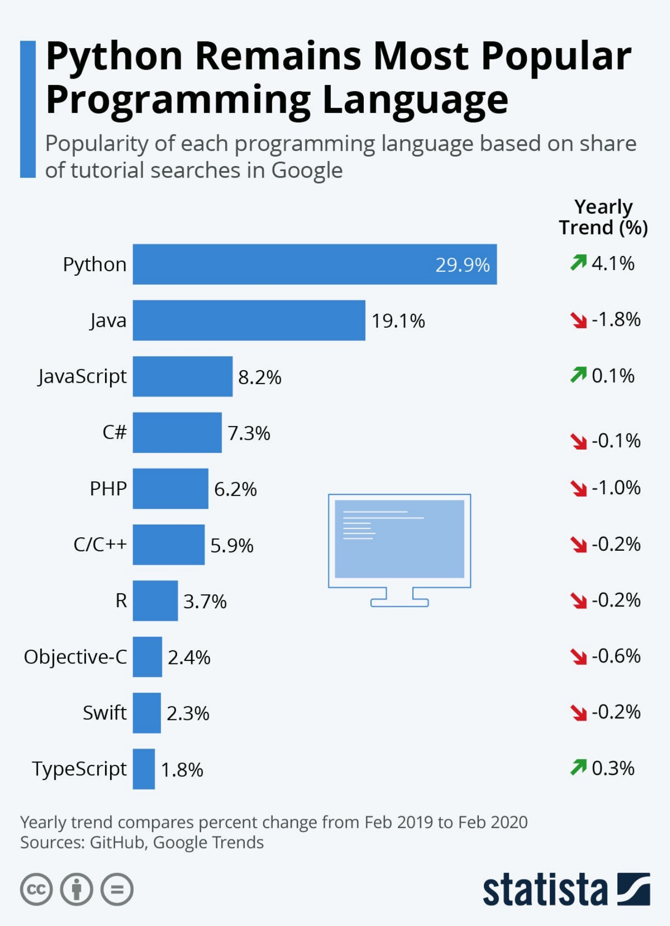
Python Growth



IEEE SPECTRUM

Rank	Language	Type	Score
1	Python		100.0
2	Java		95.3
3	C		94.6
4	C++		87.0
5	JavaScript		79.5
6	R		78.6
7	Arduino		73.2
8	Go		73.1

LECTURE 2

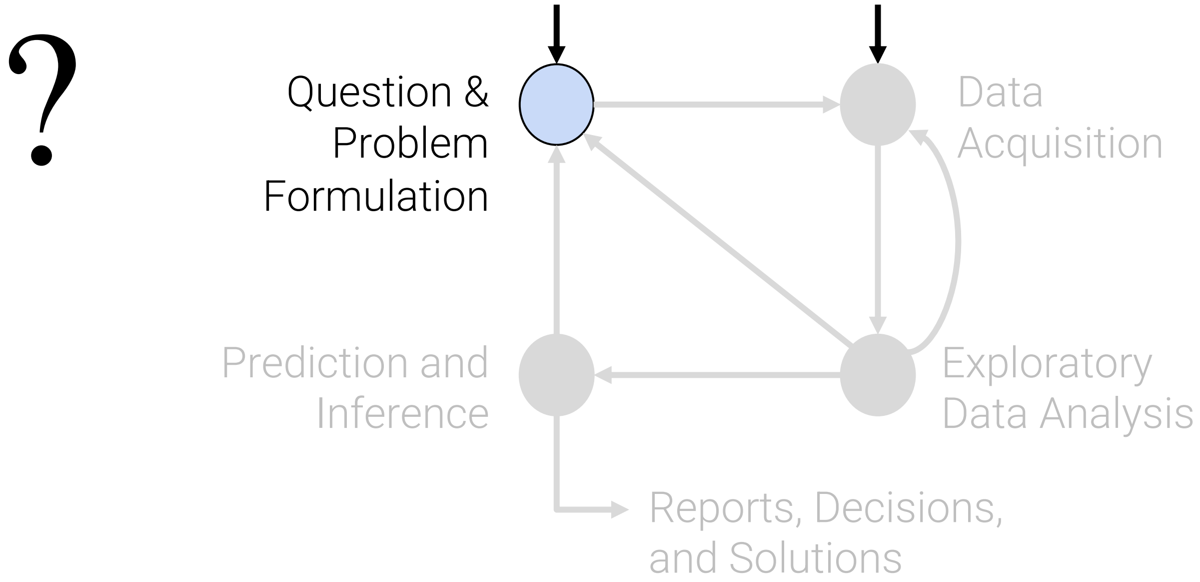
Pandas, Part I

Introduction to `pandas` syntax, operators, and functions

Data Science| Fall 24@ Knowledge Stream

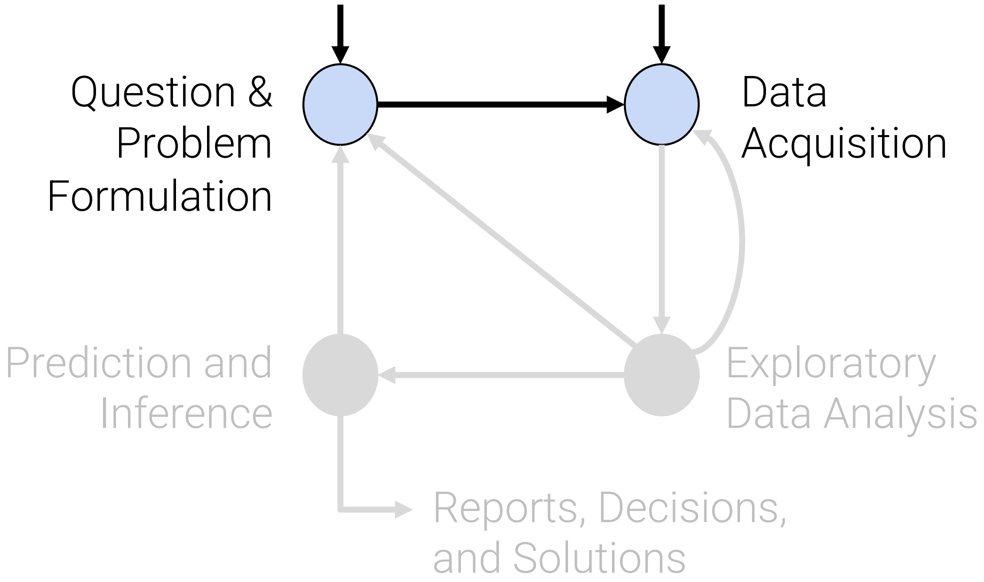
Sana Jabbar

Data Science Lifecycle



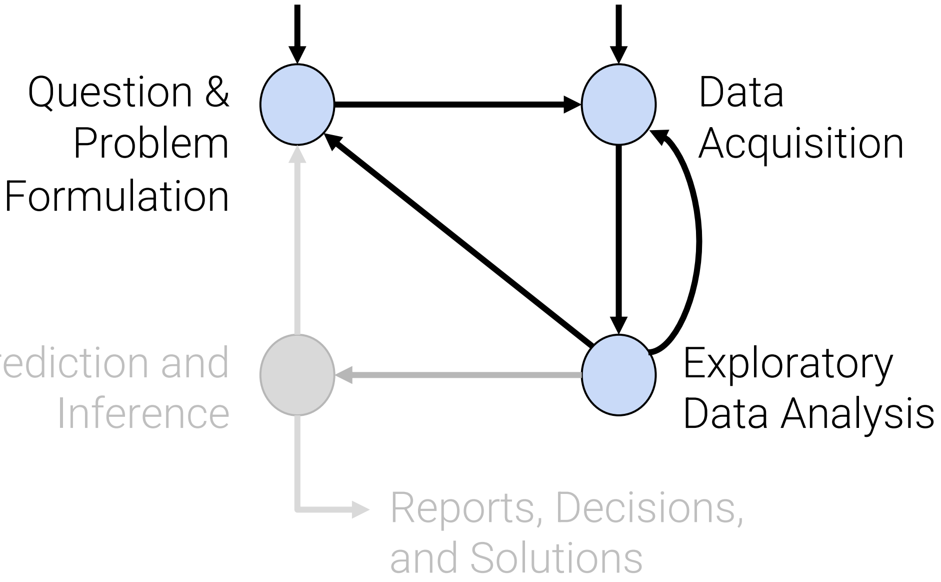
Data Science Lifecycle

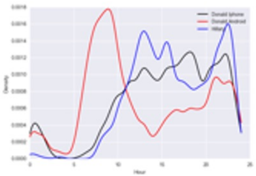
?



Data Science Lifecycle

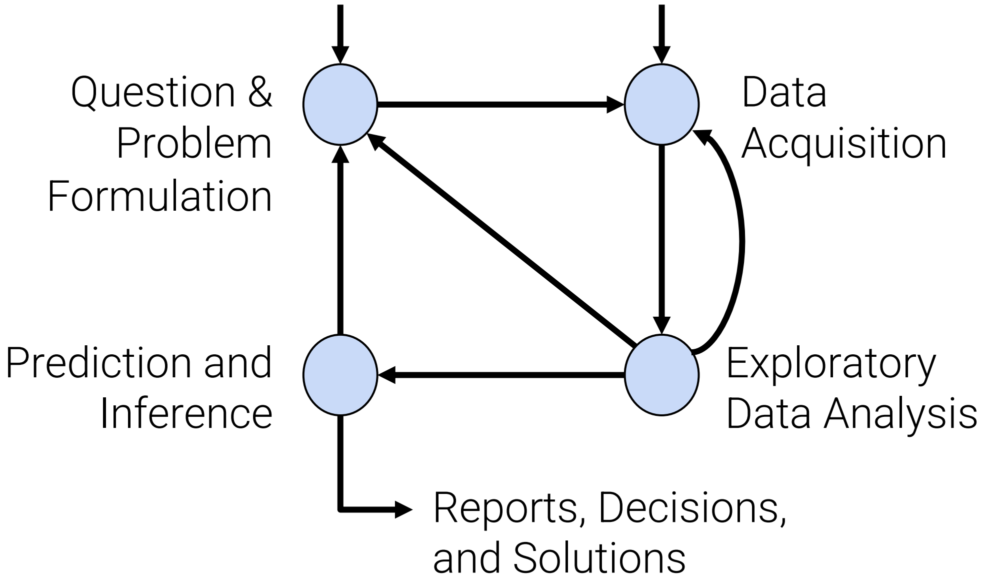
?

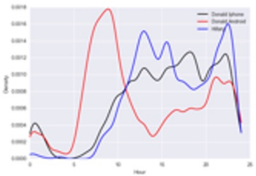




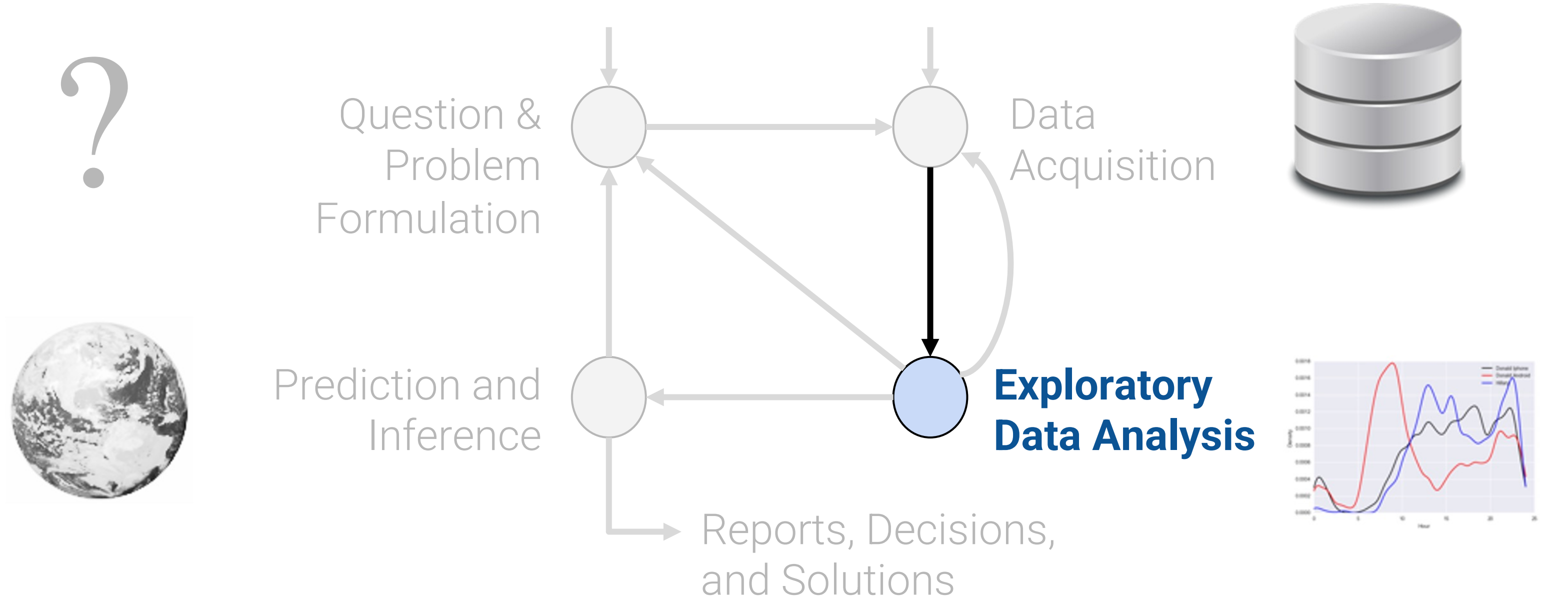
Data Science Lifecycle

?





Plan for First Few Weeks



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

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

Goals for This Lecture

Lecture 02

- 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 lecture sequence about `pandas`.

Get ready: lots of code is incoming!

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

Tabular Data

Lecture 02

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

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

"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
...	...	...	...	...	...	...
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

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).

We'll use an industry-standard library called `pandas`.

Introducing the Standard Python Data Science Tool: pandas

The Python Data
Analysis Library

Introducing the Standard Python Data Science Tool: pandas

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.
- Perform vectorized computations to speed up our analysis.

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

This Data Science course will serve as a "bootcamp" in helping you build familiarity with operating on data with pandas.

DataFrames, Series, and Indices

Lecture 2

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

DataFrames

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
...	...	...	...	...	...	...
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

A `DataFrame`

```
0         Andrew Jackson
1      John Quincy Adams
2         Andrew Jackson
3      John Quincy Adams
4         Andrew Jackson
             ...
177           Jill Stein
178         Joseph Biden
179         Donald Trump
180         Jo Jorgensen
181       Howard Hawkins
Name: Candidate, Length: 182, 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)
```

- 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`

- 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

```python
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
```

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
```

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**

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

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

- Say we want to select values in the **`Series`** that satisfy a particular condition:
  1) Apply a boolean condition to the **`Series`**. This creates a **new `Series` of boolean values**.
  2) Index into our **`Series`** using this boolean condition. **`pandas`** will select only the entries in the **`Series`** that satisfy the condition.

```python
s > 0
```

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

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

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

DataFrames of Series!

Typically, we will work with `Series` using the perspective that they are 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
       ...
177    2016
178    2020
179    2020
180    2020
181    2020
Name: Year,
```

The Series "**`Year`**"

```
0           Andrew Jackson
1        John Quincy Adams
2           Andrew Jackson
3        John Quincy Adams
4           Andrew Jackson
              ...
177             Jill Stein
178           Joseph Biden
179           Donald Trump
180           Jo Jorgensen
181         Howard Hawkins
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
...	...	...	...	...	...	...
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

The DataFrame **`elections`**

Creating a `DataFrame`

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`.

Creating a `DataFrame`

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
...	...	...	...	...	...	...
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

The DataFrame **`elections`**

Creating a `DataFrame`

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
...	...	...	...	...	...
2016	Jill Stein	Green	1457226	loss	1.073699
2020	Joseph Biden	Democratic	81268924	win	51.311515
2020	Donald Trump	Republican	74216154	loss	46.858542
2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
2020	Howard Hawkins	Green	405035	loss	0.255731

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

Creating a `DataFrame`

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`.

```
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`

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]})
```

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

	Fruit	Price
0	Strawberry	5.49
1	Orange	3.99

Specify rows of the **`DataFrame`**

Creating a `DataFrame`

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

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

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

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

	Year	Party	Popular vote	Result	%
Candidate					
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

The row labels that constitute an index do not have to be unique.

- Left: The **index** values are all unique and numeric, acting as a row number.
- 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

Modifying Indices

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

Example: Setting the index to the "Party" column.

```
elections.set_index("Party")
```

	Candidate	Year	Popular vote	Result	%
Party					
Democratic-Republican	Andrew Jackson	1824	151271	loss	57.210122
Democratic-Republican	John Quincy Adams	1824	113142	win	42.789878
Democratic	Andrew Jackson	1828	642806	win	56.203927
National Republican	John Quincy Adams	1828	500897	loss	43.796073
Democratic	Andrew Jackson	1832	702735	win	54.574789
...	...	...	...	...	...
Green	Jill Stein	2016	1457226	loss	1.073699
Democratic	Joseph Biden	2020	81268924	win	51.311515
Republican	Donald Trump	2020	74216154	loss	46.858542
Libertarian	Jo Jorgensen	2020	1865724	loss	1.177979
Green	Howard Hawkins	2020	405035	loss	0.255731

Resetting the `Index`

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

```
elections.reset_index()
```

	Candidate	Year	Popular vote	Result	%
Party					
Democratic-Republican	Andrew Jackson	1824	151271	loss	57.210122
Democratic-Republican	John Quincy Adams	1824	113142	win	42.789878
Democratic	Andrew Jackson	1828	642806	win	56.203927
National Republican	John Quincy Adams	1828	500897	loss	43.796073
Democratic	Andrew Jackson	1832	702735	win	54.574789
...	...	...	...	...	...
Green	Jill Stein	2016	1457226	loss	1.073699
Democratic	Joseph Biden	2020	81268924	win	51.311515
Republican	Donald Trump	2020	74216154	loss	46.858542
Libertarian	Jo Jorgensen	2020	1865724	loss	1.177979
Green	Howard Hawkins	2020	405035	loss	0.255731

	Candidate	Year	Party	Popular vote	Result	%
0	Andrew Jackson	1824	Democratic-Republican	151271	loss	57.210122
1	John Quincy Adams	1824	Democratic-Republican	113142	win	42.789878
2	Andrew Jackson	1828	Democratic	642806	win	56.203927
3	John Quincy Adams	1828	National Republican	500897	loss	43.796073
4	Andrew Jackson	1832	Democratic	702735	win	54.574789
...	...	...	...	...	...	...
177	Jill Stein	2016	Green	1457226	loss	1.073699
178	Joseph Biden	2020	Democratic	81268924	win	51.311515
179	Donald Trump	2020	Republican	74216154	loss	46.858542
180	Jo Jorgensen	2020	Libertarian	1865724	loss	1.177979
181	Howard Hawkins	2020	Green	405035	loss	0.255731

Column Names Are Usually Unique!

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`

Sometimes you'll want to extract the list of row and column labels.

```
elections.set_index("Party")
```

For row labels, use `DataFrame.index`:

```
elections.index
```

```
Index(['Democratic-Republican', 'Democratic-Republican', 'Democratic',
       'National Republican', 'Democratic', 'National Republican',
       'Anti-Masonic', 'Whig', 'Democratic', 'Whig',
       ...
       'Constitution', 'Republican', 'Independent', 'Libertarian',
       'Democratic', 'Green', 'Democratic', 'Republican', 'Libertarian',
       'Green'],
      dtype='object', name='Party', length=182)
```

For column labels, use `DataFrame.columns`:

```
elections.columns
```

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

For shape of the `DataFrame` we use `DataFrame.shape`:

```
elections.shape
```

```
(182, 6)
```

The Relationship Between DataFrames, Series, and Indices

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 `Series` Party `Series` % `Series` Year `Series` Result `Series`

	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

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

Lecture 02

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

Extracting Data

One of the most basic tasks for manipulating a `DataFrame` is to extract rows and columns of interest. As we'll see, the large `pandas` API means there are many ways to do things.

Common ways we may want to extract data:

- 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 data manipulation tasks.

`.head` and `.tail`

The simplest scenarios: We want to extract the first or last **`n`** rows from the **`DataFrame`**.

- **`df.head(n)`** will return the first **`n`** rows of the DataFrame **`df`**.
- **`df.tail(n)`** will return 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
...	...	...	...	...	...	...
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

`elections.head(5)`

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

`elections.tail(5)`

	Candidate	Year	Party	Popular vote	Result	%
177	Jill Stein	2016	Green	1457226	loss	1.073699
178	Joseph Biden	2020	Democratic	81268924	win	51.311515
179	Donald Trump	2020	Republican	74216154	loss	46.858542
180	Jo Jorgensen	2020	Libertarian	1865724	loss	1.177979
181	Howard Hawkins	2020	Green	405035	loss	0.255731

Label-based Extraction: `.loc`

A more complex task: We want to extract data with specific column or index labels.

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

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

- We describe "labels" as the bolded text at the top and left of a `DataFrame`.

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
...	...	...	...	...	...	...
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

Row labels

Label-based Extraction: `.loc`

Arguments to `.loc` can be:

- A list.
- A slice (syntax is 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
...	...	...	...	...	...	...
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

Label-based Extraction: `.loc`

Arguments to `.loc` can be:

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

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

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

Label-based Extraction: `.loc`

Arguments to `.loc` can be:

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

```
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`

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
...	...	...	...
177	2016	Jill Stein	loss
178	2020	Joseph Biden	win
179	2020	Donald Trump	loss
180	2020	Jo Jorgensen	loss
181	2020	Howard Hawkins	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

Label-based Extraction: `.loc`

Arguments to `.loc` can be:

- A list.
- A slice (syntax is inclusive of the right hand side of the slice).
- **A 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 string value with row label 0 and column label "Candidate".

Integer-based Extraction: `.iloc`

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

- Example: Grab the 1st, 4th, and 3rd columns of the `DataFrame`.

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

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.

Column integers

Row 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`

Arguments to `.iloc` can be:

- A list.
- A slice (syntax is **exclusive** 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
...	...	...	...	...	...	...
177	2016	Jill Stein	Green	1457226	loss	1.073699
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

Integer-based Extraction: `.iloc`

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

Select the columns at positions 0, 1, and 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

Integer-based Extraction: `.iloc`

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], 0:3]
```

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 *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	Party
0	1824	Andrew Jackson	Democratic-Republican
1	1824	John Quincy Adams	Democratic-Republican
2	1828	Andrew Jackson	Democratic
3	1828	John Quincy Adams	National Republican
4	1832	Andrew Jackson	Democratic
...	...	...	...
177	2016	Jill Stein	Green
178	2020	Joseph Biden	Democratic
179	2020	Donald Trump	Republican
180	2020	Jo Jorgensen	Libertarian
181	2020	Howard Hawkins	Green

Grab all rows of the columns at integers 0 to 2.

Integer-based Extraction: `.iloc`

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`.

We have extracted row integers 1, 2, and 3 from the column at position 1.

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

```
'Andrew Jackson'
```

We've extracted the string value with row position 0 and column position 1.

`.loc` vs `.iloc`

Remember:

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

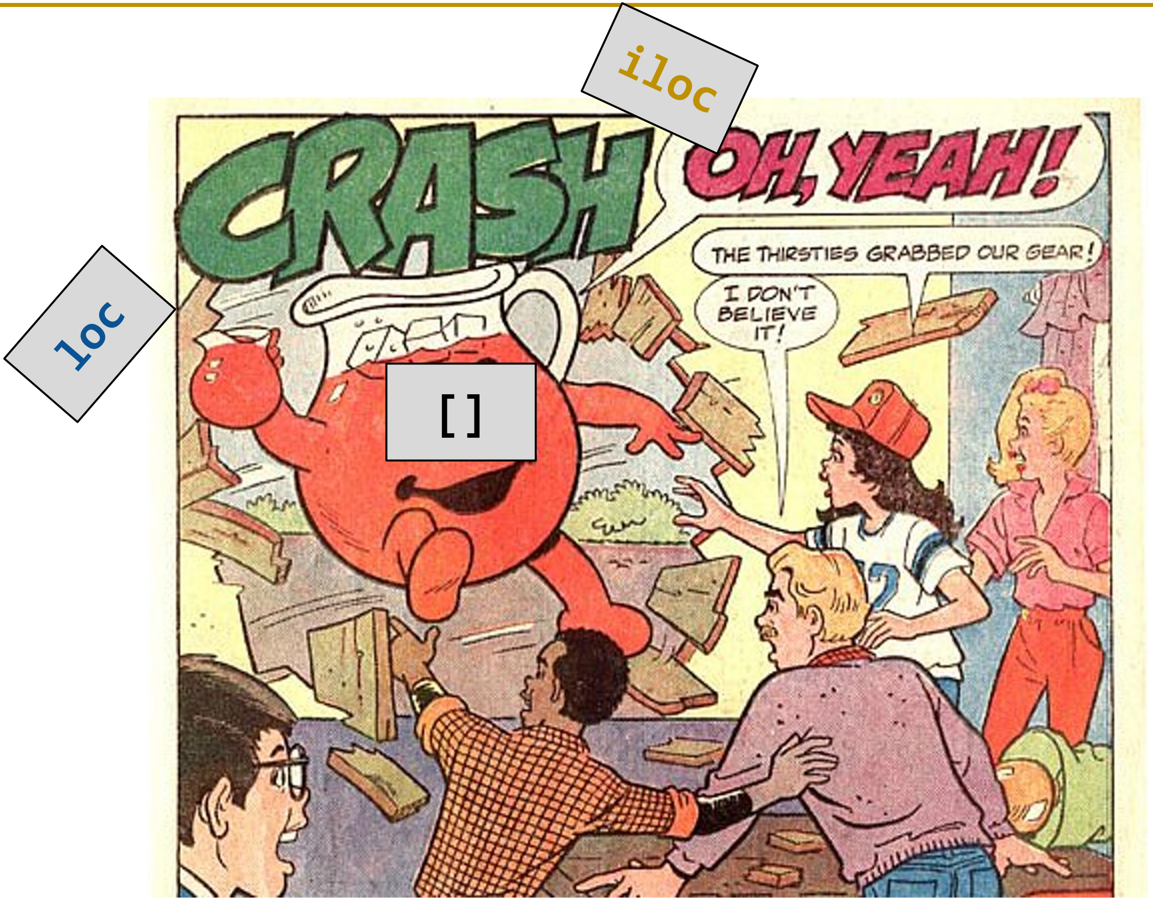
When choosing between `.loc` and `.iloc`, you'll usually choose `.loc`.

- 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` can still be useful.

- Example: If you have a `DataFrame` of movie earnings sorted by earnings, can use `.iloc` to get the median earnings for a given year (index into the middle).

… Just When It Was All Making Sense



Context-dependent Extraction: [ ]

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**.
  - A list of **column labels**.
  - A single **column label**.

That is, `[ ]` is context sensitive.

Let’s see some examples.

Context-dependent Extraction: [ ]

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

- **A slice of row integers**.
- 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: [ ]

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

- A slice of row numbers.
- **A list of column labels.**
- A single column label.

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

	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
...	...	...	...
177	2016	Jill Stein	loss
178	2020	Joseph Biden	win
179	2020	Donald Trump	loss
180	2020	Jo Jorgensen	loss
181	2020	Howard Hawkins	loss

Context-dependent extraction: [ ]

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

- A slice of row numbers.
- 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
                 ...
177               Jill Stein
178             Joseph Biden
179             Donald Trump
180             Jo Jorgensen
181           Howard Hawkins
Name: Candidate, Length: 182, dtype: object
```

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

Why Use [ ]?

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

- Consider the case where we wish to extract the "Candidate" column. It is far simpler to write `elections["Candidate"]` than it is to write `elections.loc[:, "Candidate"]`

In practice, [ ] is often used over `.iloc` and `.loc` in data science work.

Start Work on Shared Notebooks