- **JupyterLab Desktop**
- **JupyterLab**
- **VS Code, Positron, etc.**

Please type and execute this code:

```
import pandas as pd
print(pd.__version__)
```

'2.3.3'

← you should get something like this

Raise your hand if you cannot import Pandas **or if you get a version < 2.0**

Roll Call

- Your name
- Role or class year
- Department or major
- A source or form of data you regularly use or want to learn to use

ICPSR

United States® Census Bureau

eurostat

DRYAD

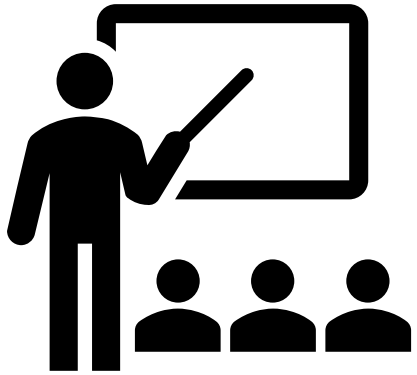
NIH National Institutes of Health

Meet Your Instructor

- Erin M. Winter, she/her
- Data Management Librarian
- MS Library Science 2023, MS Computer Science 2017
- Supports Python, HPC, GitHub, and data management at Data Services

uolib.link/data

Workshop Policies: Attendance

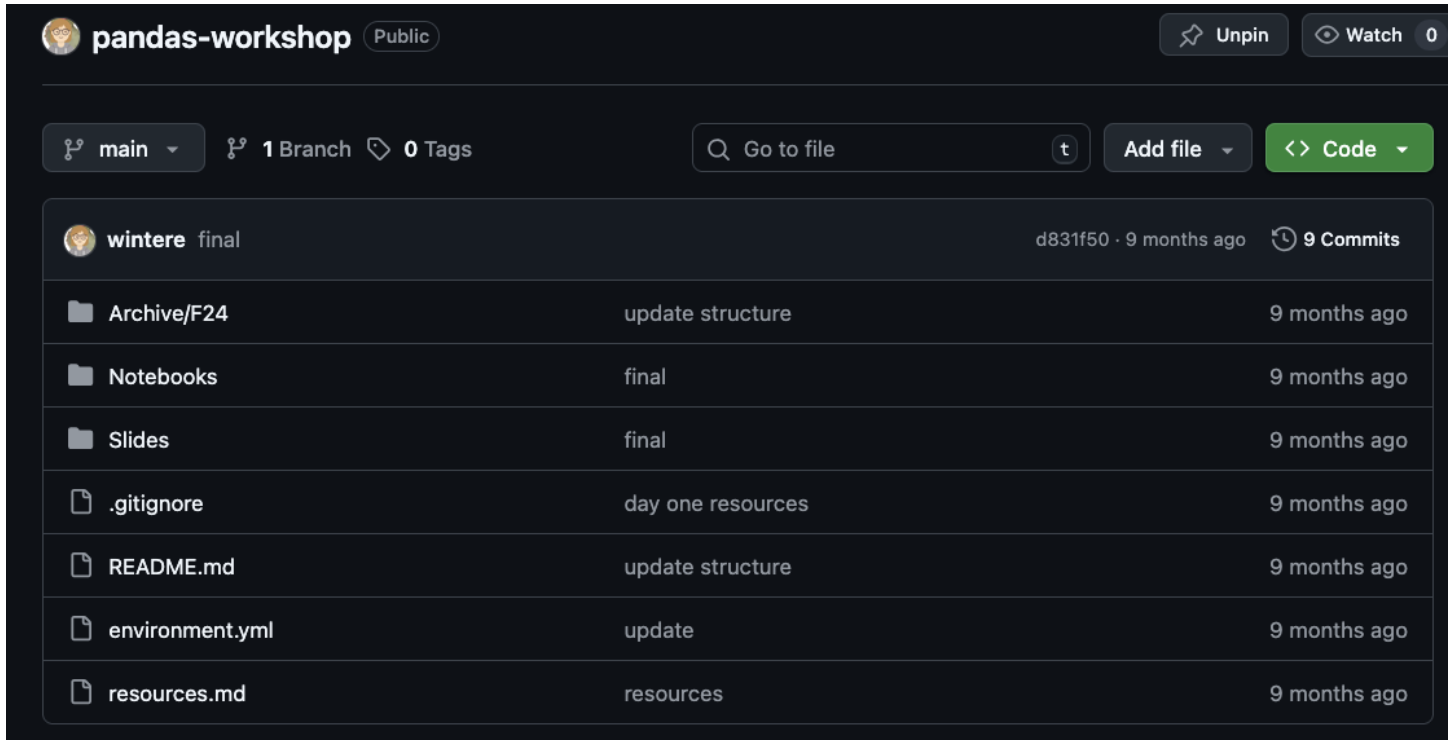


Come to Knight 267B to attend in person.

OR

Link automatically sent to the *email you registered with 3* hours before the workshop

Lecture Notes and Slides on GitHub



uolib.link/pandas-notes

GitHub Page

Notebooks

Slides

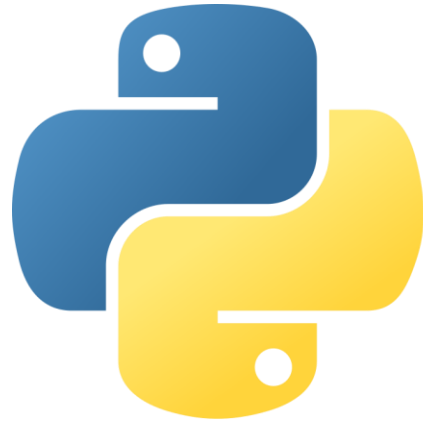
Updated a few days after the workshop

Prerequisites

- Software Carpentries: Plotting and Programming in Python

OR

  - **JupyterLab interface**
  - **variables, assignment**
  - standard Python primitives
    - int, float, string, boolean
  - control flow: if and else
  - *functions, scope*



You will get opportunities to practice these if you're still grasping them.

Workshop Objectives

Goals

- Manipulate, analyze, and visualize data in Pandas and related libraries
- Import data from a variety of sources
- Identify and correct common problems in tabular data
- Learn basic data visualization

Not Covered

- Topics in “big data”
- Math
- Non-tabular data: ie. genetic sequences, geodatabases, images

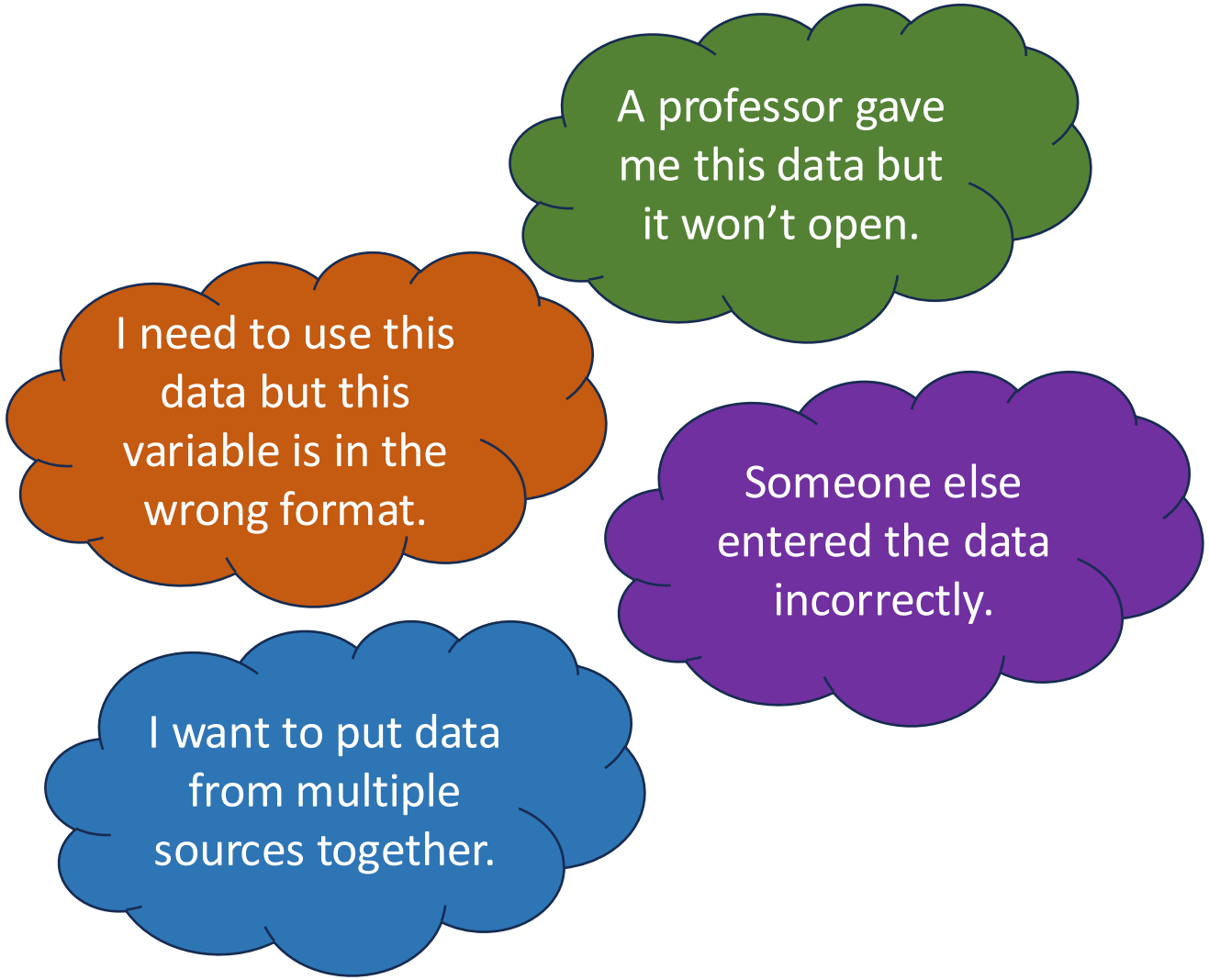
Why Pandas?

- Free, easy to learn
- Compatible with a growing data science ecosystem in Python
  - Visualization
  - Machine learning
  - Statistics
  - Scientific applications
- Flexible in terms of input and output

Data Wrangling

From Raw Data to Analysis

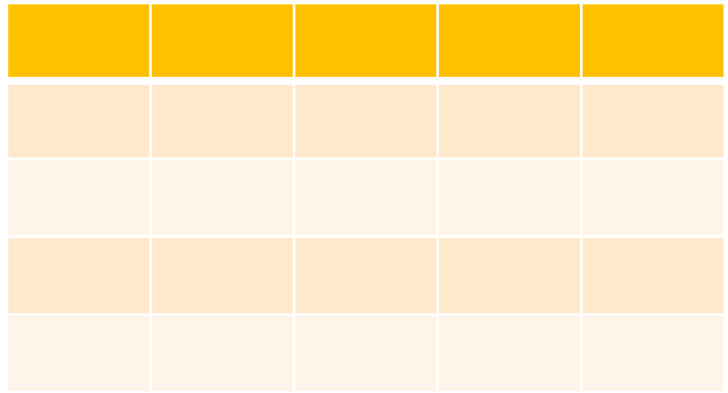
- Handle missing values
- Convert measurements, apply functions, and correct mistakes at scale
- Prepare data to be used for other applications
- Create reusable code so you don’t have to do it from scratch again



Creating DataFrame Objects

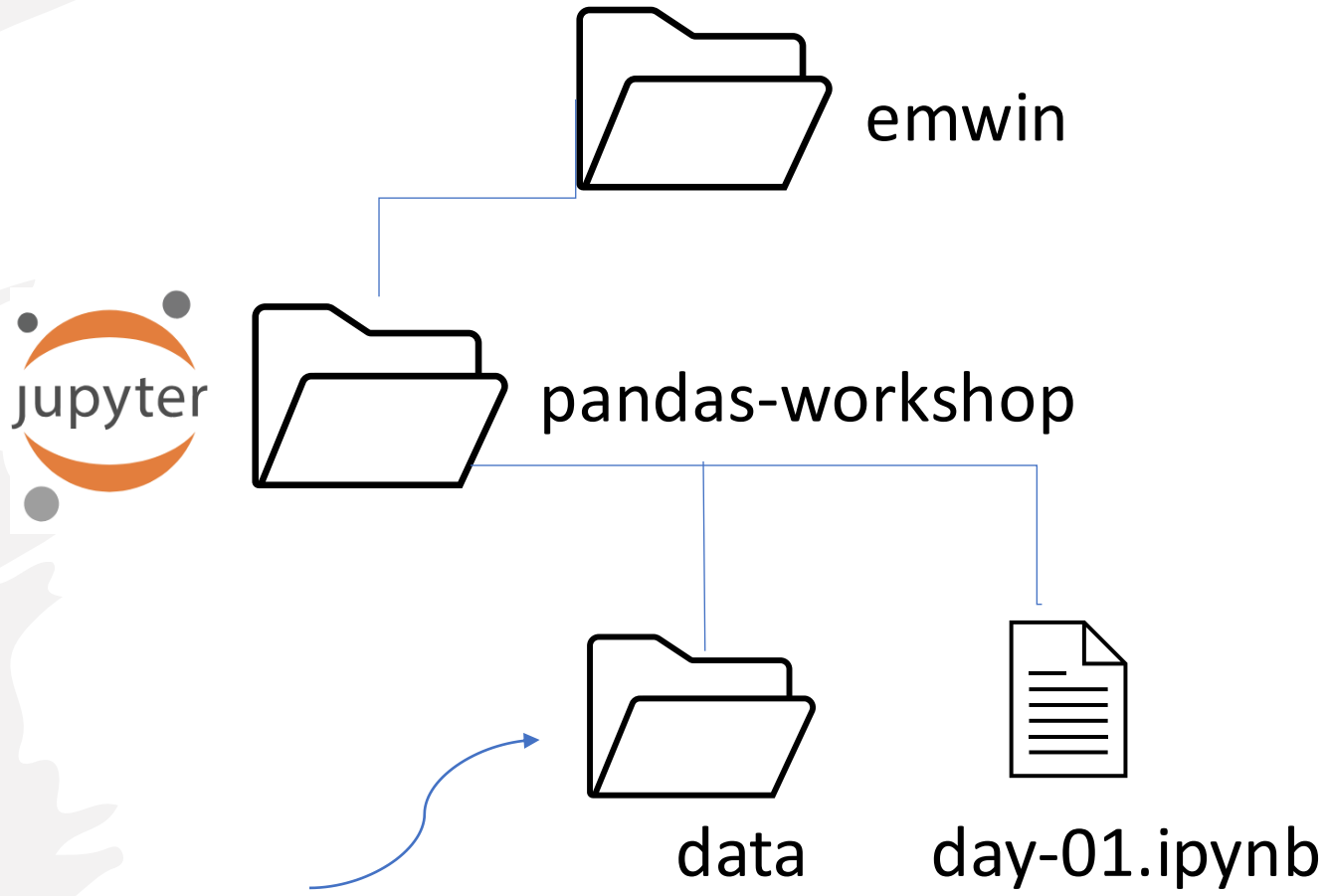
- **Can be constructed directly from Python objects like dictionaries**

- **Loaded from a file or URL:**
  - delimited text file (.txt, .tsv)
  - comma-separated values (.csv)
  - Excel sheet (.xlsx)
  - JSON dictionary (.json)
  - Stata/SAS files (.sav, .sas, .dta)



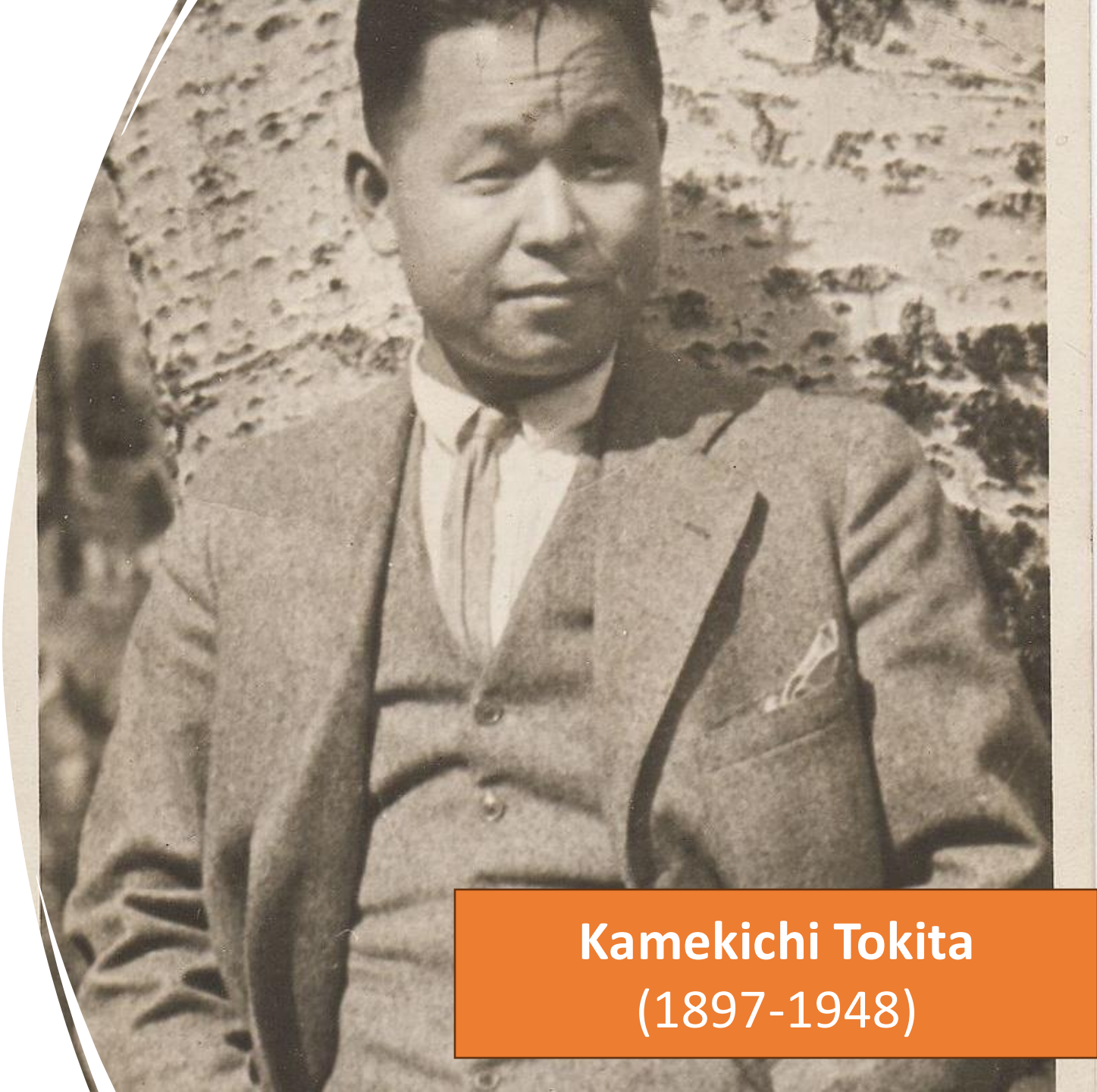
Getting Started

- Create a folder for your project named pandas-workshop
- Open the folder in JupyterLab
- Create a subfolder for your data called ***data***

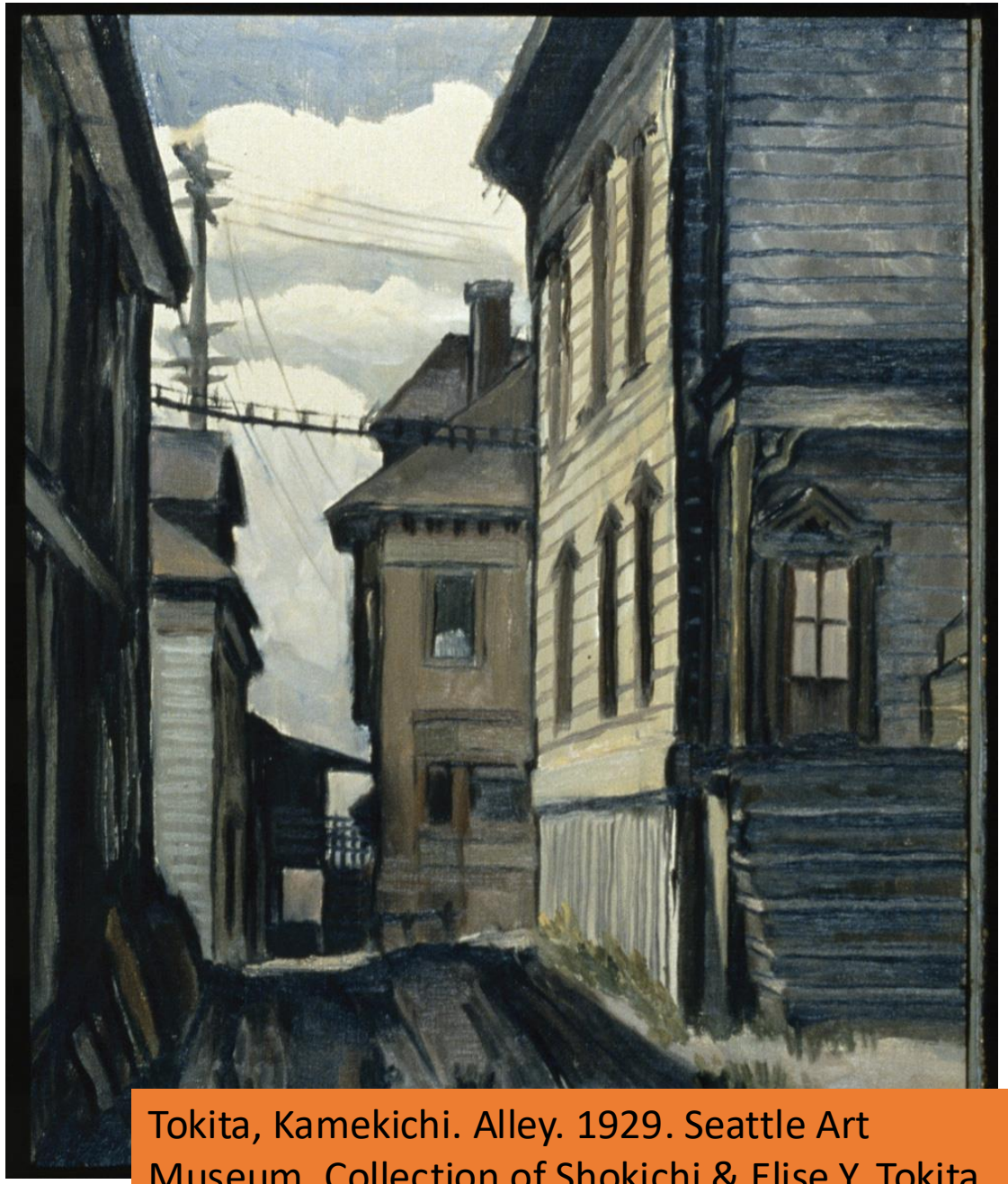


A Little Modern Art

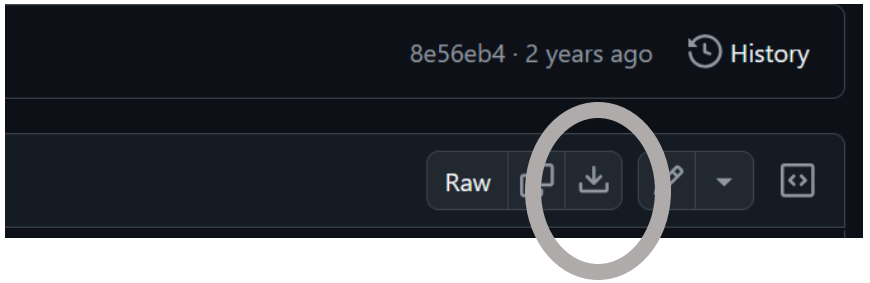
- Our first dataset is the MOMA's Watson Library Index of Asian American and Pacific Islander Artists.
- This is a collection of AAPI artists featured in the Watson Library's catalog of exhibit records and artist biographies.



Kamekichi Tokita
(1897-1948)



Tokita, Kamekichi. Alley. 1929. Seattle Art Museum. Collection of Shokichi & Elise Y. Tokita.



Loading Data From A File

- Find the download link in the *resources.md* in the class GitHub page
- Download this file from the Watson Library GitHub by clicking the download button
- Copy this *.csv* file into your JupyterLab project directorys

Reading Input

Reading files into Pandas is as simple as **matching the file type** to **the name of a function…**

- pd.read_csv()
- pd.read_json()
- pd.read_excel()
- pd.read_sql()
- pd.read_stata()
- …

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	NA
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	LaTeX	Styler.to_latex	NA
text	XML	read_xml	to_xml
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	NA
binary	HDF5 Format	read_hdf	to_hdf

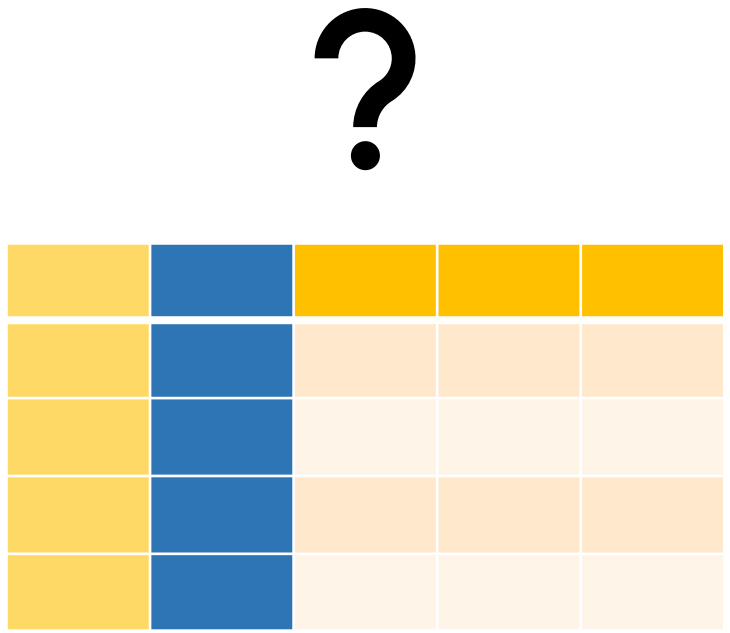
https://pandas.pydata.org/docs/dev/user_guide/io.html

Understanding Your Data (I/O)

After reading from a file to a DataFrame, check

- delimiting (are the columns separated?)
- the shape – a (row #, column #) tuple
- the types of each variable – dtypes
- which columns have nulls/nones/NaNs (and **why**)

If it is ***not*** what you expect, you probably need to adjust the formatting arguments passed to Pandas.



What Can Go in a DataFrame?

name	colour	location	seed	shape	sweetness	water_content
apple	red	canada	TRUE	round	TRUE	84

int64, float64

Integers, Floats

bool

Booleans

object, Categorical

Strings

datetime64

Dates

Columns with values of a type that Pandas cannot infer will be labeled as objects.

The Pandas DataFrame

Each **DataFrame** column is a **Series**.

.columns

axis = 1 →

.index

	name	colour	location	seed	shape	sweetness	water_content	weight
0	apple	red	canada	TRUE	round	TRUE	84	100
1	banana	yellow	mexico	FALSE	long	TRUE	75	120
2	cantaloupe	orange	spain	TRUE	round	TRUE	90	1360
3	dragon fruit	magenta	china	TRUE	round	FALSE	96	600
4	elderberry	purple	austria	FALSE	round	TRUE	80	5

axis = 0 ↓

All values in the DataFrame have an indexed position in .iloc[row_index, column_number] form.

Total elements = .size
R, C form = .shape

The Pandas Series

The **Series** is the simplest of the Pandas data structures: a one-dimensional array with an index. The alpha-numeric index is exactly as long as the data.

.index	0	1	2	3	4	5	6	7
	burgundy	red	green	gray	blue	yellow	orange	teal

All values in the Series have an indexed position in [index] or .iloc[index_num] form.

.index	a	b	c	d	e	f	g	h
	burgundy	red	green	gray	blue	yellow	orange	teal

Slicing and Indexing with .loc

Column Labels

Index	name	colour	location
a	apple	red	canada
b	banana	yellow	mexico
c	cantaloupe	orange	spain
d	dragon fruit	magenta	china
e	elderberry	purple	austria

.loc slicing is inclusive on both ends because it is intended to behave like R (not Python)

.loc – index and column based slicing

.loc['a'] returns a **pandas.Series** with the first row

.loc['a', 'location'] returns the string "canada"

.loc['b':'d', 'name': 'colour'] returns the DataFrame below

	name	colour
b	banana	yellow
c	cantaloupe	orange
d	dragon fruit	magenta

Slicing and Indexing with .iloc

Column Labels

Index	name	colour	location
a	apple	red	canada
b	banana	yellow	mexico
c	cantaloupe	orange	spain
d	dragon fruit	magenta	china
e	elderberry	purple	austria

.iloc slices behave like Python list slices (exclusive at the end of a range)

.iloc – integer-based slicing

.iloc[0] returns a **pandas.Series** with the first row

.iloc[1, 2] returns the string "mexico"

.iloc[2:, 1:3] returns the DataFrame below

	colour	location
c	orange	spain
c	magenta	china
d	purple	austria

Boolean Indexing

- Pandas .loc and .iloc can return subsets of a DataFrame specified by a boolean array.
  - Boolean arrays filter DataFrames or Series by testing values against a condition. *NaNs evaluate to False.*

```
under21 = students['age'] < 21
```

a series of size (# of rows in students)

a pd.DataFrame

numeric column

the "test", must evaluate to True or False

```
studentsUnder21 = students[under21]
```

only the rows in the students DataFrame where the column'age' < 21

x['col_name'] is shorthand for x.loc[:, 'col_name']

Boolean Indexing with Multiple Conditions

- We can combine boolean indexing with multiple conditions as follows:

```
ok_to_drink = (students['age'] >= 21) & (
students['has_id'] == 'Yes')
```

a pd.DataFrame — column — test — boolean operator — test

a series of size (# of rows in students)

```
studentDrinkers = students[ok_to_drink]
```

a DataFrame with only the rows in the students DataFrame where the column 'age' > 21 AND column 'has_id' has a 'Yes' value

What is a NaN? (null, None, etc.)

The np.NaN is the way Pandas represents **missing values** by default, but missing values occur in almost **all** domains:

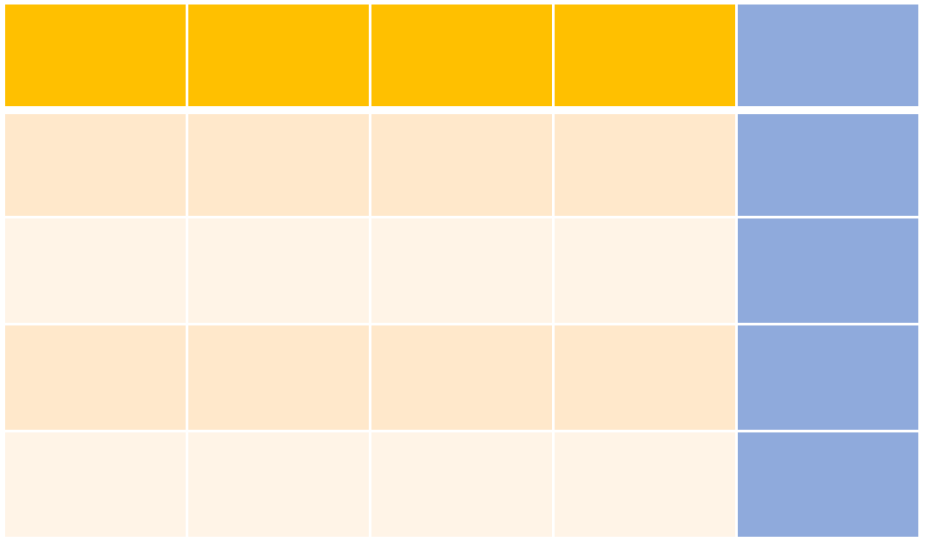
- Product of the data entry process
- Missing from the source data
- Measurement error
- Participant declined to respond
- Something went wrong computationally when creating the data

Why Dtypes Matter

continuous	discrete	discrete	ordinal/continuous
Integers, Floats	Booleans	Strings	Dates
sort/compare **arithmetic** **percentiles** filter count	**filter** count sort/compare	**count** **compare** alphabetize filter	**sort/compare** **add/subtract** filter count

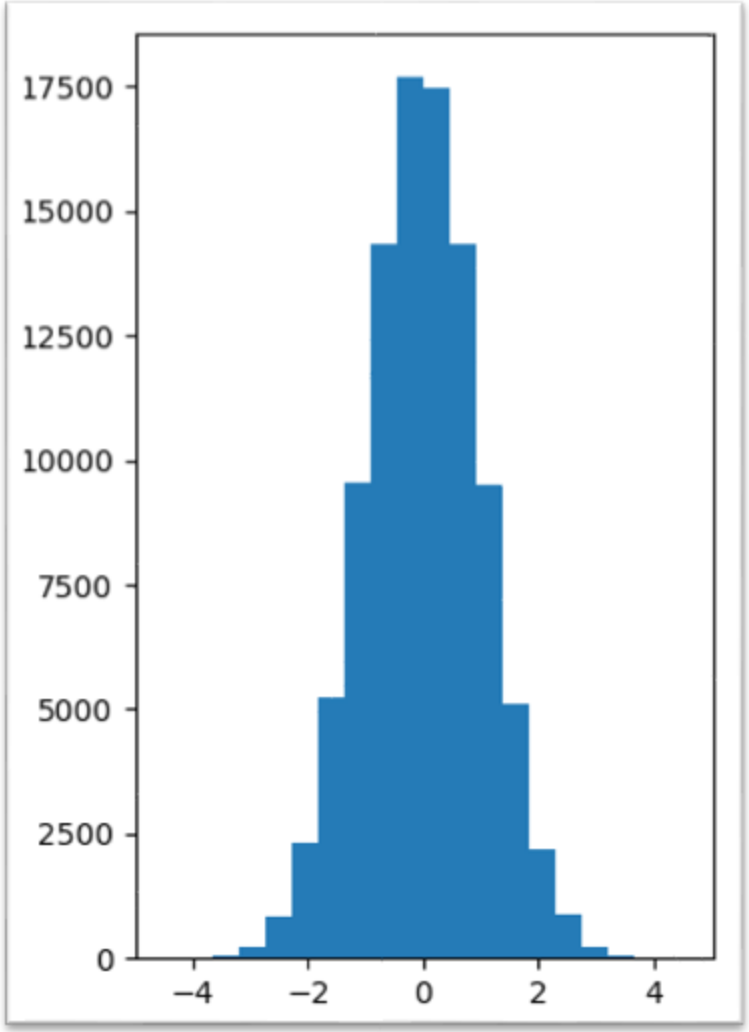
Inferring New Columns

- Infer a new column from an existing column or columns
  - Map()
  - Vectorized operations
    - F[mpg] = F[miles_per_tank]/F[gallons_per_tank]



The Histogram

Every bin of frequencies has an associated count

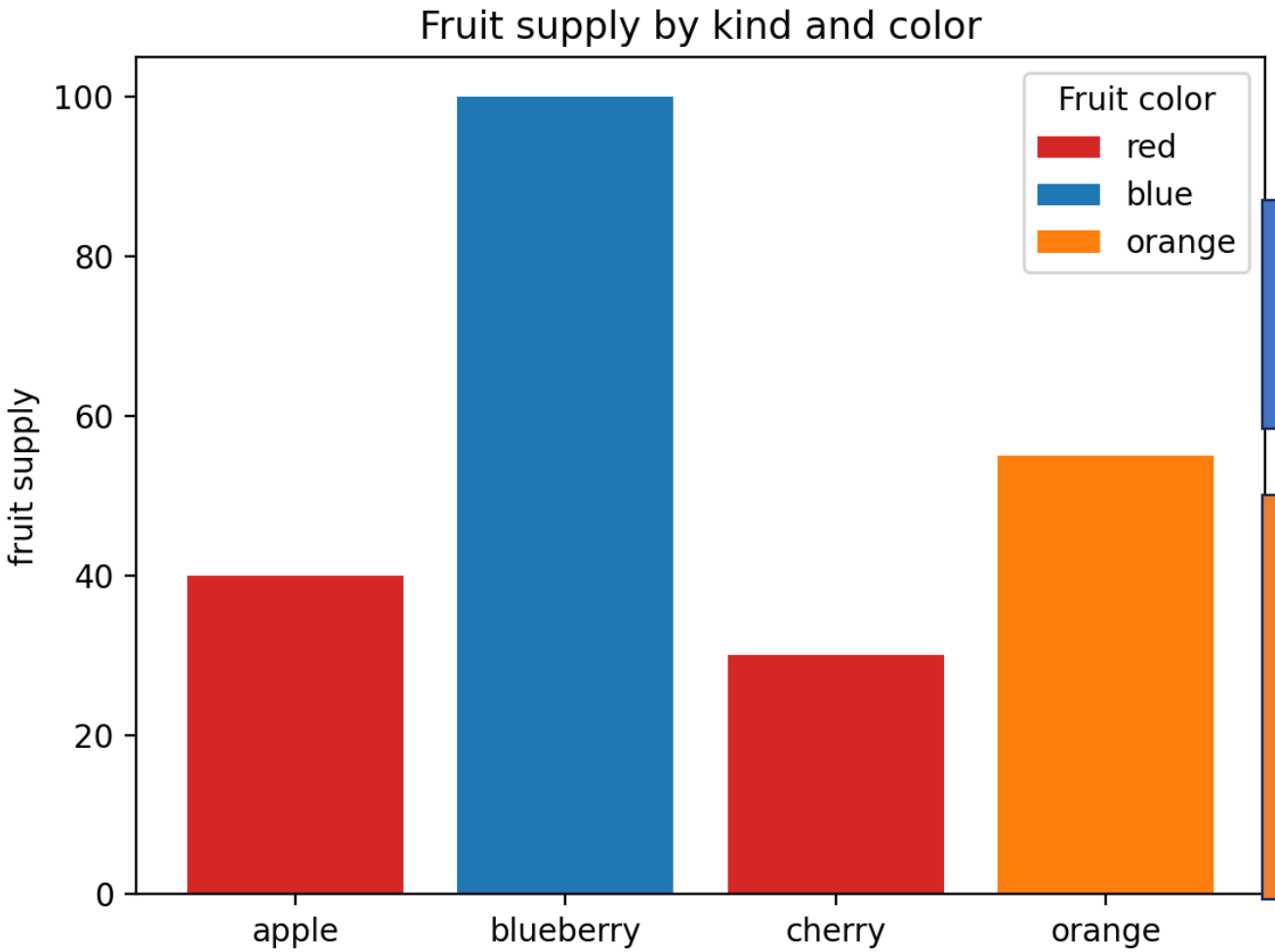


X, Y - one continuous variable
Y = frequency of the values within each bin

Gives you a sense of the shape of the distribution, very sensitive to the number of bins.

The Bar Chart

Every bar has a category X and a value Y

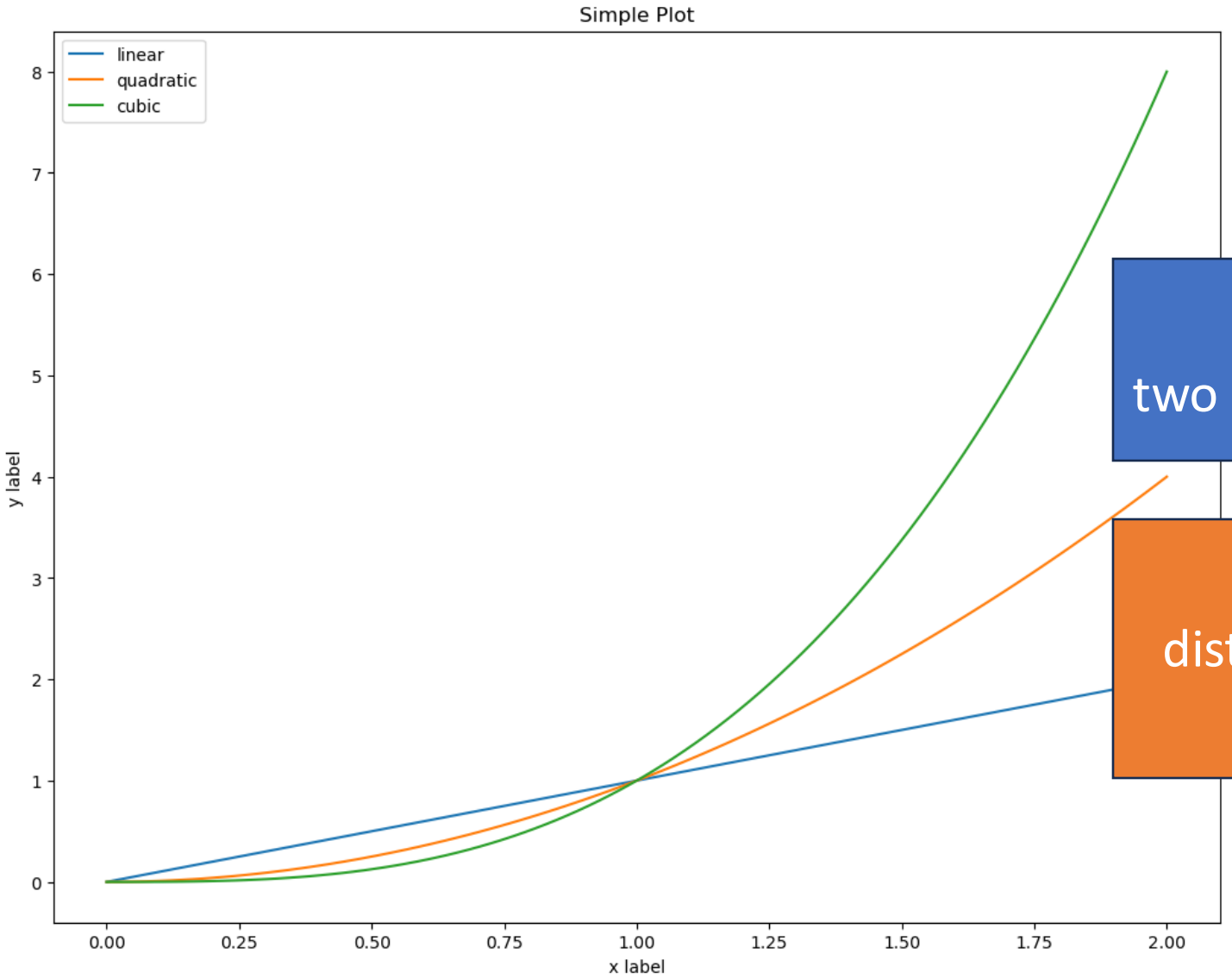


X, Y
X = categorical
Y = numeric

Color
distinguishes between bars, can also be another categorical variable
Z = fruit color

The Line Graph

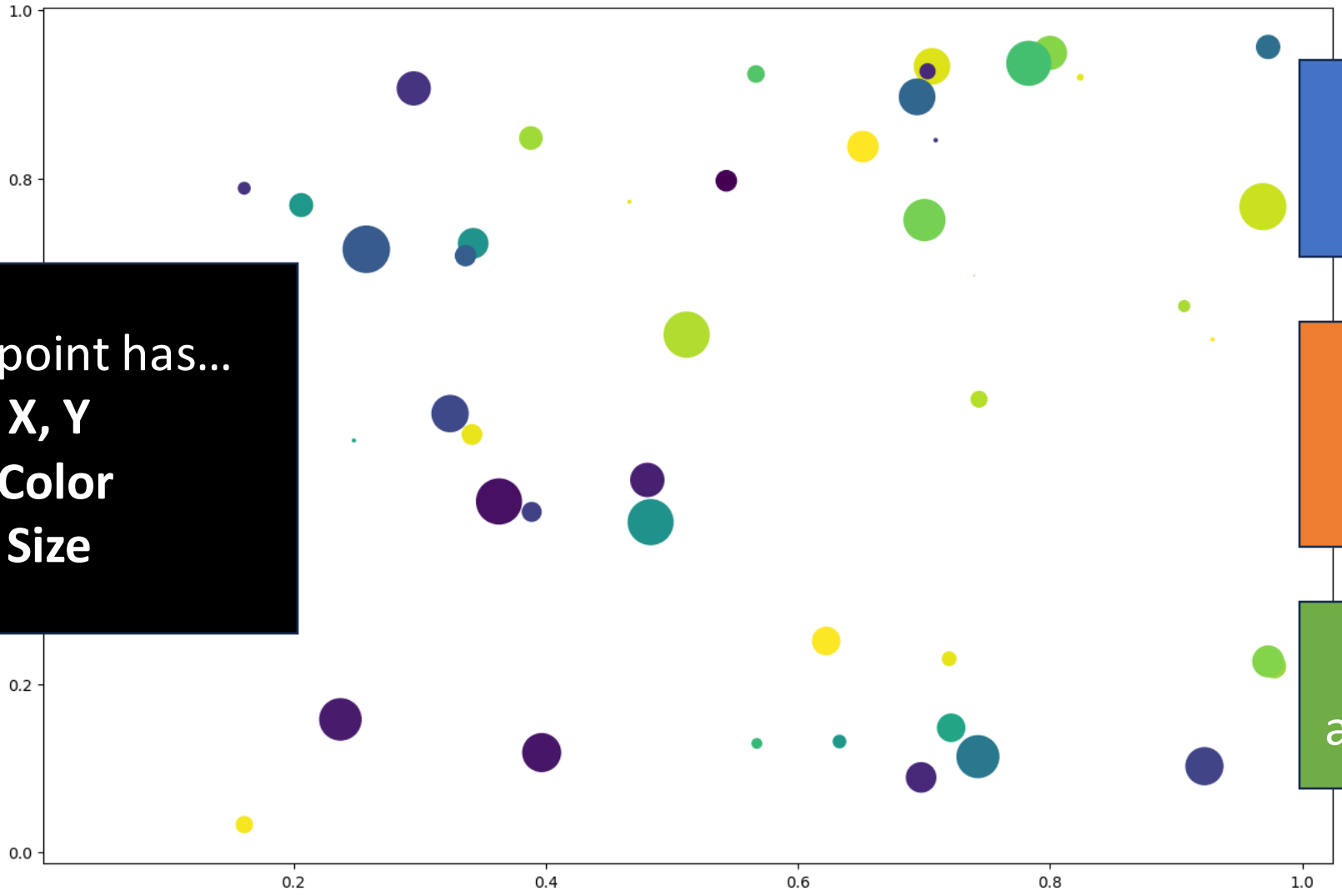
Every line has…
X, Y coordinates



X, Y
two continuous variables

Color
distinguishes between multiple lines

The Scatter Plot



Every point has…
X, Y
Color
Size

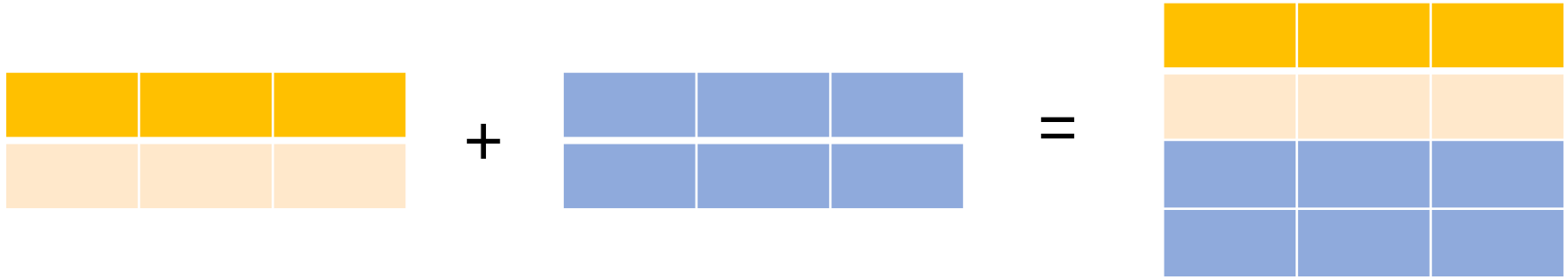
X, Y
two continuous variables

Color
continuous: shades
discrete: hues

Size
a continuous variable

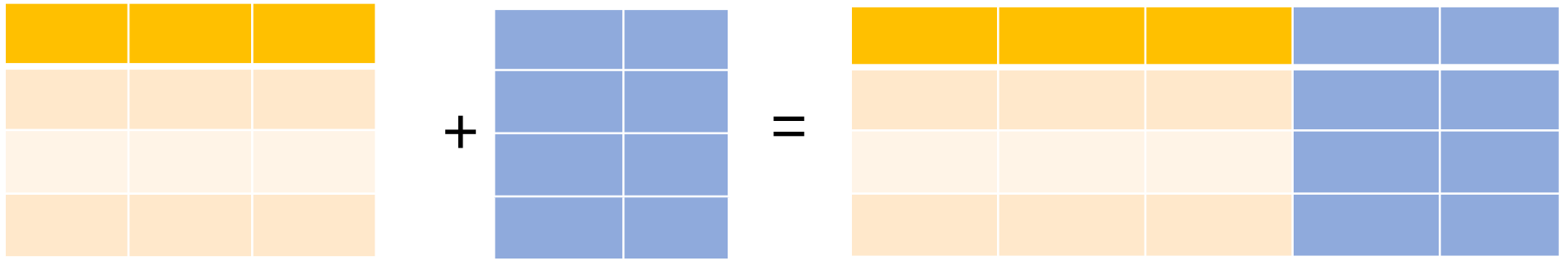
Combining Data Sources

- The simplest (and most naïve approach) to combining DataFrames is to **concatenate them**



Pandas.concat

Axis =index



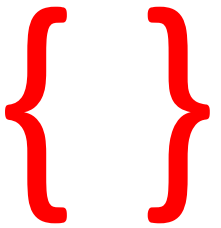
Axis = columns

JSON

A structured text-formatting standard compatible with Python, R, JavaScript etc.

Python has a robust built-in JSON parser

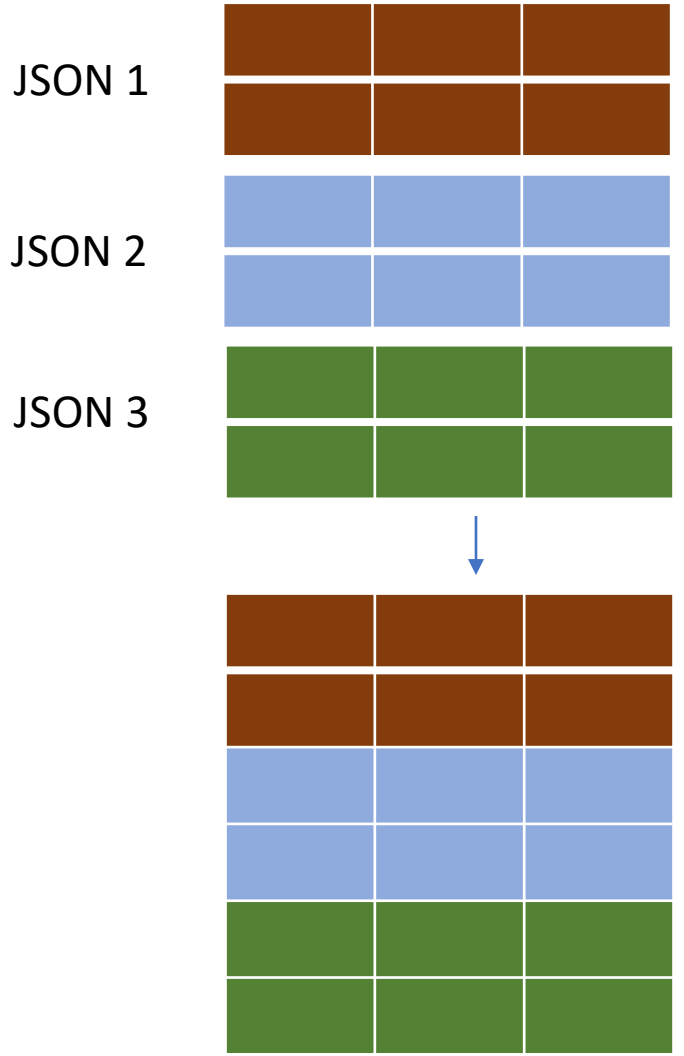
JSONs can be treated as dictionaries or lists of dictionaries



From JSON to Pandas DataFrame

A JSON file can be treated as a **list** of **dictionaries** with a shared schema

For simple JSON files, Pandas can directly read JSON files from URLs or from disk into DataFrames



Merge (Join)

Allows you to join two or more DataFrames using shared values in a shared **key** column to align the rows in the DataFrames

Akin to a SQL join, R's merge, or an Excel VLOOKUP

Thank you for attending!