slido

Join at slido.com #1845239

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

1845239

LECTURE 4

Pandas, Part III

Advanced Pandas (Sorting, Grouping, Aggregation, Pivot Tables, and Merging)

Data 100, Summer 2025 @ UC Berkeley

Josh Grossman and Michael Xiao

📣 Announcements

1845239

Lab 1 due tonight! You must pass the public tests for full credit.

Homework 1 due tomorrow! Make sure you submit both components:

- **Homework 1 Coding**
- **Homework 1 Math Prereqs**

Pre-Semester Survey also due tonight!

This weekend, students who chose the graded discussion option will be assigned to a permanent discussion section.

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

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

1845239

Today's Roadmap

Lecture 4, Data 100 Summer 2025

- Grouping
- Pivot Tables
- Joining Tables

Grouping

Lecture 4, Data 100 Summer 2025

- **Grouping**
- Pivot Tables
- Joining Tables

Why group?

1845239

Grouping allows us to perform complex+simultaneous operations and summarize trends.

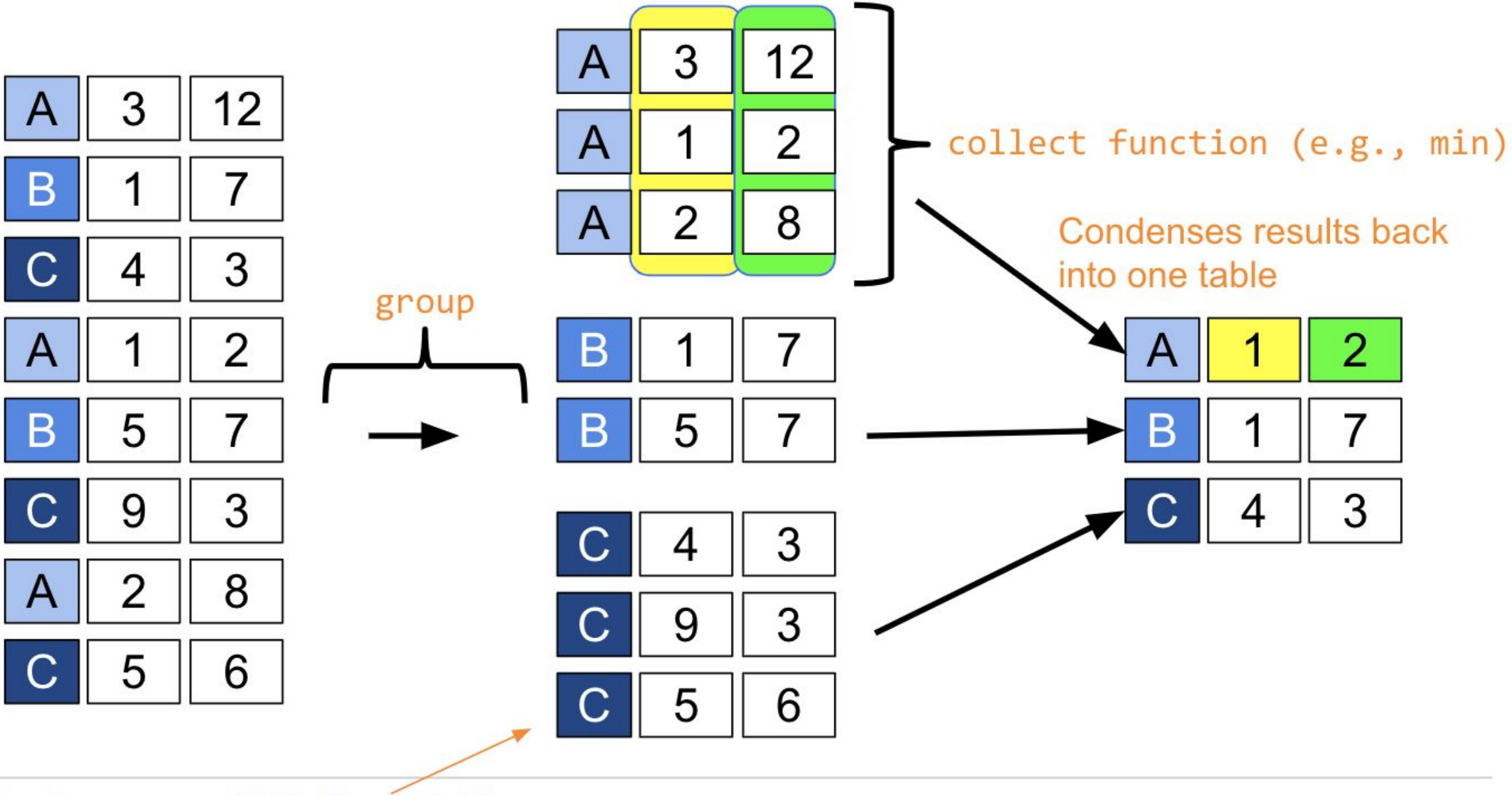
Two steps:

1. Group rows with the **same value in one or more columns**.
    - For example, rows with the same year and month.

2. For each group of rows, **aggregate** the rows using an operation.
    - For example, sum up the total # of babies born in each year and month.

Visual Review of Data 8: Grouping and Collection

1845239

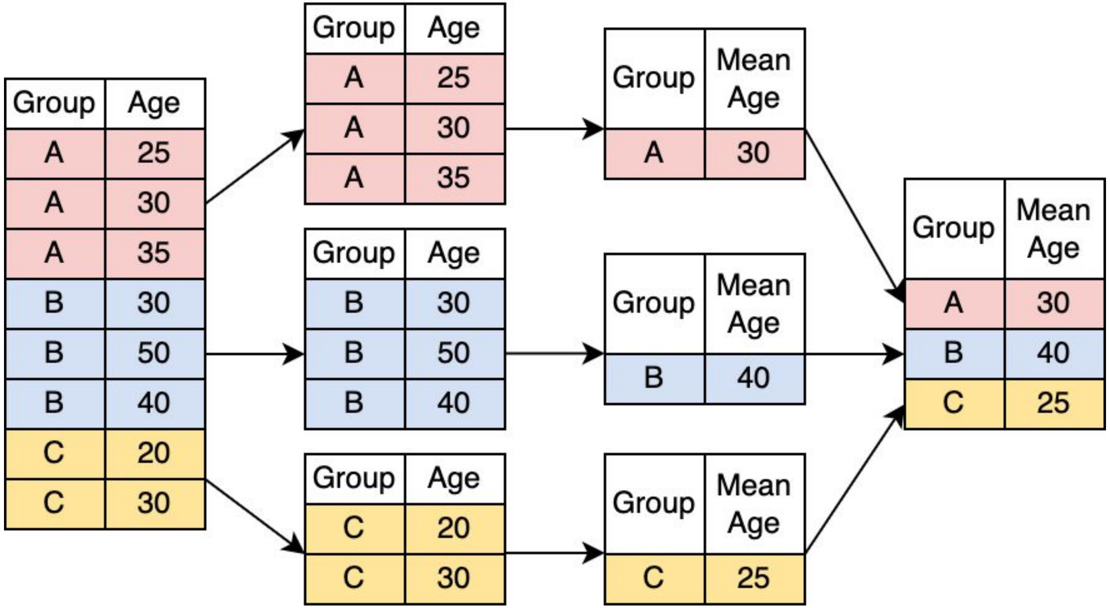


Can think of as temporary (A,B,C) sub-tables

Split-Apply-Combine Paradigm

1845239

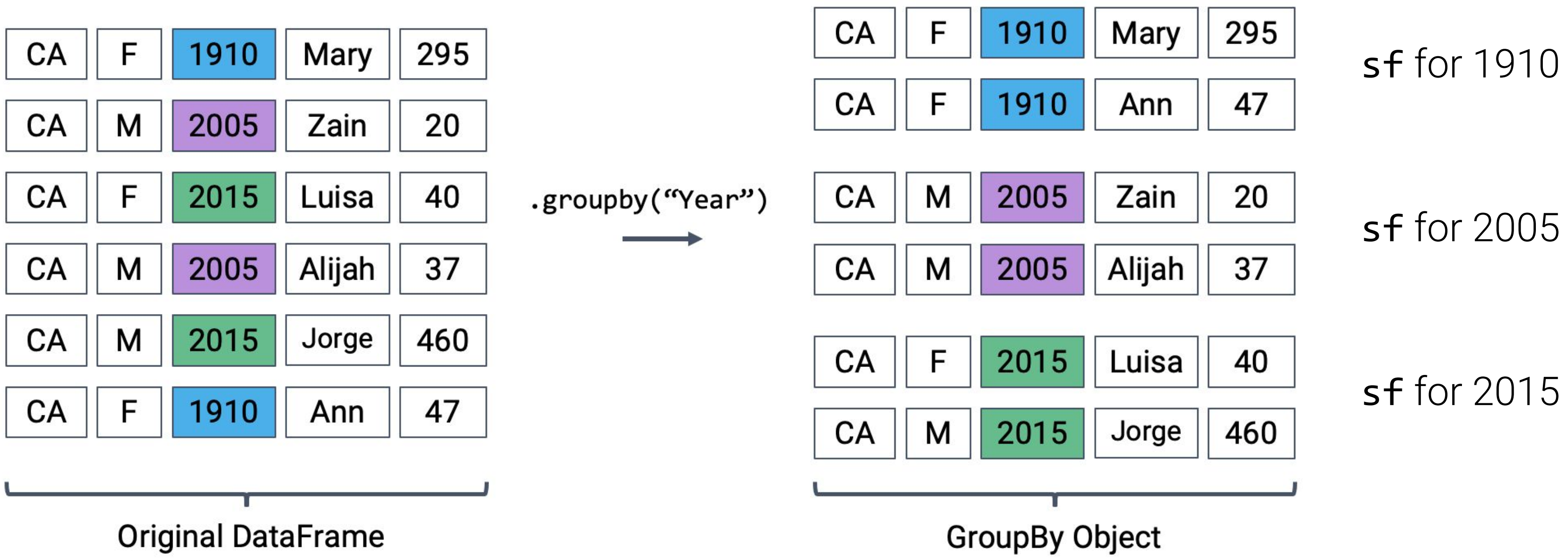
Step 1: Split Step 2: Apply Step 3: Combine



`.groupby()`

`.groupby()` **splits** a `DataFrame` (`df`) into "mini" `DataFrames` (subframes, or `sf`'s)

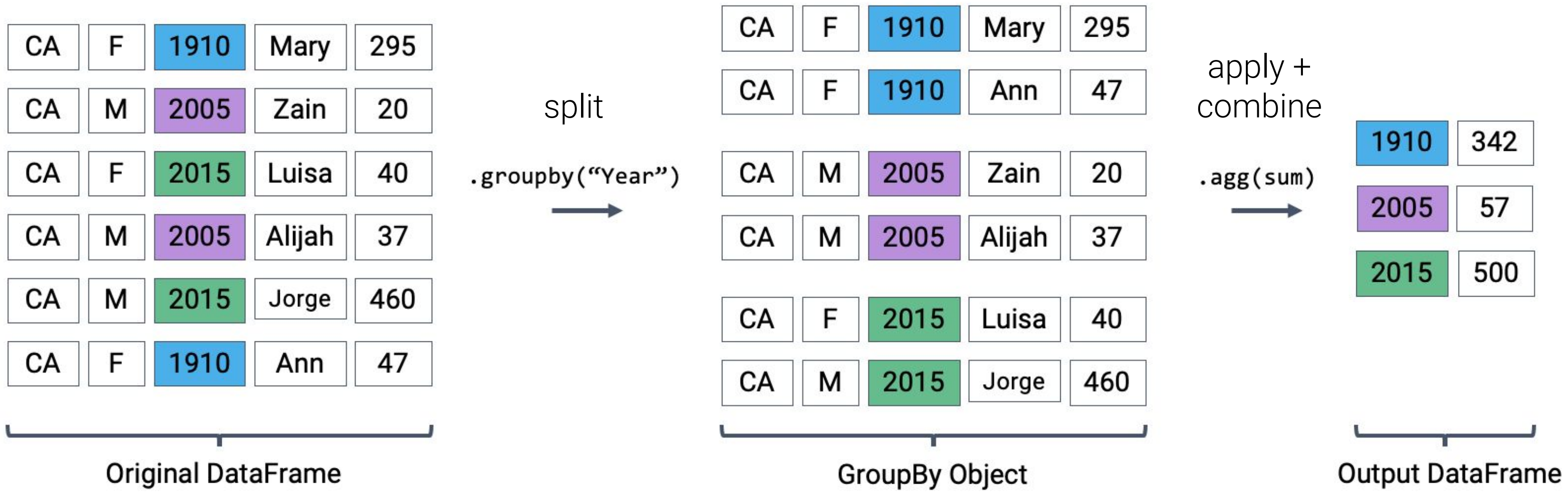
- There is one `sf` for each group.
- The `sf`'s are stored in a single `DataFrameGroupBy` object.



1845239

`.groupby()` and `.agg()`

`.agg()` **applies** an aggregation operation to each subframe (`sf`), and **combines** the `sf`'s.



Where did the non-numeric columns go? Explained on the next slide!

Note: We very rarely work directly with `DataFrameGroupBy` objects. `.groupby()` should always be followed by something else, like `.agg()` or `.filter()`

Putting It All Together

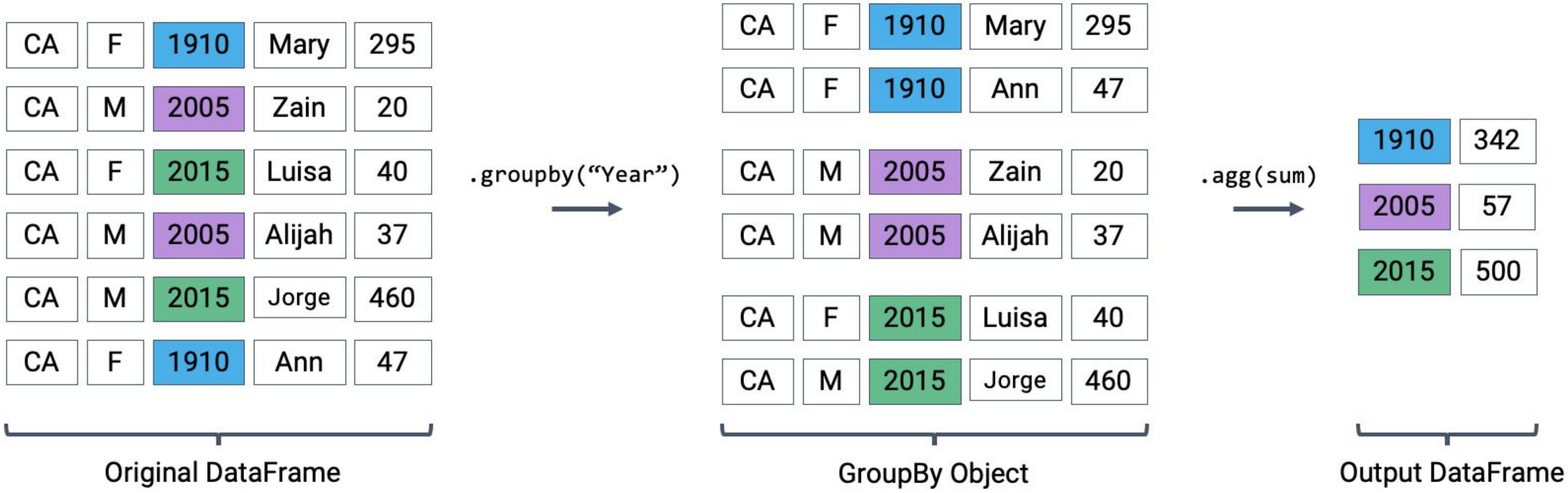
1845239

Syntax to compute the total # of babies born in each year:

```
babynames.groupby("Year")[["Count"]].agg(sum)
```

On the [Data 100 Reference Sheet]!

Only sum up the **`Count`** column.
Ignore the other columns. 🙈



1845239

What can go inside of `.agg( )`?

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

In-Built Python Functions	NumPy Functions	In-Built pandas functions
`.agg(sum)`	`.agg(np.sum)`	`.agg("sum")`
`.agg(max)`	`.agg(np.max)`	`.agg("max")`
`.agg(min)`	`.agg(np.min)`	`.agg("min")`
	`.agg(np.mean)`	`.agg("mean")`
		`.agg("first")`
		`.agg("last")`

Some aggregation functions can be called directly, without using `.agg( )`!

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

↕️

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

Plotting Birth Counts

1845239

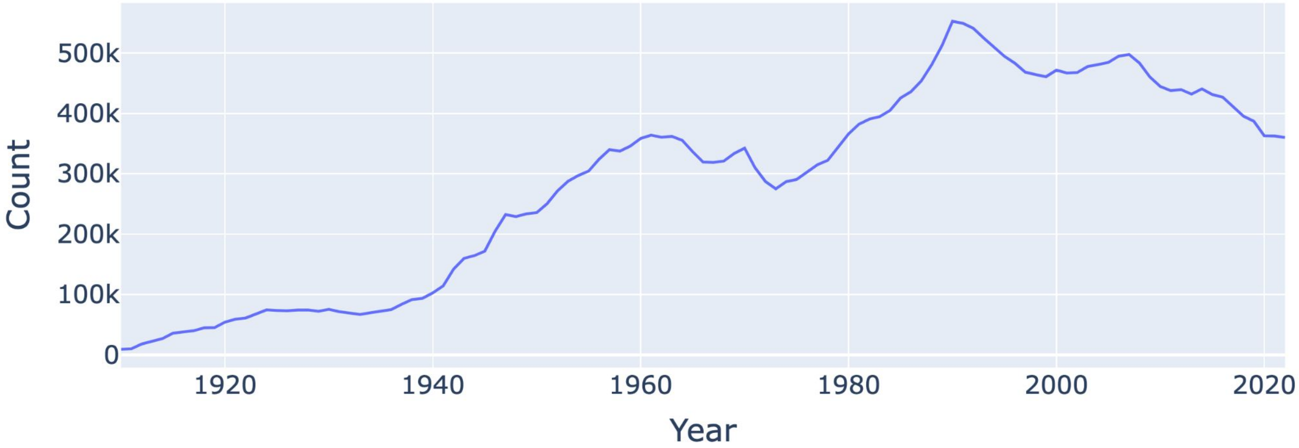
A plot of the **`babynames`** data summarized by year tells an interesting story.

```
babies_by_year = babynames.groupby("Year")[["Count"]].agg(sum)
```

Year	Count
1910	9163
1911	9983
1912	17946
1913	22094
1914	26926
...	...
2018	395436
2019	386996
2020	362882
2021	362582
2022	360023

113 rows × 1 columns

```
import plotly.express as px
px.line(babies_by_year, y="Count")
```



Alternatives …

1845239

Different ways to create groups for each year:

```
babynames.groupby("Year")[["Count"]].agg(sum)
```

or

```
babynames.groupby("Year")[["Count"]].sum()
```

or

```
babynames.groupby("Year").sum(numeric_only=True)
```

Year	Count
1910	9163
1911	9983
1912	17946
1913	22094
1914	26926
...	...
2018	395436
2019	386996
2020	362882
2021	362582
2022	360023

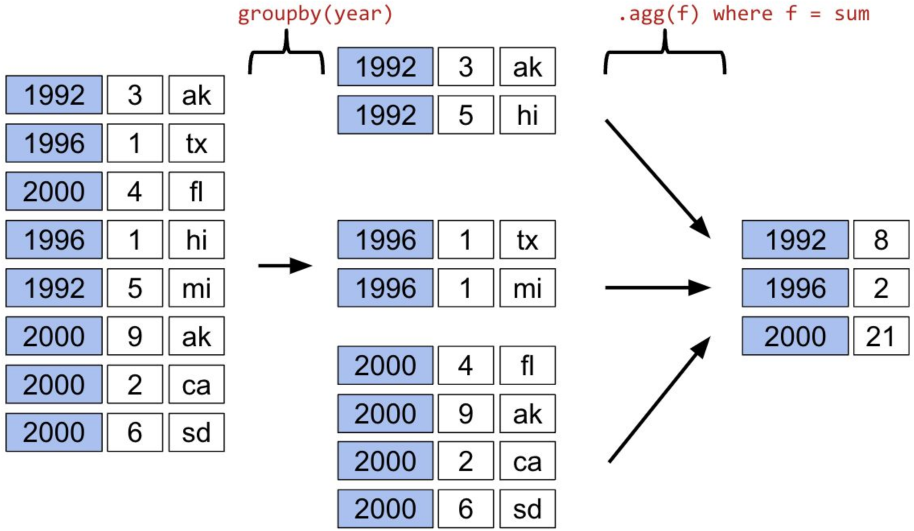
113 rows × 1 columns

Concluding `groupby.agg`

1845239

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

- So far, we've seen that `df.groupby("Year").agg(sum)`:
  - **Split** `df` into sub-`DataFrame`s based on `Year`.
  - **Apply** the `sum` function to each column of each sub-`DataFrame`.
  - **Combine** the results of `sum` into a single `DataFrame`, indexed by `Year`.



`.groupby()` mystery to ponder

`df =`

col1	col2	col3
A	3	ak
B	1	tx
C	4	fl
A	1	hi
B	5	mi
C	9	ak
A	2	ca
C	5	sd
B	6	nc

`df.groupby('col1').agg(max) =`

A	??	??
B	??	??
C	??	??

1845239

slido

What do you think will go in the first row?

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

`.groupby()` question to ponder

1845239

If we don't specify the columns to aggregate, `.agg()` aggregates <u>all</u> ungrouped columns.

`df =`

col1	col2	col3
A	3	ak
B	1	tx
C	4	fl
A	1	hi
B	5	mi
C	9	ak
A	2	ca
C	5	sd
B	6	nc

`df.groupby('col1').agg(max) =`

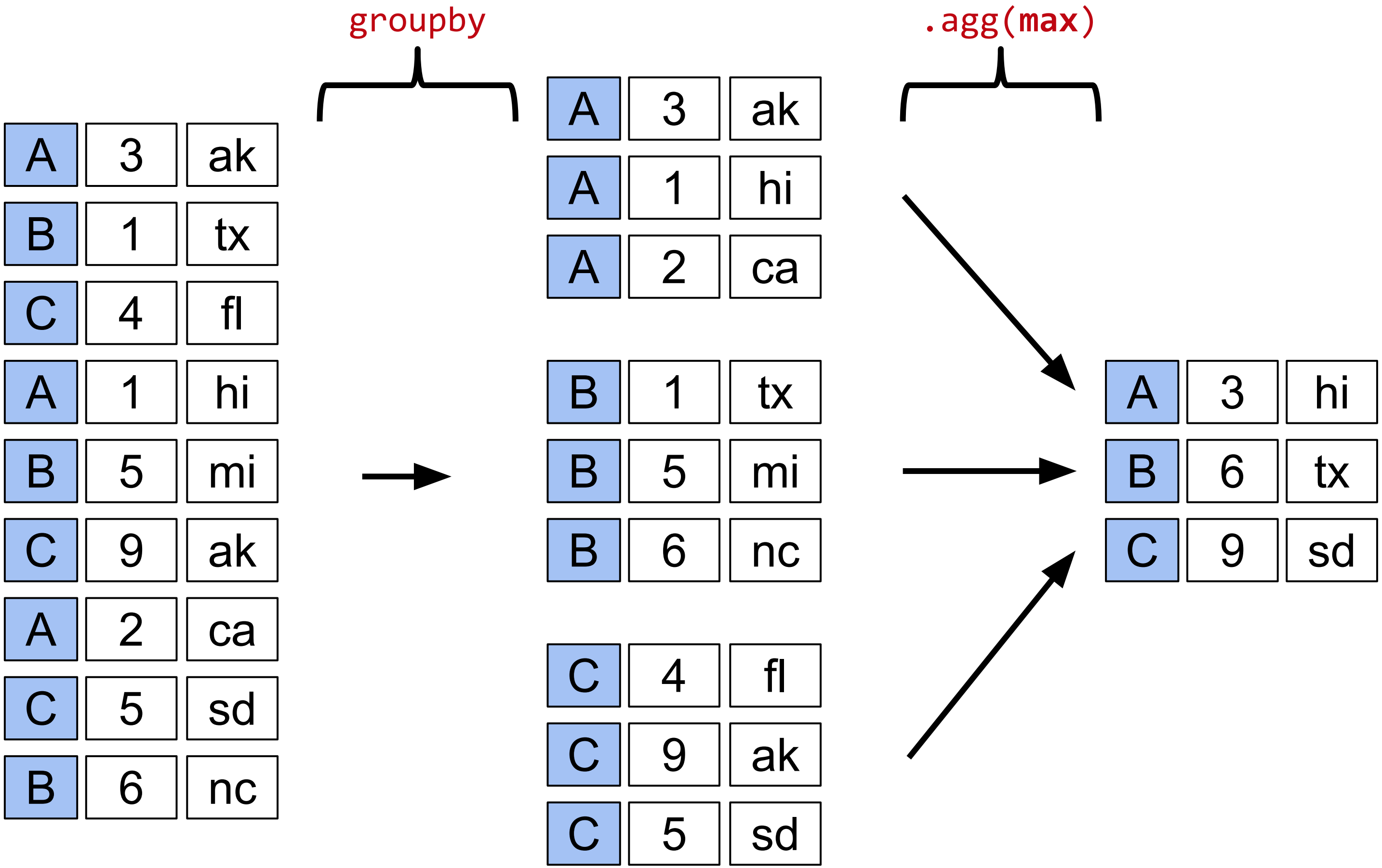
A	??	??
B	??	??
C	??	??

↕️

`df.groupby('col1')[['col2','col3']].agg(max)`

Visualizing the correct answer

1845239



Case Study: Name "Popularity"

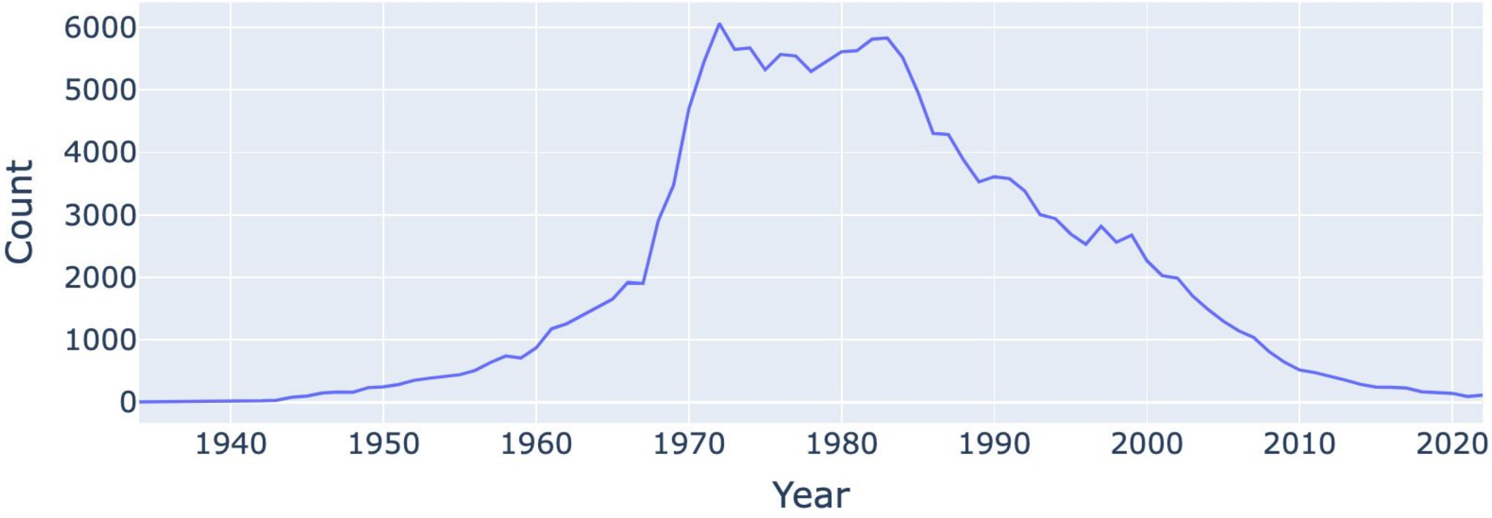
1845239

Goal: Find the baby name with sex="F" that has fallen in popularity the most in California.

```
f_babynames = babynames[babynames["Sex"]=="F"]
```

```
jenn_counts_ser = f_babynames[f_babynames["Name"]=="Jennifer"]["Count"]
```

Number of Jennifers Born in California Per Year.



What Is "Popularity"?

1845239

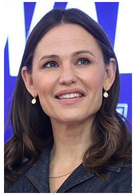
Goal: Find the baby name with sex="F" that has fallen in popularity the most in California.

How do we define "fallen in popularity?"

- Let's create a metric: **"Ratio to Peak" (RTP)**.
- RTP = (# babies w/ name in given year) / (max # of babies w/ same name from any year)

Example for "Jennifer":

- In 1972, we hit peak Jennifer! 6,065 Jennifers were born.
- In 2022, there were only 114 Jennifers.
- RTP for 2022 is 114 / 6065 ≈ 0.019



Jennifer Garner
Born in 1972

Calculating RTP

1845239

```
max_jenn = max(jenn_counts_ser)
```

```
6065
```

```
curr_jenn = jenn_counts_ser.iloc[-1]
```

```
114
```

Remember: `f_babynames` is sorted ascending by year.
`.iloc[-1]` means "grab the last element of the `Series`"

```
rtp = curr_jenn / max_jenn
```

```
0.018796372629843364
```

```
def ratio_to_peak(series):
    return series.iloc[-1] / max(series)
```

```
ratio_to_peak(jenn_counts_ser)
0.018796372629843364
```

Calculating RTP Using `.groupby()`

1845239

`.groupby()` makes it easy to compute the RTP for **all names at the same time**!

```
rtp_table = f_babynames.groupby("Name")[["Year","Count"]].agg(ratio_to_peak)
```

	Year	Count
Name		
Aadhini	1.0	1.000000
Aadhira	1.0	0.500000
Aadhya	1.0	0.660000
Aadya	1.0	0.586207
Aahana	1.0	0.269231
...	...	...
Zyanya	1.0	0.466667
Zyla	1.0	1.000000
Zylah	1.0	1.000000
Zyra	1.0	1.000000
Zyrah	1.0	0.833333

13782 rows × 2 columns

```
def ratio_to_peak(series):
    return series.iloc[-1] / max(series)
```

slido

1845239

Are there any rows for which Year is not 1.0? Recall that `babynames` is sorted ascending by year.

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

A Note on Nuisance Columns

1845239

Executing the `.agg()` call below results in a `TypeError` or warning, depending on the version of `pandas`. (How would you do mathematical division with a string, like "CA"?)

```
f_babynames.groupby("Name").agg(ratio_to_peak)
```

We need to specify the numeric column to aggregate:

```
rtp_table = f_babynames.groupby("Name")[["Count"]].agg(ratio_to_peak)
```

	Count
Name	
Aadhini	1.000000
Aadhira	0.500000
Aadhya	0.660000
Aadya	0.586207
Aahana	0.269231
...	...

Renaming Columns After Grouping

1845239

By default, `.groupby()` does not rename aggregated columns

- For example, our column is still named "Count", even though it now represents the RTP.

For better readability, we can rename "Count" to "Count RTP":

```
rtp_table = f_babynames.groupby("Name")[["Count"]].agg(ratio_to_peak)
rtp_table = rtp_table.rename(columns={"Count":"Count RTP"})
```

Name	Count
Aadhini	1.000000
Aadhira	0.500000
Aadhya	0.660000
Aadya	0.586207
Aahana	0.269231
...	...

→

Name	Count RTP
Aadhini	1.000000
Aadhira	0.500000
Aadhya	0.660000
Aadya	0.586207
Aahana	0.269231
...	...

Some Data Science Payoff

1845239

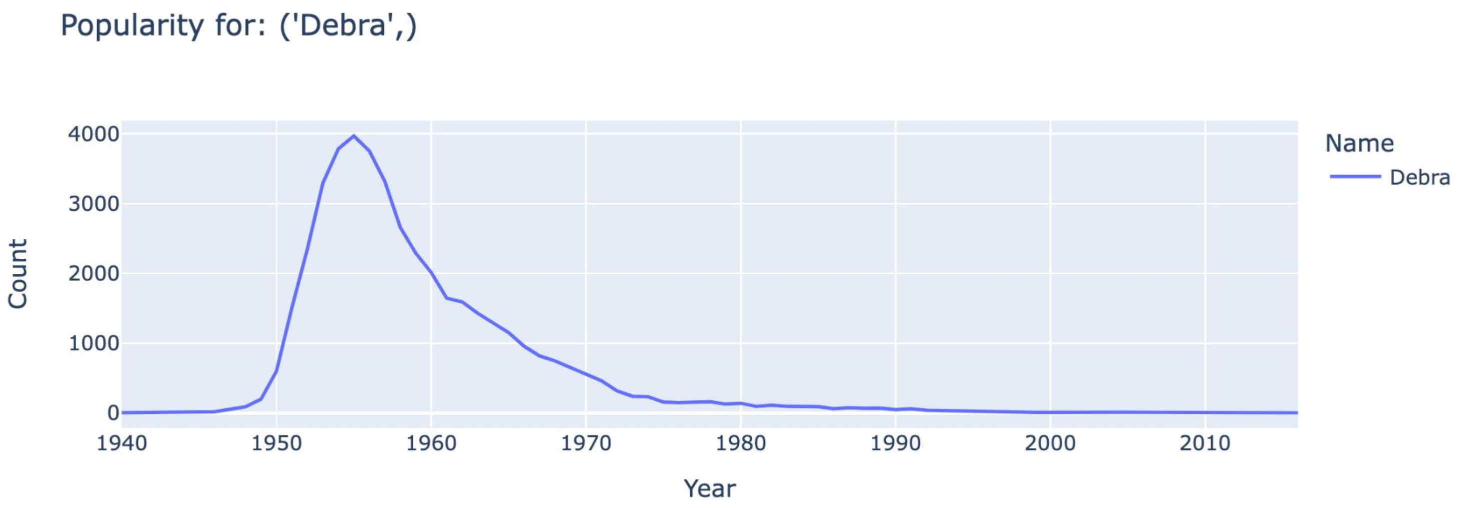
By sorting `rtp_table` we can see the names whose popularity has decreased the most.

```
rtp_table.sort_values("Count RTP")
```

Name	Count RTP
Debra	0.001260
Debbie	0.002815
Carol	0.003180
Tammy	0.003249
Susan	0.003305
...	...
Fidelia	1.000000
Naveyah	1.000000
Finlee	1.000000
Roseline	1.000000
Aadhini	1.000000

13782 rows × 1 columns

```
px.line(f_babynames[f_babynames["Name"]=="Debra"],
                        x="Year", y="Count")
```



Some Data Science Payoff

1845239

We can get the list of the top 10 names and then plot popularity:

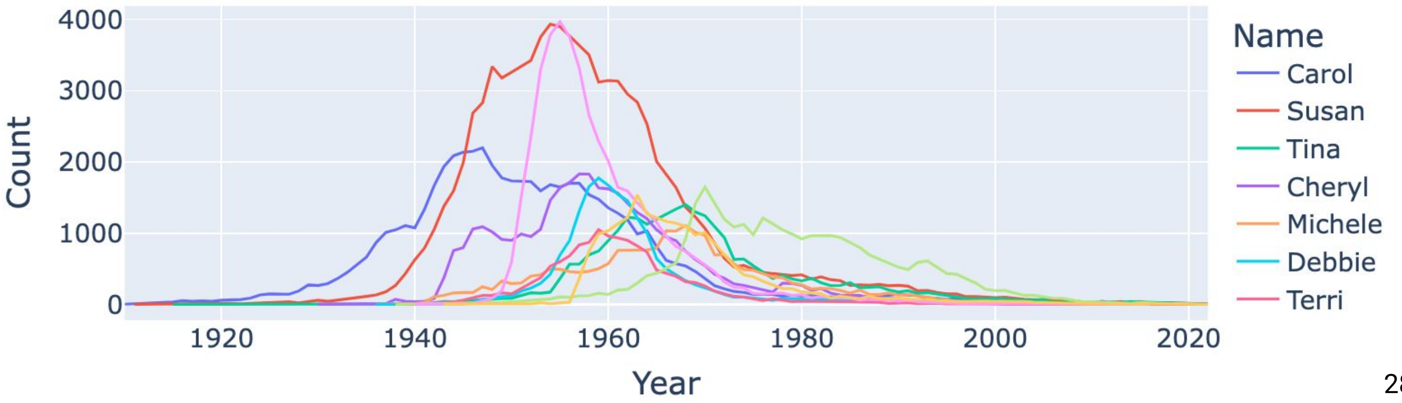
```
top10 = rtp_table.sort_values("Count RTP").head(10).index
```

	Count RTP
Name	
Debra	0.001260
Debbie	0.002815
Carol	0.003180
Tammy	0.003249
Susan	0.003305
...	...
Fidelia	1.000000
Naveyah	1.000000
Finlee	1.000000
Roseline	1.000000
Aadhini	1.000000

13782 rows × 1 columns

```
Index(['Debra', 'Debbie', 'Carol', 'Tammy', 'Susan', 'Cheryl', 'Shannon',
       'Tina', 'Michele', 'Terri'],
      dtype='object', name='Name')
```

```
px.line(f_babynames[f_babynames["Name"].isin(top10)],
                    x="Year", y="Count", color="Name")
```



Interlude

2-min stretch break!

1845239

Raw GroupBy Objects and Other Methods

1845239

The result of a groupby operation applied to a DataFrame is a **DataFrameGroupBy** object.

- It is not a **DataFrame**!

```
grouped_by_year = elections.groupby("Year")
type(grouped_by_year)
```

```
pandas.core.groupby.generic.DataFrameGroupBy
```

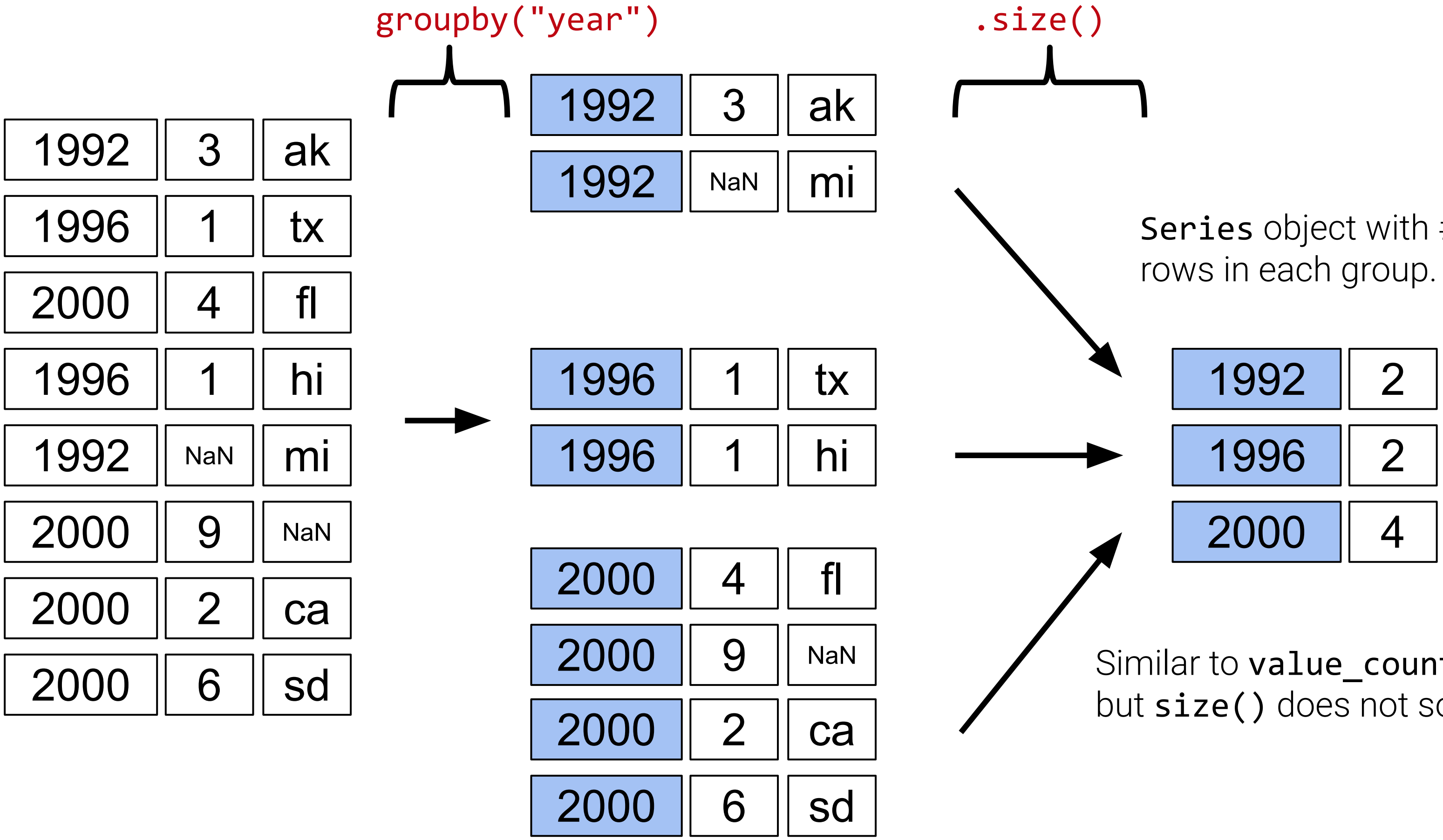
Given a **DataFrameGroupBy** object, can use various functions to generate **DataFrames** (or **Series**). **agg** is only one choice:

```
df.groupby(col).mean()      df.groupby(col).first()      df.groupby(col).filter()
df.groupby(col).sum()       df.groupby(col).last()
df.groupby(col).min()       df.groupby(col).size()
df.groupby(col).max()       df.groupby(col).count()
```

🤨What's the difference?

See pandas.pydata.org/docs/reference/groupby.html for a list of **DataFrameGroupBy** methods.

.size()

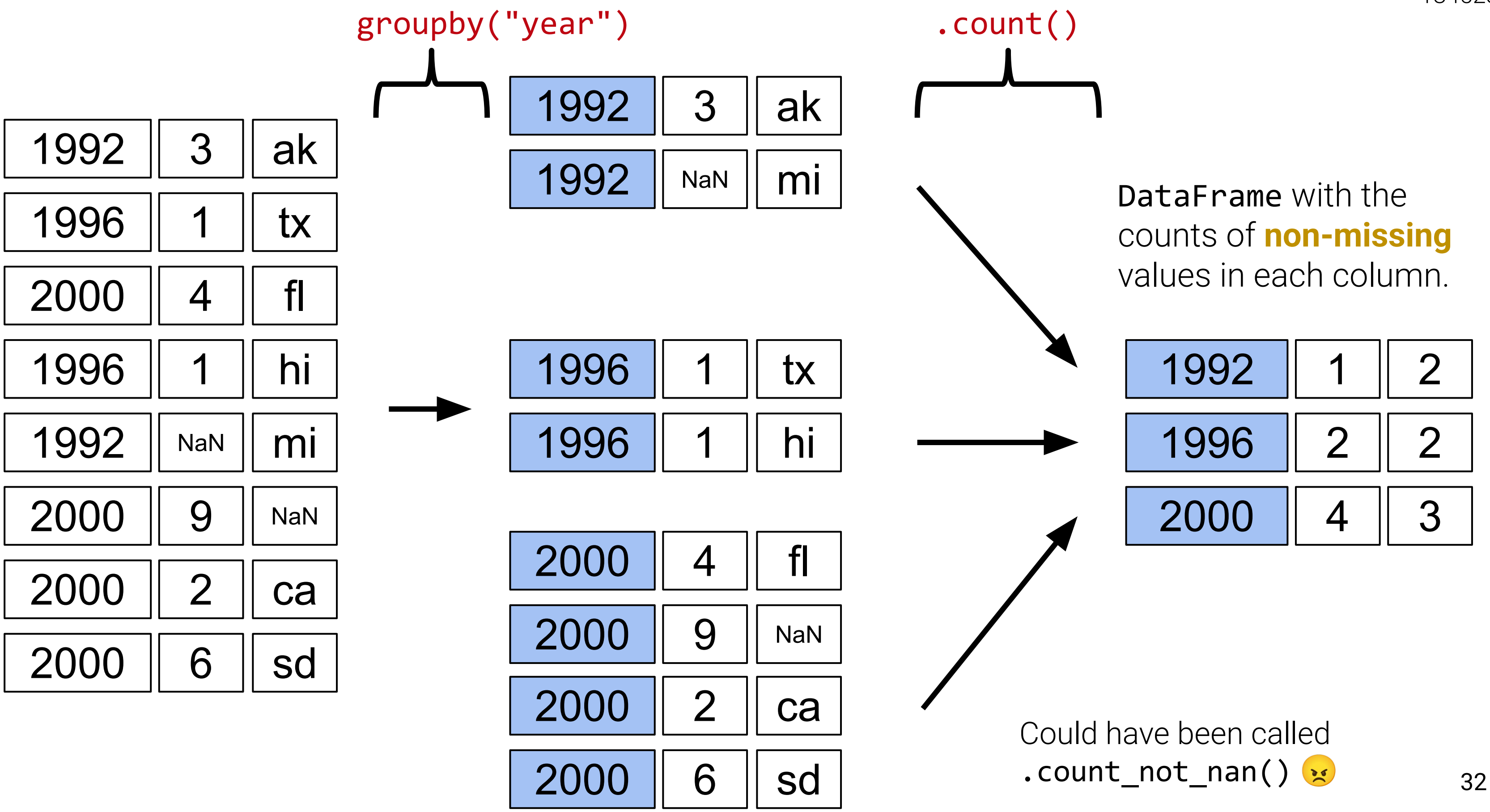


`Series` object with # of rows in each group.

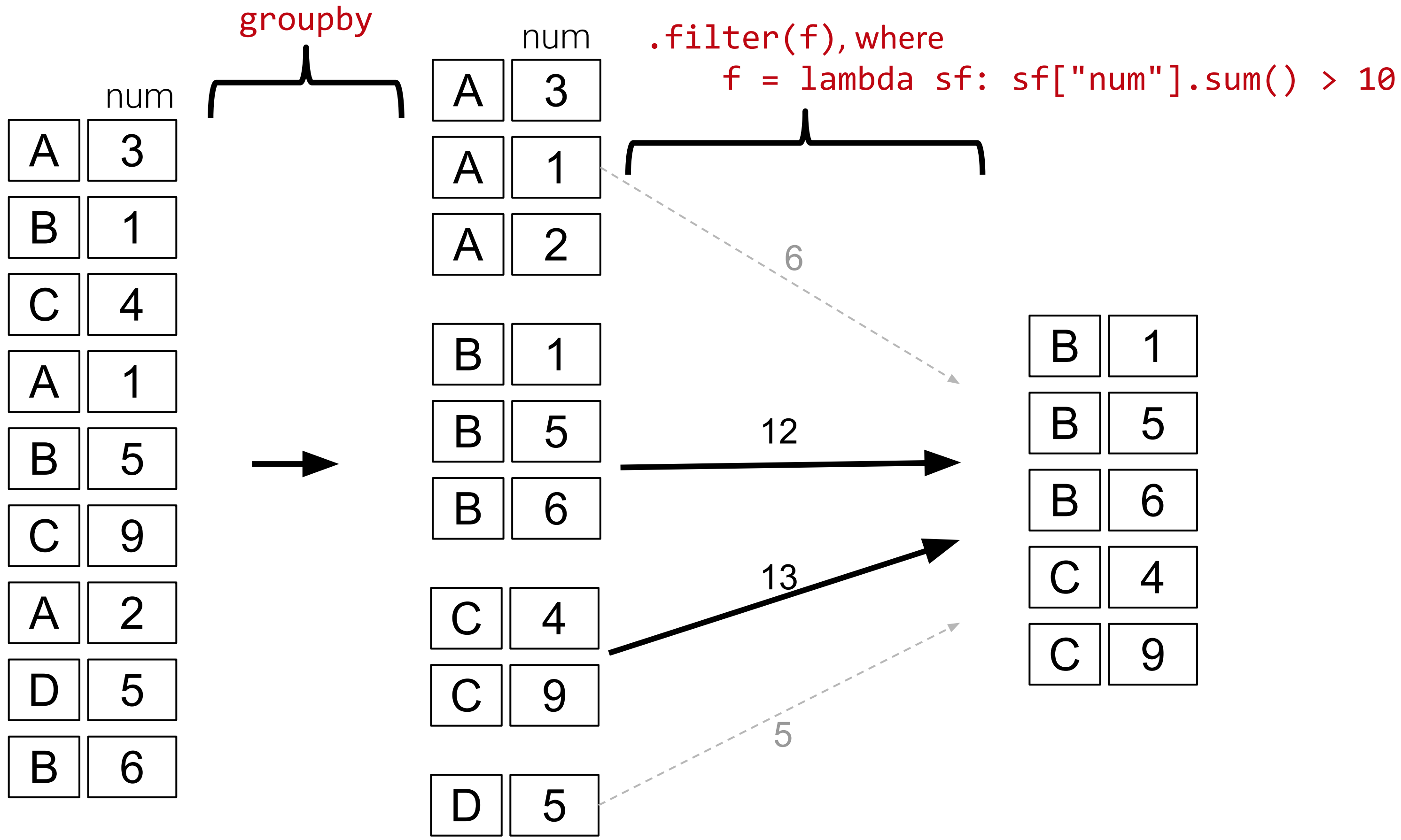
Similar to `value_counts()` but `size()` does not sort.

`.count()`

1845239



.filter()



groupby
num
.filter(f), where
f = lambda sf: sf["num"].sum() > 10
A 3
B 1
C 4
A 1
B 5
C 9
A 2
D 5
B 6
A 3
A 1
A 2
B 1
B 5
B 6
C 4
C 9
D 5
6
12
13
5
B 1
B 5
B 6
C 4
C 9

Filering by Group

1845239

Another common use for groups is to **filter** data.

- `groupby(___).filter(func)`
- Filtering is done per **group**, not per row.
- `filter` applies `func` to each group's sub-`DataFrame` (`sf`):
  - If `func` returns **True** for a `sf`, then all rows belonging to the group are **preserved**.
  - If `func` returns **False** for a `sf`, then all rows belonging to that group are **filtered out**.
- `func` must return a single `True` or `False` for each `sf`.

Unlike `.agg()`, the column we grouped on does NOT become the index!

Which of the following returns all rows of `babynames` with names that appeared for the first time after 2010?

Filtering Elections Dataset

1845239

Filtering to the years of `elections` where the max winning percentage is less than 45%.

```
elections.groupby("Year").filter(lambda sf: sf["%"].max() < 45)
```

	Year	Candidate	Party	Popular vote	Result	%
23	1860	Abraham Lincoln	Republican	1855993	win	39.699408
24	1860	John Bell	Constitutional Union	590901	loss	12.639283
25	1860	John C. Breckinridge	Southern Democratic	848019	loss	18.138998
26	1860	Stephen A. Douglas	Northern Democratic	1380202	loss	29.522311
66	1912	Eugene V. Debs	Socialist	901551	loss	6.004354
67	1912	Eugene W. Chafin	Prohibition	208156	loss	1.386325
68	1912	Theodore Roosevelt	Progressive	4122721	loss	27.457433
69	1912	William Taft	Republican	3486242	loss	23.218466
70	1912	Woodrow Wilson	Democratic	6296284	win	41.933422
115	1968	George Wallace	American Independent	9901118	loss	13.571218

`groupby` Puzzle

Puzzle: We want to know the **best election by each party**.

- Best election: The election with the highest `%` of votes.
- For example, Democrat's best election was in 1964, with candidate Lyndon Johnson winning 61.3% of votes.

	Year	Candidate	Popular vote	Result	%
Party					
American	1856	Millard Fillmore	873053	loss	21.554001
American Independent	1968	George Wallace	9901118	loss	13.571218
Anti-Masonic	1832	William Wirt	100715	loss	7.821583
Anti-Monopoly	1884	Benjamin Butler	134294	loss	1.335838
Citizens	1980	Barry Commoner	233052	loss	0.270182
Communist	1932	William Z. Foster	103307	loss	0.261069
Constitution	2008	Chuck Baldwin	199750	loss	0.152398
Constitutional Union	1860	John Bell	590901	loss	12.639283
Democratic	1964	Lyndon Johnson	43127041	win	61.344703

Attempt #1: Discuss!

1845239

Why does the table seem to claim that Woodrow Wilson won the presidency in 2020?

```python
elections.groupby("Party").max().head(10)
```

	Year	Candidate	Popular vote	Result	%
Party					
American	1976	Thomas J. Anderson	873053	loss	21.554001
American Independent	1976	Lester Maddox	9901118	loss	13.571218
Anti-Masonic	1832	William Wirt	100715	loss	7.821583
Anti-Monopoly	1884	Benjamin Butler	134294	loss	1.335838
Citizens	1980	Barry Commoner	233052	loss	0.270182
Communist	1932	William Z. Foster	103307	loss	0.261069
Constitution	2016	Michael Peroutka	203091	loss	0.152398
Constitutional Union	1860	John Bell	590901	loss	12.639283
Democratic	2020	Woodrow Wilson	81268924	win	61.344703
Democratic-Republican	1824	John Quincy Adams	151271	win	57.210122

Problem with Attempt #1

1845239

Why does the table seem to claim that Woodrow Wilson won the presidency in 2020?

```
elections.groupby("Party").max().head(10)
```

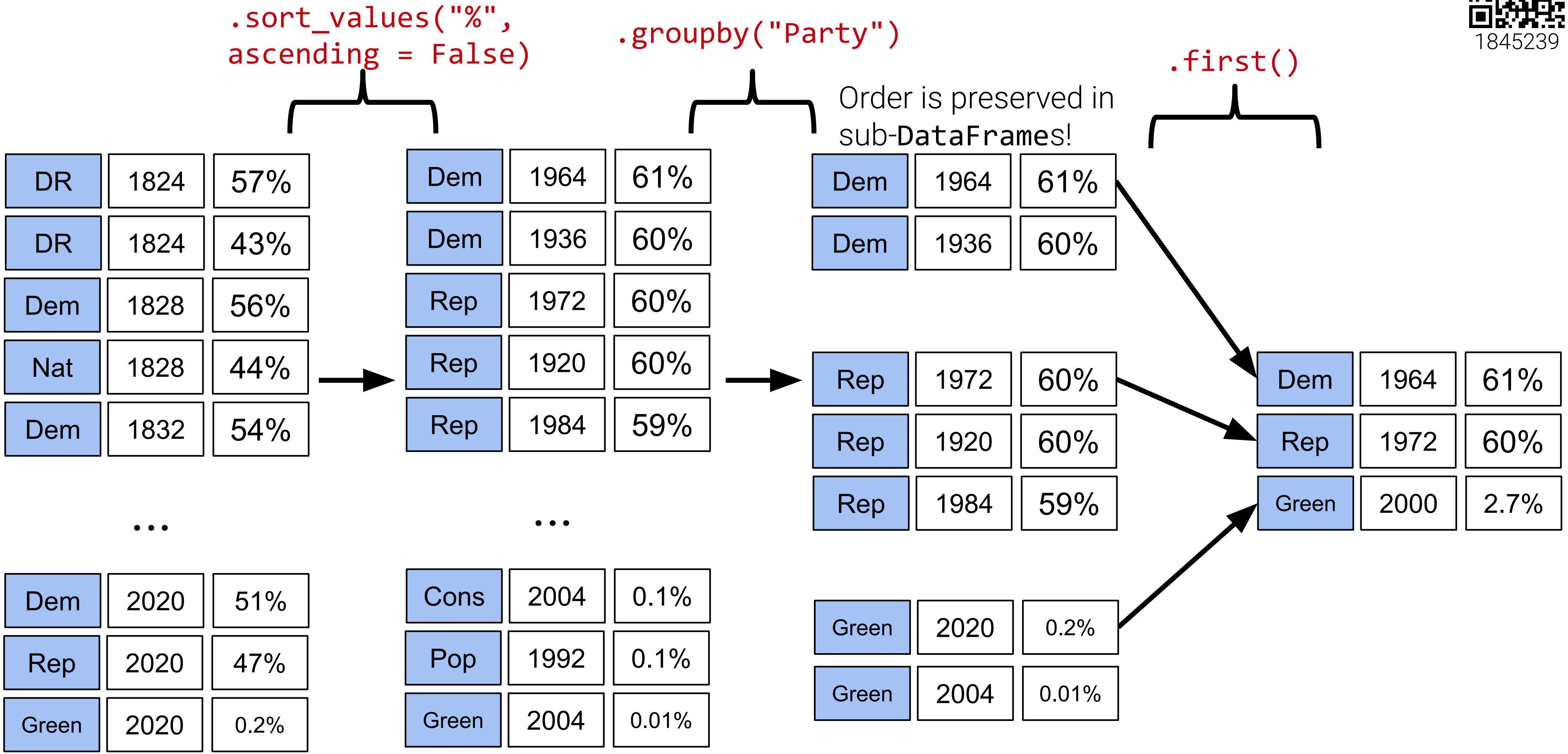
Every column is calculated independently! Among Democrats:

- Last year they ran: 2020.
- Alphabetically the latest candidate name: Woodrow Wilson.
- Highest % of vote: 61.34%.

Party	Year	Candidate	Popular vote	Result	%
American	1976	Thomas J. Anderson	873053	loss	21.554001
American Independent	1976	Lester Maddox	9901118	loss	13.571218
Anti-Masonic	1832	William Wirt	100715	loss	7.821583
Anti-Monopoly	1884	Benjamin Butler	134294	loss	1.335838
Citizens	1980	Barry Commoner	233052	loss	0.270182
Communist	1932	William Z. Foster	103307	loss	0.261069
Constitution	2016	Michael Peroutka	203091	loss	0.152398
Constitutional Union	1860	John Bell	590901	loss	12.639283
Democratic	2020	Woodrow Wilson	81268924	win	61.344703
Democratic-Republican	1824	John Quincy Adams	151271	win	57.210122

Attempt #2

1845239



Attempt #2

1845239

1. Sort the `DataFrame` so that rows are in descending order of `%`.

2. Group by Party and take the first item of each sub-`DataFrame`.

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

elections_sorted_by_percent.groupby("Party").first()
```

	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

`elections_sorted_by_percent`

→

	Year	Candidate	Popular vote	Result	%
Party					
American	1856	Millard Fillmore	873053	loss	21.554001
American Independent	1968	George Wallace	9901118	loss	13.571218
Anti-Masonic	1832	William Wirt	100715	loss	7.821583
Anti-Monopoly	1884	Benjamin Butler	134294	loss	1.335838
Citizens	1980	Barry Commoner	233052	loss	0.270182
Communist	1932	William Z. Foster	103307	loss	0.261069
Constitution	2008	Chuck Baldwin	199750	loss	0.152398
Constitutional Union	1860	John Bell	590901	loss	12.639283
Democratic	1964	Lyndon Johnson	43127041	win	61.344703

1845239

Using a **lambda** function

```
elections_sorted_by_percent = elections.sort_values("%", ascending=False)
elections_sorted_by_percent.groupby("Party").agg(lambda x : x.iloc[0])
```

Using **idxmax** function

```
best_per_party = elections.loc[elections.groupby("Party")["%"].idxmax()]
```

Using **drop_duplicates** function

```
best_per_party2 = elections.sort_values("%").drop_duplicates(["Party"], keep="last")
```

More on `DataFrameGroupby` Object

1845239

We can look into **`DataFrameGroupby`** objects in following ways:

```
grouped_by_party = elections.groupby("Party")
grouped_by_party.groups
```

```
{'American': [22, 126], 'American Independent': [115, 119, 124], 'Anti-Masonic': [6], 'Anti-Monopoly': [38], 'Citize
ns': [127], 'Communist': [89], 'Constitution': [160, 164, 172], 'Constitutional Union': [24], 'Democratic': [2, 4,
8, 10, 13, 14, 17, 20, 28, 29, 34, 37, 39, 45, 47, 52, 55, 57, 64, 70, 74, 77, 81, 83, 86, 91, 94, 97, 100, 105, 10
8, 111, 114, 116, 118, 123, 129, 134, 137, 140, 144, 151, 158, 162, 168, 176, 178, 183], 'Democratic-Republican':
[0, 1], 'Dixiecrat': [103], 'Farmer-Labor': [78], 'Free Soil': [15, 18], 'Green': [149, 155, 156, 165, 170, 177, 18
1, 184], 'Greenback': [35], 'Independent': [121, 130, 143, 161, 167, 174, 185], 'Liberal Republican': [31], 'Liberta
rian': [125, 128, 132, 138, 139, 146, 153, 159, 163, 169, 175, 180], 'Libertarian Party': [186], 'National Democrati
c': [50], 'National Republican': [3, 5], 'National Union': [27], 'Natural Law': [148], 'New Alliance': [136], 'North
ern Democratic': [26], 'Populist': [48, 61, 141], 'Progressive': [68, 82, 101, 107], 'Prohibition': [41, 44, 49, 51,
54, 59, 63, 67, 73, 75, 99], 'Reform': [150, 154], 'Republican': [21, 23, 30, 32, 33, 36, 40, 43, 46, 53, 56, 60, 6
5, 69, 72, 79, 80, 84, 87, 90, 96, 98, 104, 106, 109, 112, 113, 117, 120, 122, 131, 133, 135, 142, 145, 152, 157, 16
6, 171, 173, 179, 182], 'Socialist': [58, 62, 66, 71, 76, 85, 88, 92, 95, 102], 'Southern Democratic': [25], 'State
s' Rights': [110], 'Taxpayers': [147], 'Union': [93], 'Union Labor': [42], 'Whig': [7, 9, 11, 12, 16, 19]}
```

```
grouped_by_party.get_group("Socialist")
```

	Year	Candidate	Party	Popular vote	Result	%
58	1904	Eugene V. Debs	Socialist	402810	loss	2.985897
62	1908	Eugene V. Debs	Socialist	420852	loss	2.850866
66	1912	Eugene V. Debs	Socialist	901551	loss	6.004354
71	1916	Allan L. Benson	Socialist	590524	loss	3.194193

1845239

Pivot Tables

Lecture 4, Data 100 Summer 2025

- Grouping
- **Pivot Tables**
- Joining Tables

Grouping by Multiple Columns

1845239

We want the total # of babies born of each **sex** in each **year**. One way is to **group both columns** of interest:

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

		Count
Year	**Sex**	
1910	**F**	5950
	M	3213
1911	**F**	6602
	M	3381
1912	**F**	9804
	M	8142

Note: Resulting DataFrame is **multi-indexed**. That is, its index has multiple dimensions. Will explore in a later lecture.

Why `[["Count"]]` and not `["Count"]`? Both will work! `["Count"]` returns a `Series`.

Pivot Tables

1845239

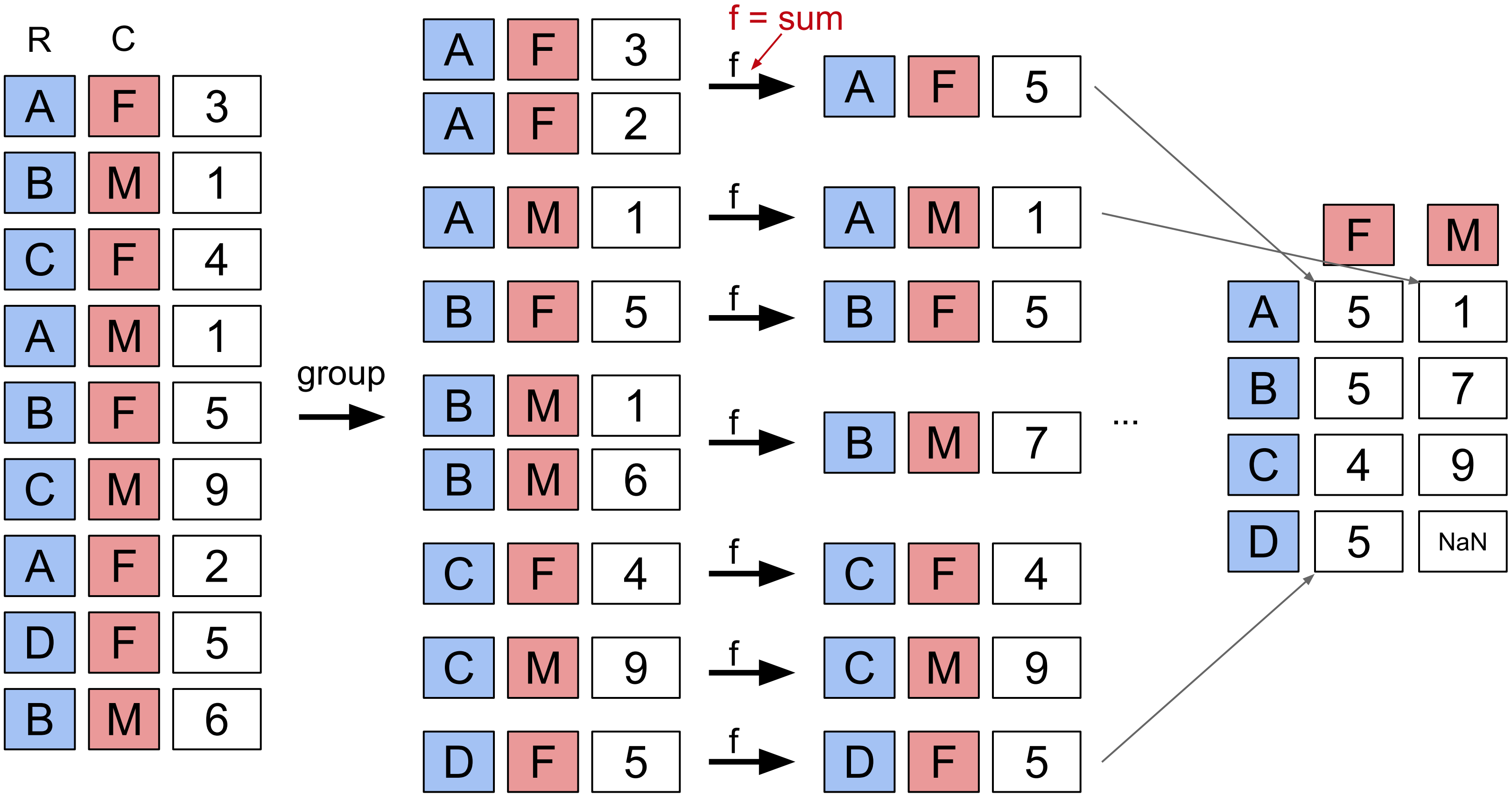
Another approach is to create a pivot table.

```
babynames_pivot = babynames.pivot_table(
    index = "Year",      # rows (turned into index)
    columns = "Sex",     # column values
    values = ["Count"], # field(s) to process in each group
    aggfunc = np.sum,    # group operation
)
babynames_pivot.head(6)
```

Sex	F	M
Year		
1910	5950	3213
1911	6602	3381
1912	9804	8142
1913	11860	10234
1914	13815	13111
1915	18643	17192

Pivot tables are especially useful as a "final" result (e.g., a lookup table provided to the public). When plotting data, it's often easier to use the output of `.groupby()`. See Tidy data paradigm (i.e., "one row per data point").

Pivot Table Mechanics



Pivot Tables with Multiple Values

1845239

We can include multiple values in our pivot tables.

```
babynames_pivot = babynames.pivot_table(
    index = "Year",      # rows (turned into index)
    columns = "Sex",     # column values
    values = ["Count", "Name"],
    aggfunc = np.max,    # group operation
)
babynames_pivot.head(6)
```

	Count		Name	
Sex	**F**	**M**	**F**	**M**
Year				
1910	295	237	Yvonne	William
1911	390	214	Zelma	Willis
1912	534	501	Yvonne	Woodrow
1913	584	614	Zelma	Yoshio
1914	773	769	Zelma	Yoshio
1915	998	1033	Zita	Yukio

Join Tables

Lecture 4, Data 100 Summer 2025

- Grouping
- Pivot Tables
- **Joining Tables**

1845239

Joining Tables

1845239

Suppose want to know the popularity of presidential candidate's names in 2022.

- Example: Dwight Eisenhower's name Dwight is not popular today, with only 5 babies born with this name in California in 2022.

To begin solving this problem, we'll have to **join** datasets.

- This will be almost exactly like `Table.join` from data 8 (Table.join - datascience 0.17.6 documentation)

Creating Table 1: Baby names in 2022

1845239

Let's set aside names in California from 2022 first:

```
babynames_2022 = babynames[babynames["Year"] == 2022]
babynames_2022
```

	State	Sex	Year	Name	Count
235835	CA	F	2022	Olivia	2178
235836	CA	F	2022	Emma	2080
235837	CA	F	2022	Camila	2046
235838	CA	F	2022	Mia	1882
235839	CA	F	2022	Sophia	1762
235840	CA	F	2022	Isabella	1733
235841	CA	F	2022	Luna	1516
235842	CA	F	2022	Sofia	1307
235843	CA	F	2022	Amelia	1289
235844	CA	F	2022	Gianna	1107

Creating Table 2: Presidents with First Names

1845239

To join our table, we'll also need to set aside the first names of each candidate.

```
elections["First Name"] = elections["Candidate"].str.split().str[0]
```

	Year	Candidate	Party	Popular vote	Result	%	First Name
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122	Andrew
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878	John
2	1828	Andrew Jackson	Democratic	642806	win	56.203927	Andrew
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073	John
4	1832	Andrew Jackson	Democratic	702735	win	54.574789	Andrew
...	...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77303568	win	49.808629	Donald
183	2024	Kamala Harris	Democratic	75019230	loss	48.336772	Kamala
184	2024	Jill Stein	Green	861155	loss	0.554864	Jill
185	2024	Robert Kennedy	Independent	756383	loss	0.487357	Robert
186	2024	Chase Oliver	Libertarian Party	650130	loss	0.418895	Chase

187 rows × 7 columns

Joining Our Tables: Two Options

1845239

```
merged = pd.merge(left = elections, right = babynames_2022,
                  left_on = "First Name", right_on = "Name")
```

```
merged = elections.merge(right = babynames_2022,
                  left_on = "First Name", right_on = "Name")
```

Alternative!

	Year_x	Candidate	Party	Popular vote	Result	%	First Name	State	Sex	Year_y	Name	Count
75	1892	Benjamin Harrison	Republican	5176108	loss	42.984101	Benjamin	CA	M	2022	Benjamin	1524
73	1884	Benjamin Butler	Anti-Monopoly	134294	loss	1.335838	Benjamin	CA	M	2022	Benjamin	1524
74	1888	Benjamin Harrison	Republican	5443633	win	47.858041	Benjamin	CA	M	2022	Benjamin	1524
45	1880	James Garfield	Republican	4453337	win	48.369234	James	CA	M	2022	James	1086
43	1880	James B. Weaver	Greenback	308649	loss	3.352344	James	CA	M	2022	James	1086
...	...	...	...	...	...	...	...	...	...	...	...	...
115	1964	Lyndon Johnson	Democratic	43127041	win	61.344703	Lyndon	CA	M	2022	Lyndon	6
92	1912	Woodrow Wilson	Democratic	6296284	win	41.933422	Woodrow	CA	M	2022	Woodrow	6
93	1916	Woodrow Wilson	Democratic	9126868	win	49.367987	Woodrow	CA	M	2022	Woodrow	6
76	1888	Clinton B. Fisk	Prohibition	249819	loss	2.196299	Clinton	CA	M	2022	Clinton	6
145	2016	Darrell Castle	Constitution	203091	loss	0.149640	Darrell	CA	M	2022	Darrell	5

152 rows × 12 columns

From here, use your new `.groupby()` tools!

Just Finished...

abcnews.go.com/Lifestyle/silly-baby-panda-falls-flat-face-public-debut/story?id=42481478

1845239

LECTURE 4

Pandas, Part III

Content credit: Acknowledgments