3385706

slido

Join at slido.com
#3385706

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

Come up front to grab a lollipop and say hi!

3385706

We encourage you to open up the Lecture 3 notebook so you can follow along!

LECTURE 3

Pandas, Part II

More on `pandas` (Selections and Utility Functions)

Data 100, Summer 2025 @ UC Berkeley

Josh Grossman and Michael Xiao

📣 Announcements - Lab 1 Due tomorrow

3385706

Mon+Wed Discussion starts today! Before discussion, **watch the discussion mini-lecture.**

Jake's **Mega** Discussion:	1:00 - 2:00pm @ Social Sciences 166
Sammie's **Data Scholars** Discussion:	1:00 - 2:30pm @ Social Sciences 140
Wesley's Discussion:	2:00 - 3:00pm @ Wheeler 130
Milena's Discussion:	3:00 - 4:00pm @ Social Sciences 140
Xiaorui's **Online** Discussion:	3:00 - 4:00pm @ Zoom

Mon+Wed **Catch-Up Session** with Sammie starts today!
4:00 - 5:00pm @ Social Sciences Building 166

Staff OH start today @ Warren 101B and Online. See website for hours.

Josh and Michael have office hours right after every lecture in HFAX B1 next door.

Agenda

Lecture 03, Data 100 Summer 2025

- Conditional selection
- Adding, removing, and modifying columns
- Useful utility functions
- Custom sorts

Conditional Selection

Lecture 03, Data 100 Summer 2025

- **Conditional selection**
- Adding, removing, and modifying columns
- Useful utility functions
- Custom sorts

Recall from last time

3385706

We can extract data using its **integer position** (`.iloc`) or its **label** (`.loc`)

```
babynames_first_10_rows = babynames.loc[:9, :]
```

	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

Question to ponder: Do we get the same result with `.iloc` or context-dependent extraction?

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

How do we extract rows that satisfy a **condition** (e.g., rows where column X > 5)?

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

```
                                    0      1      2      3      4      5      6      7      8      9
babynames_first_10_rows[[True, False, True, False, True, False, True, False, True, False]]
```

	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

	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

Boolean Array Input

3385706

We can perform the same operation using `.loc`.

```
                                    0      1      2      3      4      5      6      7      8      9
babynames_first_10_rows.loc[[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

Boolean Array Input

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".

```
babynames["Sex"]
```

```
0         F
1         F
2         F
3         F
4         F
         ..
407423    M
407424    M
407425    M
407426    M
407427    M
Name: Sex, Length: 407428, dtype: object
```

```
(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
```

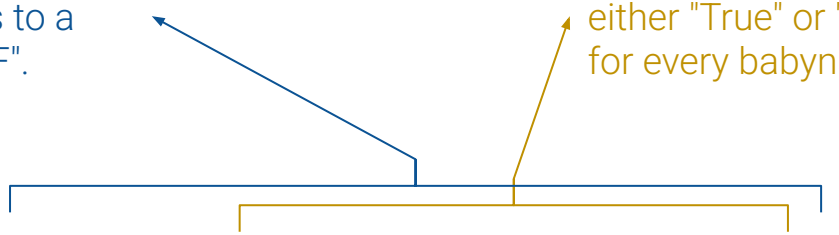
Boolean Array Input

3385706

We can use our new boolean array to filter to desired rows. Boolean mask! 🤿

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

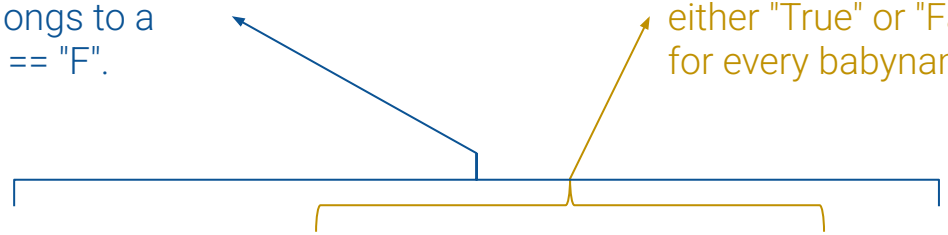
Boolean Array Input

3385706

We can also use **`.loc`**.

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.loc[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 element-wise using boolean operators.

- The `&` operator applies `boolean_series_1` **AND** `boolean_series_2`
- The `|` operator applies `boolean_series_1` **OR** `boolean_series_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 with Sex=="F" **OR** before year 2000 (or both!)

Bitwise Operators

3385706

`&` 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`	NOT `p`
`|`	`p | q`	`p` OR `q`
`&`	`p & q`	`p` AND `q`
`^`	`p ^ q`	`p` XOR `q`

p XOR q: Either p, or q, but not both. Exclusive OR!

slido

3385706

Which of the following pandas statements returns a DataFrame of the first 3 baby names with Count > 250? (Talk and work w/ your neighbor!)

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

Alternatives to Direct Boolean Array Selection

Boolean array selection is useful, but can lead to overly verbose code.

```
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`
- `.groupby.filter` (we'll see this in Lecture 4)

Alternatives to Direct Boolean Array Selection

3385706

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

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

On the Data 100 Reference Sheet!

```
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

3385706

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

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

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

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

More on the `.str` accessor in Lecture 6!

3385706

Adding, Removing, and Modifying Columns

Lecture 03, Data 100 Summer 2025

- Conditional selection
- **Adding, removing, and modifying columns**
- Useful utility functions
- Custom sorts

Syntax for Adding a Column

3385706

Making a new column with the number of characters in each name:

1. `Series` with the length of each name

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

2. Assign `Series` to a new column called "name_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

Syntax for Modifying a Column

3385706

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

	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

Syntax for Renaming a Column

3385706

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

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

	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

	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

Common typo: Flipping the keys and values in the dictionary! Forgive yourself ❤️

Syntax for Dropping a Column (or Row)

3385706

Remove columns using the `.drop` method.

- The `.drop()` method drops **rows** by default. Use `axis="columns"` to drop columns.

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

	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

An Important Note: `DataFrame` Copies

3385706

On the previous slide, we **re-assigned** `babynames` to an updated `DataFrame`.

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

By default, `pandas` creates a new **copy** of the `DataFrame`, **without changing** the original `DataFrame`.

```
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!

Which of the following returns 'b'? (Talk and work w/ your neighbor!)

3385706

Interlude

2-min stretch break!

Useful Utility Functions

Lecture 03, Data 100 Summer 2025

- Conditional selection
- Adding, removing, and modifying columns
- **Useful utility functions**
- Custom sorts

3385706

Pandas **`Series`** and **`DataFrames`** support many operations, including **`NumPy`** operations, so long as the data is numeric. [Data 8 **`NumPy`** reference.]

```
yash_count = babynames[babynames["Name"]=="Yash"]["Count"]
```

```
np.mean(yash_count)
```

```
17.142857142857142
```

```
np.max(yash_count)
```

```
29
```

	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

```
331824     8
334114     9
336390    11
338773    12
341387    10
343571    14
345767    24
348230    29
350889    24
353445    29
356221    25
358978    27
361831    29
364905    24
367867    23
370945    18
374055    14
376756    18
379660    18
383338     9
385903    12
388529    17
391485    16
394906    10
397874     9
400171    15
403092    13
406006    13
Name: Count, dtype: int64
```

Pro Tip: Just because Python shows you a lot of decimal places does not mean you should report all of them. Only use as much precision as needed. In the example above, 17.1 is preferable.

Built-In `pandas` Methods

3385706

`pandas` also provides an enormous number of useful utility functions, including:

- `size/shape`
- `describe`
- `sample`
- `value_counts`
- `unique`
- `sort_values`
- Among many others!

If you want to manipulate data in some way, there is probably a `pandas` function that does what you want. Explore the documentation, Google, or ask a large language model (LLM)!

`.shape` and `.size`

3385706

- The `.shape` attribute provides the dimensions as a tuple **(# rows, # columns)**.
- The `.size` attribute provides the total # of entries → # rows multiplied by # columns.

`babynames`

	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

```
babynames.shape
(407428, 5)
```

```
babynames.size
2037140
```

407,428 * 5 = 2,037,140

`.describe()`

3385706

- The `.describe()` method returns summary statistics of a `DataFrame` or `Series`.

`babynames`

	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

`babynames.describe()`

	Year	Count
count	407428.000000	407428.000000
mean	1985.733609	79.543456
std	27.007660	293.698654
min	1910.000000	5.000000
25%	1969.000000	7.000000
50%	1992.000000	13.000000
75%	2008.000000	38.000000
max	2022.000000	8260.000000

Note: Only the numeric columns are summarized!

`.describe()`

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

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

```
count     407428
unique         2
top            F
freq      239537
Name: Sex, dtype: object
```

`.sample()`

3385706

To randomly sample rows from a `DataFrame`, use the `.sample()` method.

- By default, we **sample without replacement**. Use `replace=True` for **replacement**.

```
babynames.sample()
```

	State	Sex	Year	Name	Count
121141	CA	F	1992	Shanelle	28

n=1 by default

Pro Tip: The top of a `DataFrame` is often not representative of the entire `DataFrame`. For this reason, I tend to prefer `df.sample()` over `df.head()` when exploring data.

Recall that "sampling with replacement" means that each row can be sampled more than one time.

Chaining methods

3385706

`pandas` methods can be chained together:

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

	Year	Name	Count
44448	1961	Karyn	36
260410	1948	Carol	7
397541	2019	Arya	11
4767	1921	Sumiko	16
104369	1987	Thomas	11

Multi-line chaining requires parentheses.

```
result = (
    babynames[babynames["Year"]==2000]
    .sample(4, replace=True)
    .iloc[:, 2:]
)
```

	Year	Name	Count
151749	2000	Iridian	7
343560	2000	Maverick	14
149491	2000	Stacy	91
149212	2000	Angel	307

.unique()

3385706

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

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

```
array(['Mary', 'Helen', 'Dorothy', ..., 'Zae', 'Zai', 'Zayvier'],
      dtype=object)
```

.value_counts()

3385706

The `Series.value_counts()` method returns the # of times each **unique** value appears.

- Return value is a sorted `Series`.

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

```
Name
Jean          223
Francis       221
Guadalupe     218
Jessie        217
Marion        214
              ...
Ozair           1
Rahm            1
Ruhan           1
Sufiyan         1
Theon           1
Name: count, Length: 20437, dtype: int64
```

.sort_values()

3385706

The `.sort_values()` method sorts a **`DataFrame`** or **`Series`**.

- `DataFrame.sort_values(column_name)` → Must specify column for sorting!

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

```
366001      Aadan
384005      Aadan
369120      Aadan
398211    Aadarsh
370306      Aaden
           ...
220691      Zyrah
197529      Zyrah
217429      Zyrah
232167      Zyrah
404544      Zyrus
Name: Name, Length: 407428, dtype: object
```

Notice the index changes order too!

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

	State	Sex	Year	Name	Count
268041	CA	M	1957	Michael	8260
267017	CA	M	1956	Michael	8258
317387	CA	M	1990	Michael	8246
281850	CA	M	1969	Michael	8245
283146	CA	M	1970	Michael	8196
...	...	...	...	...	...
317292	CA	M	1989	Olegario	5
317291	CA	M	1989	Norbert	5
317290	CA	M	1989	Niles	5
317289	CA	M	1989	Nikola	5
407427	CA	M	2022	Zylo	5

407428 rows × 5 columns

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

Common typo: Forgetting if ascending/descending is the default. Solution? Always be explicit!

Which of the following extracts the rows of a DataFrame df where the values in column A are at least as big as the smallest value in column B? (Talk and work w/ your neighbor!)

Custom Sorts

Lecture 03, Data 100 Summer 2025

- Conditional selection
- Adding, removing, and modifying columns
- Useful utility functions
- **Custom sorts**

Sorting By Length

3385706

Suppose we want to sort entries based on the length of each name.

- Familiar approach:
  - Create a new column with the length of each name and then sort on it.

Approach 1: Create a New Column and Sort Based on the New Column

3385706

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

babynames = babynames.sort_values(by="name_lengths", ascending=False)

babynames.head(5)
```

	State	Sex	Year	Name	Count	name_lengths
334166	CA	M	1996	Franciscojavier	8	15
337301	CA	M	1997	Franciscojavier	5	15
339472	CA	M	1998	Franciscojavier	6	15
321792	CA	M	1991	Ryanchristopher	7	15
327358	CA	M	1993	Johnchristopher	5	15

Approach 2: Sorting Using the `key` Argument

3385706

```
babynames.sort_values("Name", key=lambda x: x.str.len(), ascending=False).head()
```

Create a **temporary** column with the length of each name, and sort on it!

	State	Sex	Year	Name	Count
334166	CA	M	1996	Franciscojavier	8
327472	CA	M	1993	Ryanchristopher	5
337301	CA	M	1997	Franciscojavier	5
337477	CA	M	1997	Ryanchristopher	5
312543	CA	M	1987	Franciscojavier	5

Recall that `lambda x: x.str.len()` is an anonymous function (i.e., not given a name, temporary, one-time use)

Approach 3: Sorting Using the `Series.map` method

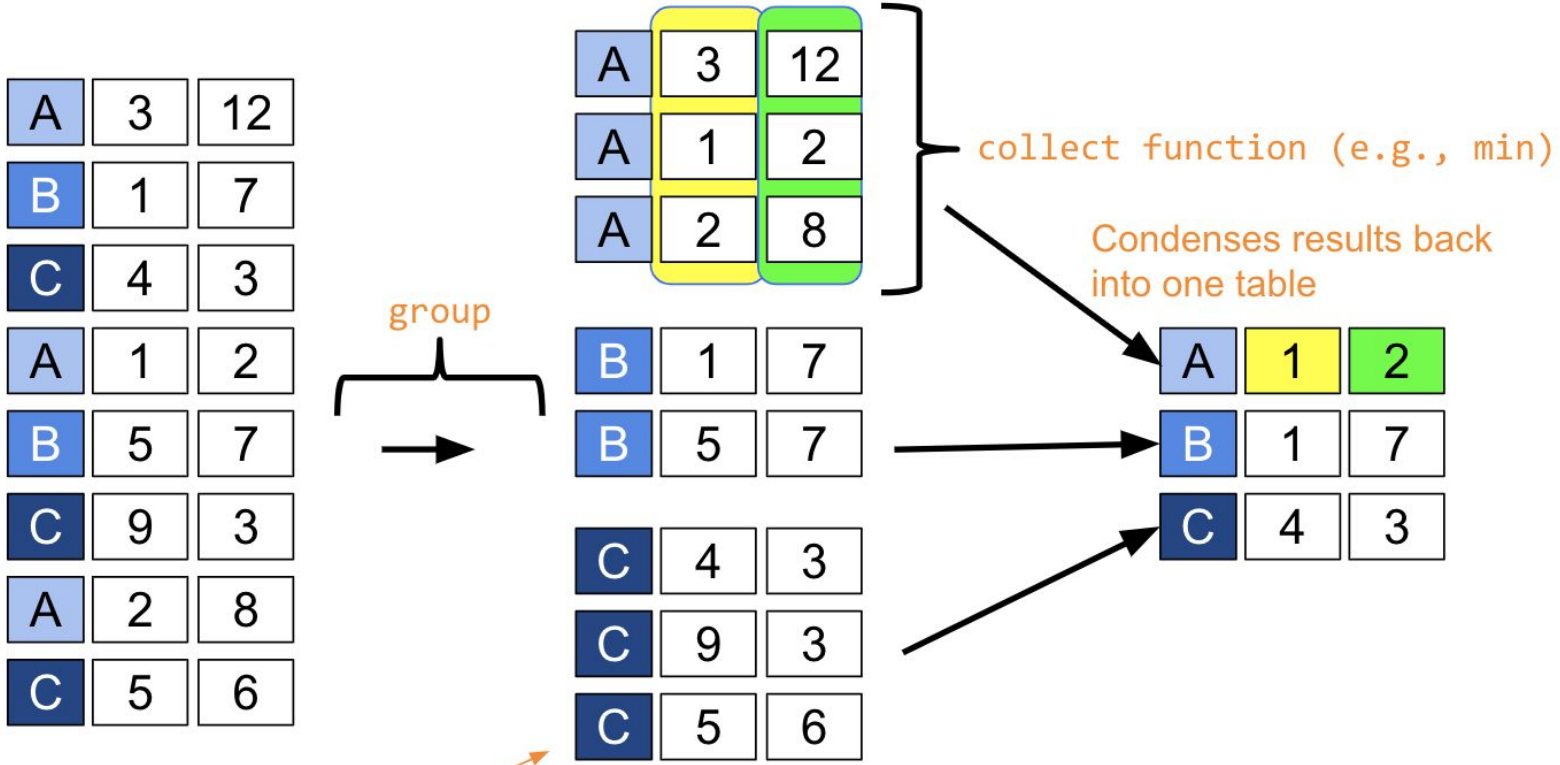
Using the **`Series.map`** method to sort by the number of occurrences of "dr" and "ea"s:

```
# Returns number of times 'dr' and 'ea' appear in `string`
def dr_ea_count(string):
    return string.count('dr') + string.count('ea')

# Apply dr_ea_count to each name in the "Name" column
babynames["dr_ea_count"] = babynames["Name"].map(dr_ea_count)

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

	State	Sex	Year	Name	Count	dr_ea_count
115957	CA	F	1990	Deandrea	5	3
101976	CA	F	1986	Deandrea	6	3
131029	CA	F	1994	Leandrea	5	3
108731	CA	F	1988	Deandrea	5	3
308131	CA	M	1985	Deandrea	6	3

A taste of next lecture! Grouping diagram from Data 8

3385706



LECTURE 3

Pandas, Part II

Content credit: Acknowledgments