

LECTURE 3

Pandas, Part I

Introduction to Pandas syntax, operators, and functions

Data 100/Data 200, Spring 2022 @ UC Berkeley

Josh Hug and Lisa Yan

Goals For This Lecture

Introduce pandas, with emphasis on:

- Key data structures (DataFrames, Series, Indices).
 - Including statistical interpretation of a DataFrame.
- How to index into these structures.
- Other basic operations on these structures.
- Solve some very basic data science problems using Jupyter/pandas.

We'll go through quite a lot of the language without full explanations.

- We expect you to fill in the gaps on homeworks, labs, projects, and through your own experimentation.

I'm assuming in this lecture that you've seen the **Table** class from the Data 8 **datascience** library.

Today might be a little boring (but important!), so please ask questions!

ds100.org/sp22/lecture/lec03/

Data 100

[Home / Schedule](#)

[Syllabus](#)

[Calendar](#)

[Resources](#)

[Staff](#)

LINKS

[Course FAQ](#)

[Datahub](#)

[Gradescope](#)

[Ed](#)

Lecture 3 – Pandas I

Presented by Josh Hug

Content by Josh Hug

- [slides](#)
- [code](#)
- recording (TBD)

Click here to get a copy of the notebook I'll use today.

Filter files by name	
/ ... / lec / lec03 /	
Name	▲
03-pandas-basics-tough-questions.ipynb	
• 03-pandas-basics.ipynb	
annoying_puzzle2.csv	
baby.csv	
• data8_translation_examples.ipynb	
elections.csv	
mottos.csv	

Extra content not covered in lecture

Primary notebook for lecture

Input file for lecture

Input file for data8 translation examples

Unofficial **datascience** -> **pandas** translations

Input file for lecture

Input file for lecture

Note: After lecture, I'll also add a solutions notebook for the tough questions.

A Quick Look at DataFrames

Lecture 03, Data 100 Spring 2022

- **A Quick Look at DataFrames**
- Indexing with loc, iloc, and []
- DataFrames, Series, and Indices
- Conditional Selection
- Handy Utility Functions
- Baby Names (Extra)

Tabular data is one of the most common data formats.

- Will be our primary focus in Data 100.
- In Data 8: The **Table** class of the **datascience** library.
- In Data 100: The **DataFrame** class of the **pandas** library.

Let's see an example in the notebook.

Tabular data is one of the most common data formats.

- In Data 100: The **DataFrame** class of the **pandas** library.

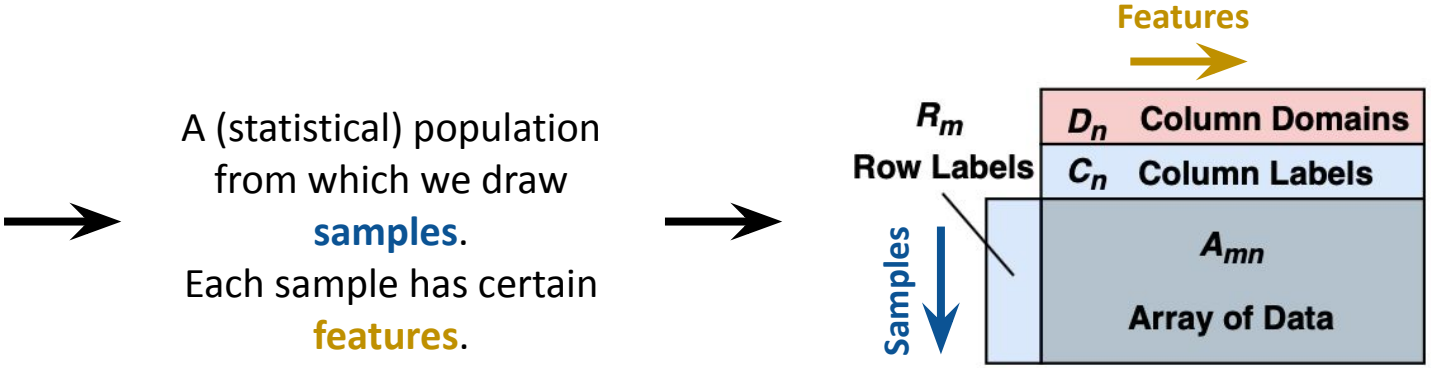
```
elections = pd.read_csv("elections.csv")
```

Example:

	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

182 rows × 6 columns

The World, a Statistician's View (I'm NOT a Statistician)



	Year	Candidate	Party	Popular vote	Result	%
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 generic DataFrame
(from <https://arxiv.org/abs/2001.00888>)

Here, our population is a census of all major party candidates since 1824.

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>

- Compare with the **Table** class from Data8: <http://data8.org/datascience/tables.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.

With that warning, let's dive in.

Indexing with loc, iloc, and []

Lecture 03, Data 100 Spring 2022

- A Quick Look at DataFrames
- **Indexing with loc, iloc, and []**
- DataFrames, Series, and Indices
- Conditional Selection
- Handy Utility Functions
- Baby Names (Extra)

Very Basic Indexing Example

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.

For example, consider the `loc` operator, which we'll learn about shortly.

```
elections.loc[0:4]
```

	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

`loc` is not a function! It's an operator. We'll see why in the next lecture.

Example Methods in the API

The Pandas library has a lot of “syntactic sugar”: Methods that are useful and lead to concise code, but not absolutely necessary for the library to function.

- Examples: `.head` and `.tail`.

```
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

Equivalent to `elections.loc[0:4]`

Example Methods in the API

The Pandas library has a lot of “syntactic sugar”: Methods that are useful and lead to concise code, but not absolutely necessary for the library to function.

- Examples: `.head` and `.tail`.

```
elections.tail(5)
```

	Year	Candidate	Party	Popular vote	Result	%
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

Equivalent to `elections.loc[177:]`

Very Basic Indexing Example

`loc` also lets us specify the columns that we want as a second argument.

```
elections.loc[0:4, "Year":"Party"]
```

	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

Fundamentally **loc** selects items by **label**.

- The labels are the **bolded** text to the top and left of our dataframe.
- Row labels shown: **177, 178, 179, 180, 181**
- Column labels: **Year, Candidate, Party, Popular vote, Result, %**

	Year	Candidate	Party	Popular vote	Result	%
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

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	%
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

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"]]
```

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

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": "%"]
```

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

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?

More on that shortly!

As we saw earlier, you can omit the second argument if you want all columns.

If you want all rows, but only some columns, you can use `:` for the left argument.

```
elections.loc[:, ["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

Pandas also supports another operator called **i**loc.

Fundamentally **i**loc selects items by **number**.

- Row numbers are 0 through 181 (in this example, same as labels!).
- Column numbers are 0 through 5.

	Year	Candidate	Party	Popular vote	Result	%
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

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	%
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

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]]
```

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

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]
```

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

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 ugly. Will discuss soon!

And just loc, if you want all rows, but only some columns, you can use `:` for the left argument.

```
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

Why Use loc vs. iloc?

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

- Safer: If the order of columns gets shuffled in a public database, your code still works.
- Legible: 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



Selection operators:

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

[] only takes one argument, which may be:

- **A slice of row numbers.**
- 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

[] 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"]].tail(5)
```

	Year	Candidate	Result
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

[] only takes one argument, which may be:

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

```
elections["Candidate"].tail(5)
```

```
177      Jill Stein
178      Joseph Biden
179      Donald Trump
180      Jo Jorgensen
181      Howard Hawkins
Name: Candidate, dtype: object
```

Two more slides, then we'll talk about the ugly formatting when we have just one column!

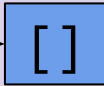
Annoying Puzzle to Test Your Knowledge


```
weird = pd.DataFrame({1:["topdog","botdog"], "1":["topcat","botcat"]})  
weird
```

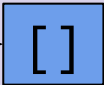
	1	1
0	topdog	topcat
1	botdog	botcat

Try to predict the output of the following:

- `weird[1]`
- `weird["1"]`
- `weird[1:]`

slice of **row numbers** →  → DataFrame
(Multiple) Row Selection

list of **column labels** →  → DataFrame
Multiple Column Selection

single **column label** →  → Series
Single Column Selection


Annoying Puzzle to Test Your Knowledge


```
weird = pd.DataFrame({1:["topdog","botdog"], "1":["topcat","botcat"]})  
weird
```

	1	1
0	topdog	topcat
1	botdog	botcat

Try to predict the output of the following:

- `weird[1]`
- `weird["1"]`
- `weird[1:]`

slice of **row numbers** →  → DataFrame
(Multiple) Row Selection

list of **column labels** →  → DataFrame
Multiple Column Selection

single **column label** →  → Series
Single Column Selection

Let's check our answers in the notebook.

When Should We Use []?

Selection operators:

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

In practice, we'll use **[]** a lot in this course, especially when selecting columns.

- Syntax for **[]** is more concise for many common uses cases!
- **[]** is much more common in real world practice than **loc**.

Note, there is yet another form of indexing called “dot notation” which is a bit of a hack but some people use: <https://www.dataschool.io/pandas-dot-notation-vs-brackets/>

DataFrames, Series, and Indices

Lecture 03, Data 100 Spring 2022

- A Quick Look at DataFrames
- Indexing with loc, iloc, and []
- **DataFrames, Series, and Indices**
- Conditional Selection
- Handy Utility Functions
- Baby Names (Extra)

Earlier, we saw that if we request a single column, our notebook outputs the data in a weird ugly format.

- This is because the returned object is a **Series**, not a **DataFrame**.
- Jupyter renders **Series** in a flatter uglier format.

```
elections["Candidate"].tail(5)
```

```
177      Jill Stein
178      Joseph Biden
179      Donald Trump
180      Jo Jorgensen
181      Howard Hawkins
Name: Candidate, dtype: object
```

```
type(elections)
```

```
pandas.core.frame.DataFrame
```

```
type(elections["Candidate"])
```

```
pandas.core.series.Series
```

Pandas Data Structures

There are three fundamental data structures in pandas:

- **Data Frame**: 2D data tabular data.
- **Series**: 1D data. I usually think of it as columnar data.
- **Index**: A sequence of row labels.

Data Frame

	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

Series

```
0    Obama
1    McCain
2    Obama
3    Romney
4    Clinton
5    Trump
Name: Candidate, dtype: object
```

Index

The Relationship Between Data Frames, Series, and Indices

We can think of a **Data Frame** 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

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

Indices Are Not Necessarily Row Numbers

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

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

```
mottos = pd.read_csv("mottos.csv", index_col = "State")
mottos.loc["Alabama":"California"]
```

	Motto	Translation	Language	Date Adopted
State				
Alabama	Audemus jura nostra defendere	We dare defend our rights!	Latin	1923
Alaska	North to the future	—	English	1967
Arizona	Ditat Deus	God enriches	Latin	1863
Arkansas	Regnat populus	The people rule	Latin	1907
California	Eureka (Εὕρηκα)	I have found it	Greek	1849

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

Column Names Are Usually Unique!

Column names in Pandas are almost always unique!

- Example: Really shouldn't have two columns named "Candidate".
- You can force duplicate columns into existence if you want.

	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

Getting a DataFrame Rather than a Series

Suppose we want our single column back as a **DataFrame**.

```
elections["Candidate"].tail(5)
```

```
177      Jill Stein
178    Joseph Biden
179    Donald Trump
180    Jo Jorgensen
181   Howard Hawkins
Name: Candidate, dtype: object
```

Getting a DataFrame Rather than a Series

Suppose we want our single column back as a **DataFrame**. Two approaches:

- Use `Series.to_frame()`.
- Provide a list containing the single column of interest.
 - Looks like double braces! But this is just a list provided to the `[]` operator.

```
elections["Candidate"].tail(5).to_frame()
```

Candidate

177 Jill Stein

178 Joseph Biden

179 Donald Trump

180 Jo Jorgensen

181 Howard Hawkins

```
elections[["Candidate"]].tail(5)
```

Candidate

177 Jill Stein

178 Joseph Biden

179 Donald Trump

180 Jo Jorgensen

181 Howard Hawkins

Occasionally Useful Fact: Retrieving Row and Column Labels

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

For row labels, use `DataFrame.index`:

`mottos.index`

```
Index(['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado',  
      'Connecticut', 'Delaware', 'Florida', 'Georgia', 'Hawaii', 'Idaho',  
      'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana',  
      'Maine', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota',  
      'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'Nevada',  
      'New Hampshire', 'New Jersey', 'New Mexico', 'New York',  
      'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon',  
      'Pennsylvania', 'Rhode Island', 'South Carolina', 'South Dakota',  
      'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia', 'Washington',  
      'West Virginia', 'Wisconsin', 'Wyoming'],  
      dtype='object', name='State')
```

For column labels, use `DataFrame.columns`:

`mottos.columns`

```
Index(['Motto', 'Translation', 'Language', 'Date Adopted'], dtype='object')
```

Conditional Selection

Lecture 03, Data 100 Spring 2022

- A Quick Look at DataFrames
- Indexing with loc, iloc, and []
- DataFrames, Series, and Indices
- **Conditional Selection**
- Handy Utility Functions
- Baby Names (Extra)

Boolean Array Input

Yet another input type supported by `loc` and `[]` is the boolean array.

```
elections[[False, False, False, False, False, False,
            False, False, True, False, False,
            True, False, False, False, True,
            False, False, False, False, False,
            False, False, True]]
```

Entry number 7

	Candidate	Party	%	Year	Result
7	Clinton	Democratic	43.0	1992	win
10	Clinton	Democratic	49.2	1996	win
14	Bush	Republican	47.9	2000	win
22	Trump	Republican	46.1	2016	win



Note: Here elections is a DataFrame with 23 rows.

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 either “True” or “False”, where “True” occurs for every Independent candidate.

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

```
0      False
1      False
2      False
3      False
4      False
...
177     False
178     False
179     False
180     False
181     False
```

True in rows 121, 130, 143, 161, 167, 174

```
Name: Party, Length: 182, dtype: bool
```

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.

```
0    False
1    False
2    False
3    False
4    False
...
177   False
178   False
179   False
180   False
181   False
Name: Party, Length: 182, dtype: bool
```

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.

```
0    False
1    False
2    False
3    False
4    False
...
177   False
178   False
179   False
180   False
181   False
Name: Party, Length: 182, dtype: bool
```

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

- Example: The & operator.
- Lab 2 covers more such operators.

```
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



Annoying Question Challenge #2

Which of the following pandas statements returns a DataFrame of the first 3 Candidate names only for candidates that won with more than 50% of the vote.

```
elections2.iloc[[0, 3, 5], [0, 3]]
```

```
elections2.loc[[0, 3, 5], "Candidate":"Year"]
```

```
elections2.loc[elections["%"] > 50, ["Candidate", "Year"]].head(3)
```

```
elections2.loc[elections["%"] > 50, ["Candidate", "Year"]].iloc[0:2, :]
```

	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
1	Carter	Democratic	41.0	1980	loss
2	Anderson	Independent	6.6	1980	loss
3	Reagan	Republican	58.8	1984	win
4	Mondale	Democratic	37.6	1984	loss
5	Bush	Republican	53.4	1988	win
6	Dukakis	Democratic	45.6	1988	loss



	Candidate	Year
0	Reagan	1980
3	Reagan	1984
5	Bush	1988

Annoying Question Challenge #2

Which of the following pandas statements returns a DataFrame of the first 3 Candidate names only for candidates that won with more than 50% of the vote.

`elections2.iloc[[0, 3, 5], [0, 3]]`

~~`elections2.loc[[0, 3, 5], "Candidate":"Year"]`~~

`elections2.loc[elections["%"] > 50, ["Candidate", "Year"]].head(3)`

~~`elections2.loc[elections["%"] > 50, ["Candidate", "Year"]].iloc[0:2, :]`~~

	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
1	Carter	Democratic	41.0	1980	loss
2	Anderson	Independent	6.6	1980	loss
3	Reagan	Republican	58.8	1984	win
4	Mondale	Democratic	37.6	1984	loss
5	Bush	Republican	53.4	1988	win
6	Dukakis	Democratic	45.6	1988	loss



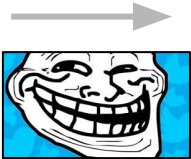
	Candidate	Year
0	Reagan	1980
3	Reagan	1984
5	Bush	1988

Note on Exam Problems

Q: Are you going to put horrible problems like these on the exam?

A: Technically such problems would be in scope, but it's very unlikely they'll be this nitpicky.

	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
1	Carter	Democratic	41.0	1980	loss
2	Anderson	Independent	6.6	1980	loss
3	Reagan	Republican	58.8	1984	win
4	Mondale	Democratic	37.6	1984	loss
5	Bush	Republican	53.4	1988	win
6	Dukakis	Democratic	45.6	1988	loss



	Candidate	Year
0	Reagan	1980
3	Reagan	1984

```
elections.loc[elections["%"] > 50, ["Candidate", "Year"]].iloc[0:2, :]
```


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 lecture 4)

	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 lecture 4)

```
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 lecture 4)

```
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.
- Warning that your TAs are unlikely to be familiar with **query**.

```
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 lecture 4)

```
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

Handy Utility Functions

Lecture 03, Data 100 Spring 2022

- A Quick Look at DataFrames
- Indexing with loc, iloc, and []
- DataFrames, Series, and Indices
- Conditional Selection
- **Handy Utility Functions**
- Baby Names (Extra)

Numpy and Built-in Function Support

Pandas Series and DataFrames support a large number of operations, including mathematical operations, so long as the data is numerical.

```
winners = elections.query('Result == "win"')['%']  
  
1      42.789878  
2      56.203927  
4      54.574789  
...  
162    53.023510  
168    51.258484  
173    46.407862  
178    51.311515  
Name: %, dtype: float64
```

`np.mean(winners)`
51.711492943

`max(winners)`
61.34470329

In addition to its rich syntax for indexing and support for other libraries (numpy, built-in functions), Pandas provides an enormous number of useful utility functions. Today, we'll discuss:

- `size/shape`
- `describe`
- `sample`
- `value_counts`
- `uniques`
- `sort_values`

	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

182 rows × 6 columns

`elections.size`
`1092`

`elections.shape`
`(182, 6)`



	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

182 rows × 6 columns

elections.describe()

	Year	Popular vote	%
count	182.000000	1.820000e+02	182.000000
mean	1934.087912	1.235364e+07	27.470350
std	57.048908	1.907715e+07	22.968034
min	1824.000000	1.007150e+05	0.098088
25%	1889.000000	3.876395e+05	1.219996
50%	1936.000000	1.709375e+06	37.677893
75%	1988.000000	1.897775e+07	48.354977
max	2020.000000	8.126892e+07	61.344703



If you want a DataFrame consisting of a random selection of rows, you can use the sample method.

- By default, *it is without replacement*. Use `replace=True` for **replacement**.
- Naturally, can be chained with other methods and operators (`query`, `iloc`, etc).

```
elections.sample(5).iloc[:, 0:2]
```

	Year	Candidate
178	2020	Joseph Biden
51	1896	Joshua Levering
41	1884	John St. John
165	2008	Cynthia McKinney
143	1992	Ross Perot

```
elections.query('Year == 2000')  
    .sample(4, replace = True)  
    .iloc[:, 0:2]
```

	Year	Candidate
151	2000	Al Gore
151	2000	Al Gore
154	2000	Pat Buchanan
155	2000	Ralph Nader

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

- Return value is also a Series.

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

```
Norman Thomas      5
Ralph Nader        4
Franklin Roosevelt  4
Eugene V. Debs     4
Andrew Jackson     3
..
Silas C. Swallow   1
Alton B. Parker    1
John G. Woolley    1
Joshua Levering    1
Howard Hawkins     1
Name: Candidate, Length: 132, dtype: int64
```

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

```
elections["Party"].unique()
```

```
array(['Democratic-Republican', 'Democratic', 'National Republican',  
      'Anti-Masonic', 'Whig', 'Free Soil', 'Republican', 'American',  
      'Constitutional Union', 'Southern Democratic',  
      'Northern Democratic', 'National Union', 'Liberal Republican',  
      'Greenback', 'Anti-Monopoly', 'Prohibition', 'Union Labor',  
      'Populist', 'National Democratic', 'Socialist', 'Progressive',  
      'Farmer-Labor', 'Communist', 'Union', 'Dixiecrat',  
      "States' Rights", 'American Independent', 'Independent',  
      'Libertarian', 'Citizens', 'New Alliance', 'Taxpayers',  
      'Natural Law', 'Green', 'Reform', 'Constitution'], dtype=object)
```

The `DataFrame.sort_values` and `Series.sort_values` methods sort a `DataFrame` (or `Series`).

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

75      Aaron S. Watkins
27      Abraham Lincoln
23      Abraham Lincoln
108     Adlai Stevenson
105     Adlai Stevenson
...
19      Winfield Scott
37  Winfield Scott Hancock
74      Woodrow Wilson
70      Woodrow Wilson
16      Zachary Taylor
Name: Candidate, Length: 182, dtype: object
```


The `DataFrame.sort_values` and `Series.sort_values` methods sort a `DataFrame` (or `Series`).

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

`elections.sort_values("%", ascending = False)`

	Year	Candidate	Party	Popular vote	Result	%
114	1964	Lyndon Johnson	Democratic	43127041	win	61.344703
91	1936	Franklin Roosevelt	Democratic	27752648	win	60.978107
120	1972	Richard Nixon	Republican	47168710	win	60.907806
79	1920	Warren Harding	Republican	16144093	win	60.574501
133	1984	Ronald Reagan	Republican	54455472	win	59.023326
...
165	2008	Cynthia McKinney	Green	161797	loss	0.123442
148	1996	John Hagelin	Natural Law	113670	loss	0.118219
160	2004	Michael Peroutka	Constitution	143630	loss	0.117542
141	1992	Bo Gritz	Populist	106152	loss	0.101918
156	2004	David Cobb	Green	119859	loss	0.098088

Baby Names (Extra)

Lecture 03, Data 100 Spring 2022

- A Quick Look at DataFrames
- Indexing with loc, iloc, and []
- DataFrames, Series, and Indices
- Conditional Selection
- Handy Utility Functions
- **Baby Names (Extra)**

Wrapping Up

Time permitting, let's try answering some questions about a list of California baby names.

I'll start with my own goal, and if we still somehow have more time, I will take suggested goals from you and try to write code to achieve your goals.