Join our iClicker class:

https://join.iclicker.com/XSFZ

LECTURE 2

Pandas Bootcamp: Part 1

Intro to useful Pandas functions for Exploratory Data Analysis

CSCI 3022 @ CU Boulder

Maribeth Oscamou

Content credit: Acknowledgments

Meet The Course Team

Isabella Longo

Course Manager

Vincent Bowen

Course Manager

Kevin Buhler

Course Assistant

Grace Mudd

Course Assistant

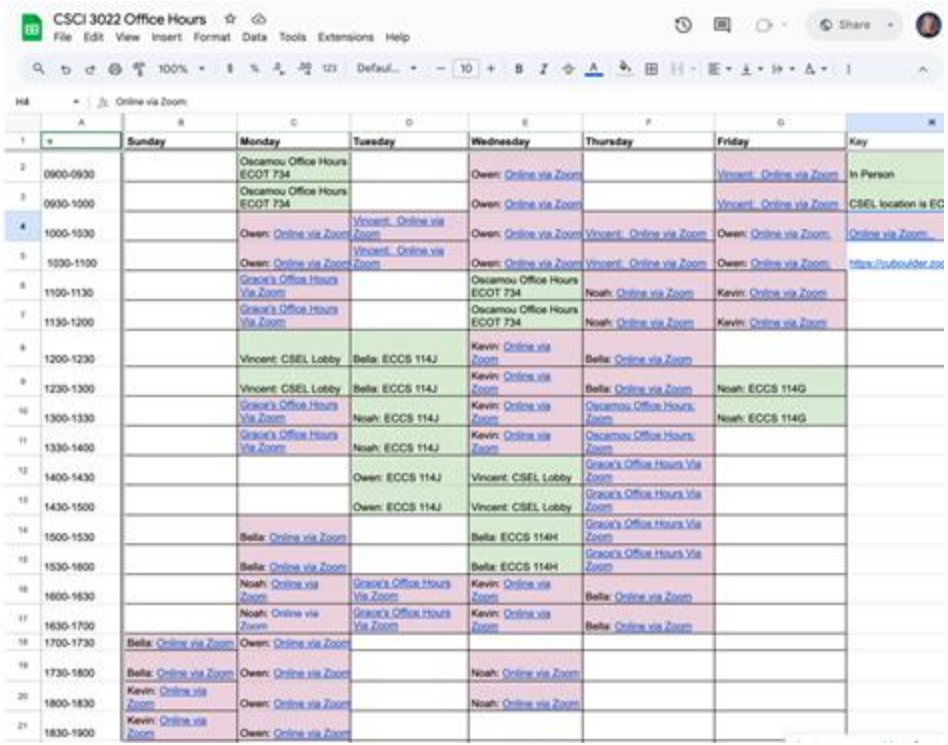
Noah Turner

Course Assistant

Owen Vangermeersch

Course Assistant

Office Hours:



CSCI 3022 Office Hours

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Key
0900-0930		Oscarnou Office Hours ECOT 734		Owen: Online via Zoom		Vincent: Online via Zoom	In Person
0930-1000		Oscarnou Office Hours ECOT 734		Owen: Online via Zoom		Vincent: Online via Zoom	CSEL location is EC
1000-1030		Owen: Online via Zoom	Vincent: Online via Zoom	Owen: Online via Zoom	Vincent: Online via Zoom	Owen: Online via Zoom;	Online via Zoom:
1030-1100		Owen: Online via Zoom	Vincent: Online via Zoom	Owen: Online via Zoom	Vincent: Online via Zoom	Owen: Online via Zoom;	https://ouboulder.zoo
1100-1130		Gracie's Office Hours Via Zoom		Oscarnou Office Hours ECOT 734	Noah: Online via Zoom	Kevin: Online via Zoom	
1130-1200		Gracie's Office Hours Via Zoom		Oscarnou Office Hours ECOT 734	Noah: Online via Zoom	Kevin: Online via Zoom	
1200-1230		Vincent: CSEL Lobby	Bella: ECCS 114J	Kevin: Online via Zoom	Bella: Online via Zoom		
1230-1300		Vincent: CSEL Lobby	Bella: ECCS 114J	Kevin: Online via Zoom	Bella: Online via Zoom	Noah: ECCS 114G	
1300-1330		Gracie's Office Hours Via Zoom	Noah: ECCS 114J	Kevin: Online via Zoom	Oscarnou Office Hours: Zoom	Noah: ECCS 114G	
1330-1400		Gracie's Office Hours Via Zoom	Noah: ECCS 114J	Kevin: Online via Zoom	Oscarnou Office Hours: Zoom		
1400-1430			Owen: ECCS 114J	Vincent: CSEL Lobby	Gracie's Office Hours Via Zoom		
1430-1500			Owen: ECCS 114J	Vincent: CSEL Lobby	Gracie's Office Hours Via Zoom		
1500-1530		Bella: Online via Zoom		Bella: ECCS 114H	Gracie's Office Hours Via Zoom		
1530-1600		Bella: Online via Zoom		Bella: ECCS 114H	Gracie's Office Hours Via Zoom		
1600-1630		Noah: Online via Zoom	Gracie's Office Hours Via Zoom	Kevin: Online via Zoom	Bella: Online via Zoom		
1630-1700		Noah: Online via Zoom	Gracie's Office Hours Via Zoom	Kevin: Online via Zoom	Bella: Online via Zoom		
1700-1730	Bella: Online via Zoom	Owen: Online via Zoom					
1730-1800	Bella: Online via Zoom	Owen: Online via Zoom		Noah: Online via Zoom			
1800-1830	Kevin: Online via Zoom	Owen: Online via Zoom		Noah: Online via Zoom			
1830-1900	Kevin: Online via Zoom	Owen: Online via Zoom					

https://canvas.colorado.edu/courses/117881/pages/hw-slash-office-hours

Jupyter Notebook and LaTeX Troubleshooting and Tips

Make sure before submitting to double check that your PDF includes all of the manually graded questions and plots, and that all code is fully visible in your PDF.

General best practices

- Make sure you have not renamed the .ipynb file. For example, HW 2 must be named hw02.ipynb
- Make sure you haven't inserted any new cells into the notebook.
- Make sure that you're in the 3022 instance of CSEL DataHub. You can do this by signing out of JupyterHub and then re-clicking the link. It should lead you to the page where you have to select the course "3022". The 3022 course has otter-grader installed in it. Other courses in the DataHub may not.
- If you make changes in your HW and run your export cell in your notebook more than once you should first delete the PDF (in the folder where the notebook is) and then re-run. It's possible that the version you submit is an earlier version of your HW.

First fixes to try

- **Save everything, delete the zip and pdf files and shut your browser window. Then open a new browser window and then restart your kernel and run through all of the cells and SAVE the nb before running the final export cell.**
- **As an extension, log out of coding.csel completely (after saving any work), close your browser, then launch a new one. Make sure you have selected CSCI 3022 as your coding environment.**

Latex Issues

- Check that there aren't any spaces after your dollar signs in LaTex:

https://docs.google.com/document/d/1ndr3Wj1PSF5qzlLMaBJznwh6QGeEXjd5TAJ6nf9EJvo/edit?usp=sharing

Course Logistics: Your First Week At A Glance

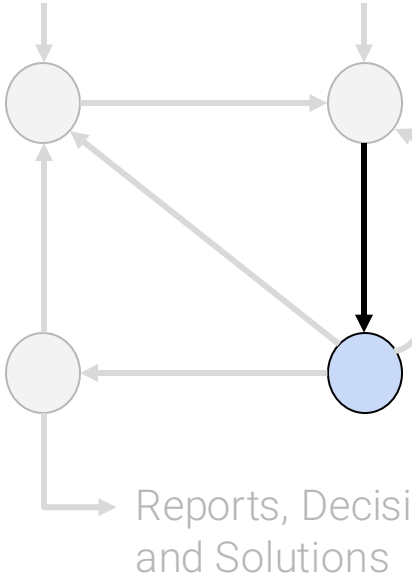
Mon 1/13	Tues 1/14	Wed 1/15	Thurs 1/16	Fri 1/17	
Attend & Participate in Class Office Hours Begin (See Schedule on Canvas)		Attend & Participate in Class	HW 1 Due 11:59pm via Gradescope (Includes Intro to CSCI 3022 Video assignment)	Attend & Participate in Class ~~In Class Quiz (beginning of class)~~	
				HW 2 released	

Getting To Know You:

I'd like to get a chance to be introduced to each of you!

1. **Please sign-up for a 15 min. timeslot (link on first announcement on Canvas and Piazza)** to meet with me during the first couple weeks to briefly introduce yourself and meet a few other classmates.

Plan for first 2 weeks



(Weeks 1 and 2)

EDA, Wrangling, and Data Visualization

Lesson Learning Objectives:

- **Explain the relationship between DataFrames, Series and Indices in Pandas**
- **Understand and implement methods for extracting data: .loc, .iloc, and [ ].**
- **Understand and implement methods for conditional selection in Pandas**
- **Modify columns in a Pandas DataFrame**
- **Manipulate and transform Series and DataFrames using common utility functions:**
  - **(value_counts, describe, info, unique, shape, sort_values)**

Roadmap

Lecture 3, CSCI 3022

- Pandas Bootcamp:
  - Pandas Data Structures
  - Extracting Data
  - Conditional Selection
  - Adding/Modifying/Deleting Columns
  - Useful Utility Functions

Supporting Material:

More with Conditional Selection

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.

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

- We will only consider a tiny portion of this API.

We want you to get familiar with the real world programming practice of… Googling!

- Answers to your questions are often found in the **`pandas`** documentation, Stack Overflow, etc.

Very Useful Resource: Data Wrangling with Pandas Cheat Sheet

Learning Objectives

- **Understand the relationship between DataFrames and Indices in Pandas**
- **Create DataFrames**
- **Manipulate indices**

Pandas Data Structures

- Pandas Bootcamp:
  - **Pandas Data Structures**
  - Extracting Data
  - Conditional Selection
  - Adding/Modifying/Deleting Columns
  - Useful Utility Functions

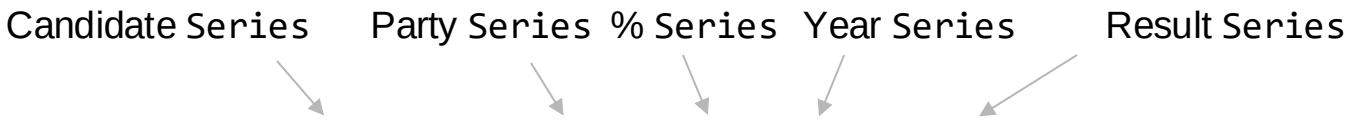
Supporting Material:

More with Conditional Selection

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	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

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

Year	Candidate	Party	%	Result
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

Labels

.

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

	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

Accessing a DataFrame's labels: .index and .columns methods

```
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

```
elections.index
```

```
Index([2024, 2024, 2024, 2024, 2024, 2024, 2024, 2020, 2020, 2020,
       ...
       1836, 1836, 1836, 1832, 1832, 1832, 1828, 1828, 1824, 1824],
      dtype='int64', name='Year', length=189)
```

```
elections.columns
```

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

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

Creating a `DataFrame`

Many approaches exist for creating a `DataFrame`.

- **From a CSV file**. (most common method in our class)
- Using a list and column name(s).
- From a dictionary.
- From a `Series`.

See Supporting Materials for examples of creating DataFrames using the other 3 methods

```
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`

Indices Are Not Necessarily Row Numbers

```
# 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

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")
```

Party	Candidate	Year	Popular vote	Result	%
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()
```

Party	Candidate	Year	Popular vote	Result	%
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

Learning Objective:

Extract data from DataFrames using .loc, .iloc and [ ]

Data Extraction

- Pandas Bootcamp:
  - Pandas Data Structures
  - **Extracting Data**
  - Conditional Selection
  - Adding/Modifying/Deleting Columns
  - Useful Utility Functions

Supporting Material:

More with Conditional Selection

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`

Suppose 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 to extract.

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
...	...	...	...	...	...	...
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`

```
elections.loc[3, "Candidate"]
```

```
'John Quincy Adams'
```

```
elections.loc[[1, 4, 5], ["Party, "Candidate", "Result"]]
```

	Party	Candidate	Result
1	Democratic-Republican	John Quincy Adams	win
4	Democratic	Andrew Jackson	win
5	National Republican	Henry Clay	loss

```
elections.loc[[1, 4, 5],"Year”:"Party" ]
```

	Year	Candidate	Party
1	1824	John Quincy Adams	Democratic-Republican
4	1832	Andrew Jackson	Democratic
5	1832	Henry Clay	National Republican

Integer-based Extraction: `.iloc`

Suppose we want to extract data according to its *position*.

```
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.

	0	1	2	3	4	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
...	...	...	...	...	...	...

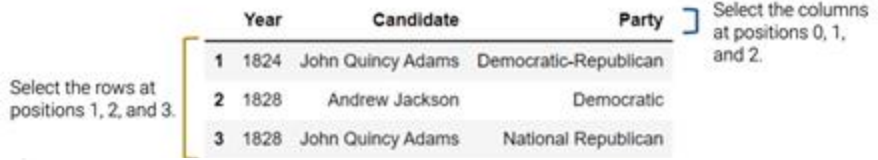
The DataFrame `elections`

Ex:

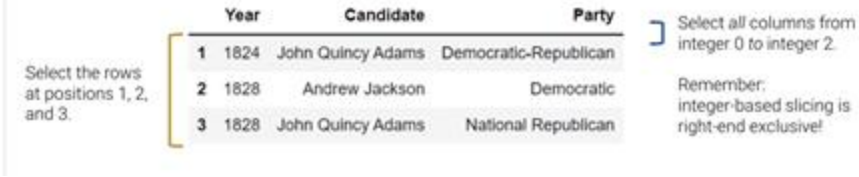
```
elections.iloc[0, 1]
```

```
'Andrew Jackson'
```

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




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



Selection Operators in Pandas

- `loc` performs label-based extraction. 1st argument is rows, 2nd argument is columns:
  `df.loc[row_labels, column_labels]`

- `.iloc` performs integer-based extraction. 1st argument is rows, 2nd argument is columns:
  `df.iloc[row_integers, column_integers]`

- `SHORTCUT OPERATOR FOR 3 COMMON TYPES OF SELECTIONS: []`

  Only takes one argument, which may be:

  - A list of **column labels**. `df[["Year", "Result"]]` (shortcut for `df.loc[:, ["Year", "Result"]]`)
  - A single **column label**. `df["Candidate"]` (shortcut for `df.loc[:, "Candidate"]`)
  - A slice of **row numbers** `df.iloc[3:7,:]` `df[3:7]` (shortcut for

  That is, `[]` is context sensitive.

Lesson Learning Objectives:

- Understand and implement methods for extracting data: .loc, .iloc, and [ ].
- Understand and implement methods for conditional selection in Pandas

Conditional Selection

- Pandas Bootcamp:
  - Data Structures
  - Extracting Data
  - **Conditional Selection**
  - Adding/Modifying/Deleting Columns
  - Useful Utility Functions

Boolean Arrays

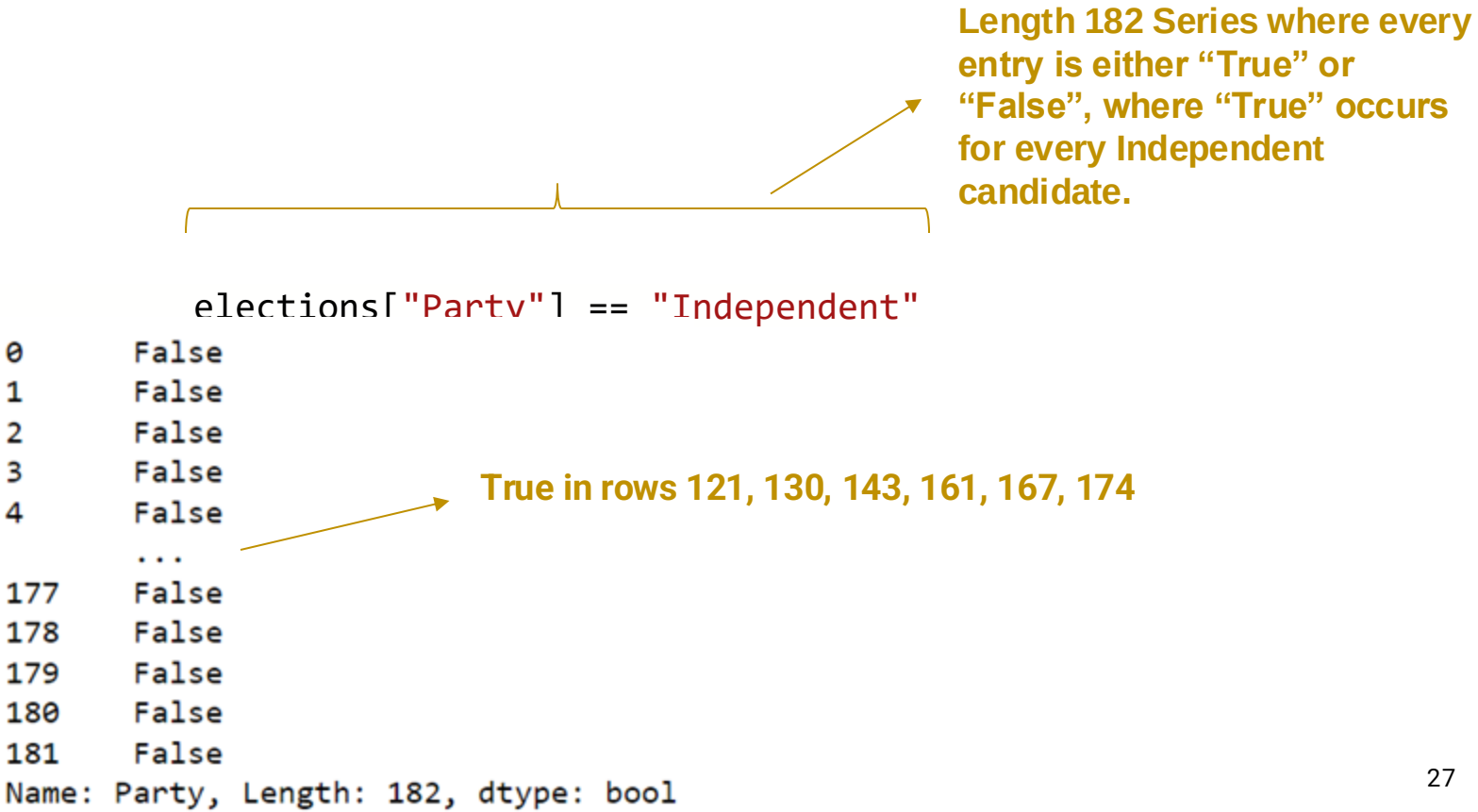
A boolean array is an array that contains only Boolean values (True or False)

```
a = np.array([True, False, True, False, True, False, False, False, False, False])
```

iClicker: What is the output of `a.sum()`

Boolean Array Input

Useful because boolean arrays can be generated by using logical operators on Series.



Boolean Array Input

Useful because boolean arrays can be generated by using logical operators on Series.

Length 182 Series where every entry is "Republican", "Democratic", "Independent", etc.

Length 182 Series where every entry is either "True" or "False", where "True" occurs for every Independent candidate.

```
elections[elections["Party"] == "Independent"]
```

	Year	Candidate	Party	Popular vote	Result	%
121	1976	Eugene McCarthy	Independent	740460	loss	0.911649
130	1980	John B. Anderson	Independent	5719850	loss	6.631143
143	1992	Ross Perot	Independent	19743821	loss	18.956298
161	2004	Ralph Nader	Independent	465151	loss	0.380663
167	2008	Ralph Nader	Independent	739034	loss	0.563842
174	2016	Evan McMullin	Independent	732273	loss	0.539546

Boolean Array Input

Useful because boolean arrays can be generated by using logical operators on Series.

Length 182 Series where every entry is “Republican”, “Democratic”, “Independent”, etc.

Length 182 Series where every entry is either “True” or “False”, where “True” occurs for every Independent candidate.

```
elections[elections["Party"] == "Independent"]
```

	Year	Candidate	Party	Popular vote	Result	%
121	1976	Eugene McCarthy	Independent	740460	loss	0.911649
130	1980	John B. Anderson	Independent	5719850	loss	6.631143
143	1992	Ross Perot	Independent	19743821	loss	18.956298
161	2004	Ralph Nader	Independent	465151	loss	0.380663
167	2008	Ralph Nader	Independent	739034	loss	0.563842
174	2016	Evan McMullin	Independent	732273	loss	0.539546

Boolean Array Input

Can also use **.loc**.

Length 182 Series where every entry is “Republican”, “Democratic”, “Independent”, etc.

Length 182 Series where every entry is either “True” or “False”, where “True” occurs for every Independent candidate.

```
elections.loc[elections["Party"] == "Independent"]
```

	Year	Candidate	Party	Popular vote	Result	%
121	1976	Eugene McCarthy	Independent	740460	loss	0.911649
130	1980	John B. Anderson	Independent	5719850	loss	6.631143
143	1992	Ross Perot	Independent	19743821	loss	18.956298
161	2004	Ralph Nader	Independent	465151	loss	0.380663
167	2008	Ralph Nader	Independent	739034	loss	0.563842
174	2016	Evan McMullin	Independent	732273	loss	0.539546

Boolean Array Input

Boolean Series can be combined using various operators, allowing filtering of results by multiple criteria.

- Example: The & operator.

```
elections[(elections["Result"] == "win") & (elections["%"] < 47)]
```

	Year	Candidate	Party	Popular vote	Result	%
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
20	1856	James Buchanan	Democratic	1835140	win	45.306080
23	1860	Abraham Lincoln	Republican	1855993	win	39.699408
47	1892	Grover Cleveland	Democratic	5553898	win	46.121393
70	1912	Woodrow Wilson	Democratic	6296284	win	41.933422
117	1968	Richard Nixon	Republican	31783783	win	43.565246
140	1992	Bill Clinton	Democratic	44909806	win	43.118485
173	2016	Donald Trump	Republican	62984828	win	46.407862

Bitwise Operators

`&` and `|` are examples of **bitwise operators**. They allow us to apply multiple logical conditions.

If `p` and `q` are boolean arrays or `Series`:

Symbol	Usage	Meaning
`~`	`~p`	Negation of `p`
`|`	`p | q`	`p` OR `q`
`&`	`p & q`	`p` AND `q`
`^`	`p ^ q`	`p` XOR `q` (exclusive or)

Alternatives to Boolean Array Selection

Boolean array selection is a useful tool, but can lead to overly verbose code for complex conditions.

```
elections[(elections["Party"] == "Anti-Masonic")  |
          (elections["Party"] == "American")      |
          (elections["Party"] == "Anti-Monopoly") |
          (elections["Party"] == "American Independent")]
```

Pandas provides **many** alternatives, for example:

- `.query`
- `.isin`
- `.str.startswith`
- `.groupby.filter`

	Year	Candidate	Party	Popular vote	Result	%
6	1832	William Wirt	Anti-Masonic	100715	loss	7.821583
22	1856	Millard Fillmore	American	873053	loss	21.554001
38	1884	Benjamin Butler	Anti-Monopoly	134294	loss	1.335838
115	1968	George Wallace	American Independent	9901118	loss	13.571218
119	1972	John G. Schmitz	American Independent	1100868	loss	1.421524
124	1976	Lester Maddox	American Independent	170274	loss	0.209640
126	1976	Thomas J. Anderson	American	158271	loss	0.194862

Alternatives to Boolean Array Selection

Pandas provides **many** alternatives, for example:

- **`.query`**: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.query.html
- `.isin`
- `.str.startswith`
- `.groupby.filter`

See supporting materials for examples of these other alternatives

```
elections.query('Year >= 2000 and Result == "win"')
```

	Year	Candidate	Party	Popular vote	Result	%
152	2000	George W. Bush	Republican	50456002	win	47.974666
157	2004	George W. Bush	Republican	62040610	win	50.771824
162	2008	Barack Obama	Democratic	69498516	win	53.023510
168	2012	Barack Obama	Democratic	65915795	win	51.258484
173	2016	Donald Trump	Republican	62984828	win	46.407862
178	2020	Joseph Biden	Democratic	81268924	win	51.311515

Query Example

Query has a rich syntax.

- Can access Python variables with the special @ character.

```
parties = ["Republican", "Democratic"]
elections.query('Result == "win" and Party not in @parties')
```

	Year	Candidate	Party	Popular vote	Result	%
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
11	1840	William Henry Harrison	Whig	1275583	win	53.051213
16	1848	Zachary Taylor	Whig	1360235	win	47.309296
27	1864	Abraham Lincoln	National Union	2211317	win	54.951512

Lesson Learning Objectives:

- Understand and implement methods for extracting data: .loc, .iloc, and [ ].
- Understand and implement methods for conditional selection in Pandas
- Add, modify and delete columns in Pandas DataFrames

Transforming Columns

- Pandas Bootcamp:
  - Extracting Data
  - Conditional Selection
  - **Adding/Modifying/Deleting Columns**
  - Useful Utility Functions

Syntax for Adding a Column

Adding a column is easy:

Option 1: Use `[ ]` to reference the desired new column.

a. Assign this column to a `Series` or array of the appropriate length.

```
# Create a Series of the length of each name
babyname_lengths = babynames["Name"].str.len()

# Add a column named "name_lengths" that
# includes the length of each name
babynames["name_lengths"] = babyname_lengths
```

	State	Sex	Year	Name	Count	name_lengths
0	CA	F	1910	Mary	295	4
1	CA	F	1910	Helen	239	5
2	CA	F	1910	Dorothy	220	7
3	CA	F	1910	Margaret	163	8
4	CA	F	1910	Frances	134	7
...	...	...	...	...	...	...
407423	CA	M	2022	Zayvier	5	7
407424	CA	M	2022	Zia	5	3
407425	CA	M	2022	Zora	5	4
407426	CA	M	2022	Zuriel	5	6
407427	CA	M	2022	Zylo	5	4

407428 rows × 6 columns

Option 2: Use `df.assign()`

```
babynames= babynames.assign(name_lengths = babyname_lengths)
```

An Important Note: `DataFrame` Copies

Notice that we *re-assigned* **`babynames`** to an updated value on the previous slide.

```
babynames= babynames.assign(name_lengths = babyname_lengths)
```

By default, **`pandas`** methods create a **copy** of the **`DataFrame`**, without changing the original **`DataFrame`** at all. To apply our changes, we must update our **`DataFrame`** to this new, modified copy.

```
babynames.assign(name_lengths = babyname_lengths)
babynames
```

	State	Sex	Year	Name	Count	Length
0	CA	F	1910	Mary	295	3
1	CA	F	1910	Helen	239	4
2	CA	F	1910	Dorothy	220	6
3	CA	F	1910	Margaret	163	7
4	CA	F	1910	Frances	134	6
...	...	...	...	...	...	...

Our change was not applied!

Syntax for Modifying a Column

Modifying a column is very similar to adding a column.

- Use `[ ]` to reference the existing column.
- Assign this column to a new `Series` or array of the appropriate length.

```
# Modify the "name_lengths" column to be one less than its original value
babynames["name_lengths"] = babynames["name_lengths"]-1
```

	State	Sex	Year	Name	Count	name_lengths
0	CA	F	1910	Mary	295	3
1	CA	F	1910	Helen	239	4
2	CA	F	1910	Dorothy	220	6
3	CA	F	1910	Margaret	163	7
4	CA	F	1910	Frances	134	6
...	...	...	...	...	...	...
407423	CA	M	2022	Zayvier	5	6
407424	CA	M	2022	Zia	5	2
407425	CA	M	2022	Zora	5	3
407426	CA	M	2022	Zuriel	5	5
407427	CA	M	2022	Zylo	5	3

407428 rows × 6 columns

See more examples in supporting materials

Syntax for Renaming a Column

Rename a column using the (creatively named) `.rename()` method.

- `.rename()` takes in a **dictionary** that maps old column names to their new ones.

```
# Rename "name_lengths" to "Length"
babynames = babynames.rename(columns={"name_lengths":"Length"})
```

By default, `pandas` methods create a **copy** of the `DataFrame`, without changing the original `DataFrame` at all. To apply our changes, we must update our `DataFrame` to this new, modified copy.

	State	Sex	Year	Name	Count	Length
0	CA	F	1910	Mary	295	3
1	CA	F	1910	Helen	239	4
2	CA	F	1910	Dorothy	220	6
3	CA	F	1910	Margaret	163	7
4	CA	F	1910	Frances	134	6
...	...	...	...	...	...	...
407423	CA	M	2022	Zayvier	5	6
407424	CA	M	2022	Zia	5	2
407425	CA	M	2022	Zora	5	3
407426	CA	M	2022	Zuriel	5	5
407427	CA	M	2022	Zylo	5	3

407428 rows × 6 columns

Syntax for Dropping a Column (or Row)

Remove columns using the (also creatively named) `.drop` method.

- The `.drop()` method assumes you're dropping a row by default. Use `axis = "columns"` to drop a column instead.

```
babynames = babynames.drop("Length", axis = "columns")
```

	State	Sex	Year	Name	Count	Length
0	CA	F	1910	Mary	295	3
1	CA	F	1910	Helen	239	4
2	CA	F	1910	Dorothy	220	6
3	CA	F	1910	Margaret	163	7
4	CA	F	1910	Frances	134	6
...	...	...	...	...	...	...
407423	CA	M	2022	Zayvier	5	6
407424	CA	M	2022	Zia	5	2
407425	CA	M	2022	Zora	5	3
407426	CA	M	2022	Zuriel	5	5
407427	CA	M	2022	Zylo	5	3

407428 rows × 6 columns

	State	Sex	Year	Name	Count
0	CA	F	1910	Mary	295
1	CA	F	1910	Helen	239
2	CA	F	1910	Dorothy	220
3	CA	F	1910	Margaret	163
4	CA	F	1910	Frances	134
...	...	...	...	...	...
407423	CA	M	2022	Zayvier	5
407424	CA	M	2022	Zia	5
407425	CA	M	2022	Zora	5
407426	CA	M	2022	Zuriel	5
407427	CA	M	2022	Zylo	5

407428 rows × 5 columns

Learning Objective

- **Manipulate and Transform Series and DataFrames using built-in methods**

Useful Utility Functions

- **Useful Utility Functions**
  - `Shape`
  - `Describe`
  - `Info`
  - `Value_counts`
  - `Unique`
  - `Sort_values`

NumPy

Pandas **`Series`** and **`DataFrames`** support a large number of operations, including mathematical operations, so long as the data is numerical. [NumPy reference.]

	State	Sex	Year	Name	Count	Length
0	CA	F	1910	Mary	295	3
1	CA	F	1910	Helen	239	4
2	CA	F	1910	Dorothy	220	6
3	CA	F	1910	Margaret	163	7
4	CA	F	1910	Frances	134	6
...	...	...	...	...	...	...
407423	CA	M	2022	Zayvier	5	6
407424	CA	M	2022	Zia	5	2
407425	CA	M	2022	Zora	5	3
407426	CA	M	2022	Zuriel	5	5
407427	CA	M	2022	Zylo	5	3

407428 rows × 6 columns

```
devon_count = babynames[babynames["Name"] == "Devon"]["Count"]
```

```
19053      5
20481      6
21016      6
24795      6
25157     13
         ..
109089    14
110093     9
112039     6
112768     9
113765     7
Name: Count, Length: 83, dtype: int64
```

```
np.mean(devon_count)
```

20.53012048192771

```
np.max(devon_count)
```

72

Pandas

Pandas provides an enormous number of useful utility functions.

Here are a few we will use frequently in this class:

- `info`
- `value_counts`
- `unique`
- `shape`
- `describe`
- `sort_values`

In the next slide we'll explain value_counts
See supporting materials for info about the rest of these

iClicker poll

Match the following functions to their descriptions

1. `info`
2. `value_counts`
3. `unique`
4. `shape`
5. `describe`

A). returns (number of rows, number of columns) of dataframe

B). Outputs the column integer positions, column labels, data types, memory usage, and the number of non-null cells in each column of a dataframe

C). Counts the number of occurrences of a each unique value in a series or dataframe

D). Generates descriptive statistics of each column

E). Returns unique values of a series

Series.value_counts()

The `Series.value_counts` method counts the number of occurrences of a each unique value in a `Series`.

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

```
0            Joseph Biden
1            Donald Trump
2            Jo Jorgensen
3          Howard Hawkins
4          Darrell Castle
              ...
177          William Wirt
178        Andrew Jackson
179     John Quincy Adams
180        Andrew Jackson
181     John Quincy Adams
Name: Candidate, Length: 182, dtype: object
```

- Return value is also a `Series`.

```
elections["Candidate"].value_counts()
```

```
Norman Thomas          5
Franklin Roosevelt     4
Eugene V. Debs         4
Ralph Nader            4
Andrew Jackson         3
                      ..
Roger MacBride         1
Lester Maddox          1
Gerald Ford            1
Eugene McCarthy        1
Wendell Willkie        1
Name: Candidate, Length: 132, dtype: int64
```

dataframe.value_counts()

The **Dataframe.value_counts** method returns a Series containing the frequency of each distinct row in a **DataFrame**

```
elections[["Year","Party"]]
```

	Year	Party
0	2020	Democratic
1	2020	Republican
2	2020	Libertarian
3	2020	Green
4	2016	Constitution
...	...	...
177	1832	Anti-Masonic
178	1828	Democratic
179	1828	National Republican
180	1824	Democratic-Republican
181	1824	Democratic-Republican

182 rows × 2 columns

- Return value is also a (multi-indexed) **Series**.

```
elections[["Year","Party"]].value_counts()
```

```
Year  Party
1824  Democratic-Republican    2
1836  Whig                     2
1976  Republican               1
1968  Republican               1
1972  American Independent     1
                              ..
1908  Republican               1
      Socialist                1
1912  Democratic               1
      Progressive              1
2020  Republican               1
Length: 180, dtype: int64
```

Series.sort_values()

The `Series.sort_values` method sorts a `Series` (by default, sorted in ascending order)

```
babynames["Name"].sort_values()
```

```
380256      Aadan
362255      Aadan
365374      Aadan
394460    Aadarsh
366561      Aaden
           ...
232144      Zyrah
217415      Zyrah
197519      Zyrah
220674      Zyrah
400761      Zyrus
Name: Name, Length: 400762, dtype: object
```

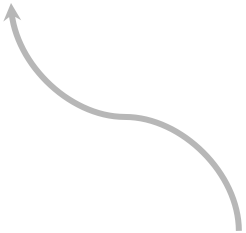
DataFrame.sort_values()

The **DataFrame** version requires an argument specifying the column on which to sort.

```
babynames.sort_values(by = "Count", ascending=False)
```

	State	Sex	Year	Name	Count
263272	CA	M	1956	Michael	8262
264297	CA	M	1957	Michael	8250
313644	CA	M	1990	Michael	8247
278109	CA	M	1969	Michael	8244
279405	CA	M	1970	Michael	8197
...	...	...	...	...	...
159967	CA	F	2002	Arista	5
159966	CA	F	2002	Arisbeth	5
159965	CA	F	2002	Arisa	5
159964	CA	F	2002	Arionna	5
400761	CA	M	2021	Zyrus	5

400762 rows × 5 columns



By default, rows are sorted in ***ascending*** order.

Lesson Learning Objectives:

- Understand and implement methods for extracting data: .loc, .iloc, and [ ].
- Understand and implement methods for conditional selection in Pandas
- Modify columns in a Pandas DataFrames
- Recognize situations where aggregation is useful and implement the correct technique for performing an aggregation

GroupBy

- Pandas Bootcamp:
  - Extracting Data
  - Conditional Selection
  - Adding/Modifying/Deleting Columns
  - **Aggregating Data**

Why Group?

Our goal:

- Group together rows that fall under the same category.
  - For example, group together all rows from the same year.
- Perform an operation that *aggregates* across all rows in the category.
  - For example, sum up the total number of babies born in that year.

Grouping is a powerful tool to

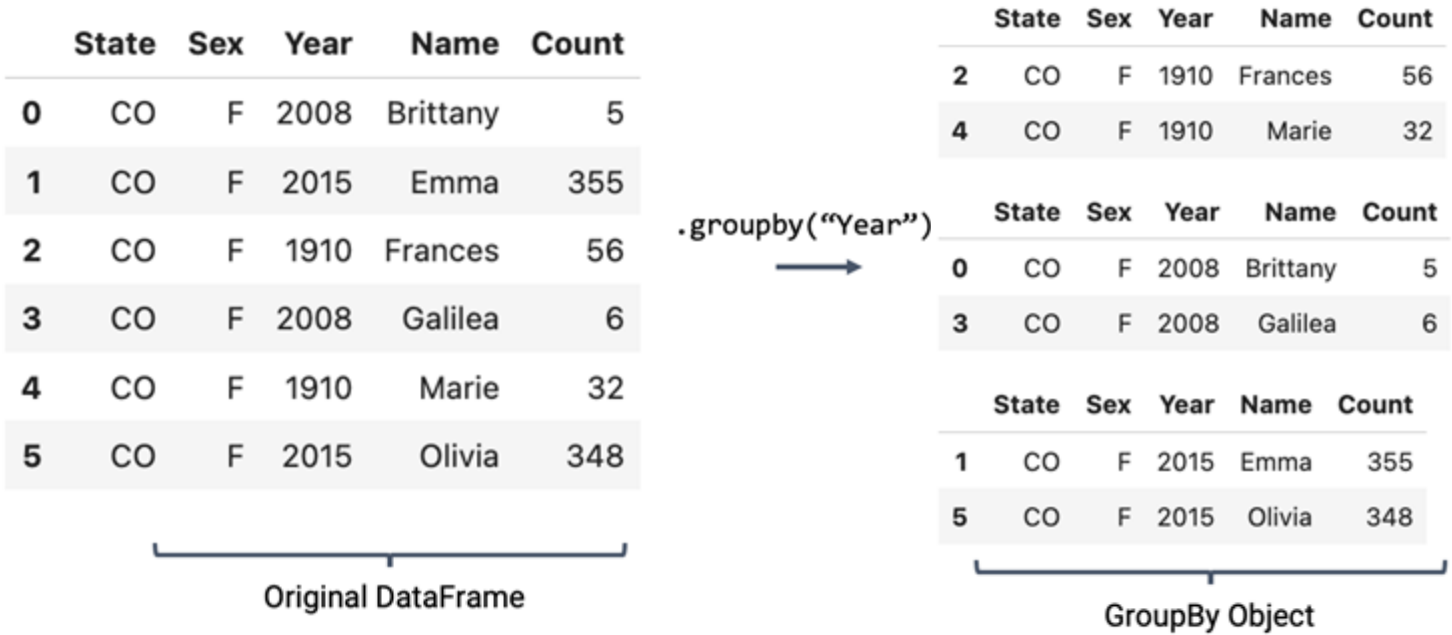
1) perform large operations, all at once

and 2) summarize trends in a dataset.

`.groupby()`

A `.groupby()` operation involves some combination of **splitting the object, applying a function**, and **combining the results**.

- Calling `.groupby()` generates `DataFrameGroupBy` objects → "mini" sub-DataFrames
- Each subframe contains all rows that correspond to the same group (here, a particular year)



	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

	State	Sex	Year	Name	Count
2	CO	F	1910	Frances	56
4	CO	F	1910	Marie	32

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
3	CO	F	2008	Galilea	6

	State	Sex	Year	Name	Count
1	CO	F	2015	Emma	355
5	CO	F	2015	Olivia	348

.groupby().agg()

- We cannot work directly with **DataFrameGroupBy** objects! The diagram below is to help understand what goes on conceptually – in reality, we can't "see" the result of calling `.groupby`.
- Instead, we transform a **DataFrameGroupBy** object back into a DataFrame using `.agg`
  - `.agg` is how we apply an aggregation operation to the data.

```
babynames_temp.groupby("Year")
```

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

Original DataFrame

	State	Sex	Year	Name	Count
2	CO	F	1910	Frances	56
4	CO	F	1910	Marie	32

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
3	CO	F	2008	Galilea	6

	State	Sex	Year	Name	Count
1	CO	F	2015	Emma	355
5	CO	F	2015	Olivia	348

GroupBy Object

```
.agg({"Count":"sum"})
```

or `[["Count"]].agg(sum)`

	Count
Year	
1910	88
2008	11
2015	703

Output DataFrame

Index of output is the col you grouped on

A Note on Nuisance Columns

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

```
babynames_temp.groupby("Year").agg({"Count":"sum"})
babynames_temp.groupby("Year")[["Count"]].agg(sum)
```

Year	Count
1910	88
2008	11
2015	703

If you don't specify the column to aggregate, the aggregation function will be applied to all columns:

```
babynames_temp.groupby("Year").agg(sum)
```

Year	State	Sex	Name	Count
1910	COCO	FF	FrancesMarie	88
2008	COCO	FF	BrittanyGalilea	11
2015	COCO	FF	EmmaOlivia	703

If the aggregation function can't be applied to all columns it results in a `TypeError`.

```
babynames_temp.groupby("Year").agg(mean)
```

```
NotImplementedError                       Traceback (most recent call
File /opt/conda/lib/python3.10/site-packages/pandas/core/groupby/grou
lues)
   1489 try:
-> 1490     result = self.grouper._cython_operation(
   1491         "aggregate",
   1492         values,
   1493         how,
```

Aggregation Functions

What goes inside of `.agg( )`?

- Any function that aggregates several values into one summary value. Common examples:

Built-in Python Functions	NumPy Functions	Built-In `pandas` functions	
`.agg(sum)`	`.agg(np.sum)`	`.agg("sum")`	Returns sum of each col in each group
`.agg(max)`	`.agg(np.max)`	`.agg("max")`	Returns max of each col in each group
`.agg(min)`	`.agg(np.min)`	`.agg("min")`	Returns min of each col in each group
	`.agg(np.mean)`	`.agg("mean")`	Returns mean of each col in each group
		`.agg("first")`	Returns first/last non-null entry in each group for each column
		`.agg("last")`	
		`.agg("count")`	Returns counts of non-null values in each col of each group
		`.agg("size")`	Returns series counting # of rows in each group, including missing values

- You can also define your own function!

The built-in Pandas functions listed above can also be called directly, without the explicit use of `.agg( )`

```
babynames.groupby("Year").mean()
```

groupby.first()

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

Original DataFrame

.groupby("Year")

	State	Sex	Year	Name	Count
2	CO	F	1910	Frances	56
4	CO	F	1910	Marie	32

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
3	CO	F	2008	Galilea	6

	State	Sex	Year	Name	Count
1	CO	F	2015	Emma	355
5	CO	F	2015	Olivia	348

GroupBy Object

.first()

Year	State	Sex	Name	Count
1910	CO	F	Frances	56.0
2008	CO	F	Brittany	5.0
2015	CO	F	Brittany	355.0

The “first” row in each sub-DataFrame depends on how the original DataFrame is sorted:

```
babynames_temp.sort_values(by="Count").groupby("Year").first()
```

Year	State	Sex	Name	Count
1910	CO	F	Marie	32.0
2008	CO	F	Brittany	5.0
2015	CO	F	Olivia	348.0

Aggregating the Same Column Using Multiple Aggregation Functions

```
babynames_temp.groupby("Year").agg({"Count":[max, min, sum]})
```

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

Original DataFrame

	Count		
	max	min	sum
Year			
1910	56	32	88
2008	6	5	11
2015	355	348	703

Aggregating Different Columns Using Different Functions

```
babynames_temp.groupby("Year").agg({"Count":max, "Name":min})
```

	State	Sex	Year	Name	Count
0	CO	F	2008	Brittany	5
1	CO	F	2015	Emma	355
2	CO	F	1910	Frances	56
3	CO	F	2008	Galilea	6
4	CO	F	1910	Marie	32
5	CO	F	2015	Olivia	348

Original DataFrame

Year	Count	Name
1910	56	Frances
2008	6	Brittany
2015	355	Emma

Notice, the column names don't indicate how they've been aggregated.

```
.rename(columns={"Count":"MaxCount", "Name":"MinName"})
```

Year	MaxCount	MinName
1910	56	Frances
2008	6	Brittany
2015	355	Emma

Grouping by Multiple Columns

Suppose we want to build a table showing the total number of babies born of each sex in each year. One way is to **`groupby`** *using both columns* of interest:

```
babynames.groupby(["Year", "Sex"])[["Count"]].agg(sum).head(6)
```

		Count
Year	**Sex**	
1910	**F**	2471
	M	1138
1911	**F**	2343
	M	1307
1912	**F**	3251
	M	2633

Note: Resulting `DataFrame` is multi-indexed. That is, its index has multiple dimensions.

Can reset using

```
.reset_index()
```

	Year	Sex	Count
0	1910	F	2471
1	1910	M	1138
2	1911	F	2343
3	1911	M	1307
4	1912	F	3251
5	1912	M	2633

Supporting Materials

- Supporting Materials
  - More on Conditional Selection
  - More on useful utility functions

Learning Objective:

- **Use Boolean conditions to extract data**

Conditional Selection

Conditional Selection in Pandas

Alternatives to Boolean Array Selection

Boolean array selection is a useful tool, but can lead to overly verbose code for complex conditions.

```
elections[(elections["Party"] == "Anti-Masonic")  |
          (elections["Party"] == "American")      |
          (elections["Party"] == "Anti-Monopoly") |
          (elections["Party"] == "American Independent")]
```

Pandas provides **many** alternatives, for example:

- `.isin`
- `.str.startswith`
- `.query`
- `.groupby.filter` (see next lecture)

	Year	Candidate	Party	Popular vote	Result	%
6	1832	William Wirt	Anti-Masonic	100715	loss	7.821583
22	1856	Millard Fillmore	American	873053	loss	21.554001
38	1884	Benjamin Butler	Anti-Monopoly	134294	loss	1.335838
115	1968	George Wallace	American Independent	9901118	loss	13.571218
119	1972	John G. Schmitz	American Independent	1100868	loss	1.421524
124	1976	Lester Maddox	American Independent	170274	loss	0.209640
126	1976	Thomas J. Anderson	American	158271	loss	0.194862

Alternatives to Boolean Array Selection

Pandas provides **many** alternatives, for example:

- `.isin`
- `.str.startswith`
- `.query`
- `.groupby.filter` (see next lecture)

```
a_parties = ["Anti-Masonic", "American", "Anti-Monopoly", "American Independent"]
elections[elections["Party"].isin(a_parties)]
```

	Year	Candidate	Party	Popular vote	Result	%
6	1832	William Wirt	Anti-Masonic	100715	loss	7.821583
22	1856	Millard Fillmore	American	873053	loss	21.554001
38	1884	Benjamin Butler	Anti-Monopoly	134294	loss	1.335838
115	1968	George Wallace	American Independent	9901118	loss	13.571218
119	1972	John G. Schmitz	American Independent	1100868	loss	1.421524
124	1976	Lester Maddox	American Independent	170274	loss	0.209640
126	1976	Thomas J. Anderson	American	158271	loss	0.194862

Alternatives to Boolean Array Selection

Pandas provides **many** alternatives, for example:

- `.isin`
- **`.str.startswith`**
- `.query`
- `.groupby.filter` (see next lecture)

```
elections[elections["Party"].str.startswith("A")]
```

	Year	Candidate	Party	Popular vote	Result	%
6	1832	William Wirt	Anti-Masonic	100715	loss	7.821583
22	1856	Millard Fillmore	American	873053	loss	21.554001
38	1884	Benjamin Butler	Anti-Monopoly	134294	loss	1.335838
115	1968	George Wallace	American Independent	9901118	loss	13.571218
119	1972	John G. Schmitz	American Independent	1100868	loss	1.421524
124	1976	Lester Maddox	American Independent	170274	loss	0.209640
126	1976	Thomas J. Anderson	American	158271	loss	0.194862

One More Query Example

Query has a rich syntax.

- Can access Python variables with the special @ character.
- We won't cover `query` syntax in detail in our class, but you're welcome to use it.

```
parties = ["Republican", "Democratic"]
elections.query('Result == "win" and Party not in @parties')
```

	Year	Candidate	Party	Popular vote	Result	%
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
11	1840	William Henry Harrison	Whig	1275583	win	53.051213
16	1848	Zachary Taylor	Whig	1360235	win	47.309296
27	1864	Abraham Lincoln	National Union	2211317	win	54.951512

Note: I use `query` frequently in my own work. It's great!

Alternatives to Boolean Array Selection

Pandas provides **many** alternatives, for example:

- `.isin`
- `.str.startswith`
- **`.query`**
- `.groupby.filter` (see next lecture)

```
elections.query('Year >= 2000 and Result == "win"')
```

	Year	Candidate	Party	Popular vote	Result	%
152	2000	George W. Bush	Republican	50456002	win	47.974666
157	2004	George W. Bush	Republican	62040610	win	50.771824
162	2008	Barack Obama	Democratic	69498516	win	53.023510
168	2012	Barack Obama	Democratic	65915795	win	51.258484
173	2016	Donald Trump	Republican	62984828	win	46.407862
178	2020	Joseph Biden	Democratic	81268924	win	51.311515

Alternatives to Direct Boolean Array Selection

`pandas` provides **many** alternatives, for example:

- `.query`
- `.isin`
- `.str.startswith`

```
names = ["Bella", "Alex", "Narges", "Lisa"]
babynames[babynames["Name"].isin(names)]
```

Returns a Boolean `Series` that is `True` when the corresponding name in `babynames` is Bella, Alex, Narges, or Lisa.

```
0         False
1         False
2         False
3         False
4         False
          ...
407423    False
407424    False
407425    False
407426    False
407427    False
Name: Name, Length: 407428, dtype: bool
```

Alternatives to Boolean Array Selection

`pandas` provides **many** alternatives, for example:

- `.isin`
- **`.str.startswith`**
- `.groupby.filter` (stay tuned)

```
babynames[babynames["Name"].str.startswith("N")]
```

Returns a Boolean `Series` that is `True` when the corresponding name in `babynames` starts with "N".

```
0         False
1         False
2         False
3         False
4         False
          ...
407423    False
407424    False
407425    False
407426    False
407427    False
Name: Name, Length: 407428, dtype: bool
```

	State	Sex	Year	Name	Count
76	CA	F	1910	Norma	23
83	CA	F	1910	Nellie	20
127	CA	F	1910	Nina	11
198	CA	F	1910	Nora	6
310	CA	F	1911	Nellie	23
...	...	...	...	...	...
407319	CA	M	2022	Nilan	5
407320	CA	M	2022	Niles	5
407321	CA	M	2022	Nolen	5
407322	CA	M	2022	Noriel	5
407323	CA	M	2022	Norris	5

12229 rows × 5 columns

Learning Objective

- **Manipulate and Transform Series and DataFrames using built-in methods**

Useful Utility Functions

- **Useful Utility Functions**
  - `Shape`
  - `Describe`
  - `Info`
  - `Sample`
  - `Value_counts`
  - `Unique`
  - `Sort_values`

.shape

`babynames`

	State	Sex	Year	Name	Count
0	CO	F	1910	Mary	193
1	CO	F	1910	Helen	112
2	CO	F	1910	Dorothy	87
3	CO	F	1910	Ruth	68
4	CO	F	1910	Margaret	67
...	...	...	...	...	...
114948	CO	M	2022	Wynn	5
114949	CO	M	2022	Zephaniah	5
114950	CO	M	2022	Zephyr	5
114951	CO	M	2022	Zeus	5
114952	CO	M	2022	Zyon	5

114953 rows × 5 columns

- returns the shape of a `DataFrame` or `Series` in the form (number of rows, number of columns)

```
babynames.shape
(114953, 5)
```

DataFrame.describe()

babynames

	State	Sex	Year	Name	Count
0	CO	F	1910	Mary	193
1	CO	F	1910	Helen	112
2	CO	F	1910	Dorothy	87
3	CO	F	1910	Ruth	68
4	CO	F	1910	Margaret	67
...	...	...	...	...	...
114948	CO	M	2022	Wynn	5
114949	CO	M	2022	Zephaniah	5
114950	CO	M	2022	Zephyr	5
114951	CO	M	2022	Zeus	5
114952	CO	M	2022	Zyon	5

114953 rows × 5 columns

babynames.describe()

	Year	Count
count	114953.000000	114953.000000
mean	1981.727106	33.623533
std	30.313836	65.501008
min	1910.000000	5.000000
25%	1960.000000	7.000000
50%	1989.000000	12.000000
75%	2008.000000	30.000000
max	2022.000000	1037.000000

Series.describe()

- A different set of statistics will be reported if `.describe()` is called on a `Series`.

```
babynames["Count"].describe()
```

```
count    114953.000000
mean         33.623533
std          65.501008
min           5.000000
25%           7.000000
50%          12.000000
75%          30.000000
max        1037.000000
Name: Count, dtype: float64
```

```
babynames["Sex"].describe()
```

```
count     114953
unique         2
top            F
freq       63777
Name: Sex, dtype: object
```

.info()

Outputs the column integer positions, column labels, data types, memory usage, and the number of non-null cells in each column

babynames

	State	Sex	Year	Name	Count
0	CO	F	1910	Mary	193
1	CO	F	1910	Helen	112
2	CO	F	1910	Dorothy	87
3	CO	F	1910	Ruth	68
4	CO	F	1910	Margaret	67
...	...	...	...	...	...
114948	CO	M	2022	Wynn	5
114949	CO	M	2022	Zephaniah	5
114950	CO	M	2022	Zephyr	5
114951	CO	M	2022	Zeus	5
114952	CO	M	2022	Zyon	5

114953 rows × 5 columns

```
babynames.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114953 entries, 0 to 114952
Data columns (total 5 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   State   114953 non-null  object
 1   Sex     114953 non-null  object
 2   Year    114953 non-null  int64 
 3   Name    114953 non-null  object
 4   Count   114953 non-null  int64 
dtypes: int64(2), object(3)
memory usage: 4.4+ MB
```

`Series.value_counts()`

The `Series.value_counts` method counts the number of occurrences of a each unique value in a `Series`.

- Return value is also a `Series`.

```
babyname["Name"].value_counts()
```

```
Jean         221
Francis      219
Guadalupe    216
Jessie       215
Marion       213
             ...
Janin          1
Jilliann       1
Jomayra        1
Karess         1
Zyrus          1
Name: Name, Length: 20239, dtype: int64
```

.unique()

The `Series.unique` method returns an array of every unique value in a `Series`.

```
babynames["Name"].unique()
```

```
array(['Mary', 'Helen', 'Dorothy', ..., 'Zyire', 'Zylo', 'Zyrus'],
      dtype=object)
```