- **JupyterLab**
- **JupyterNotebook**
  - or an IDE of your choice…

Please type and execute this code:

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

```
'2.2.3'
```

← 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 tabular data you regularly use or want to learn to use

ICPSR

United States® Census Bureau

eurostat

DRYAD

NIH

National Institutes of Health

Workshop Policies

Lecture Notes
(*.ipynb*)

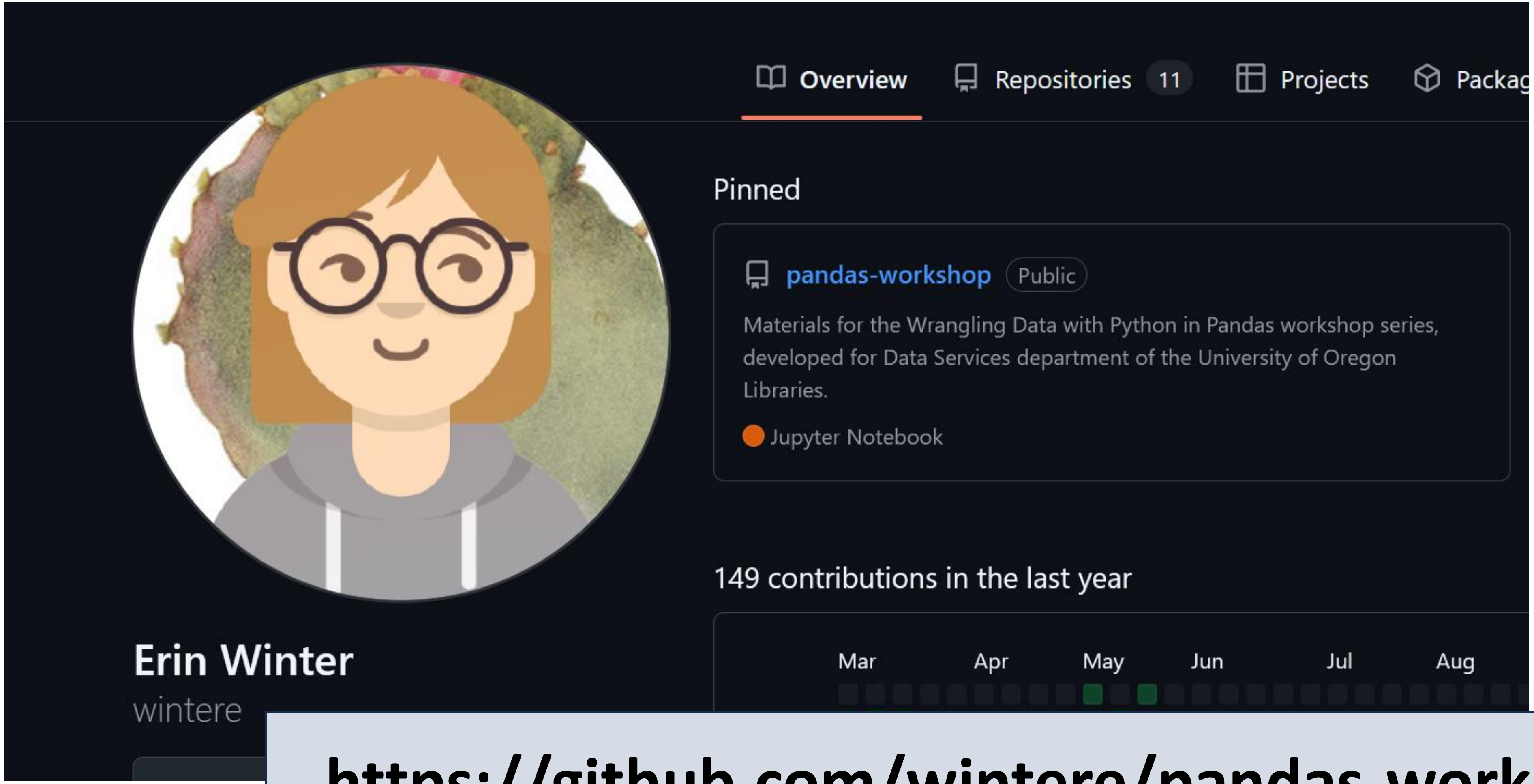
Slides

Updated a few days after the workshop

zoom

Link automatically sent to the *email you registered with* 24 hours before the workshop

Lecture Notes and Slides on GitHub



Pinned

pandas-workshop Public

Materials for the Wrangling Data with Python in Pandas workshop series, developed for Data Services department of the University of Oregon Libraries.

Jupyter Notebook

149 contributions in the last year

Erin Winter

wintere

https://github.com/wintere/pandas-workshop

Prerequisites

- Software Carpentries: Plotting and Programming in Python

OR

- **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 techniques

Not Covered

- Topics in “big data” Optimization
- Math
- Data in formats that are not tabular: 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

bokeh

jupyter

stumpy

pandas

NumPy

scikit learn

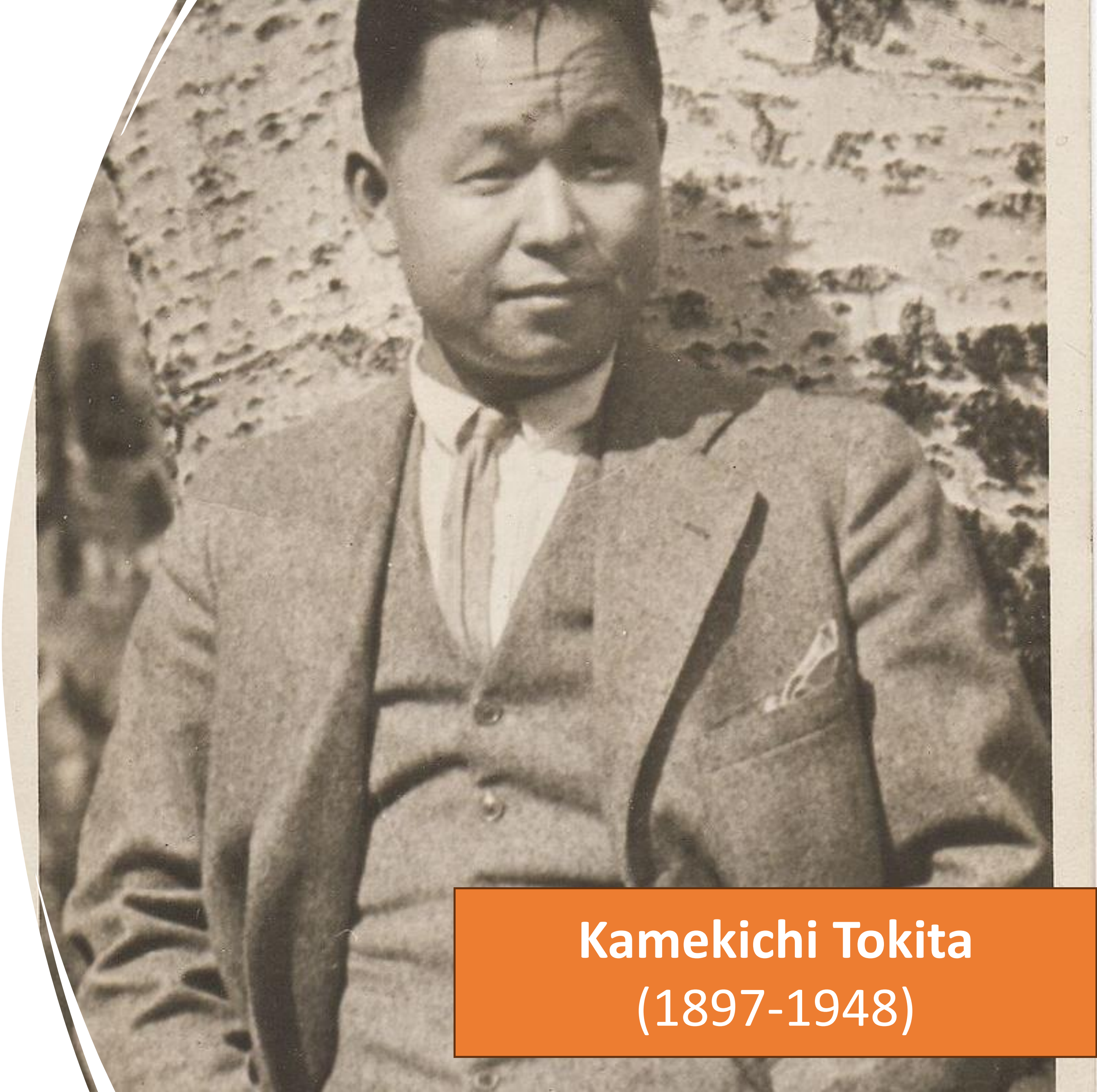
Creating DataFrame Objects

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

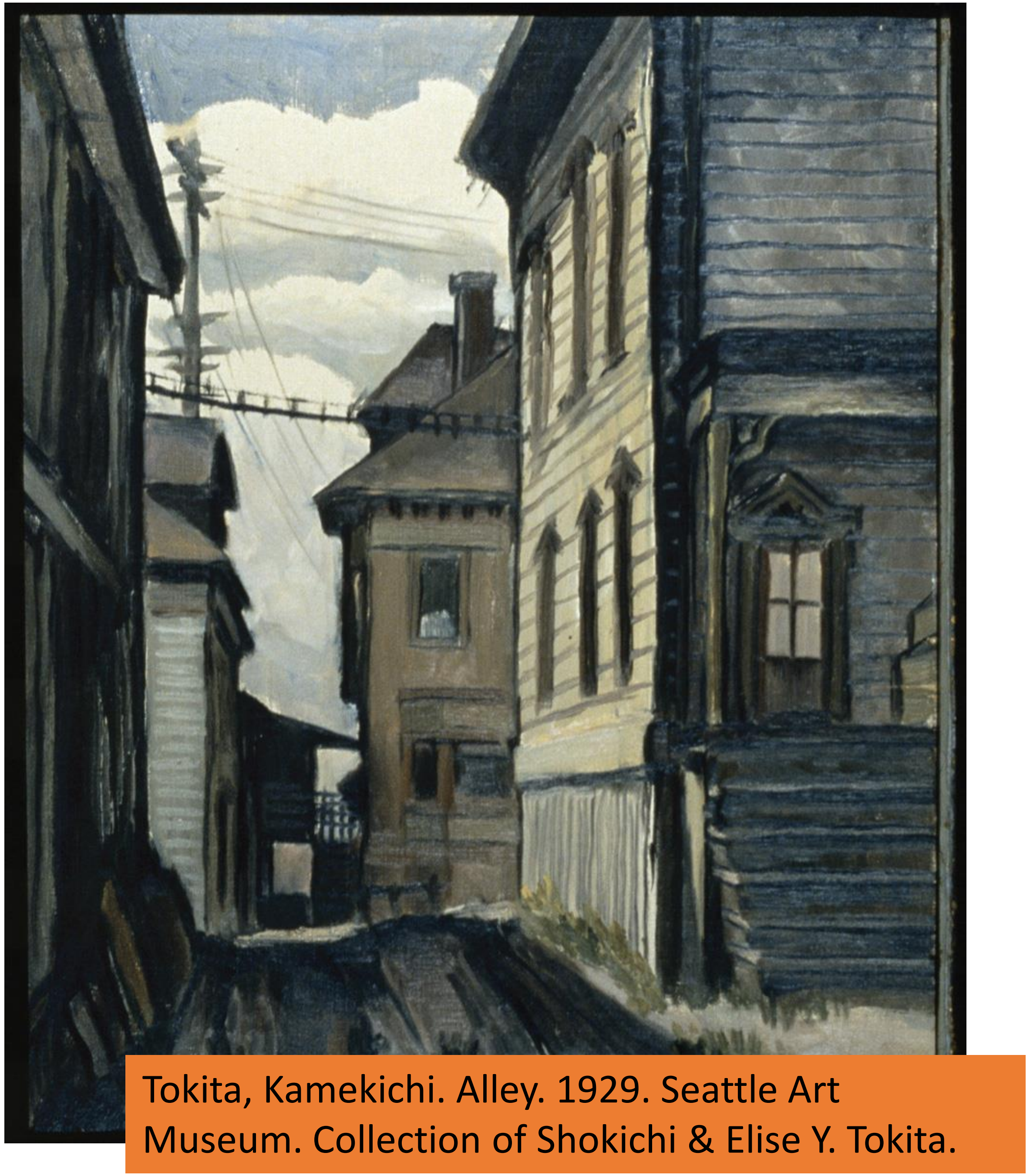
- **Loaded from a file or URL:**
  - delimited text file (.txt, .tsv)
  - comma-separated values (.csv)
  - Excel sheet (.xlsx)
  - JSON dictionary (.json)
  - Stata/SAS files (.sav, .sas, .dta)

A Little Modern Art

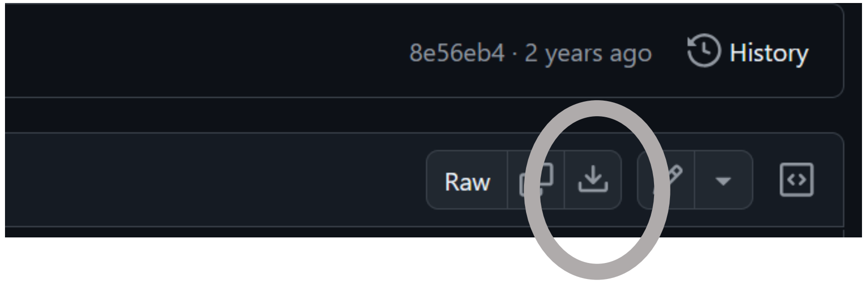
- Our first dataset is the MOMA’s Watson Library Index of Asian American and Pacific Islander Artists.
- This is a collection of AAPI artists featured in the Watson Library’s catalog of exhibit records and artist biographies.



Kamekichi Tokita
(1897-1948)



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



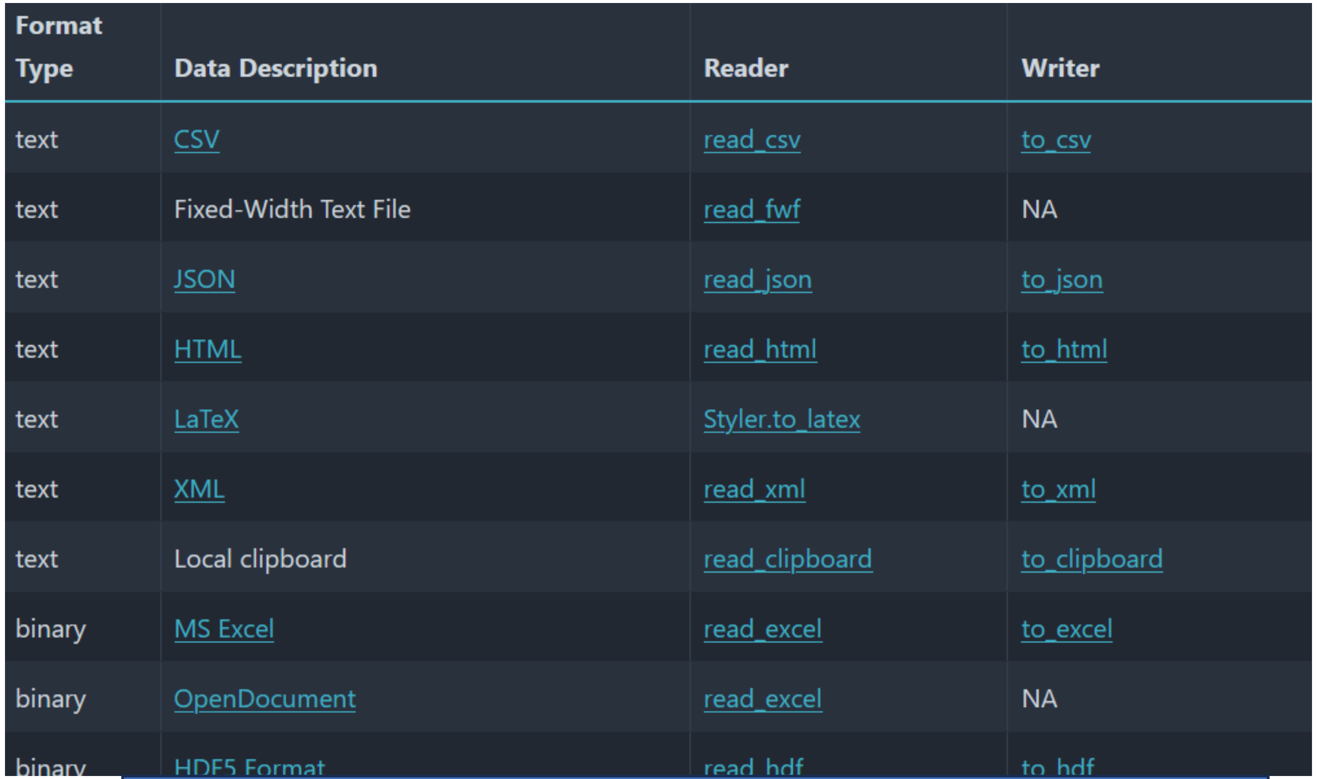
Loading Data From A File

- 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 directory

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()
- …



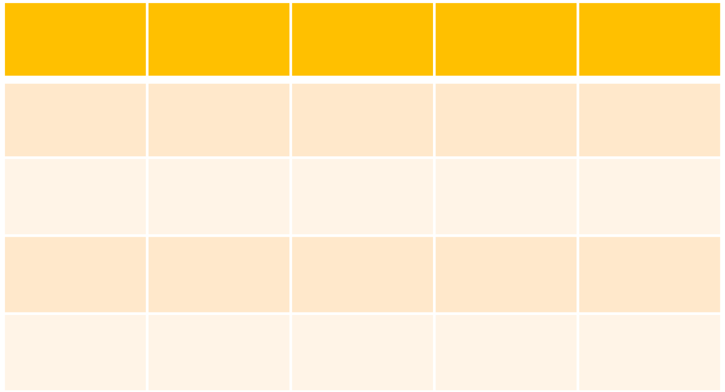
Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	NA
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	LaTeX	Styler.to_latex	NA
text	XML	read_xml	to_xml
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel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

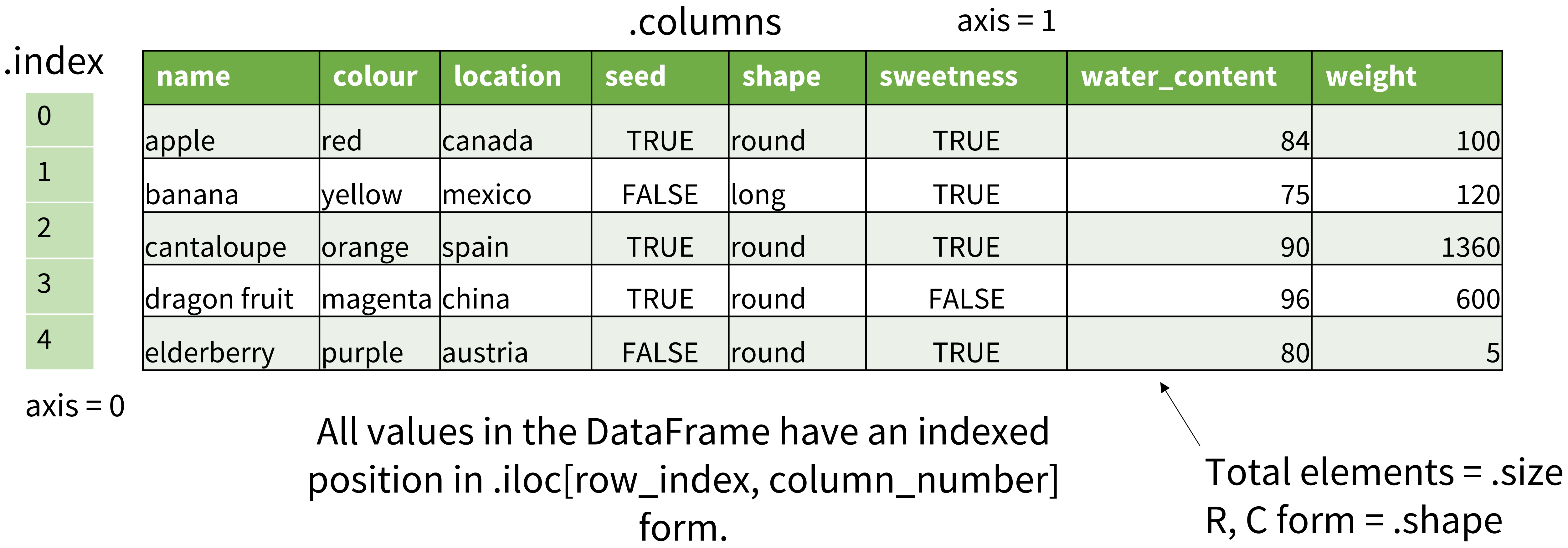
- delimiting (are the columns separated?)
- the shape – a (row #, column #) tuple
- the types of each variable – dtypes
- which columns have nulls/nones/NaNs (and **why**)



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

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:

a pd.DataFrame column test boolean operator

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

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