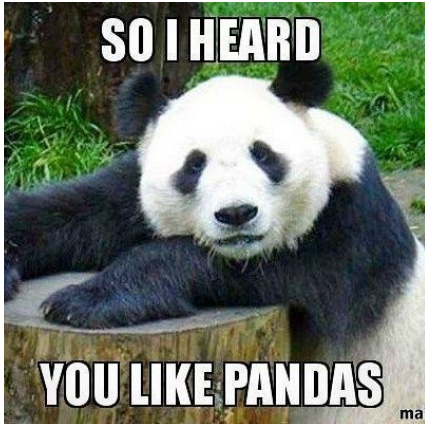


LECTURE 3

Pandas For EDA: Day 2

Overview of useful Pandas functions for Exploratory Data Analysis

CSCI 3022 @ CU Boulder

Maribeth Oscamou

Content credit: Acknowledgments

Course Logistics: Your Second Week At A Glance

Mon 1/22	Tues 1/23	Wed 1/24	Thurs 1/25	Fri 1/26	Sat 1/27	
	(Optional): Attend Notebook Discussion with our TA (5-6pm Zoom)	Attend & Participate in Class	~~HW 2 Due 11:59pm via Gradescope~~	Quiz 1: Scope - Prerequisites, Syllabus & HW 1 Attend & Participate in Class HW 2 Due: 11:59pm via Gradescope		
			Graded HW 1 posted	HW 3 Released Discussion NB 3 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 on Zoom during the first couple weeks to briefly introduce yourself and meet a few other classmates.

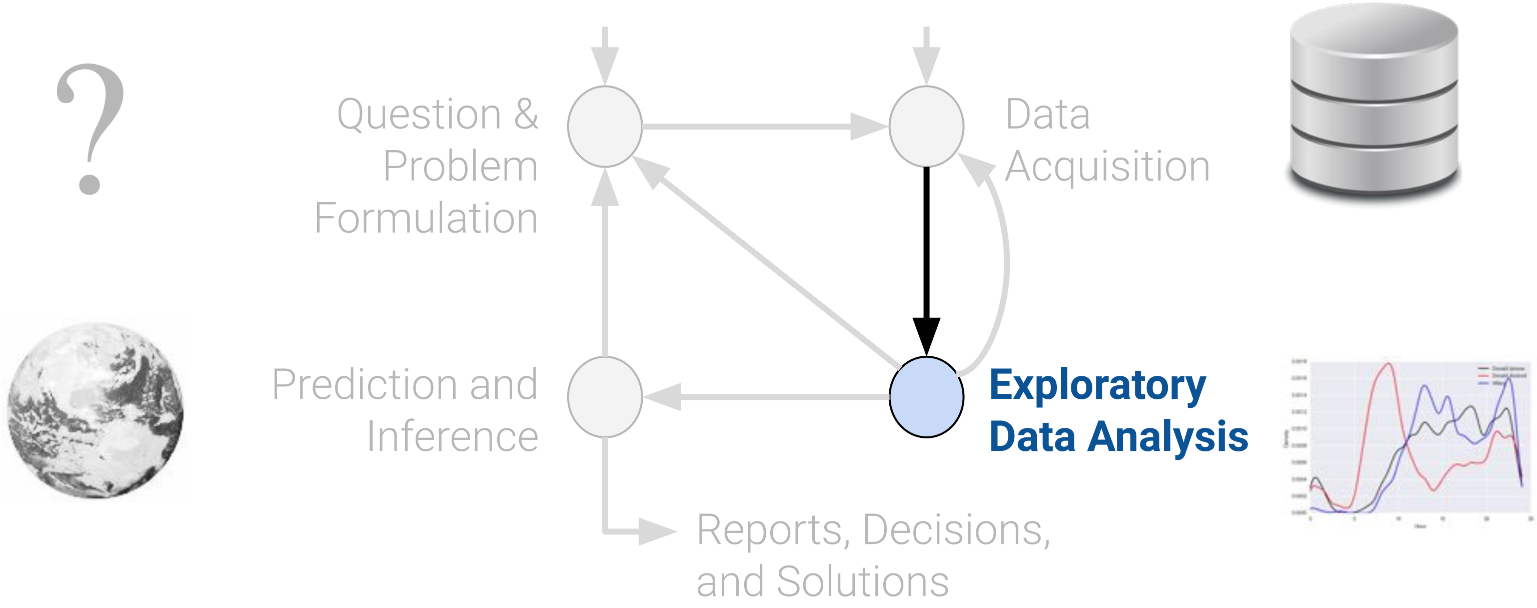
HW 2: Due Date Extended to Friday because link wasn’t live until Saturday

https://canvas.colorado.edu/courses/101142/assignments/

iClicker Poll:

What questions do you have about the syllabus and/or class format?

Plan for first 2 weeks



Question & Problem Formulation

Data Acquisition

Exploratory Data Analysis

Prediction and Inference

Reports, Decisions, and Solutions

(Weeks 1 and 2)

EDA, Wrangling, and Data Visualization

Lesson Learning Objectives:

- Identify 5 key data properties to consider when doing Exploratory Data Analysis
- Understand methods for extracting data: .loc, .iloc, and [].
- Use conditional selection in Pandas
- Add/modify and delete columns

Roadmap

Lecture 3, CSCI 3022

- Exploratory Data Analysis - Key properties
- Pandas Bootcamp:
    - Extracting Data
    - Conditional Selection
    - Adding/Modifying/Deleting Columns

Key Data Properties to Consider in EDA

Structure -- the “shape” of a data file

Granularity -- what does each record represent?

Scope -- how (in)complete is the data

Temporality -- how is the data situated in time

Faithfulness -- how well does the data capture “reality”

Lesson Learning Objectives:

- Identify 5 key data properties to consider when doing Exploratory Data Analysis
- Understand methods for extracting data: .loc, .iloc, and [].
-

Pandas Bootcamp

Lecture 03, CSCI 3022

- Exploratory Data Analysis - Key properties
- Pandas Bootcamp:
  - **Extracting Data**
  - Conditional Selection
  - Adding/Modifying/Deleting Columns
  - Grouping
  - Joining

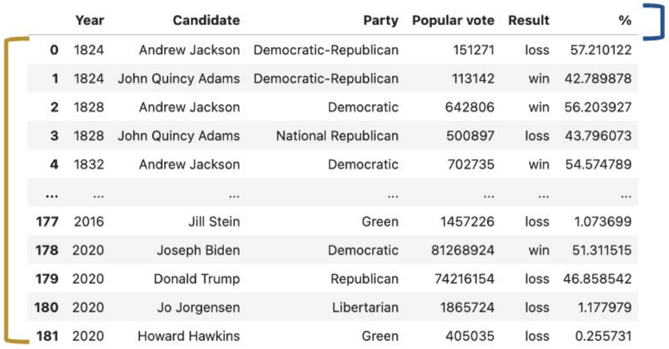
Label-based Extraction: `.loc`

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

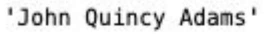


	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832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

Column labels

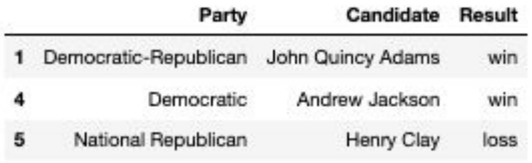
The DataFrame `elections`

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



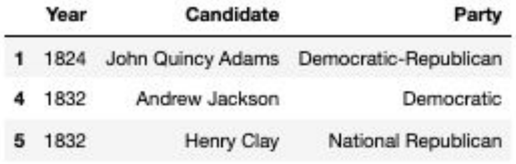
```
'John Quincy Adams'
```

```
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

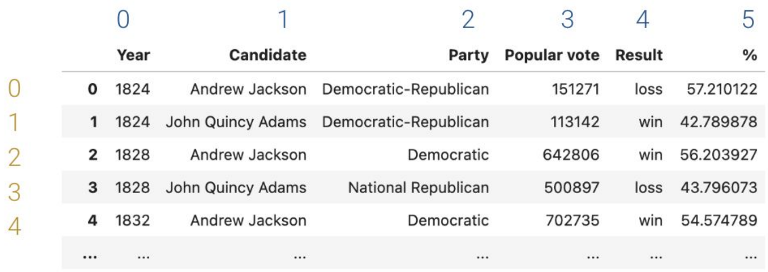
Integer-based Extraction: `.iloc`

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	**0**	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	**1**	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	**2**	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	**3**	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	**4**	1832	Andrew Jackson	Democratic	702735	win	54.574789
	...	...	...	...	...	...	...

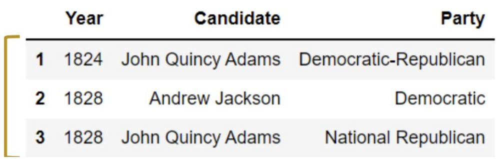
The DataFrame `elections`

Ex:

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

```
'Andrew Jackson'
```

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

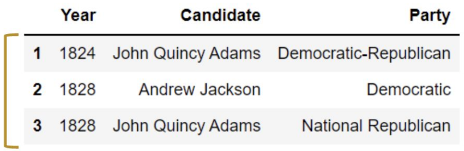


	Year	**Candidate**	**Party**
1	1824	John Quincy Adams	Democratic-Republican
2	1828	Andrew Jackson	Democratic
3	1828	John Quincy Adams	National Republican

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

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

```
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

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

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

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

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[3:7]` `(shortcut for  df.iloc[3:7,:]`

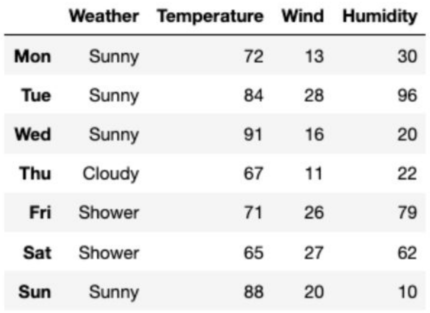
  That is, `[]` is context sensitive.

Poll

. The following dataframe contains data about weather for a one week period:

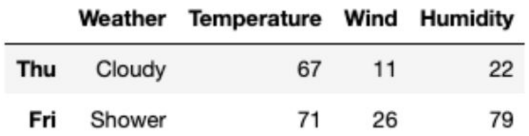
```
import pandas as pd

df=pd.read_csv("data/weather.csv", index_col=["Day"])
```



	Weather	Temperature	Wind	Humidity
Mon	Sunny	72	13	30
Tue	Sunny	84	28	96
Wed	Sunny	91	16	20
Thu	Cloudy	67	11	22
Fri	Shower	71	26	79
Sat	Shower	65	27	62
Sun	Sunny	88	20	10

(a) (5 pts) Which code will return the output shown here? Select all that apply.



	Weather	Temperature	Wind	Humidity
Thu	Cloudy	67	11	22
Fri	Shower	71	26	79

□ `df[4:5]`

□ `df[3:5]`

□ `df[3:6]`

□ `df.iloc[3:4,:]`

□ `df.iloc[3:5,:]`

□ `df.iloc[:,3:5]`

□ `df.loc["Thu","Fri"]`

□ `df.loc["Thu":"Fri", "Weather":"Humidity"]`

□ `df.loc[["Thu","Fri"],:]`

□ `df[["Thu","Fri"]]`

□ `df.iloc[[3, 4], [0, 1, 2, 3]]`

Learning Objectives:

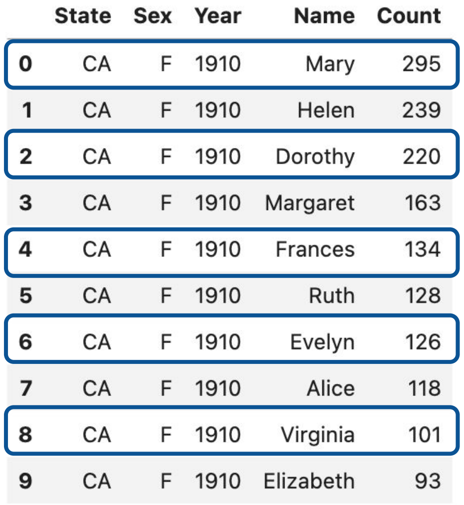
- Identify 5 key data properties to consider when doing Exploratory Data Analysis
- Understand methods for extracting data: .loc, .iloc, and [].
- Use conditional selection in Pandas

Pandas: Conditional Selection

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

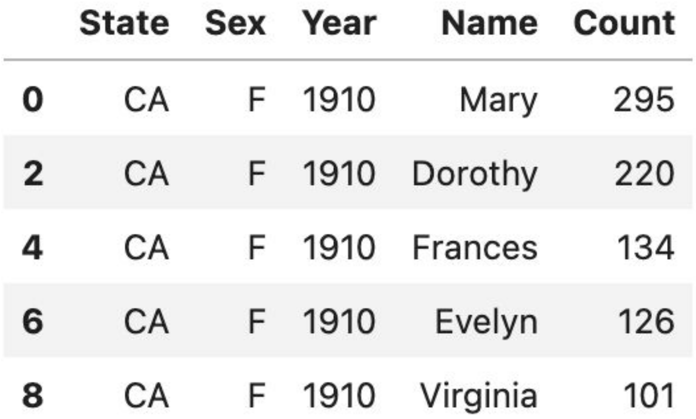
Boolean Array Input for `.loc` and `[ ]`

- `.loc` and `[ ]` also accept boolean arrays as input.
- Rows corresponding to **`True`** are extracted; rows corresponding to **`False`** are not.



	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
5	CA	F	1910	Ruth	128
6	CA	F	1910	Evelyn	126
7	CA	F	1910	Alice	118
8	CA	F	1910	Virginia	101
9	CA	F	1910	Elizabeth	93

```
babynames[[True, False, True, False,
True, False, True, False, True, False]]
```



	State	Sex	Year	Name	Count
0	CA	F	1910	Mary	295
2	CA	F	1910	Dorothy	220
4	CA	F	1910	Frances	134
6	CA	F	1910	Evelyn	126
8	CA	F	1910	Virginia	101

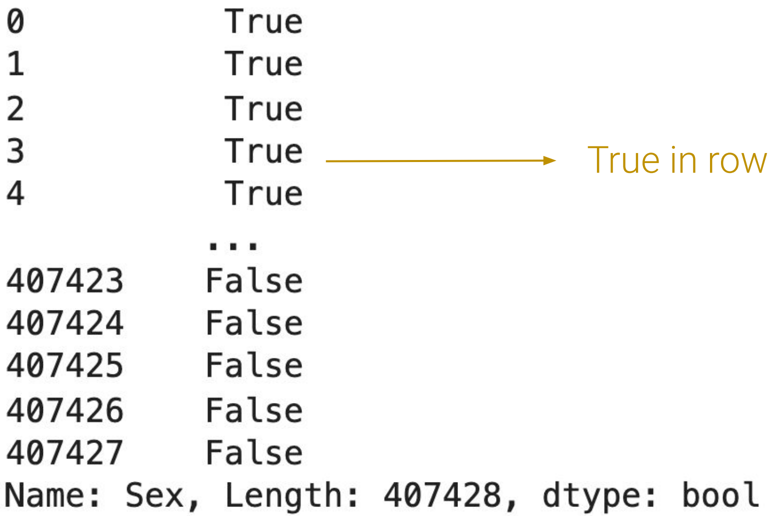
```
babynames.loc[[True, False, True, False, True, False, True, False, True, False], :]
```

Boolean Array Input

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

Length 407428 `Series` where every entry is either "True" or "False", where "True" occurs for every babyname with "Sex" = "F".

```
logical_operator = (babynames["Sex"] == "F")
```

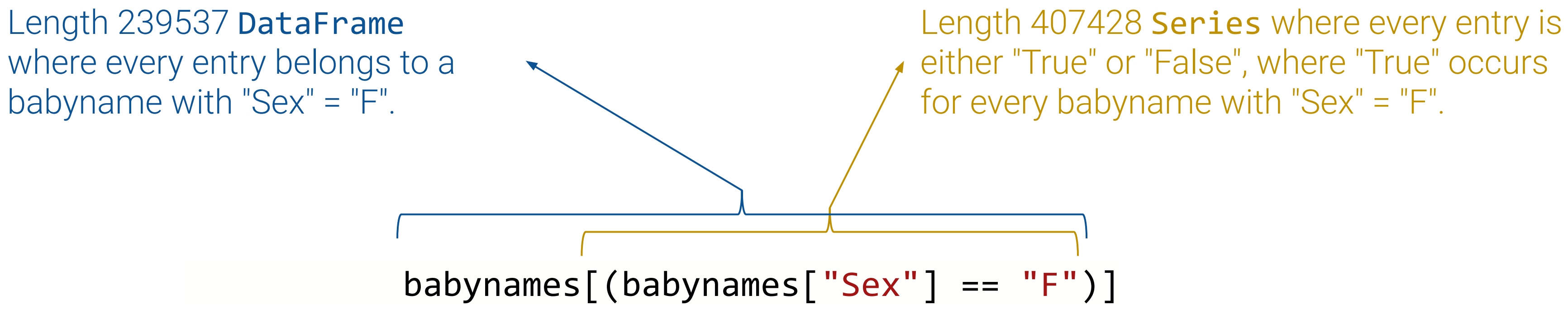


```
0          True
1          True
2          True
3          True
4          True
          ...
407423    False
407424    False
407425    False
407426    False
407427    False
Name: Sex, Length: 407428, dtype: bool
```

True in rows 0, 1, 2, ...

Boolean Array Input

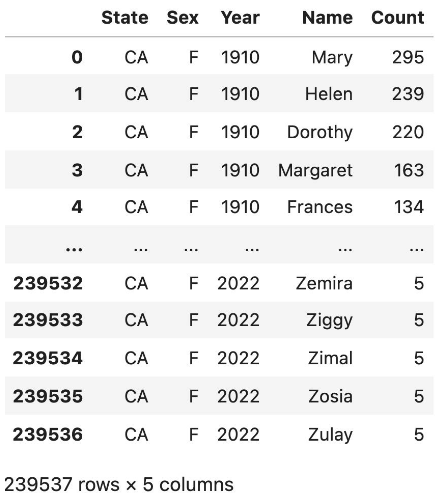
Useful because boolean arrays can be generated by using logical operators on `Series`.



Length 239537 `DataFrame` where every entry belongs to a babyname with "Sex" = "F".

Length 407428 `Series` where every entry is either "True" or "False", where "True" occurs for every babyname with "Sex" = "F".

```
babynames[(babynames["Sex"] == "F")]
```



	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
...	...	...	...	...	...
239532	CA	F	2022	Zemira	5
239533	CA	F	2022	Ziggy	5
239534	CA	F	2022	Zimal	5
239535	CA	F	2022	Zosia	5
239536	CA	F	2022	Zulay	5

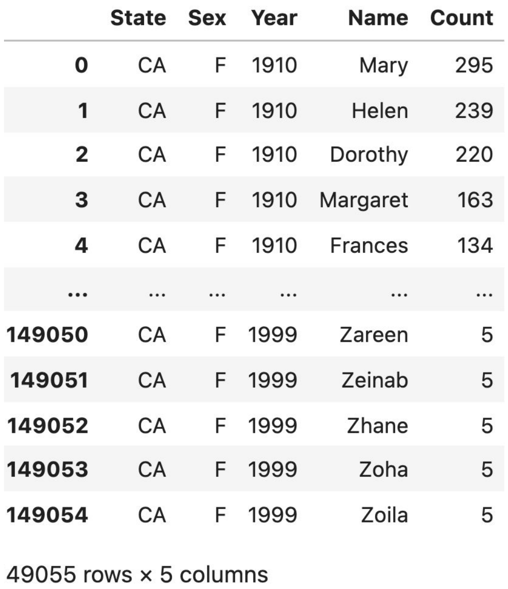
239537 rows × 5 columns

Boolean Array Input

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

- **The & operator allows us to apply `logical_operator_1` *and* `logical_operator_2`**
- The | operator allows us to apply `logical_operator_1` *or* `logical_operator_2`

```
babynames[(babynames["Sex"] == "F") & (babynames["Year"] < 2000)]
```



	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
...	...	...	...	...	...
149050	CA	F	1999	Zareen	5
149051	CA	F	1999	Zeinab	5
149052	CA	F	1999	Zhane	5
149053	CA	F	1999	Zoha	5
149054	CA	F	1999	Zoila	5

49055 rows × 5 columns

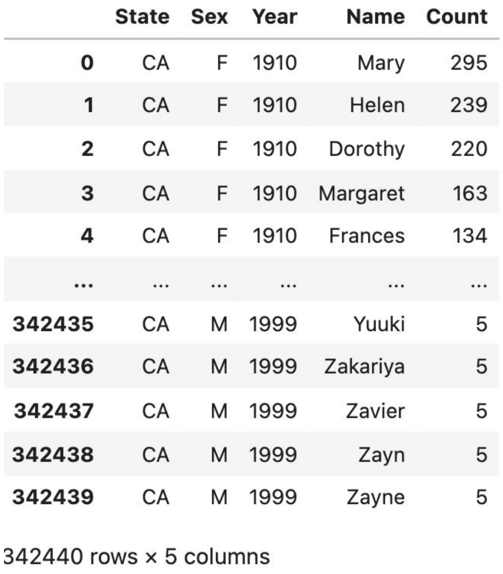
Rows that have a Sex of "F" *and* are earlier than the year 2000

Boolean Array Input

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

- The & operator allows us to apply `logical_operator_1` *and* `logical_operator_2`
- **The | operator allows us to apply `logical_operator_1` *or* `logical_operator_2`**

```
babynames[(babynames["Sex"] == "F") | (babynames["Year"] < 2000)]
```



	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
...	...	...	...	...	...
342435	CA	M	1999	Yuuki	5
342436	CA	M	1999	Zakariya	5
342437	CA	M	1999	Zavier	5
342438	CA	M	1999	Zayn	5
342439	CA	M	1999	Zayne	5

342440 rows × 5 columns

Rows that have a Sex of "F" *or* are earlier than the year 2000 (or both!)

Bitwise Operators

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

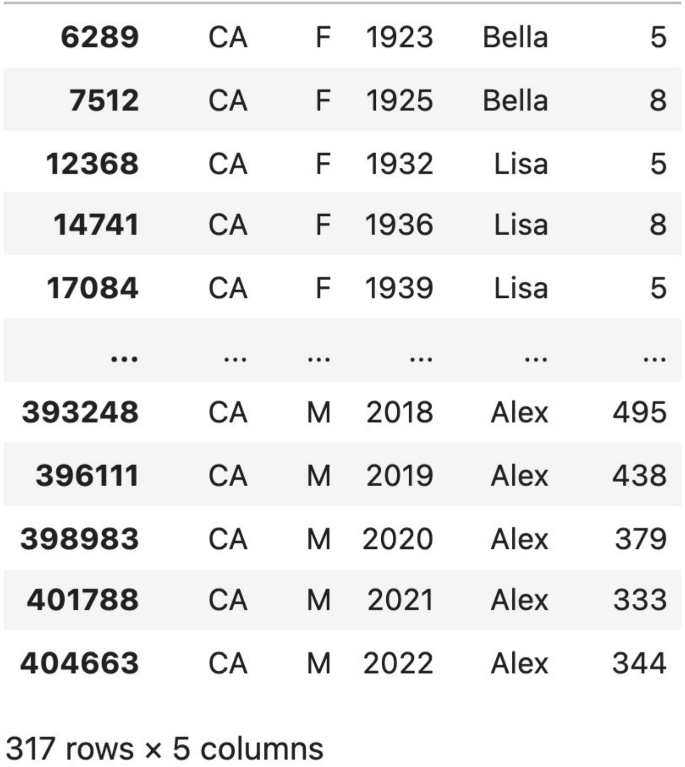
If `p` and `q` are boolean arrays or `Series`:

Symbol	Usage	Meaning
`~`	`~p`	Negation of `p`
`|`	`p | q`	`p` OR `q`
`&`	`p & q`	`p` AND `q`
`^`	`p ^ q`	`p` XOR `q` (exclusive or)
`!=`		`Not equal to`

Alternatives to Direct Boolean Array Selection

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

```
babynames[(babynames["Name"] == "Bella") |
          (babynames["Name"] == "Alex") |
          (babynames["Name"] == "Narges") |
          (babynames["Name"] == "Lisa")]
```



6289	CA	F	1923	Bella	5
7512	CA	F	1925	Bella	8
12368	CA	F	1932	Lisa	5
14741	CA	F	1936	Lisa	8
17084	CA	F	1939	Lisa	5
...	...	...	...	...	...
393248	CA	M	2018	Alex	495
396111	CA	M	2019	Alex	438
398983	CA	M	2020	Alex	379
401788	CA	M	2021	Alex	333
404663	CA	M	2022	Alex	344

317 rows × 5 columns

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

- `.isin`
- `str.startswith (see Appendix)`
- `.groupby.filter` (see Appendix)

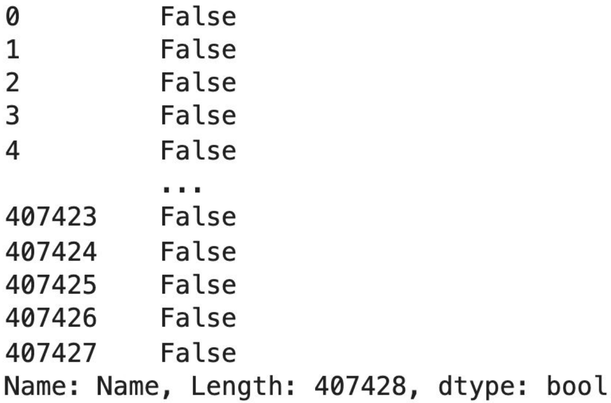
Alternatives to Direct Boolean Array Selection

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

- `.isin`
- `.str.startswith (see Appendix)`
- `.groupby.filter` (see next lesson)

```
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.

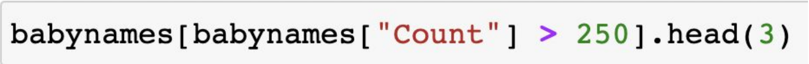


```
0         False
1         False
2         False
3         False
4         False
          ...
407423    False
407424    False
407425    False
407426    False
407427    False
Name: Name, Length: 407428, dtype: bool
```

iClicker Question

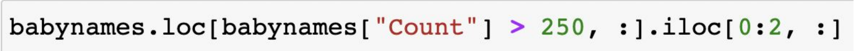
Which of the following pandas statements returns a DataFrame with the same columns as the babynames DataFrame but only the rows of the first 3 baby names with Count > 250? (Select all that apply)

A)



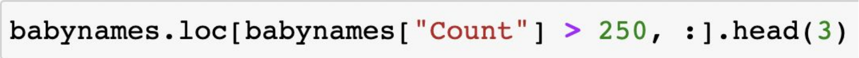
```
babynames[babynames["Count"] > 250].head(3)
```

B)



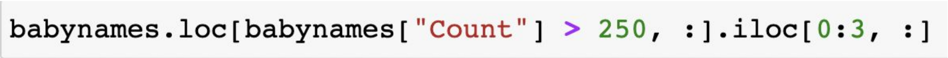
```
babynames.loc[babynames["Count"] > 250, :].iloc[0:2, :]
```

C)



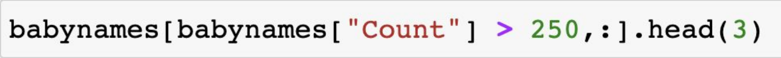
```
babynames.loc[babynames["Count"] > 250, :].head(3)
```

D)

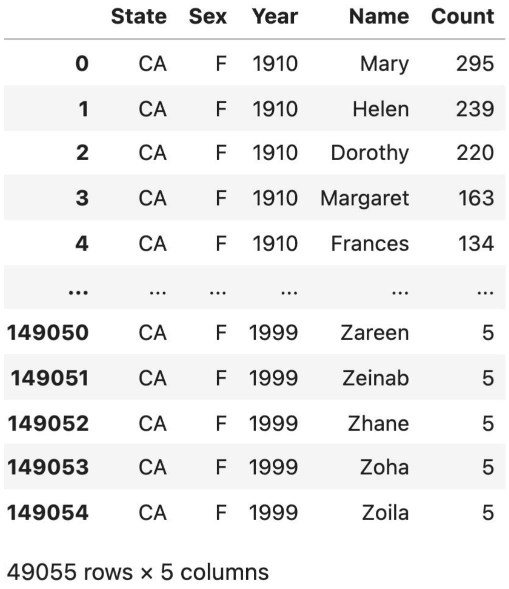


```
babynames.loc[babynames["Count"] > 250, :].iloc[0:3, :]
```

E).



```
babynames[babynames["Count"] > 250,:].head(3)
```



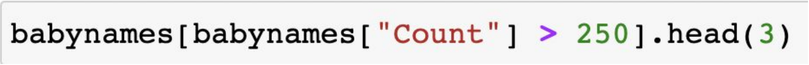
	State	Sex	Year	Name	Count
0	CA	F	1910	Mary	295
1	CA	F	1910	Helen	239
2	CA	F	1910	Dorothy	220
3	CA	F	1910	Margaret	163
4	CA	F	1910	Frances	134
...	...	...	...	...	...
149050	CA	F	1999	Zareen	5
149051	CA	F	1999	Zeinab	5
149052	CA	F	1999	Zhane	5
149053	CA	F	1999	Zoha	5
149054	CA	F	1999	Zoila	5

49055 rows × 5 columns

iClicker Question

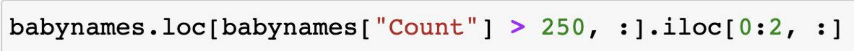
Which of the following pandas statements returns a DataFrame with the same columns as the babynames DataFrame but only the rows of the first 3 baby names with Count > 250? (Select all that apply)

A)



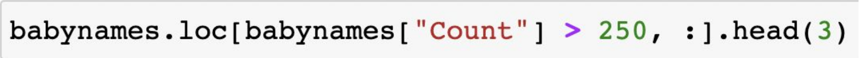
```
babynames[babynames["Count"] > 250].head(3)
```

B)



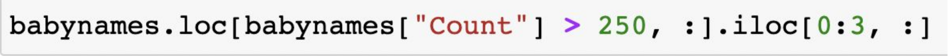
```
babynames.loc[babynames["Count"] > 250, :].iloc[0:2, :]
```

C)



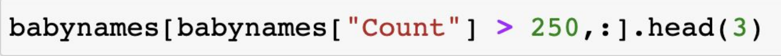
```
babynames.loc[babynames["Count"] > 250, :].head(3)
```

D)

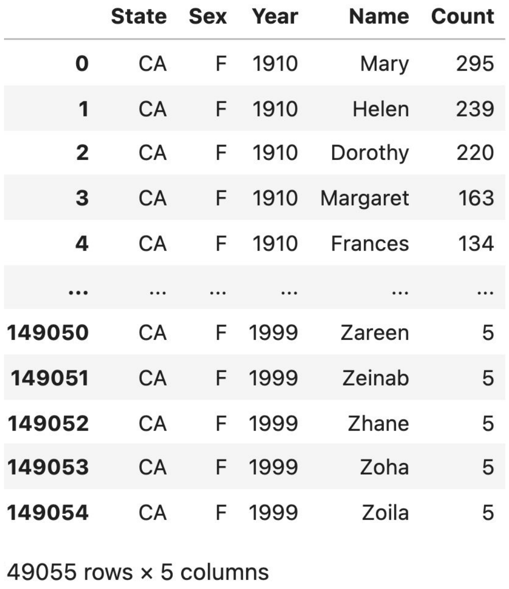


```
babynames.loc[babynames["Count"] > 250, :].iloc[0:3, :]
```

E).



```
babynames[babynames["Count"] > 250,:].head(3)
```



	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
...	...	...	...	...	...
149050	CA	F	1999	Zareen	5
149051	CA	F	1999	Zeinab	5
149052	CA	F	1999	Zhane	5
149053	CA	F	1999	Zoha	5
149054	CA	F	1999	Zoila	5

49055 rows × 5 columns

Pandas: Modifying Columns

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

Syntax for Adding a Column

Suppose we wanted to add a column with the length of each name:

```
# Create a Series of the length of each name

babyname_lengths = babynames["Name"].str.len()
```

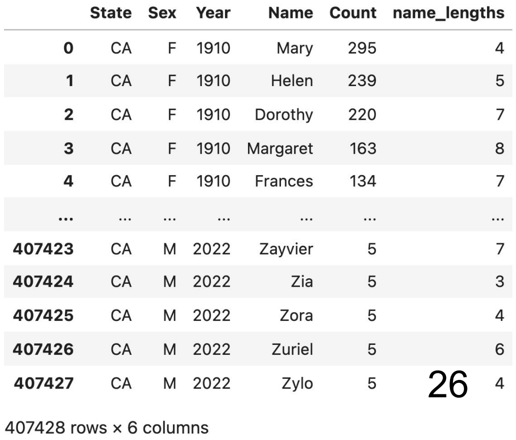
Adding a column is easy:

Option 1: Use `df.assign()`

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

Option 2: Use `[ ]` to reference the desired new column.

```
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

Syntax for Modifying a Column

Modifying a column is very similar to adding a column.

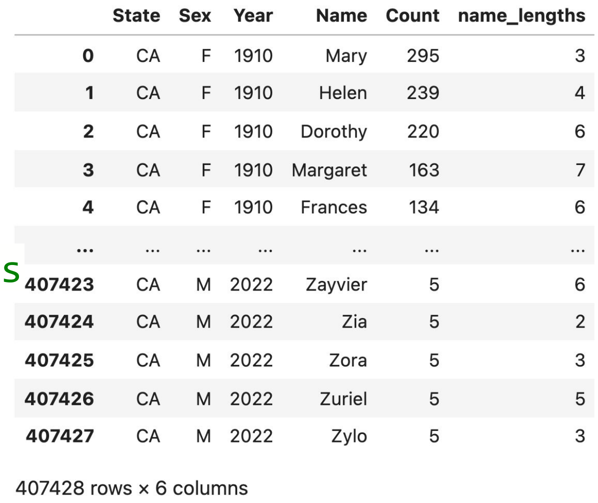
Option 1: Use `df.assign()`

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

Option 2: 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

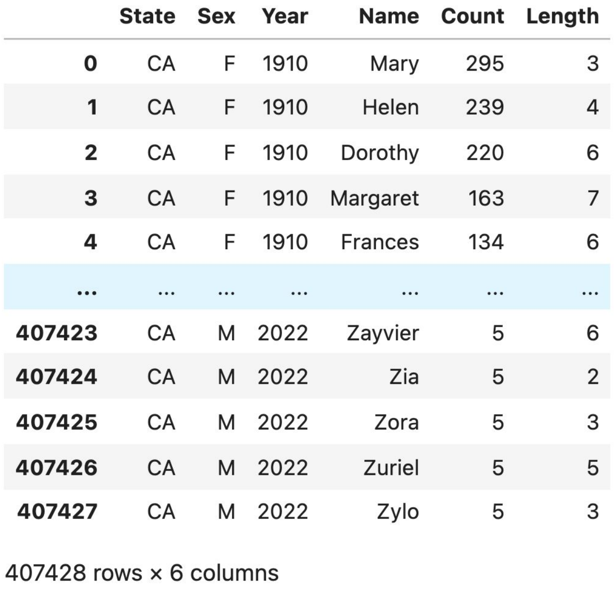
407428 rows × 6 columns

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"})
```



	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

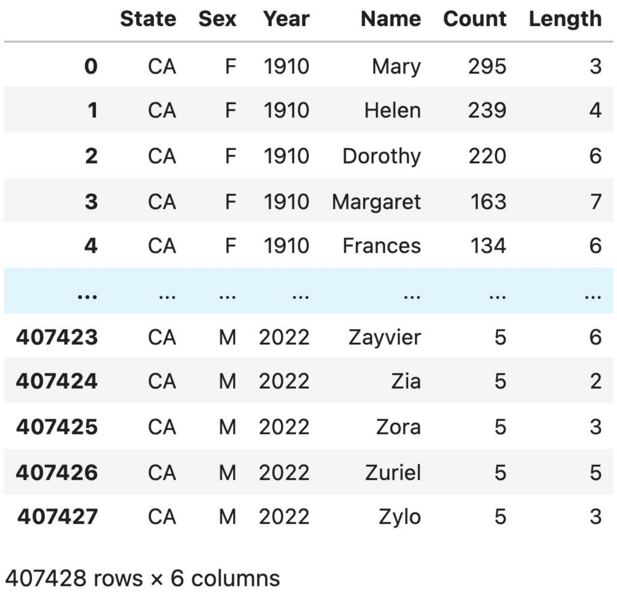
407428 rows × 6 columns

Syntax for Dropping a Column (or Row)

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

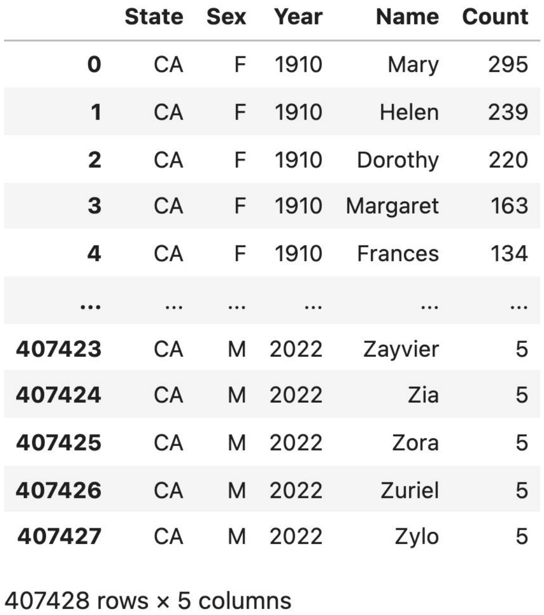
- The `.drop()` method assumes you're dropping a row by default. Use `axis = "columns"` to drop a column instead.

```
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

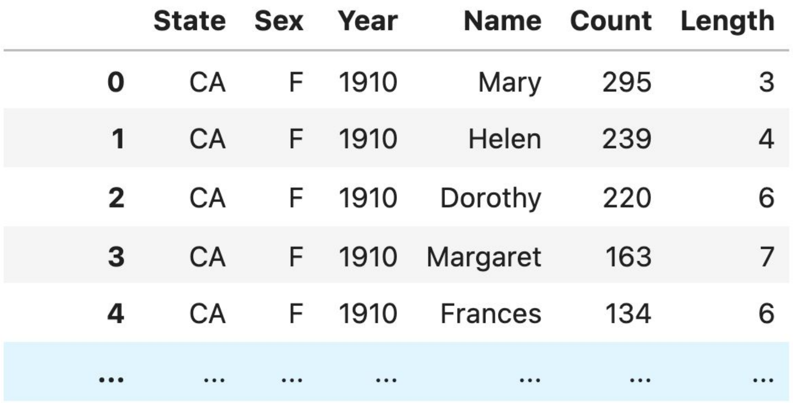
An Important Note: `DataFrame` Copies

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

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

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.drop("Length", axis = "columns")
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!

Appendix: Alternatives to Boolean Array Selection

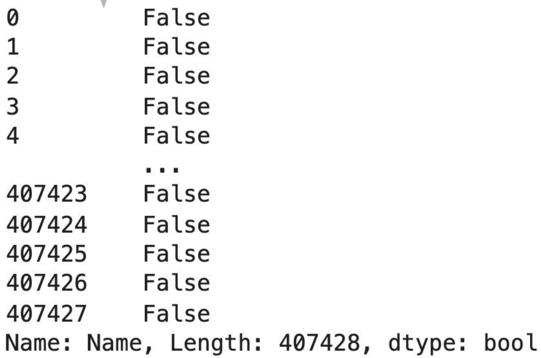
Alternatives to Boolean Array Selection

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

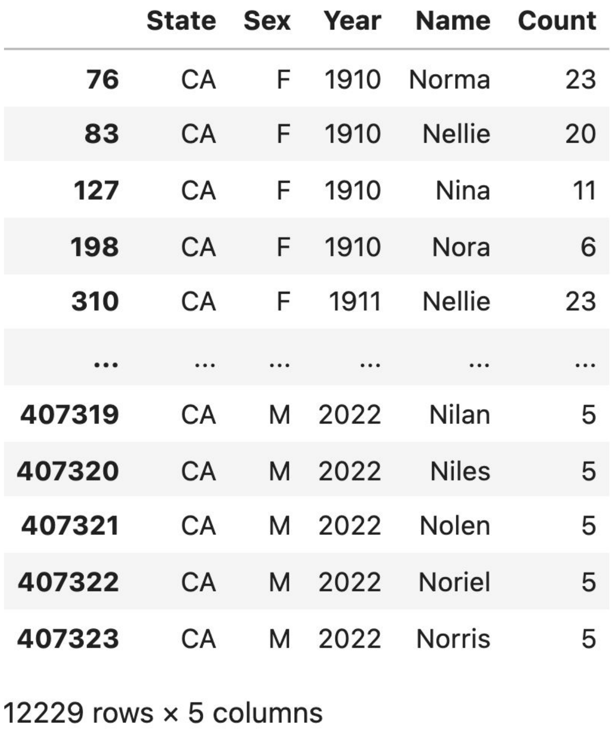
- `.isin`
- `.str.startswith`
- `.groupby.filter` (see lecture 4)

```
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