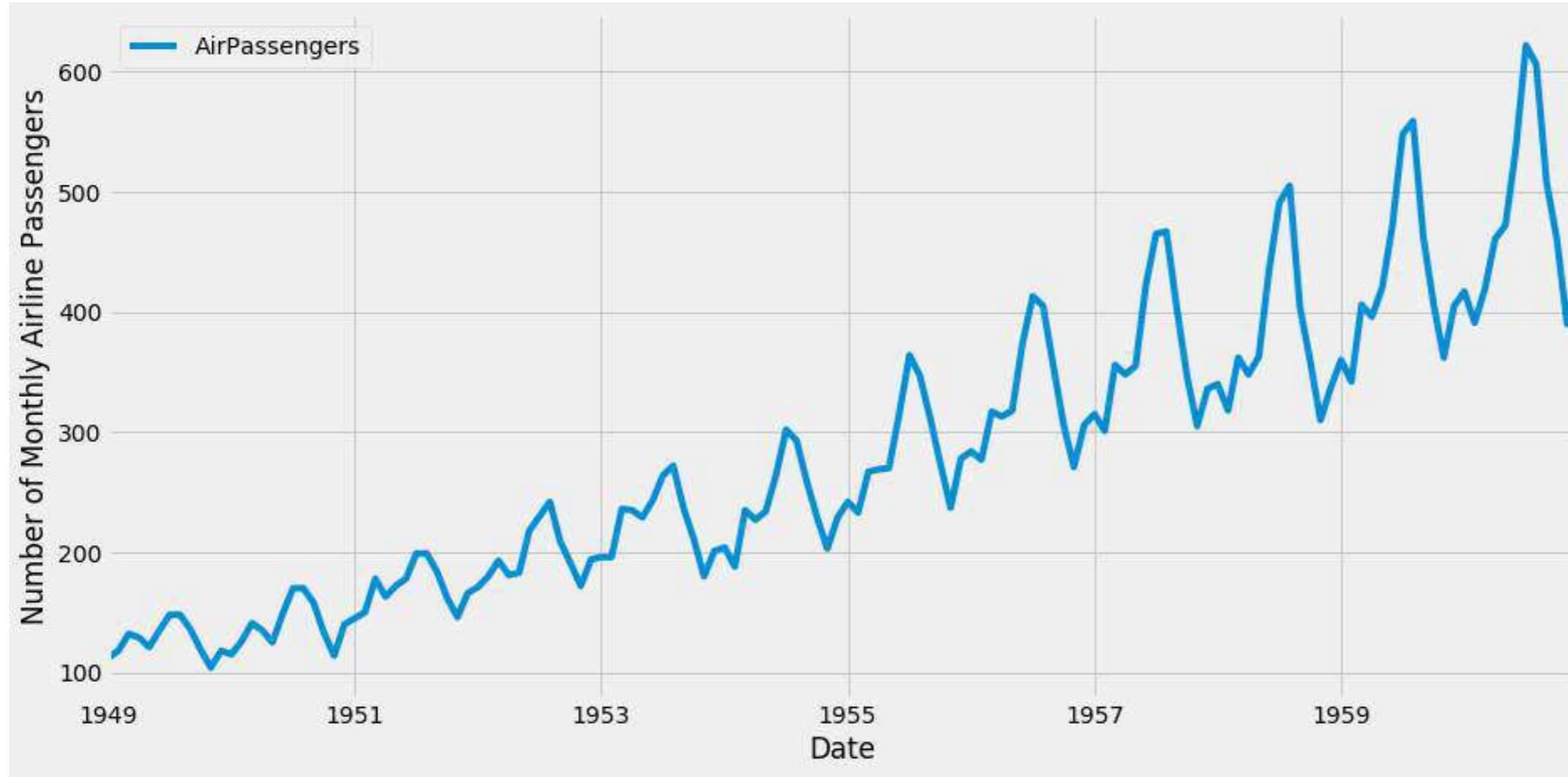
Head of Data Science, Getty Images



So far ...

- Visualize aggregates of time series data
- Extract statistical summaries
- Autocorrelation and Partial autocorrelation
- Time series decomposition

The airline dataset

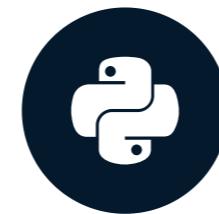


Let's analyze this data!

VISUALIZING TIME SERIES DATA IN PYTHON

Working with more than one time series

VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

Head of Data Science, Getty Images

Working with multiple time series

An isolated time series

date	ts1
1949-01	112
1949-02	118
1949-03	132

A file with multiple time series

date	ts1	ts2	ts3	ts4	ts5	ts6	ts7
2012-01-01	2113.8	10.4	1987.0	12.1	3091.8	43.2	476.7
2012-02-01	2009.0	9.8	1882.9	12.3	2954.0	38.8	466.8
2012-03-01	2159.8	10.0	1987.9	14.3	3043.7	40.1	502.1

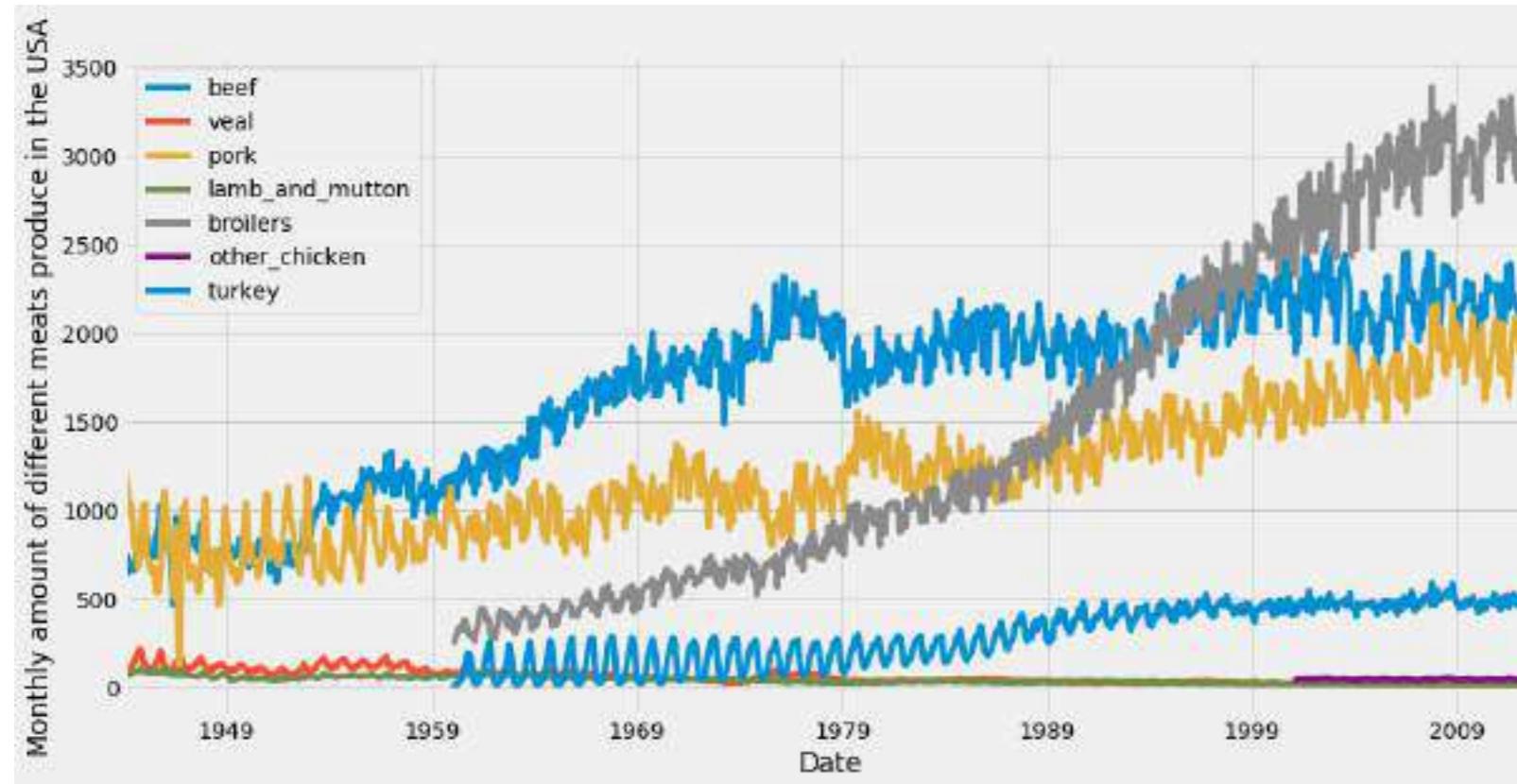
The Meat production dataset

```
import pandas as pd  
meat = pd.read_csv("meat.csv")  
print(meat.head(5))
```

```
      date    beef    veal    pork lamb_and_mutton broilers  
0  1944-01-01  751.0   85.0  1280.0            89.0       NaN  
1  1944-02-01  713.0   77.0  1169.0            72.0       NaN  
2  1944-03-01  741.0   90.0  1128.0            75.0       NaN  
3  1944-04-01  650.0   89.0   978.0            66.0       NaN  
4  1944-05-01  681.0  106.0  1029.0            78.0       NaN  
  
other_chicken turkey  
0             NaN     NaN  
1             NaN     NaN  
2             NaN     NaN  
3             NaN     NaN  
4             NaN     NaN
```

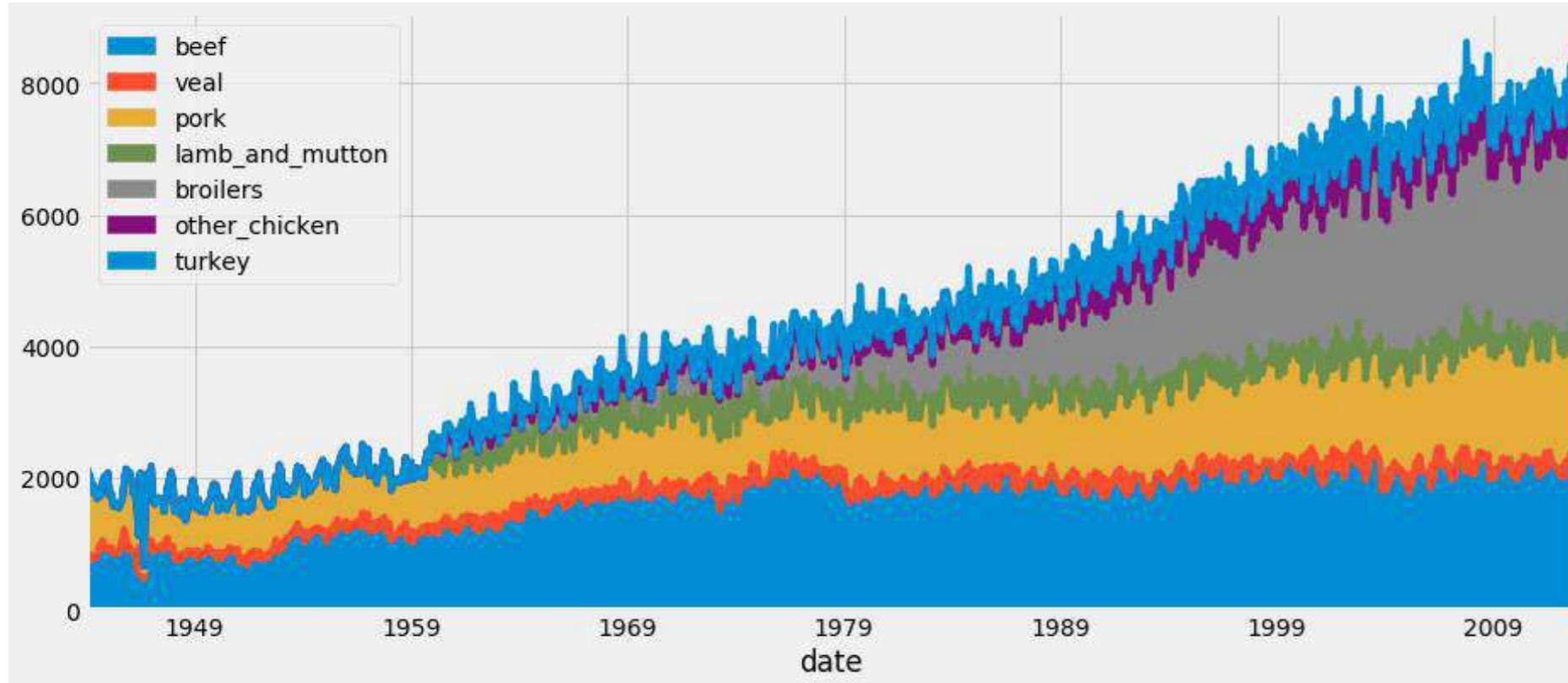
Summarizing and plotting multiple time series

```
import matplotlib.pyplot as plt  
plt.style.use('fivethirtyeight')  
ax = df.plot(figsize=(12, 4), fontsize=14)  
  
plt.show()
```



Area charts

```
import matplotlib.pyplot as plt  
plt.style.use('fivethirtyeight')  
ax = df.plot.area(figsize=(12, 4), fontsize=14)  
  
plt.show()
```

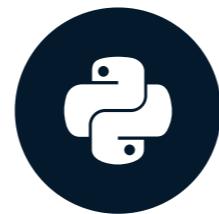


Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Plot multiple time series

VISUALIZING TIME SERIES DATA IN PYTHON

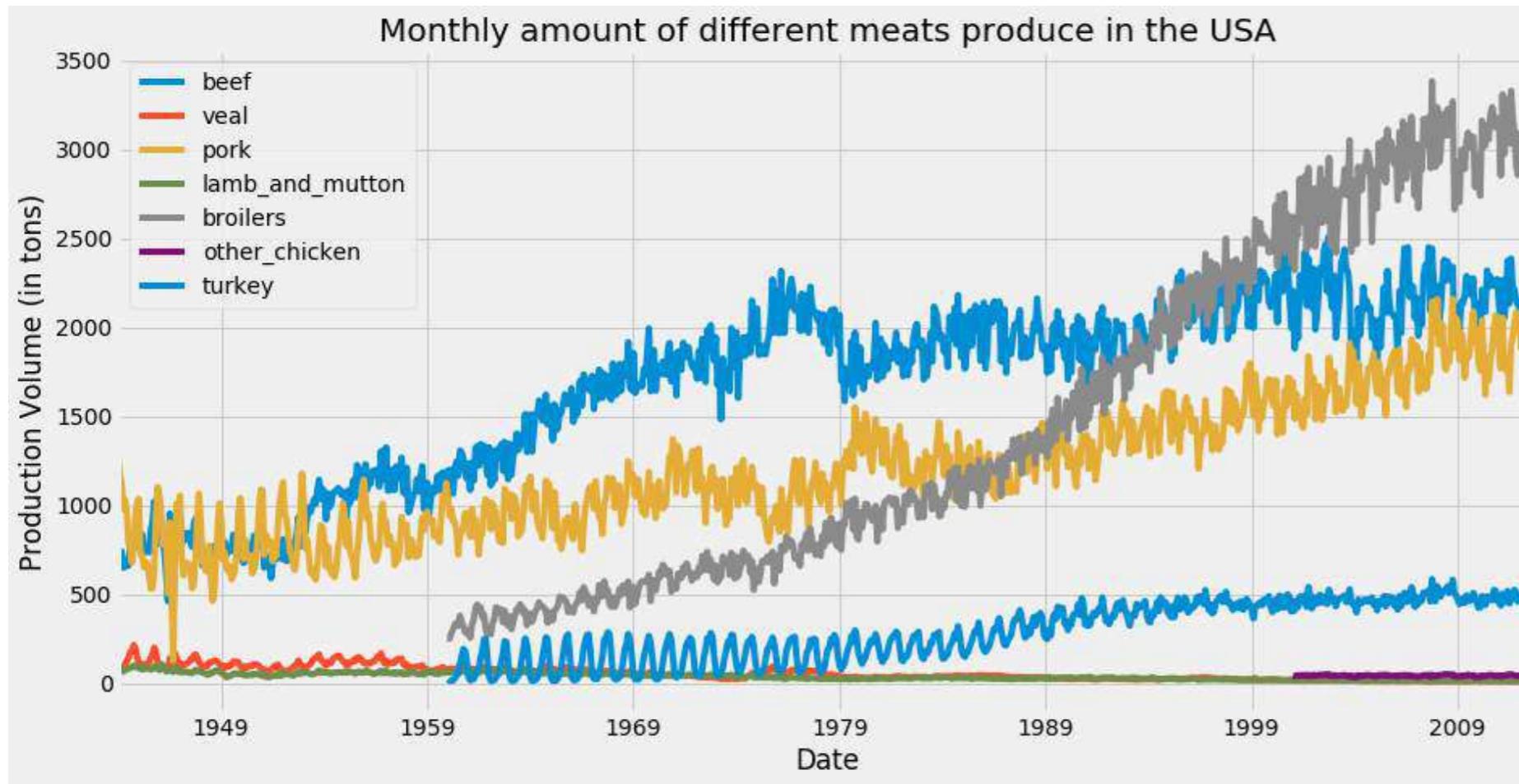


Thomas Vincent

Head of Data Science, Getty Images

Clarity is key

In this plot, the default `matplotlib` color scheme assigns the same color to the `beef` and `turkey` time series.

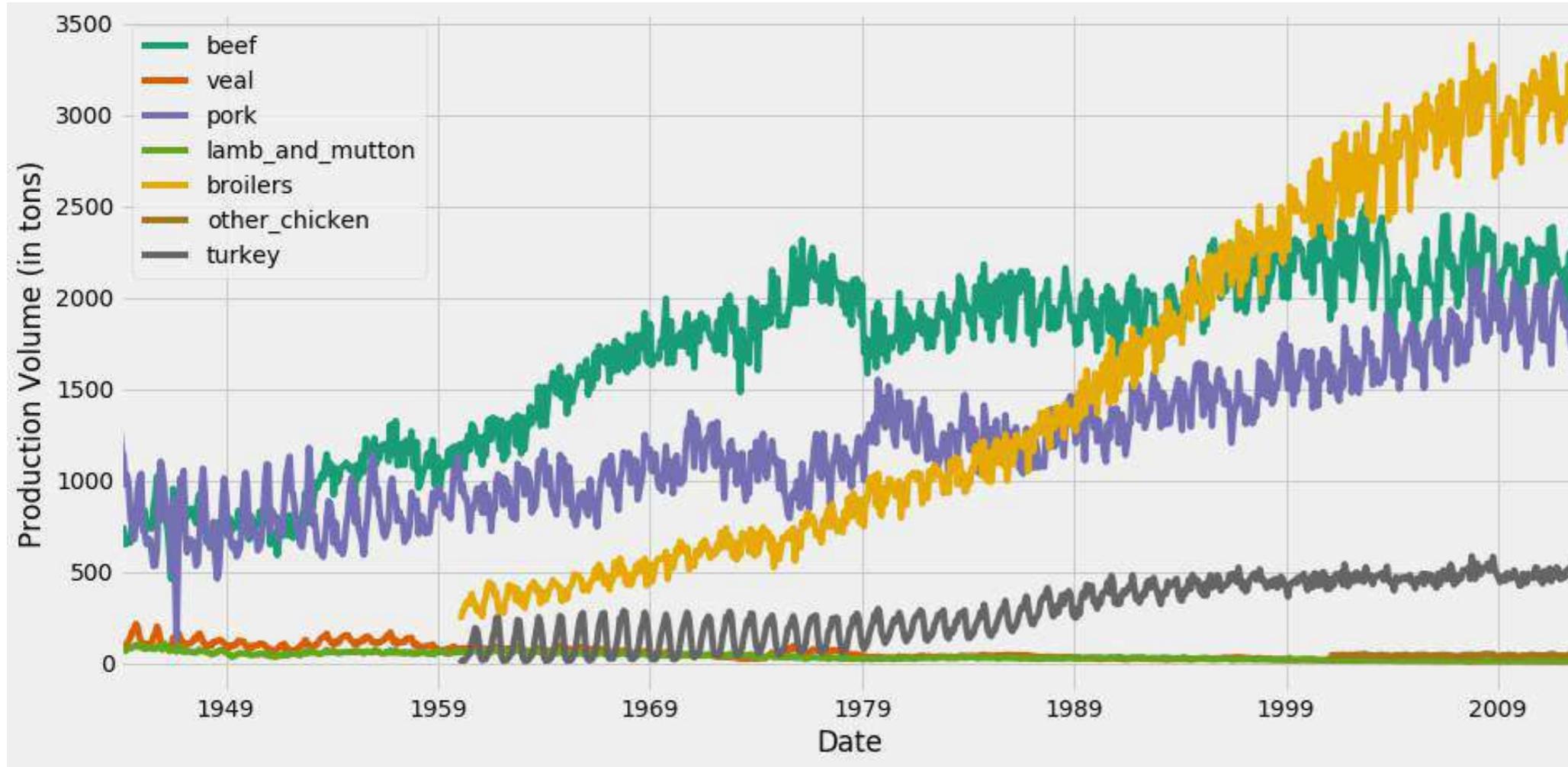


The colormap argument

```
ax = df.plot(colormap='Dark2', figsize=(14, 7))  
ax.set_xlabel('Date')  
ax.set_ylabel('Production Volume (in tons)')  
  
plt.show()
```

For the full set of available colormaps, click [here](#).

Changing line colors with the colormap argument



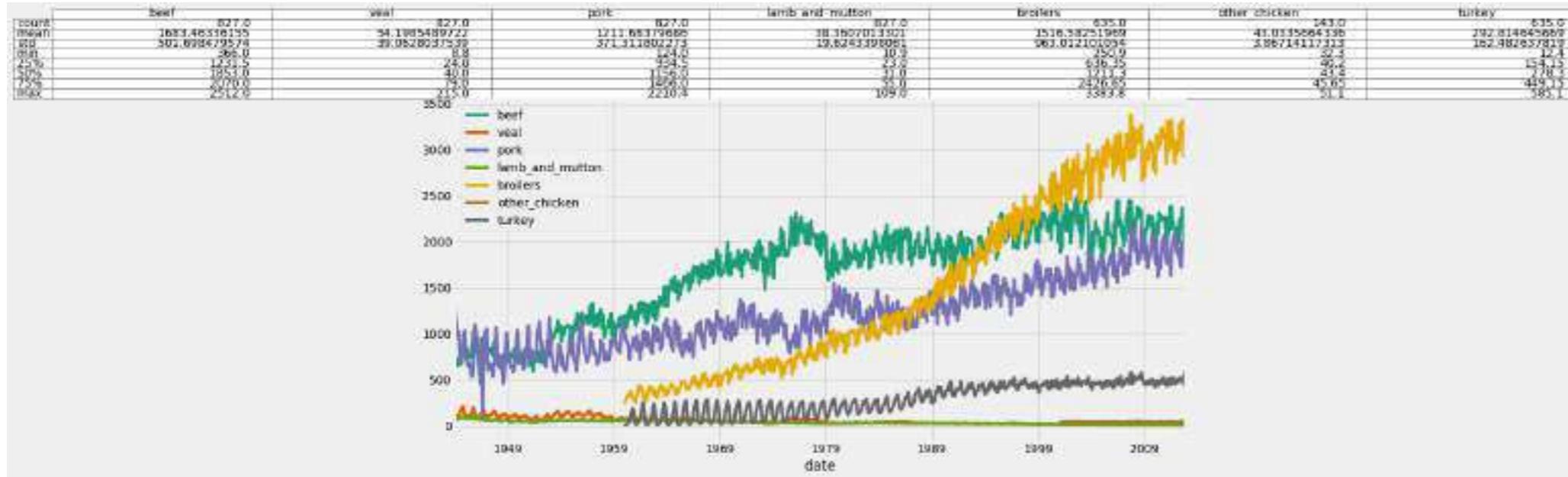
Enhancing your plot with information

```
ax = df.plot(colormap='Dark2', figsize=(14, 7))
df_summary = df.describe()

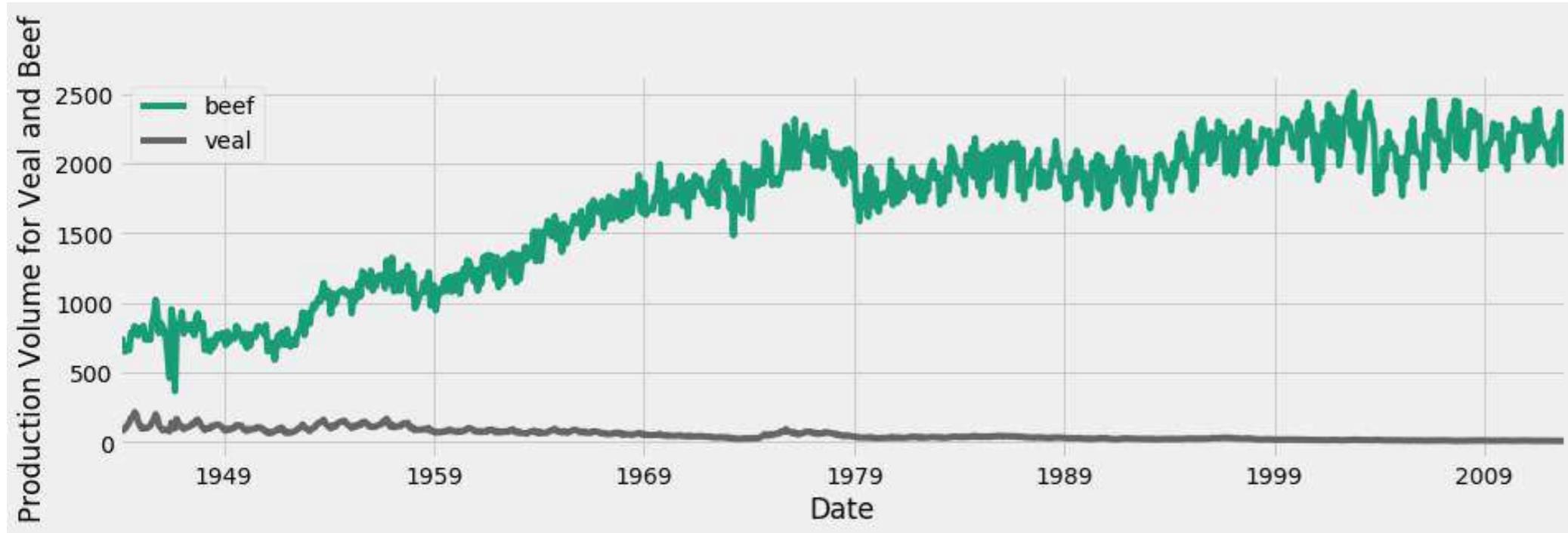
# Specify values of cells in the table
ax.table(cellText=df_summary.values,
          # Specify width of the table
          colWidths=[0.3]*len(df.columns),
          # Specify row labels
          rowLabels=df_summary.index,
          # Specify column labels
          colLabels=df_summary.columns,
          # Specify location of the table
          loc='top')

plt.show()
```

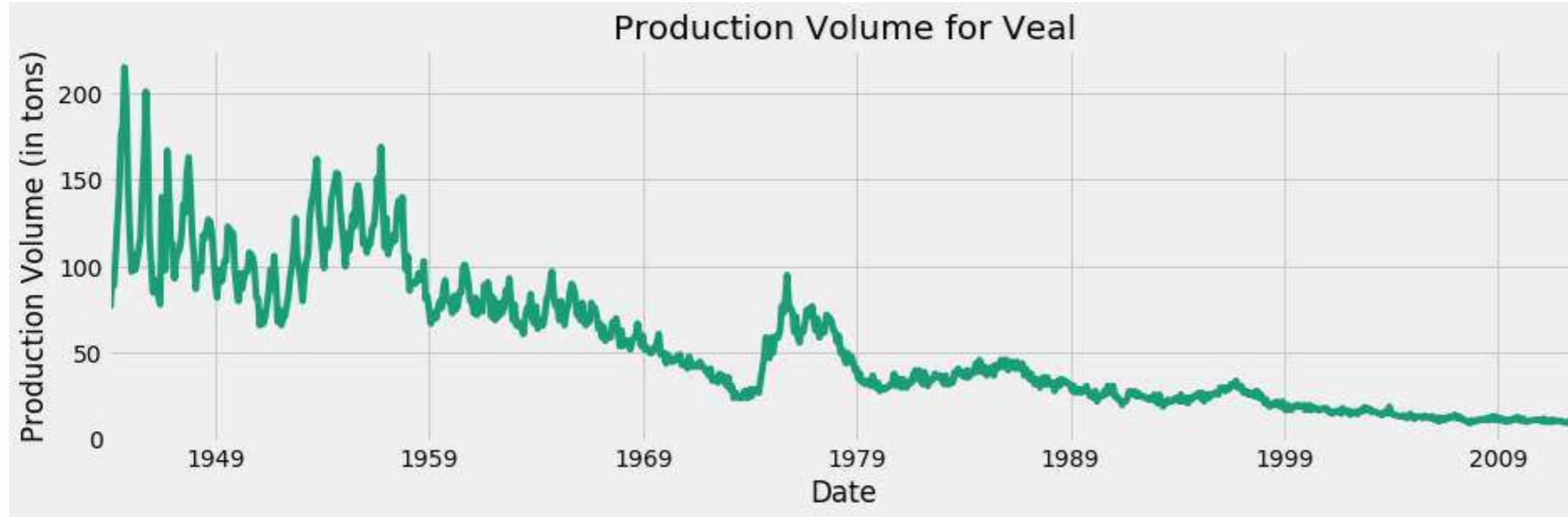
Adding Statistical summaries to your plots



Dealing with different scales

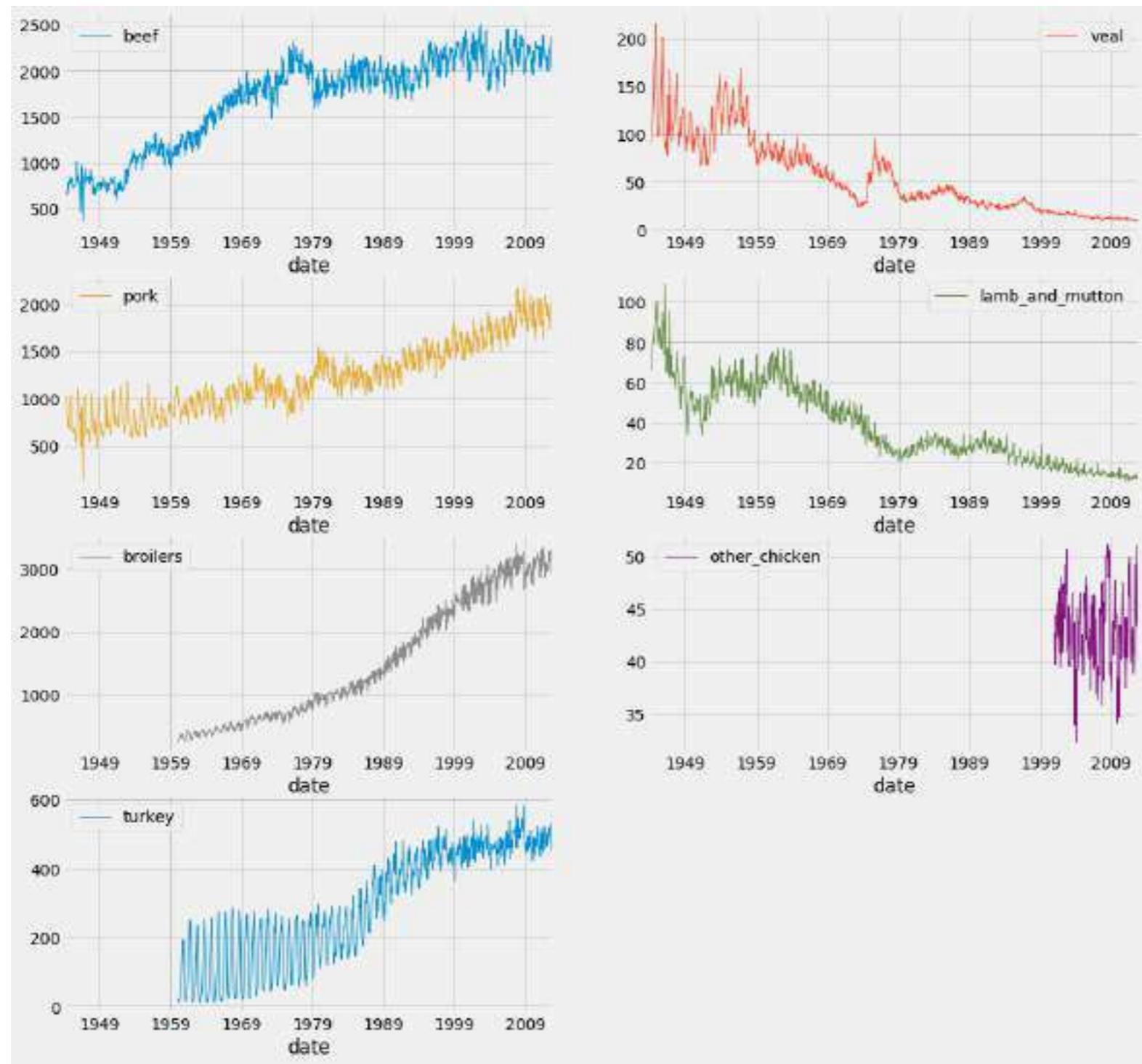


Only veal



Facet plots

```
df.plot(subplots=True,  
        linewidth=0.5,  
        layout=(2, 4),  
        figsize=(16, 10),  
        sharex=False,  
        sharey=False)  
  
plt.show()
```



Time for some action!

VISUALIZING TIME SERIES DATA IN PYTHON

Find relationships between multiple time series

VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

Head of Data Science, Getty Images

Correlations between two variables

- In the field of Statistics, the correlation coefficient is a measure used to determine the strength or lack of relationship between two variables:
 - Pearson's coefficient can be used to compute the correlation coefficient between variables for which the relationship is thought to be linear
 - Kendall Tau or Spearman rank can be used to compute the correlation coefficient between variables for which the relationship is thought to be non-linear

Compute correlations

```
from scipy.stats.stats import pearsonr  
from scipy.stats.stats import spearmanr  
from scipy.stats.stats import kendalltau  
x = [1, 2, 4, 7]  
y = [1, 3, 4, 8]  
pearsonr(x, y)
```

```
SpearmanResult(correlation=0.9843, pvalue=0.01569)
```

```
spearmanr(x, y)
```

```
SpearmanResult(correlation=1.0, pvalue=0.0)
```

```
kendalltau(x, y)
```

```
KendalltauResult(correlation=1.0, pvalue=0.0415)
```

What is a correlation matrix?

- When computing the correlation coefficient between more than two variables, you obtain a correlation matrix
 - Range: [-1, 1]
 - 0: no relationship
 - 1: strong positive relationship
 - -1: strong negative relationship

What is a correlation matrix?

- A correlation matrix is always "symmetric"
- The diagonal values will always be equal to 1

	x	y	z
x	1.00	-0.46	0.49
y	-0.46	1.00	-0.61
z	0.49	-0.61	1.00

Computing Correlation Matrices with Pandas

```
corr_p = meat[['beef', 'veal', 'turkey']].corr(method='pearson')
print(corr_p)
```

```
          beef      veal     turkey
beef    1.000   -0.829    0.738
veal   -0.829    1.000   -0.768
turkey  0.738   -0.768    1.000
```

```
corr_s = meat[['beef', 'veal', 'turkey']].corr(method='spearman')
print(corr_s)
```

```
          beef      veal     turkey
beef    1.000   -0.812    0.778
veal   -0.812    1.000   -0.829
turkey  0.778   -0.829    1.000
```

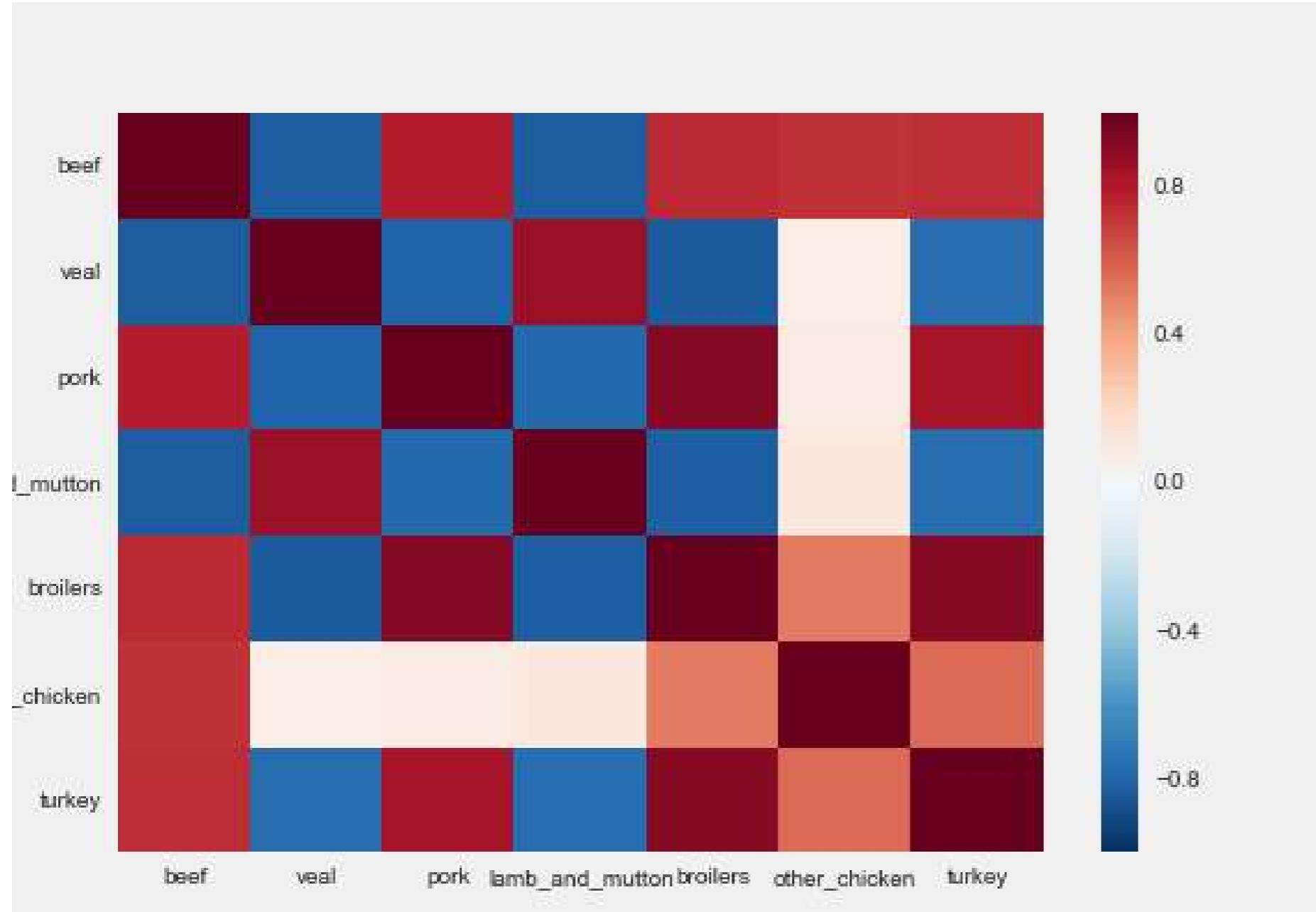
Computing Correlation Matrices with Pandas

```
corr_mat = meat.corr(method='pearson')
```

Heatmap

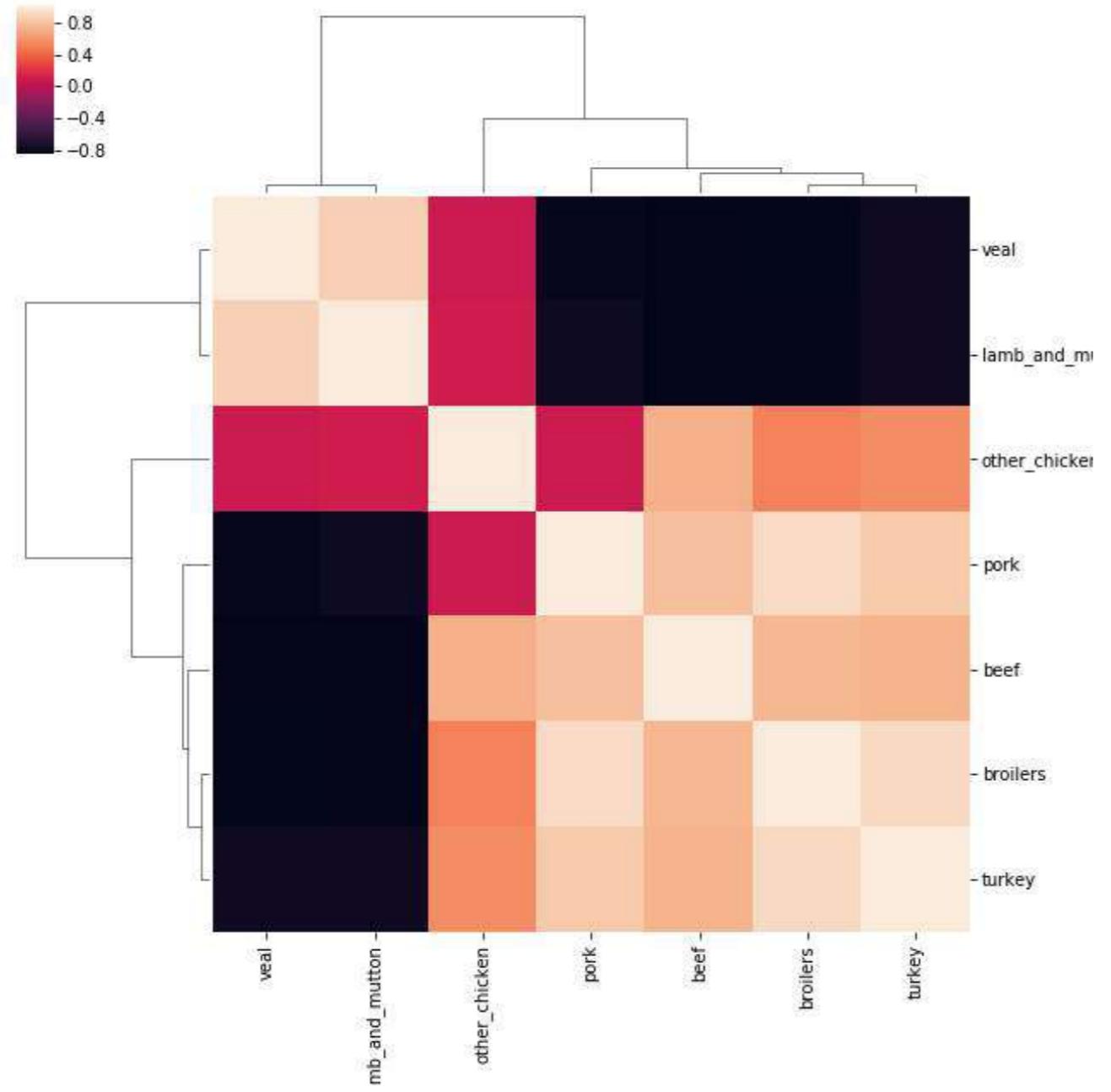
```
import seaborn as sns  
sns.heatmap(corr_mat)
```

Heatmap



Clustermap

```
sns.clustermap(corr_mat)
```



Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Apply your knowledge to a new dataset

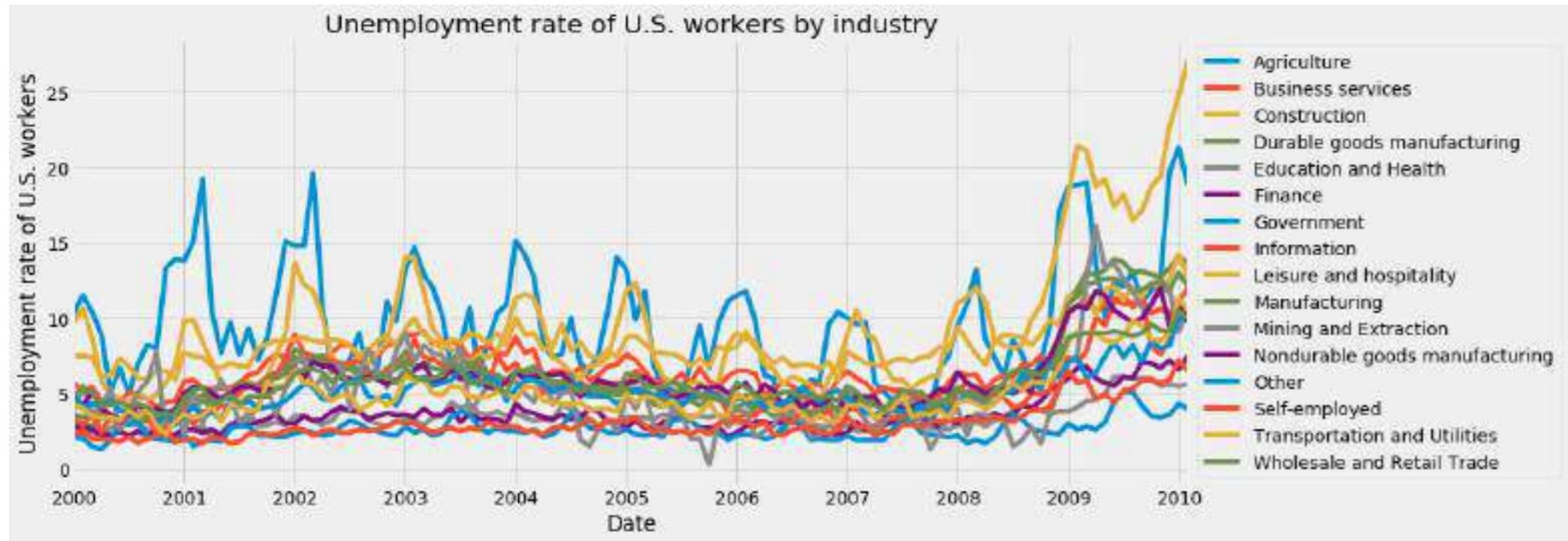
VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

Head of Data Science, Getty Images

The Jobs dataset

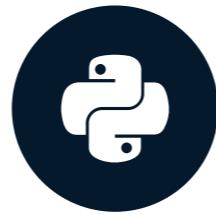


Let's get started!

VISUALIZING TIME SERIES DATA IN PYTHON

Beyond summary statistics

VISUALIZING TIME SERIES DATA IN PYTHON

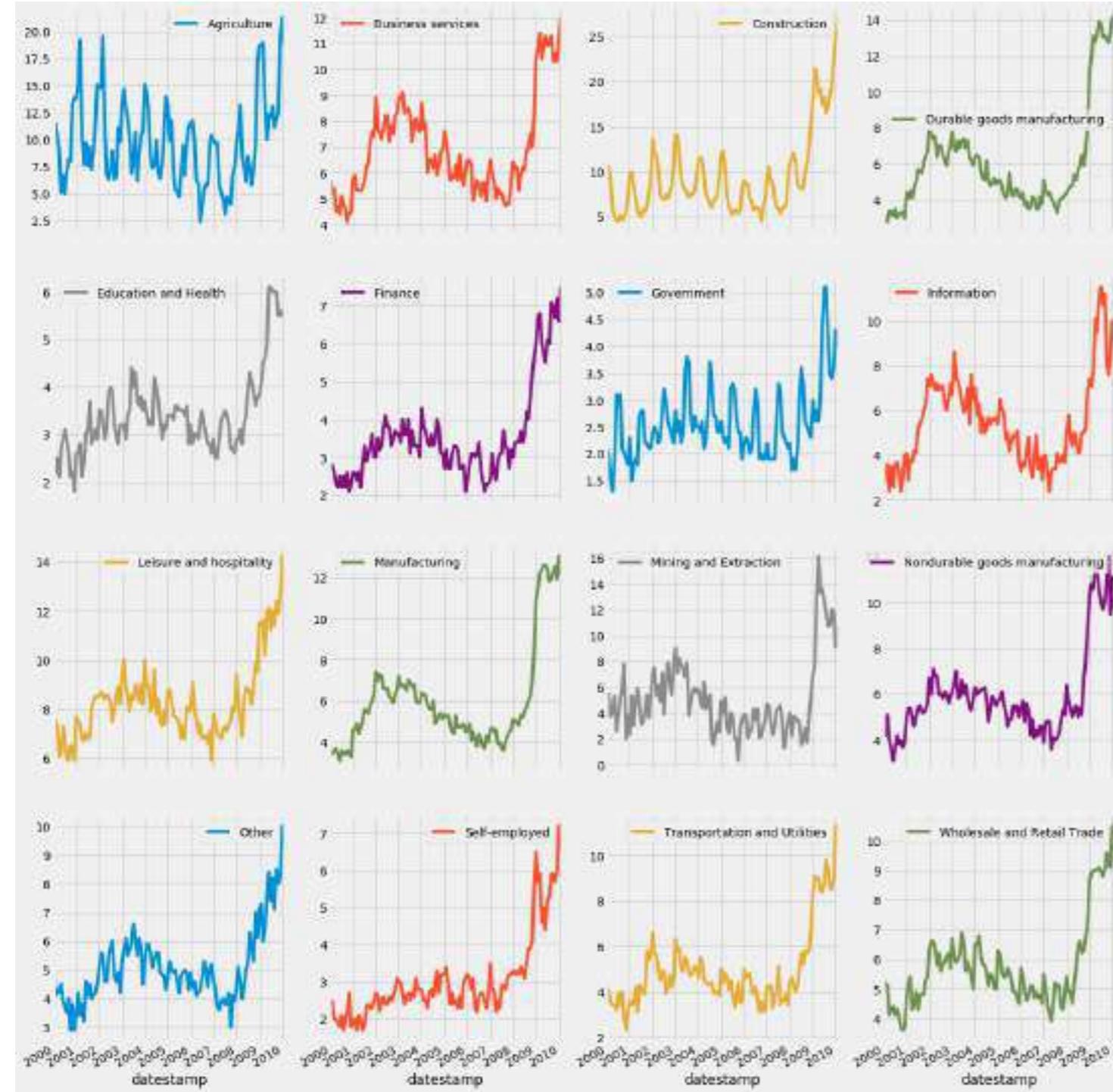


Thomas Vincent

Head of Data Science, Getty Images

Facet plots of the jobs dataset

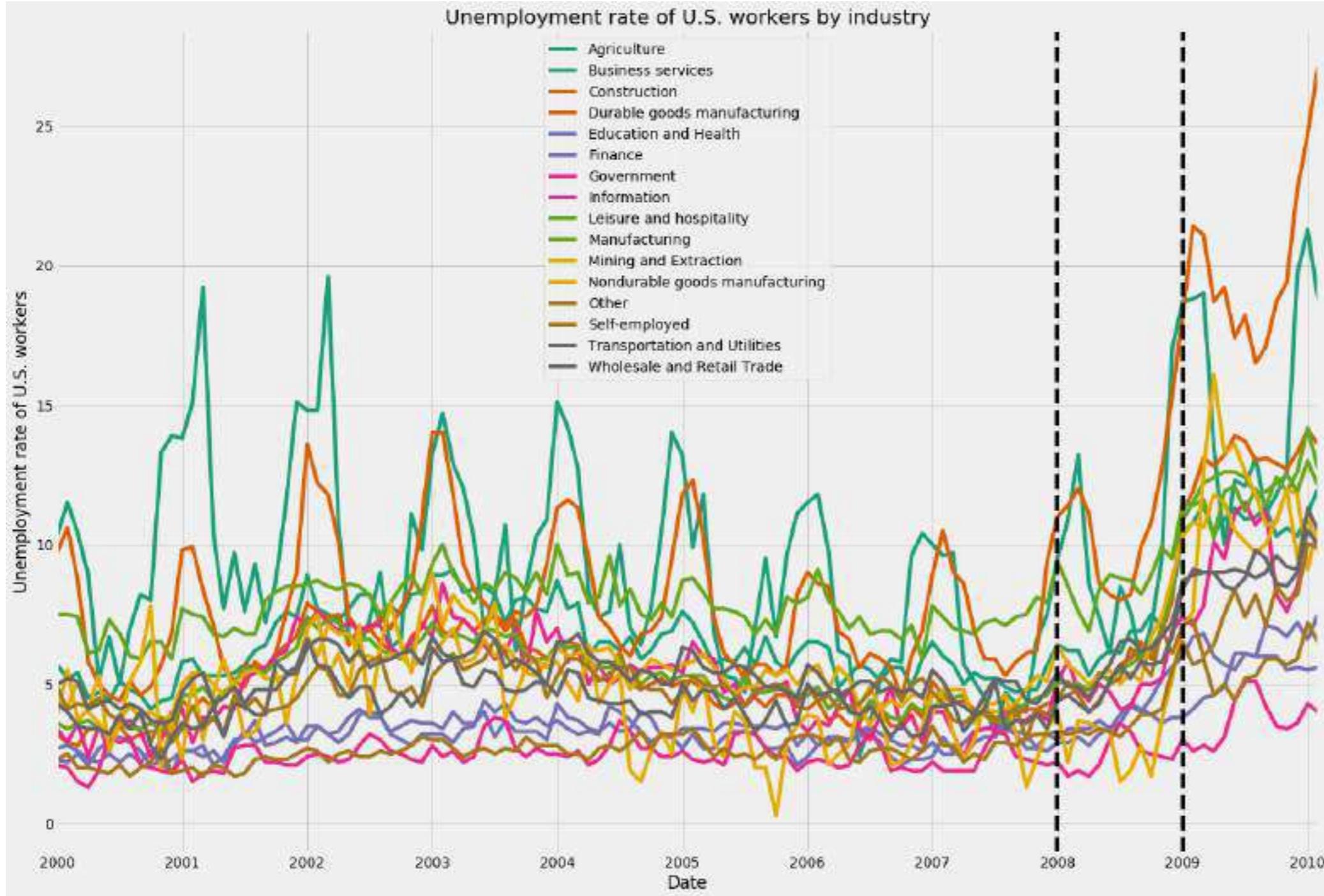
```
jobs.plot(subplots=True,  
          layout=(4, 4),  
          figsize=(20, 16),  
          sharex=True,  
          sharey=False)  
  
plt.show()
```



Annotating events in the jobs dataset

```
ax = jobs.plot(figsize=(20, 14), colormap='Dark2')
ax.axvline('2008-01-01', color='black',
           linestyle='--')
ax.axvline('2009-01-01', color='black',
           linestyle='--')
```

Unemployment rate of U.S. workers by industry



Taking seasonal average in the jobs dataset

```
print(jobs.index)
```

```
DatetimeIndex(['2000-01-01', '2000-02-01', '2000-03-01',
'2000-04-01', '2009-09-01', '2009-10-01',
'2009-11-01', '2009-12-01', '2010-01-01', '2010-02-01'],
dtype='datetime64[ns]', name='datestamp',
length=122, freq=None)
```

```
index_month = jobs.index.month
jobs_by_month = jobs.groupby(index_month).mean()
print(jobs_by_month)
```

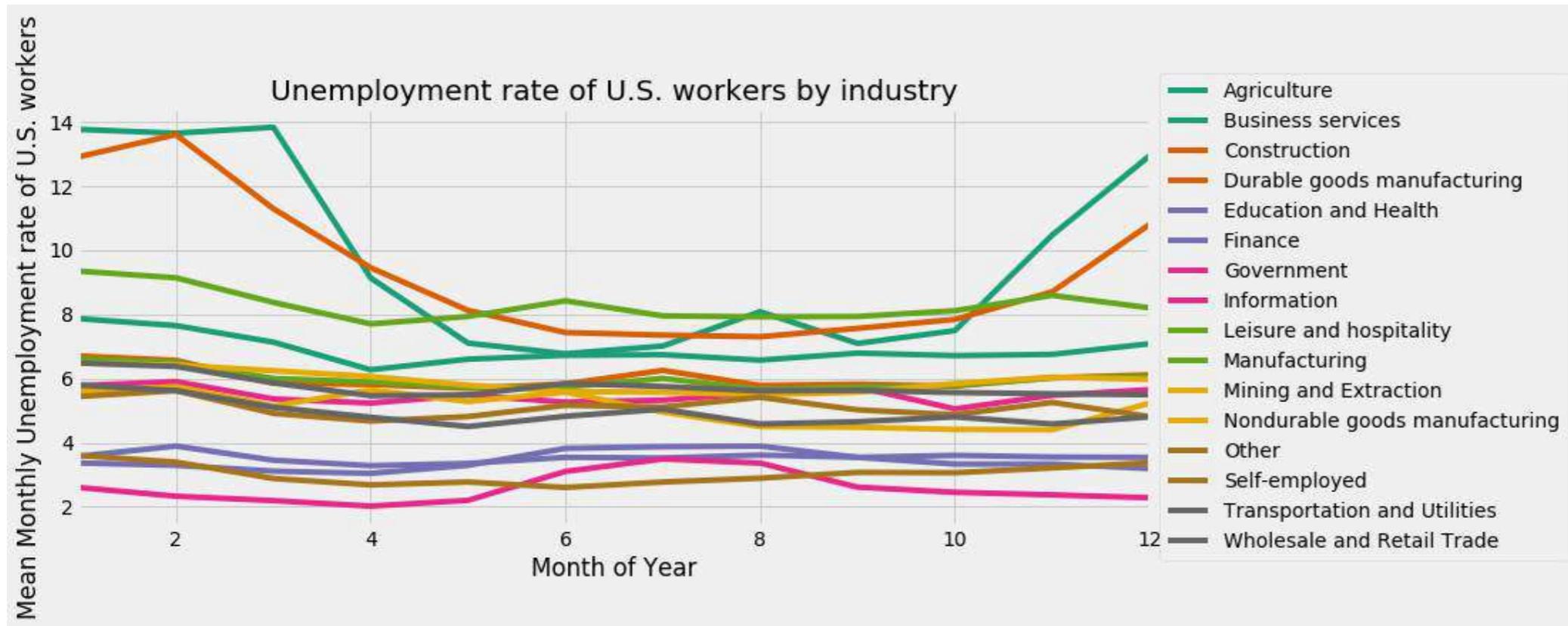
datestamp	Agriculture	Business services	Construction
1	13.763636	7.863636	12.909091
2	13.645455	7.645455	13.600000
3	13.830000	7.130000	11.290000
4	9.130000	6.270000	9.450000
5	7.100000	6.600000	8.120000
...			

Monthly averages in the jobs dataset

```
ax = jobs_by_month.plot(figsize=(12, 5),  
colormap='Dark2')
```

```
ax.legend(bbox_to_anchor=(1.0, 0.5),  
loc='center left')
```

Monthly averages in the jobs dataset

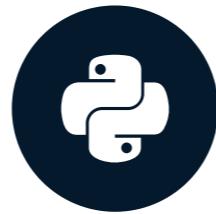


Time to practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Decompose time series data

VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

Head of Data Science, Getty Images

Python dictionaries

```
# Initialize a Python dictionnary
my_dict = {}

# Add a key and value to your dictionnary
my_dict['your_key'] = 'your_value'

# Add a second key and value to your dictionnary
my_dict['your_second_key'] = 'your_second_value'

# Print out your dictionnary
print(my_dict)
```

```
{'your_key': 'your_value',
 'your_second_key': 'your_second_value'}
```

Decomposing multiple time series with Python dictionaries

```
# Import the statsmodel library
import statsmodels.api as sm
# Initialize a dictionary
my_dict = {}
# Extract the names of the time series
ts_names = df.columns
print(ts_names)
```

```
['ts1', 'ts2', 'ts3']
```

```
# Run time series decomposition
for ts in ts_names:
    ts_decomposition = sm.tsa.seasonal_decompose(jobs[ts])
    my_dict[ts] = ts_decomposition
```

Extract decomposition components of multiple time series

```
# Initialize a new dictionary
my_dict_trend = {}
# Extract the trend component
for ts in ts_names:
    my_dict_trend[ts] = my_dict[ts].trend
# Convert to a DataFrame
trend_df = pd.DataFrame.from_dict(my_dict_trend)
print(trend_df)
```

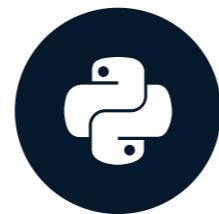
```
      ts1  ts2  ts3
datestamp
2000-01-01  2.2  1.3  3.6
2000-02-01  3.4  2.1  4.7
...
...
```

Python dictionaries for the win!

VISUALIZING TIME SERIES DATA IN PYTHON

Compute correlations between time series

VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

Head of Data Science, Getty Images

Trends in Jobs data

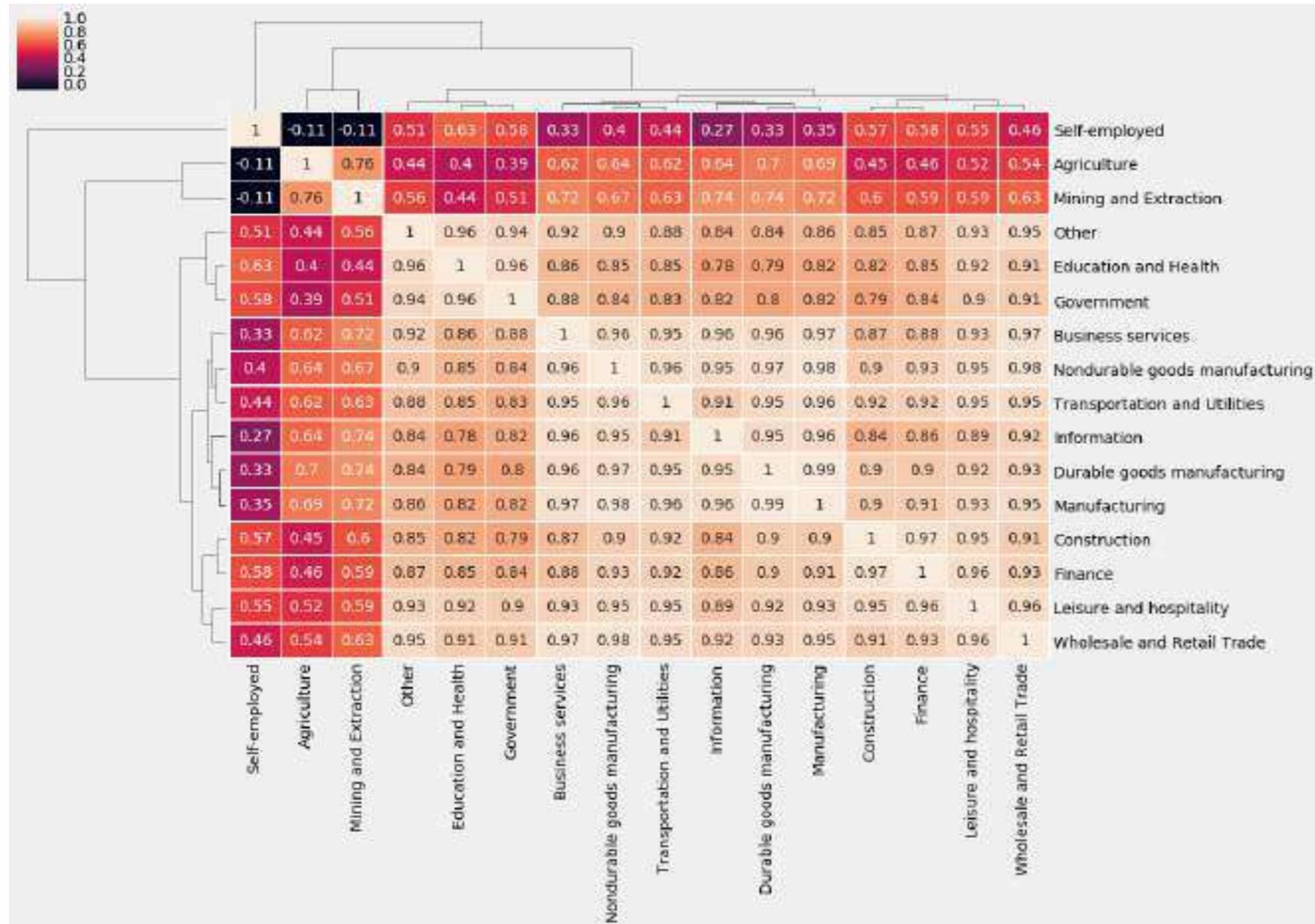
```
print(trend_df)
```

datestamp	Agriculture	Business services	Construction
2000-01-01	NaN	NaN	NaN
2000-02-01	NaN	NaN	NaN
2000-03-01	NaN	NaN	NaN
2000-04-01	NaN	NaN	NaN
2000-05-01	NaN	NaN	NaN
2000-06-01	NaN	NaN	NaN
2000-07-01	9.170833	4.787500	6.329167
2000-08-01	9.466667	4.820833	6.304167
...			

Plotting a clustermap of the jobs correlation matrix

```
# Get correlation matrix of the seasonality_df DataFrame  
trend_corr = trend_df.corr(method='spearman')  
  
# Customize the clustermap of the seasonality_corr  
correlation matrix  
fig = sns.clustermap(trend_corr, annot=True, linewidth=0.4)  
  
plt.setp(fig.ax_heatmap.yaxis.get_majorticklabels(),  
rotation=0)  
  
plt.setp(fig.ax_heatmap.xaxis.get_majorticklabels(),  
rotation=90)
```

The jobs correlation matrix

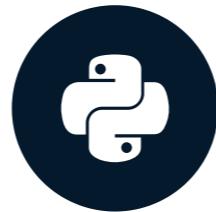


Let's practice!

VISUALIZING TIME SERIES DATA IN PYTHON

Congratulations!

VISUALIZING TIME SERIES DATA IN PYTHON



Thomas Vincent

Head of Data Science, Getty Images

Going further with time series

- Data from Zillow Research
- Kaggle competitions
- Reddit Data

Going further with time series

- The importance of time series in business:
 - to identify seasonal patterns and trends
 - to study past behaviors
 - to produce robust forecasts
 - to evaluate and compare company achievements

Getting to the next level

- Manipulating Time Series Data in Python
- Importing & Managing Financial Data in Python
- Statistical Thinking in Python (Part 1)
- Supervised Learning with scikit-learn

Thank you!

VISUALIZING TIME SERIES DATA IN PYTHON