2530521

slido

Join at slido.com
#2530521

ⓘ Click **Present with Slido** or install our Chrome extension to display joining instructions for participants while presenting.

⚠️ Reminder to start the Zoom recording!

2530521

🍬 Come up to the front to say hi and get some Smarties!

LECTURE 2

Pandas, Part I

Introduction to `pandas` syntax, operators, and functions

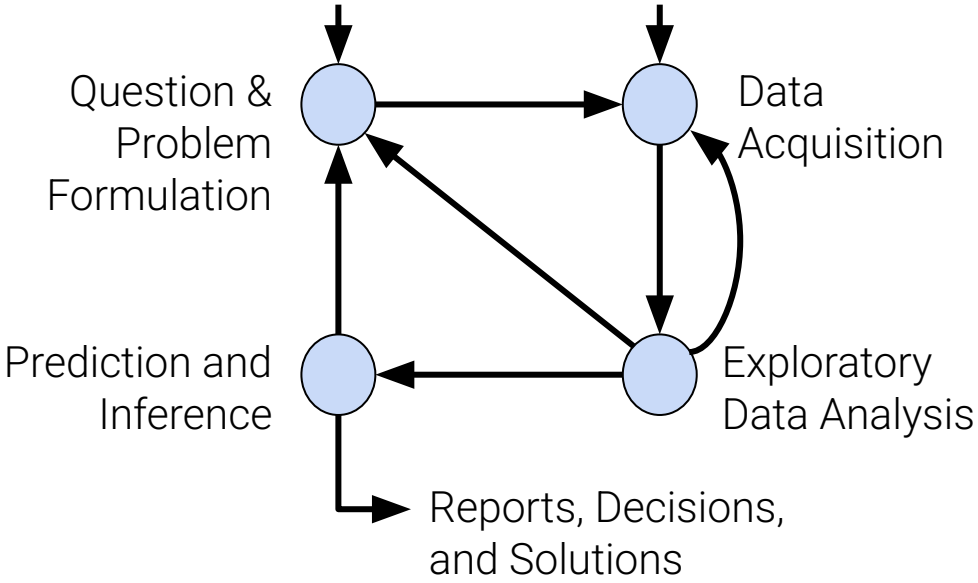
Data 100, Summer 2025 @ UC Berkeley

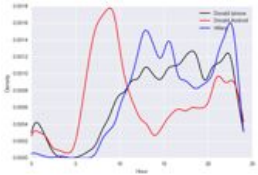
Josh Grossman and Michael Xiao

Recall the Data Science Lifecycle

2530521

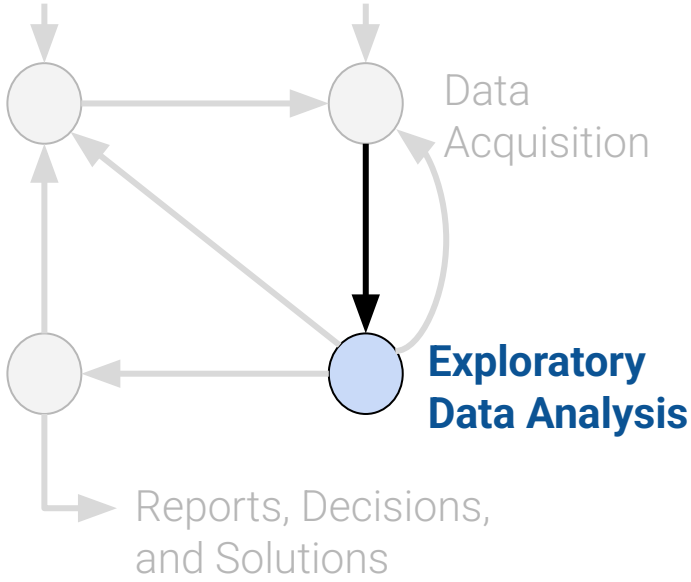
?

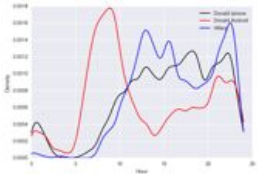




Plan for First Few Weeks

2530521





(This week)

Exploring and Cleaning Tabular Data
From `datascience` to `pandas`

(Next week)

Data Science in Practice
EDA, Data Cleaning, Text processing (regular expressions)

Goals for This Lecture

Lecture 02, Data 100 Summer 2025

2530521

- Introduce **pandas**, an important Python library for working with data
- Key data structures: DataFrames, Series, Indices
- Extracting data: `loc, iloc, []`

This is the first of a three-lecture sequence about **pandas**.

Get ready: lots of code incoming!

- Lecture: introduce high-level concepts
- Lab, homework: practical experimentation

2530521

Agenda

Lecture 02, Data 100 Summer 2025

- Tabular data
- Series, DataFrames, and Indices
- Data extraction with `loc`, `iloc`, and `[]`

Tabular Data

Lecture 02, Data 100 Summer 2025

- **Tabular data**
- Series, DataFrames, and Indices
- Data extraction with `loc`, `iloc`, and `[]`

2530521

Congratulations!!!

You **have collected** or **have been given** a box of data.

What does this "data" actually look like?
How will you work with it?

Data Scientists Love Tabular Data

2530521

"Tabular data" = data in a table.

Typically:

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

A **row** represents one **observation** (here, a single person running for president in a particular year).

A **column** represents some characteristic, or **feature**, of that observation (here, the political party of that person).

In Data 8, you worked with the `datascience` library using `Tables`.

In Data 100 (and beyond), we'll use an industry-standard library called `pandas`.

Introducing the Standard Python Data Science Tool: pandas

2530521

The Python Data Analysis Library

Stands for "panel data"

The Data 100 logo

a cartoon panda

Introducing the Standard Python Data Science Tool: pandas

2530521

pandas is the standard tool across research and industry for working with tabular data.

Using pandas, we can:

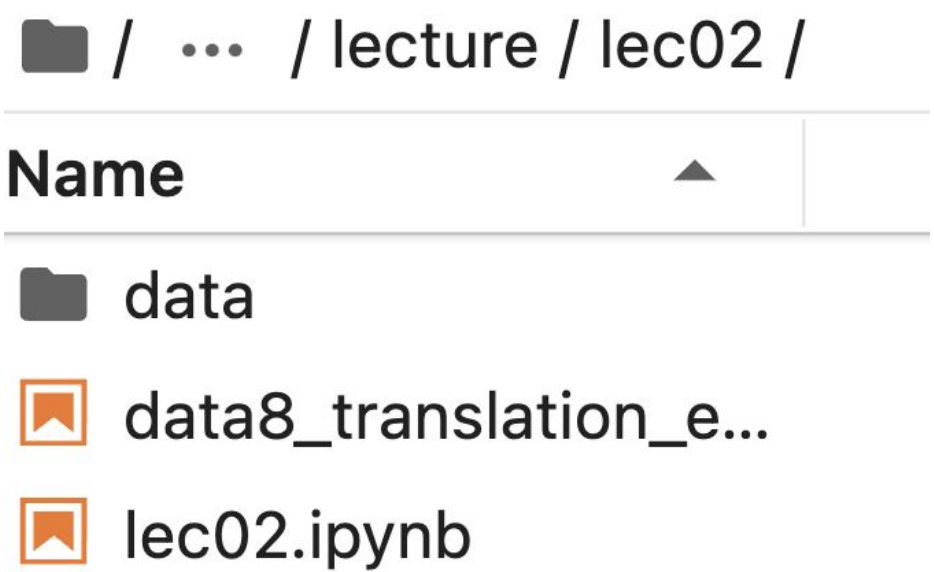
- **Arrange** data in a tabular format.
- **Extract** useful information filtered by specific conditions.
- **Operate** on data to gain new insights.
- **Apply** `NumPy` functions to our data (our friends from Data 8).
- Perform **vectorized** computations to speed up our analysis (Lab 1).

The first week of Data 100 will serve as a "bootcamp" in helping you build familiarity with operating on data with pandas.

Your Data 8 knowledge will serve you well! Much of our work will be in translating syntax.

Today's Lecture Code

2530521



Data used in this lecture

Unofficial `datascience` → `pandas` translations

Primary notebook for lecture

See course website → Lecture 2 for link

During demos, you are encouraged to have a laptop+notebook open to work alongside us!

DataFrames, Series, and Indices

Lecture 02, Data 100 Summer 2025

- Tabular data (Covered last lecture!)
- **DataFrames, Series, and Indices**
- Data extraction with `loc`, `iloc`, and `[]`

DataFrames

2530521

In the "language" of pandas, we call a table a **`DataFrame`**.

We think of **`DataFrame`**s as collections of named columns, called **`Series`**.

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

A `DataFrame`

```
0          Andrew Jackson
1        John Quincy Adams
2          Andrew Jackson
3        John Quincy Adams
4          Andrew Jackson
              ...
182          Donald Trump
183         Kamala Harris
184            Jill Stein
185        Robert Kennedy
186          Chase Oliver
Name: Candidate, Length: 187, dtype: object
```

A `Series` named "Candidate"

Series

A **`Series`** is a 1-dimensional array-like object. It contains:

- A sequence of **values** of the same type.
- A sequence of data labels, called the **index**.

`pd` is the conventional alias for **`pandas`**

```
import pandas as pd
s = pd.Series(["welcome", "to", "data 100"])
```

```
0      welcome
1           to
2     data 100
dtype: object
```

`Index`, accessed by calling `s.index`

```
RangeIndex(start=0, stop=3, step=1)
```

`Values`, accessed by calling `s.values`

```
array(['welcome', 'to', 'data 100'], dtype=object)
```

Constructing a `Series`

2530521

- We can provide index labels for items in a **`Series`** by passing an index list.

```
s = pd.Series([-1, 10, 2], index = ["a", "b", "c"])
```

```
a    -1
b    10
c     2
dtype: int64
```

```
s.index
```

```
Index(['a', 'b', 'c'], dtype='object')
```

- A **`Series`** index can also be changed.

```
s.index = ["first", "second", "third"]
```

```
first     -1
second    10
third      2
dtype: int64
```

```
s.index
```

```
Index(['first', 'second', 'third'], dtype='object')
```

Selection in `Series`

2530521

- We can select a single value or a set of values in a **`Series`** using:
  - A single label
  - A list of labels
  - A filtering condition

```
s = pd.Series([4, -2, 0, 6], index = ["a", "b", "c", "d"])
```

```
a    4
b   -2
c    0
d    6
dtype: int64
```

Selection in `Series`

- We can select a single value or a set of values in a **`Series`** using:
  - **A single label**
  - A list of labels
  - A filtering condition

```
s = pd.Series([4, -2, 0, 6], index = ["a", "b", "c", "d"])
```

```
a    4
b   -2
c    0
d    6
dtype: int64
```

```
s["a"]
```

`4`

A single string label

A single value

Selection in `Series`

- We can select a single value or a set of values in a **`Series`** using:
  - A single label
  - **A list of labels**
  - A filtering condition

```
s = pd.Series([4, -2, 0, 6], index = ["a", "b", "c", "d"])
```

```
a    4
b   -2
c    0
d    6
dtype: int64
```

```
s[["a", "c"]]
```

```
a    4
c    0
dtype: int64
```

A list of string labels

A `Series`

Selection in `Series`

2530521

- We can select a single value or a set of values in a **`Series`** using:
  - A single label
  - A list of labels
  - **A filtering condition**

```
s = pd.Series([4, -2, 0, 6], index = ["a", "b", "c", "d"])
```

```
a    4
b   -2
c    0
d    6
dtype: int64
```

How to select values in the **`Series`** that satisfy a condition:

1) Apply a boolean condition to the **`Series`**, creating a **new boolean `Series`** (often called a **"boolean mask"**).
2) Index into our original **`Series`** using the boolean mask. **`pandas`** selects only the entries in the **`Series`** that satisfy the condition.

```
s > 0
```

```
a     True
b    False
c    False
d     True
dtype: bool
```

```
s[s > 0]
```

Boolean mask

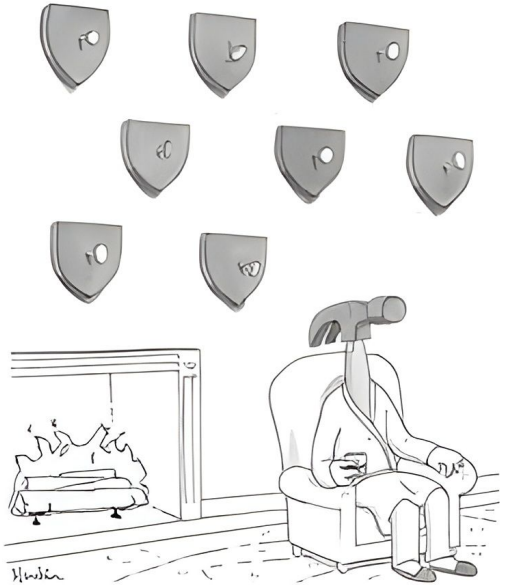
```
a    4
d    6
dtype: int64
```

Demo Slides

lec02.ipynb

Nuts and bolts of `Series`

2530521



slido

2530521

What is the output of the following code?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

DataFrames of Series!

2530521

In Data 100, we primarily think of **`Series`** as columns in a **`DataFrame`**.

We can think of a **`DataFrame`** as a collection of **`Series`** that all share the same **`Index`**.

```
0       1824
1       1824
2       1828
3       1828
4       1832
       ...
182     2024
183     2024
184     2024
185     2024
186     2024
Name: Year,
```

The Series **`"Year"`**

```
0          Andrew Jackson
1       John Quincy Adams
2          Andrew Jackson
3       John Quincy Adams
4          Andrew Jackson
              ...
182          Donald Trump
183         Kamala Harris
184             Jill Stein
185         Robert Kennedy
186           Chase Oliver
Name: Candidate,
```

The Series **`"Candidate"`**

[...]

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

The DataFrame **`elections`**

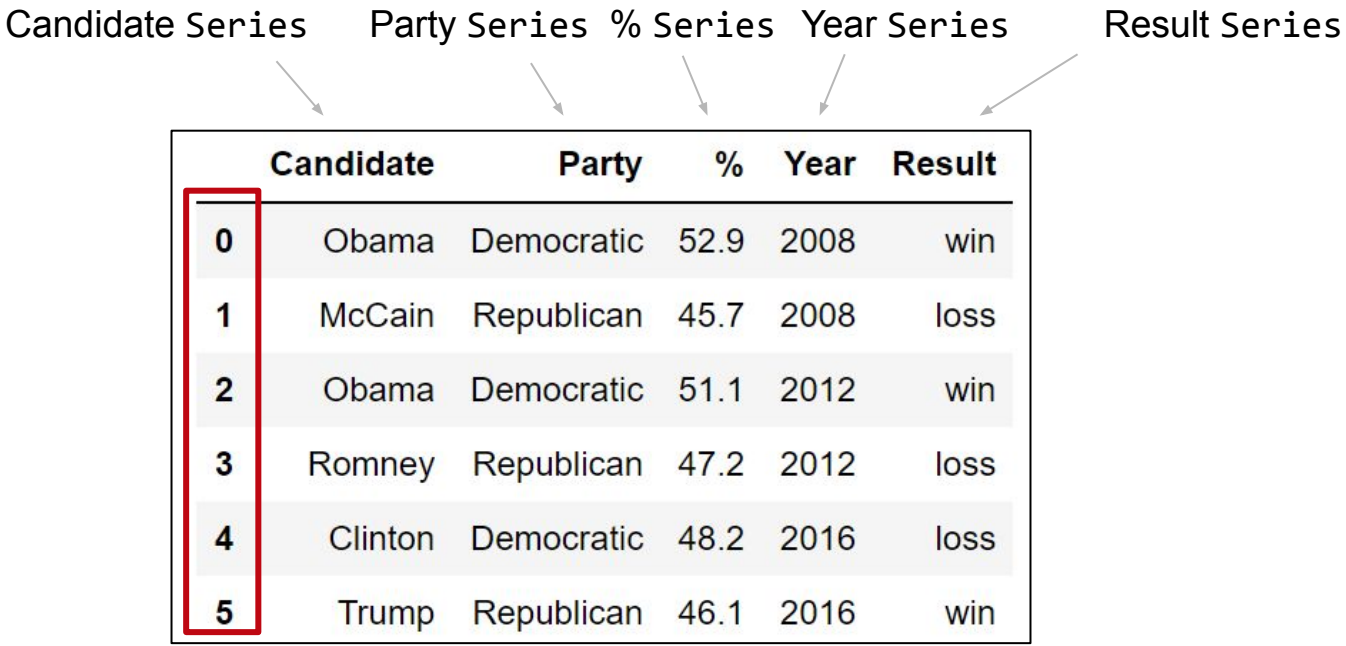
Non-native English speaker note: The plural of "series" is "series". Sorry.

The Relationship Between DataFrames, Series, and Indices

2530521

We can think of a **`DataFrame`** as a collection of **`Series`** that all share the same **`Index`**.

- Candidate, Party, %, Year, and Result **`Series`** all share an **`Index`** from 0 to 5.



	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Creating a `DataFrame`

2530521

The syntax of creating **`DataFrame`** is:

```
pandas.DataFrame(data, index, columns)
```

Many approaches exist for creating a **`DataFrame`**. Here, we will go over the most popular ones.

- From a CSV file.
- Using a list and column name(s).
- From a dictionary.
- From a **`Series`**.

Understanding the syntax we show you is far more important than **memorizing** it.

- Great data scientists look up syntax all the time (even basic syntax!).

Knowing **what** you want to do is more essential than knowing *precisely* **how** to code it.

- LLMs, Google, and documentation are good at the "how".

Creating a `DataFrame`

2530521

The syntax of creating **`DataFrame`** is:

```
pandas.DataFrame(data, index, columns)
```

Many approaches exist for creating a **`DataFrame`**. Here, we will go over the most popular ones.

- **From a CSV file**.
- Using a list and column name(s).
- From a dictionary.
- From a **`Series`**.

```
elections = pd.read_csv("data/elections.csv")
```

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

The DataFrame **`elections`**

Creating a `DataFrame`

2530521

The syntax of creating **`DataFrame`** is:

```
pandas.DataFrame(data, index, columns)
```

Many approaches exist for creating a **`DataFrame`**. Here, we will go over the most popular ones.

- **From a CSV file**.
- Using a list and column name(s).
- From a dictionary.
- From a **`Series`**.

```
elections = pd.read_csv("data/elections.csv", index_col="Year")
```

Year	Candidate	Party	Popular vote	Result	%
1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
1828	Andrew Jackson	Democratic	642806	win	56.203927
1828	John Quincy Adams	National Republican	500897	loss	43.796073
1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...
2024	Donald Trump	Republican	77303568	win	49.808629
2024	Kamala Harris	Democratic	75019230	loss	48.336772
2024	Jill Stein	Green	861155	loss	0.554864
2024	Robert Kennedy	Independent	756383	loss	0.487357
2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

The DataFrame **`elections`** with **`"Year"`** as **`Index`**

Creating a `DataFrame`

2530521

Many approaches exist for creating a **`DataFrame`**. Here, we will go over the most popular ones.

- From a CSV file.
- **Using a list and column name(s)**.
- From a dictionary.
- From a **`Series`**.

Each sublist is a **row** or **observation**.

```
pd.DataFrame([1, 2, 3],
             columns=["Numbers"])
```

	Numbers
0	1
1	2
2	3

```
pd.DataFrame([[1, "one"], [2, "two"]],
             columns = ["Number", "Description"])
```

	Number	Description
0	1	one
1	2	two

Creating a `DataFrame`

2530521

Many approaches exist for creating a **`DataFrame`**. Here, we will go over the most popular ones.

- From a CSV file.
- Using a list and column name(s).
- **From a dictionary**.
- From a **`Series`**.

Specify columns of the **`DataFrame`**

```
pd.DataFrame({"Fruit":["Strawberry", "Orange"],
              "Price": [5.49, 3.99]})
```

Looks like JSON file format!
Future lecture.

```
pd.DataFrame([{"Fruit":"Strawberry", "Price":5.49},
              {"Fruit":"Orange", "Price":3.99}])
```

Specify rows of the **`DataFrame`**

	Fruit	Price
0	Strawberry	5.49
1	Orange	3.99

Creating a `DataFrame`

2530521

Many approaches exist for creating a **`DataFrame`**. Here, we will go over the most popular ones.

- From a CSV file.
- Using a list and column name(s).
- From a dictionary.
- **From a `Series`**.

```
s_a = pd.Series(["a1", "a2", "a3"], index = ["r1", "r2", "r3"])
s_b = pd.Series(["b1", "b2", "b3"], index = ["r1", "r2", "r3"])

pd.DataFrame({"A-column":s_a, "B-column":s_b})
```

	A-column	B-column
r1	a1	b1
r2	a2	b2
r3	a3	b3

```
pd.DataFrame(s_a)
```

```
s_a.to_frame()
```

	0
r1	a1
r2	a2
r3	a3

Indices Are Not Necessarily Row Numbers

2530521

The **`Index`** (a.k.a. row labels) can also:

- Be non-numeric.
- Have a name, e.g., "Candidate".

```
# Creating a DataFrame from a CSV file and specifying the Index column
elections = pd.read_csv("data/elections.csv", index_col = "Candidate")
```

Candidate	Year	Party	Popular vote	Result	%
Andrew Jackson	1824	Democratic-Republican	151271	loss	57.210122
John Quincy Adams	1824	Democratic-Republican	113142	win	42.789878
Andrew Jackson	1828	Democratic	642806	win	56.203927
John Quincy Adams	1828	National Republican	500897	loss	43.796073
Andrew Jackson	1832	Democratic	702735	win	54.574789

Indices Are Not Necessarily Unique

2530521

The row labels of an index can be non-unique.

- Left: The **index** values are unique and numeric, acting as a row number. The default!
- Right: The **index** values are named and non-unique.

	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

	Candidate	Party	%	Result
Year				
2008	Obama	Democratic	52.9	win
2008	McCain	Republican	45.7	loss
2012	Obama	Democratic	51.1	win
2012	Romney	Republican	47.2	loss
2016	Clinton	Democratic	48.2	loss
2016	Trump	Republican	46.1	win

When are non-unique indices useful? Note that performance is worse when non-unique.

Modifying Indices

2530521

- We can select a new column and set it as the index of the `DataFrame`.

```
elections = elections.set_index("Candidate")
```

Candidate	Year	Party	Popular vote	Result	%
Andrew Jackson	1824	Democratic-Republican	151271	loss	57.210122
John Quincy Adams	1824	Democratic-Republican	113142	win	42.789878
Andrew Jackson	1828	Democratic	642806	win	56.203927
John Quincy Adams	1828	National Republican	500897	loss	43.796073
Andrew Jackson	1832	Democratic	702735	win	54.574789
...	...	...	...	...	...
Donald Trump	2024	Republican	77303568	win	49.808629
Kamala Harris	2024	Democratic	75019230	loss	48.336772
Jill Stein	2024	Green	861155	loss	0.554864
Robert Kennedy	2024	Independent	756383	loss	0.487357
Chase Oliver	2024	Libertarian Party	650130	loss	0.418895

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

On the Data 100 Reference Sheet!

Resetting the `Index`

2530521

- We can change our mind and reset the **`Index`** back to the default list of integers.

```
elections = elections.reset_index()
```

Candidate	Year	Party	Popular vote	Result	%
Andrew Jackson	1824	Democratic-Republican	151271	loss	57.210122
John Quincy Adams	1824	Democratic-Republican	113142	win	42.789878
Andrew Jackson	1828	Democratic	642806	win	56.203927
John Quincy Adams	1828	National Republican	500897	loss	43.796073
Andrew Jackson	1832	Democratic	702735	win	54.574789
...	...	...	...	...	...
Donald Trump	2024	Republican	77303568	win	49.808629
Kamala Harris	2024	Democratic	75019230	loss	48.336772
Jill Stein	2024	Green	861155	loss	0.554864
Robert Kennedy	2024	Independent	756383	loss	0.487357
Chase Oliver	2024	Libertarian Party	650130	loss	0.418895

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

On the Data 100 Reference Sheet!

Column Names Are Usually Unique!

2530521

Column names in **pandas** are almost always unique.

- Example: Really shouldn't have two columns named "`Candidate`".

	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Retrieving the `Index`, `Columns`, and `shape`

2530521

```
elections = elections.set_index("Candidate")
```

To extract row labels, use `DataFrame.index`:

```
elections.index
```

```
Index(['Democratic-Republican', 'Democratic-Republican', 'Democratic',
       'National Republican', 'Democratic', 'National Republican',
       'Anti-Masonic', 'Whig', 'Democratic', 'Whig',
       ...
       'Green', 'Democratic', 'Republican', 'Libertarian', 'Green',
       'Republican', 'Democratic', 'Green', 'Independent',
       'Libertarian Party'],
      dtype='object', name='Party', length=187)
```

To extract column labels, use `DataFrame.columns`:

```
elections.columns
```

```
Index(['Candidate', 'Year', 'Popular vote', 'Result', '%'], dtype='object')
```

For dimensions use `DataFrame.shape`:

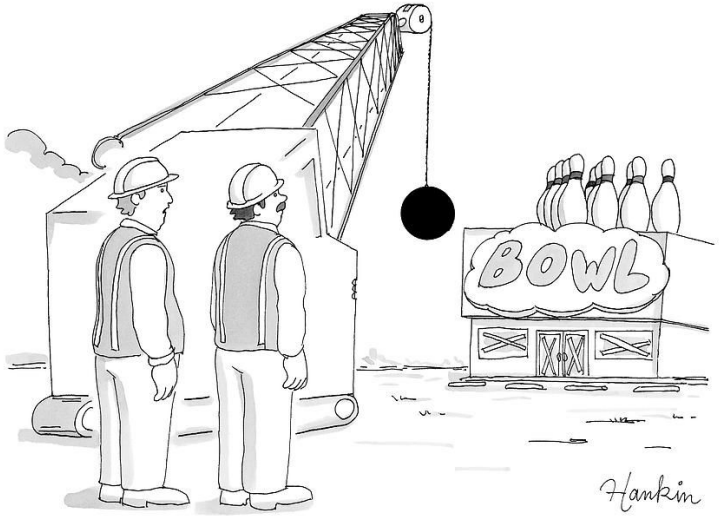
```
elections.shape
```

```
(187, 6)
```

2530521

Demo Slides

lec02.ipynb



"A strike sends us home early."

slido

2530521

Which of the following lines of code creates this DataFrame?

Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

The `DataFrame` API

The API for the **`DataFrame`** class is enormous.

- API: "Application Programming Interface".
- The API is the set of abstractions supported by the class (e.g., **`DataFrame`**.index)

Full documentation is at pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

- Compare with the **`Table`** class from Data8: data8.org/datascience/tables.html
- We will only consider a tiny portion of the **`pandas`** API.

We want you to get familiar with the real world programming practice of… looking up syntax!

- Ask an LLM, check the **`pandas`** documentation, read Stack Overflow, Google, etc.

2530521

Interlude

2-min stretch break!

Data Extraction with `loc`, `iloc`, and `[ ]`

Lecture 02, Data 100 Summer 2025

- Tabular data
- DataFrames, Series, and Indices
- **Data extraction with `loc`, `iloc`, and `[]`**

Extracting Data

2530521

There are lots of different ways to extract rows and columns of interest from a `DataFrame`:

- Grab the **first or last `n` rows** in the `DataFrame`
- Grab data with a certain **label**.
- Grab data at a certain **position**.

We'll find that all three of these methods are useful to us in different contexts.

`.head` and `.tail`

2530521

The simplest scenarios: Extract the first or last `n` rows.

- `df.head(n)` returns the first `n` rows of the DataFrame `df`.
- `df.tail(n)` returns the last `n` rows.

`elections`

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

`elections.head(5)`

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789

`elections.tail(5)`

	Year	Candidate	Party	Popular vote	Result	%
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

Note: If you don't specify `n`, `elections.head()` returns 5 rows by default. Common practice!

Label-based Extraction: `.loc`

2530521

A more complex task: Extract data with specific column or index labels.

```
df.loc[row_labels, column_labels]
```

`.loc` stands for location

The `.loc` accessor allows us to specify **labels** of rows and columns we wish to extract.

- Labels are the bolded text at the top and left of a `DataFrame`.

Row labels

Column labels

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

Label-based Extraction: `.loc`

2530521

Each of the **two** arguments to `.loc` can be:

- A list.
- A slice (inclusive of the right hand side of the slice).
- A single value.

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

Label-based Extraction: `.loc`

Each of the **two** arguments to `.loc` can be:

- **A list.**
- A slice (inclusive of the right hand side of the slice).
- A single value.

List List

```
elections.loc[[87, 25, 179], ["Year", "Candidate", "Result"]]
```

Select the rows with labels 87, 25, and 179.

	Year	Candidate	Result
87	1932	Herbert Hoover	loss
25	1860	John C. Breckinridge	loss
179	2020	Donald Trump	loss

Select the columns with labels "Year", "Candidate", and "Result".

Label-based Extraction: `.loc`

2530521

Each of the **two** arguments to `.loc` can be:

- A list.
- **A slice** (**syntax is <u>inclusive</u> of the right hand side of the slice**).
- A single value.

List Slice

```
elections.loc[[87, 25, 179], "Popular vote":"%"]
```

Select all columns *starting* from "Popular vote" *until* "%".

Select the rows with labels 87, 25, and 179.

	Popular vote	Result	%
87	15761254	loss	39.830594
25	848019	loss	18.138998
179	74216154	loss	46.858542

Label-based Extraction: `.loc`

2530521

To extract *all* rows or *all* columns, use a colon (`:`)

```
elections.loc[:, ["Year", "Candidate", "Result"]]
```

All rows for the columns with labels "Year", "Candidate", and "Result".

Ellipses (...) indicate more rows not shown.

	Year	Candidate	Result
0	1824	Andrew Jackson	loss
1	1824	John Quincy Adams	win
2	1828	Andrew Jackson	win
3	1828	John Quincy Adams	loss
4	1832	Andrew Jackson	win
...	...	...	...
182	2024	Donald Trump	win
183	2024	Kamala Harris	loss
184	2024	Jill Stein	loss
185	2024	Robert Kennedy	loss
186	2024	Chase Oliver	loss

```
elections.loc[[87, 25, 179], :]
```

All columns for the rows with labels 87, 25, 179.

	Candidate	Year	Party	Popular vote	Result	%
87	Herbert Hoover	1932	Republican	15761254	loss	39.830594
25	John C. Breckinridge	1860	Southern Democratic	848019	loss	18.138998
179	Donald Trump	2020	Republican	74216154	loss	46.858542

Technical note: `elections.loc[[87, 25, 179]]` is the same as `elections.loc[[87, 25, 179], :]`.
With one argument, `.loc` assumes the second argument is a colon (:).

Label-based Extraction: `.loc`

2530521

Each of the **two** arguments to `.loc` can be:

- A list.
- A slice (inclusive of the right hand side of the slice).
- **A single value.**

List Single value

```
elections.loc[[87, 25, 179], "Popular vote"]
```

```
87     15761254
25       848019
179    74216154
Name: Popular vote, dtype: int64
```

Wait, what? Why did everything get so ugly?

We've extracted a subset of the "Popular vote" column as a `Series`.

```
elections.loc[0, "Candidate"]
```

```
'Andrew Jackson'
```

We've extracted the single element with row label 0 and column label "Candidate".

Demo Slides

lec02.ipynb

Extraction with .loc

2530521

.loc: Which of the following statements correctly returns the value "blue fish" from the "weird" DataFrame?

Integer-based Extraction: `.iloc`

2530521

A different scenario: We want to extract data according to its **numeric** *position*.

- Example: Grab the 1st, 2nd, and 3rd columns of the `DataFrame`.

```
df.iloc[row_integers, column_integers]
```

`.iloc` stands for integer location

The `.iloc` accessor allows us to specify the **integers** of rows and columns we wish to extract.

- Python convention: The first position has integer index 0.

Row integers

Column integers

	0	1	2	3	4	5
	Year	**Candidate**	**Party**	**Popular vote**	**Result**	**%**
0	**0** 1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	**1** 1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	**2** 1828	Andrew Jackson	Democratic	642806	win	56.203927
3	**3** 1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	**4** 1832	Andrew Jackson	Democratic	702735	win	54.574789
	... ...	...	...	...	...	...

Integer-based Extraction: `.iloc`

2530521

Each of the **two** arguments to `.iloc` can be:

- A list.
- A slice (syntax is **exclusive** of the right hand side of the slice, akin to `range(a,b)`).
- A single value.

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772
184	2024	Jill Stein	Green	861155	loss	0.554864
185	2024	Robert Kennedy	Independent	756383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895

Integer-based Extraction: `.iloc`

2530521

Each of the **two** arguments to `.iloc` can be:

- **A list.**
- A slice (syntax is **exclusive** of the right hand side of the slice, akin to `range(a,b)`).
- A single value.

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

Select the rows at positions 1, 2, and 3.

	Year	Candidate	Party
1	1824	John Quincy Adams	Democratic-Republican
2	1828	Andrew Jackson	Democratic
3	1828	John Quincy Adams	National Republican

Select the columns at positions 0, 1, and 2.

Integer-based Extraction: `.iloc`

2530521

Each of the **two** arguments to `.iloc` can be:

- A list.
- **A slice (syntax is <u>exclusive</u> of the right hand side of the slice, akin to `range(a,b)`).**
- A single value.

```
elections.iloc[[1, 2, 3], 0:3]
```

	0 Year	1 Candidate	2 Party
1	1824	John Quincy Adams	Democratic-Republican
2	1828	Andrew Jackson	Democratic
3	1828	John Quincy Adams	National Republican

Select the rows at positions 1, 2, **and 3**.

Select *all* columns from integer 0 *to* **integer 2**.

Remember:
Integer-based slicing is right-end exclusive!

Integer-based Extraction: `.iloc`

Just like `.loc`, we can use a colon with `.iloc` to extract all rows or all columns.

```
elections.iloc[:, 0:3]
```

	Year	Candidate	Result
0	1824	Andrew Jackson	loss
1	1824	John Quincy Adams	win
2	1828	Andrew Jackson	win
3	1828	John Quincy Adams	loss
4	1832	Andrew Jackson	win
...	...	...	...
182	2024	Donald Trump	win
183	2024	Kamala Harris	loss
184	2024	Jill Stein	loss
185	2024	Robert Kennedy	loss
186	2024	Chase Oliver	loss

Grab **all rows** of columns at integer positions 0 to 2.

Integer-based Extraction: `.iloc`

2530521

Each of the **two** arguments to `.iloc` can be:

- A list.
- A slice (syntax is exclusive of the right hand side of the slice).
- **A single value.**

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

```
1    John Quincy Adams
2       Andrew Jackson
3    John Quincy Adams
Name: Candidate, dtype: object
```

As before, the result for a single value argument is a `Series`.

Extract row integers 1, 2, and 3 from the column at position 1.

```
elections.iloc[0, 1]
```

```
'Andrew Jackson'
```

Extract the single element with row position 0 and column position 1.

Demo Slides

lec02.ipynb

Extraction with .iloc

2530521

.iloc: Which of the following statements correctly returns the value "blue fish" from the "weird" DataFrame?

`.loc` vs `.iloc`

2530521

Remember:

- `.loc` performs **l**abel-based extraction
- `.iloc` performs **i**nteger-based extraction

When choosing between `.loc` and `.iloc`, you'll usually choose `.loc`.

- **Safer**: If the order of data gets shuffled in a public database, your code still works.
- **Readable**: Easier to understand what `elections.loc[:, ["Year", "Candidate", "Result"]]` means than `elections.iloc[:, [0, 1, 4]]`

`.iloc` is still useful.

- Example: `DataFrame` of movies sorted by box office earnings. Use `.iloc` to get the median earnings (i.e., index into the middle).

… Just When It Was All Making Sense

2530521

Context-dependent Extraction: [ ]

2530521

Selection operators:

- `.loc` selects items by **label**. First argument is rows, second argument is columns.
- `.iloc` selects items by **integer**. First argument is rows, second argument is columns.

- `[]` only takes one argument, which may be:
  - A slice of **row numbers** (right-hand exclusive).
  - A list of **column labels**.
  - A single **column label**.

That is, `[]` is context sensitive! But, it's the most **popular**.

Context-dependent Extraction: [ ]

2530521

[ ] only takes one argument, which may be:

- **A slice of row integers** (right-hand exclusive).
- A list of column labels.
- A single column label.

```
elections[3:7]
```

	Year	Candidate	Party	Popular vote	Result	%
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
5	1832	Henry Clay	National Republican	484205	loss	37.603628
6	1832	William Wirt	Anti-Masonic	100715	loss	7.821583

Context-dependent Extraction: [ ]

2530521

`[ ]` only takes one argument, which may be:

- A slice of row numbers (right-hand exclusive).
- **A list of column labels.**
- A single column label.

```
elections[["Year", "Candidate", "Result"]]
```

Common typo: Forget inner brackets!

	Year	Candidate	Result
0	1824	Andrew Jackson	loss
1	1824	John Quincy Adams	win
2	1828	Andrew Jackson	win
3	1828	John Quincy Adams	loss
4	1832	Andrew Jackson	win
...	...	...	...
182	2024	Donald Trump	win
183	2024	Kamala Harris	loss
184	2024	Jill Stein	loss
185	2024	Robert Kennedy	loss
186	2024	Chase Oliver	loss

Context-dependent Extraction: [ ]

2530521

`[ ]` only takes one argument, which may be:

- A slice of row numbers (right-hand exclusive).
- A list of column labels.
- **A single column label.**

```
elections["Candidate"]
```

```
0          Andrew Jackson
1       John Quincy Adams
2          Andrew Jackson
3       John Quincy Adams
4          Andrew Jackson
              ...
182          Donald Trump
183         Kamala Harris
184            Jill Stein
185        Robert Kennedy
186          Chase Oliver
Name: Candidate, Length: 187, dtype: object
```

Extract the "Candidate" column as a `Series`.

`elections[["Candidate"]]` returns a one-column `DataFrame`.

Demo Slides

lec02.ipynb

Context-based extraction

2530521

Context-dependent extraction with [ ]: Which of the following statements correctly returns the value "blue fish" from the "weird" DataFrame?

Why Use [ ]?

2530521

In short: [ ] can be much more concise than `.loc` or `.iloc`

- Example: Extracting the "Candidate" column. It is far simpler to write `elections["Candidate"]` than `elections.loc[:, "Candidate"]`

In practice, [ ] is often used over `.iloc` and `.loc` in data science work. Typing time adds up!

LECTURE 2

Pandas, Part I

Content credit: Acknowledgments