slido

2173124

Join at slido.com
#2173124

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

CC BY NC SA

2173124

Pandas, Part III

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

Data 100/Data 200, Spring 2025 @ UC Berkeley

Narges Norouzi and Josh Grossman

CC BY NC SA

2173124

Today's Roadmap

Lecture 4, Data 100 Spring 2025

- Custom Sorts
- Grouping
- Pivot Tables
- Joining Tables

2173124

Custom Sorts

Lecture 4, Data 100 Spring 2025

- **Custom Sorts**
- Grouping
- Pivot Tables
- Joining Tables

CC BY NC SA

Sorting DataFrame by Length - Approach 1: Using a Temporary Column

2173124

Sorting the `DataFrame` as usual:

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

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

Sorting DataFrame by Length - Approach 2: Sorting Using the `key` Argument

2173124

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

	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

CC BY NC SA

Sorting DataFrame by `dr_ea_count` - Approach 3: Sorting Using the `map` Function

2173124

Suppose we want to sort by the number of occurrences of "dr" and "ea"s.

- Use the **`Series.map`** method.

```
def dr_ea_count(string):
    return string.count('dr') + string.count('ea')

# Use map to apply dr_ea_count to each name in the "Name" column
babynames["dr_ea_count"] = babynames["Name"].map(dr_ea_count)
babynames = babynames.sort_values(by="dr_ea_count", ascending=False)
babynames.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

CC BY NC SA

2173124

Grouping

Lecture 4, Data 100 Spring 2025

- Custom Sorts
- **Grouping**
- Pivot Tables
- Joining Tables

Why Group?

2173124

Our goal:

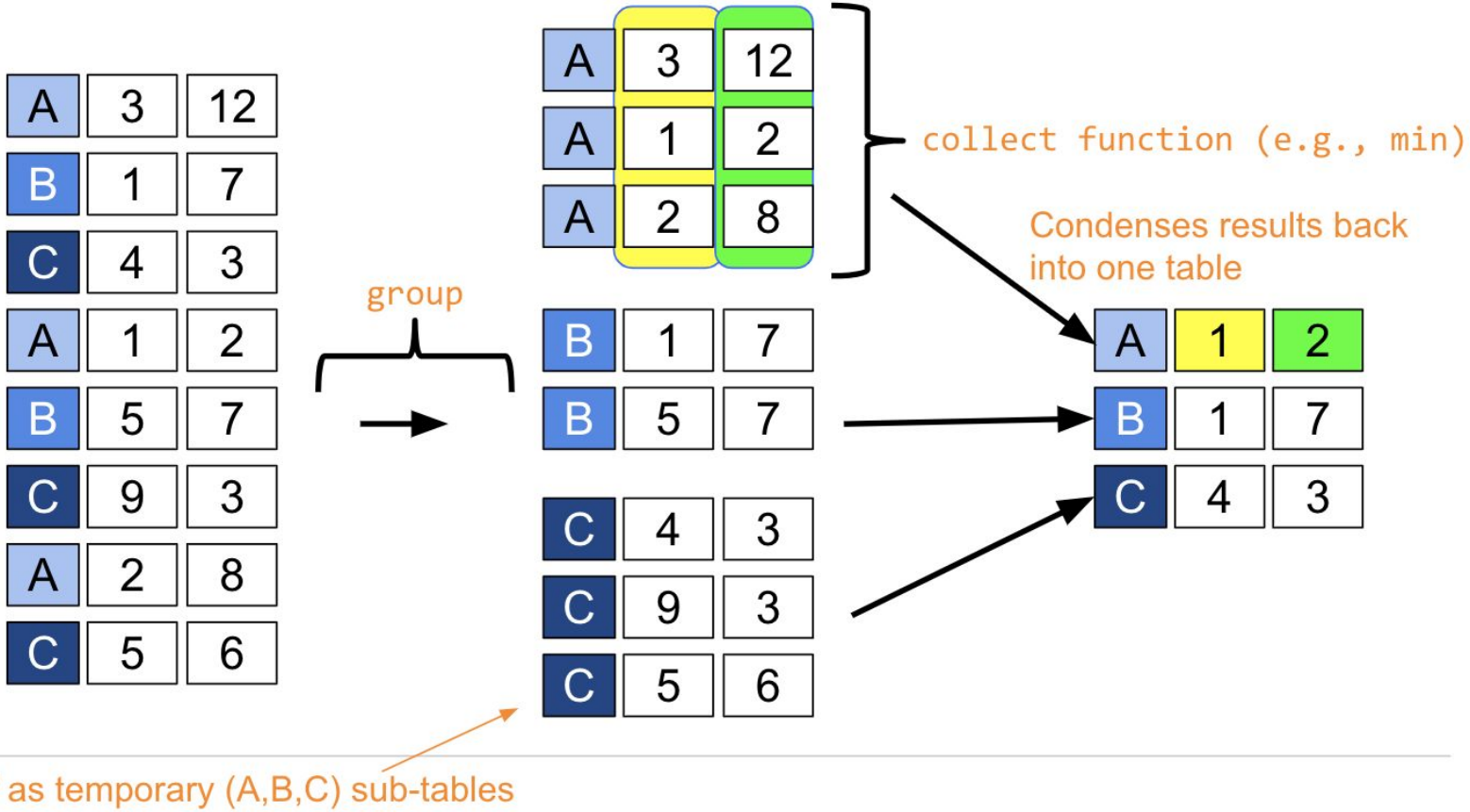
- Group together rows that fall under the **same category**.
  - For example, group together all rows from the same year.
- Perform an operation that ***aggregates*** across all rows in the category.
  - For example, sum up the total number of babies born in that year.

Grouping is a powerful tool to 1) perform large operations, all at once and 2) summarize trends in a dataset.

CC BY NC SA

Visual Review of Data 8: Grouping and Collection

2173124



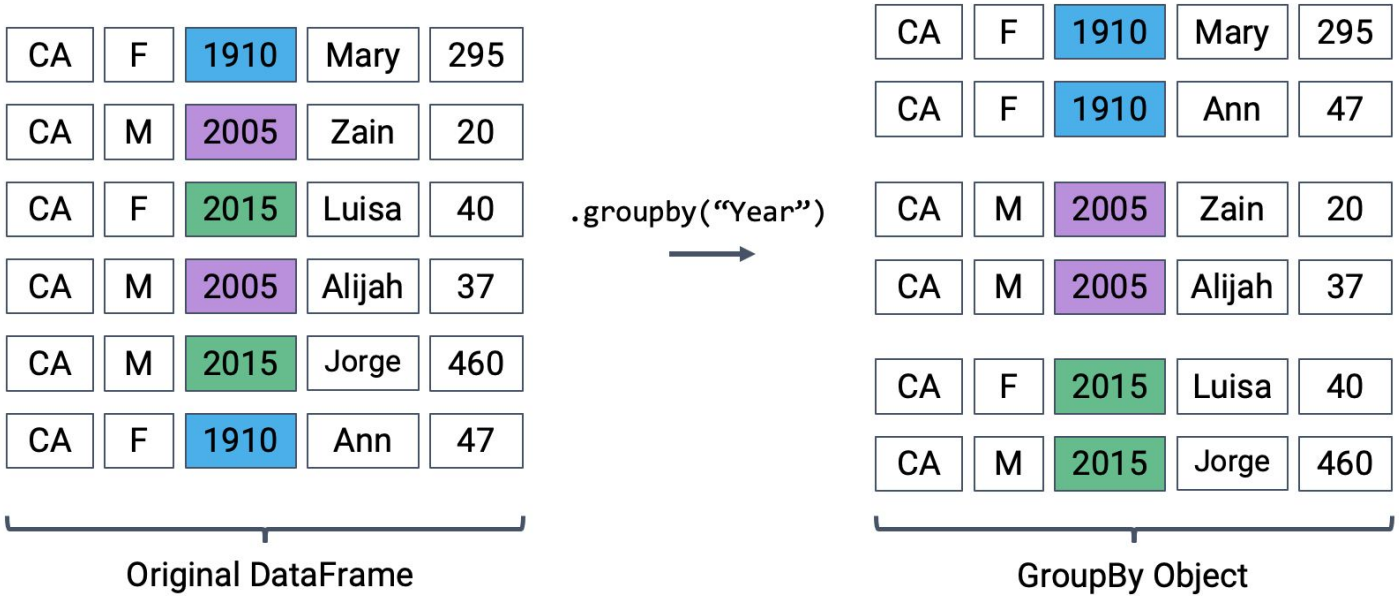
CC BY NC SA

`.groupby()`

2173124

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

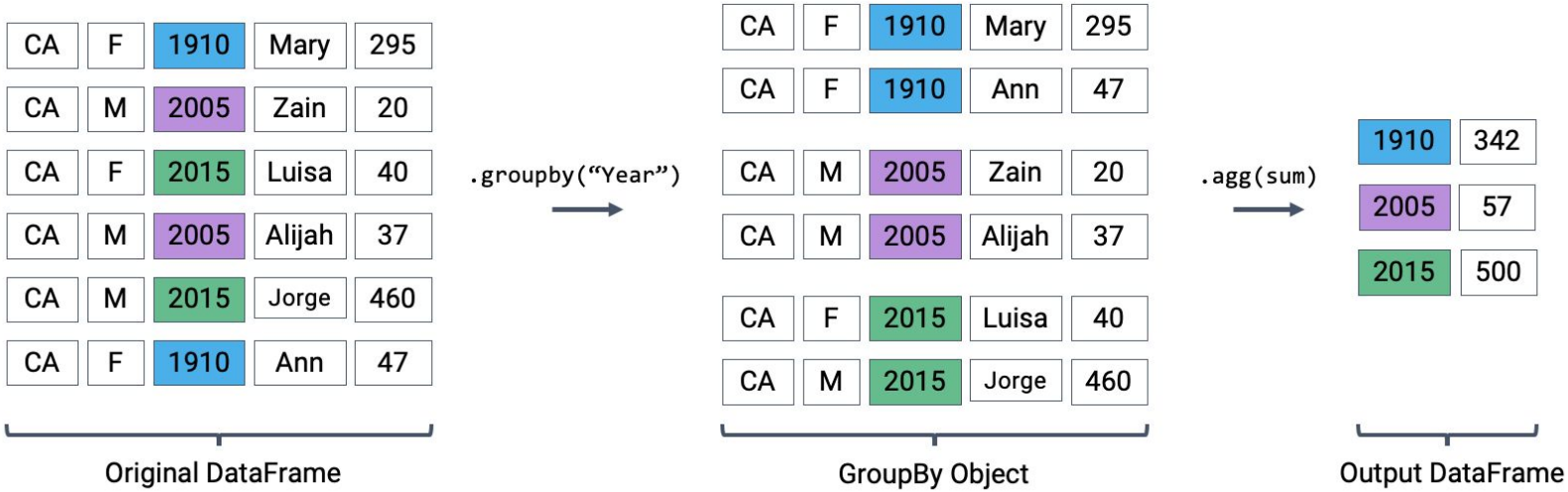
- Calling `.groupby()` generates `DataFrameGroupBy` objects → "mini" sub-DataFrames
- Each subframe contains all rows that correspond to the same group (here, a particular year)



`.groupby().agg()`

2173124

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



Where did the non-numeric columns go? We'll talk about this in a bit.

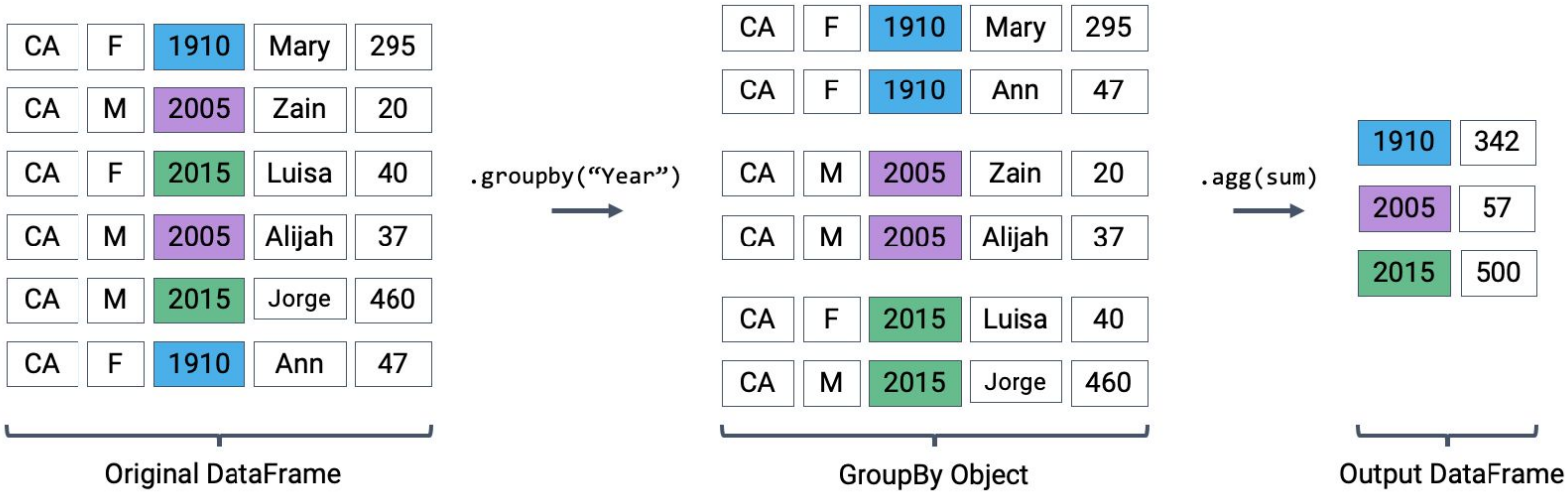
CC BY NC SA

Putting It All Together

2173124

```
dataframe.groupby(column_name).agg(aggregation_function)
```

`babynames[["Year", "Count"]].groupby("Year").agg(sum)` computes the total number of babies born in each year.



Aggregation Functions

2173124

What goes 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 commonly-used aggregation functions can even be called directly, without the explicit use of `.agg( )`

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


CC BY NC SA

Alternatives …

2173124

Now, we 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

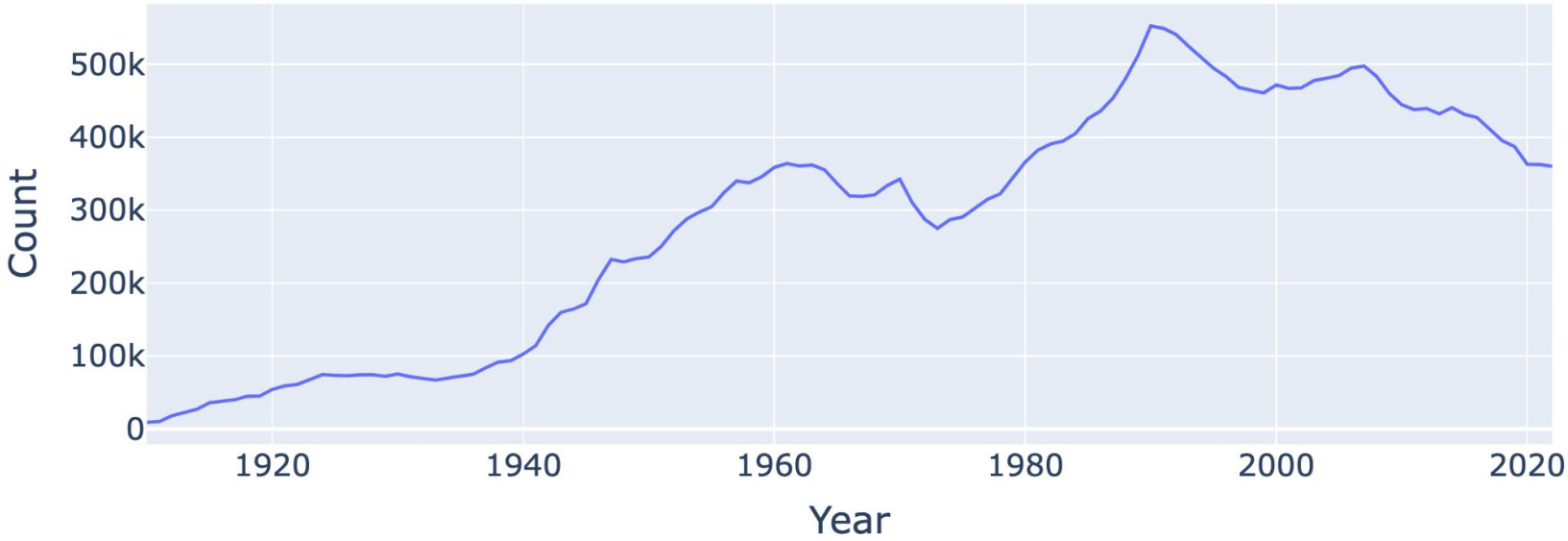
CC BY NC SA

Plotting Birth Counts

2173124

Plotting the **`DataFrame`** we just generated tells an interesting story.

```
babies_by_year = babynames.groupby("Year")[["Count"]].agg(sum)
px.line(babies_by_year, y="Count")
```



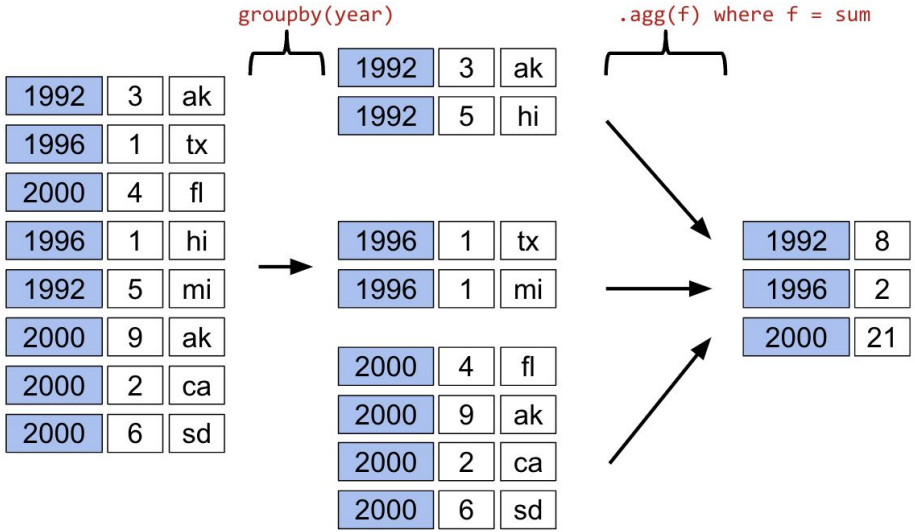
CC BY NC SA

Concluding `groupby.agg`

2173124

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

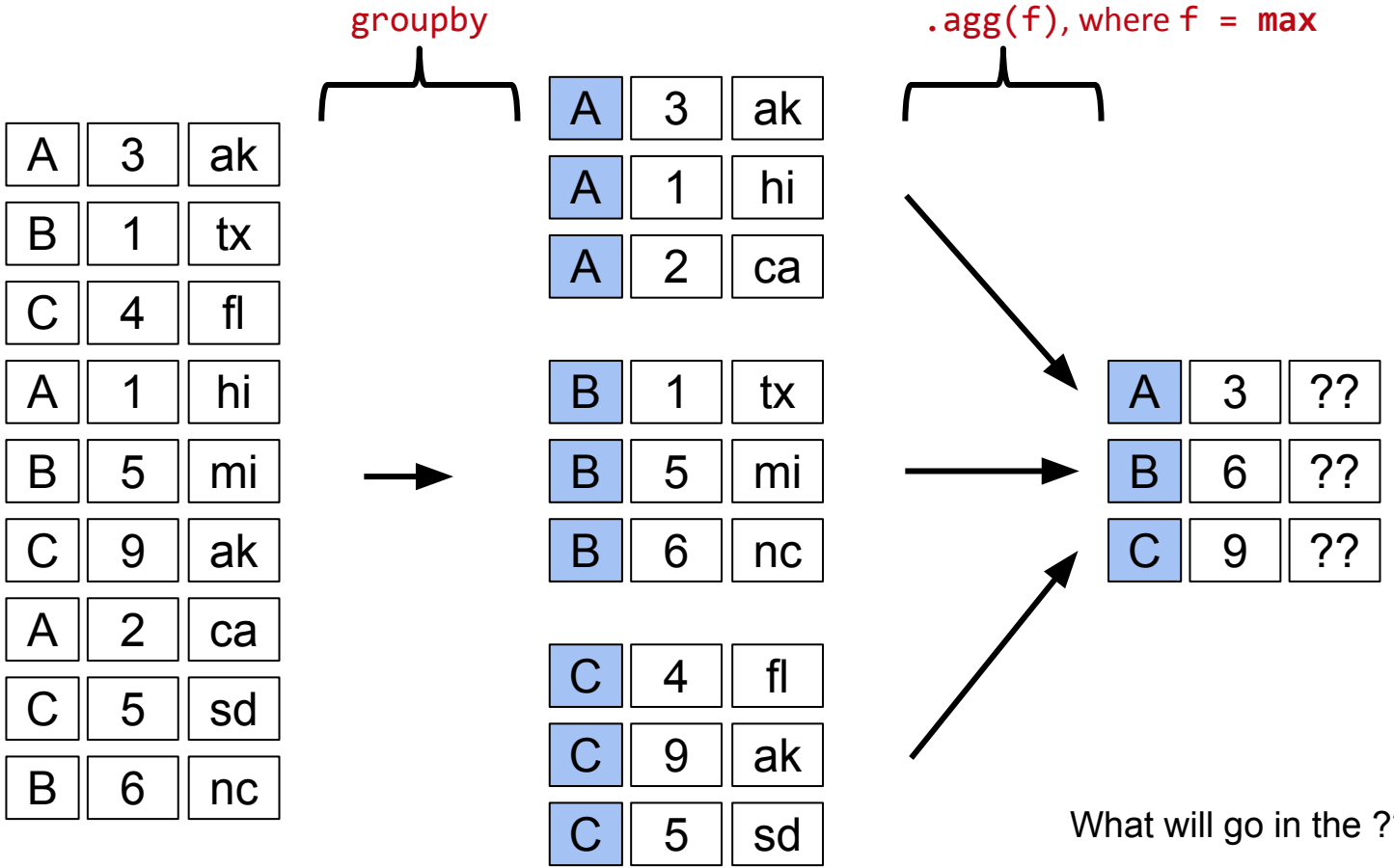
- So far, we've seen that `df.groupby("Year").agg(sum)`:
  - **Split** `df` into sub-`DataFrame`s based on `Year`.
  - **Apply** the `sum` function to each column of each sub-`DataFrame`.
  - **Combine** the results of `sum` into a single `DataFrame`, indexed by `Year`.



CC BY NC SA

Groupby Review Question

2173124



What will go in the ??

slido

2173124

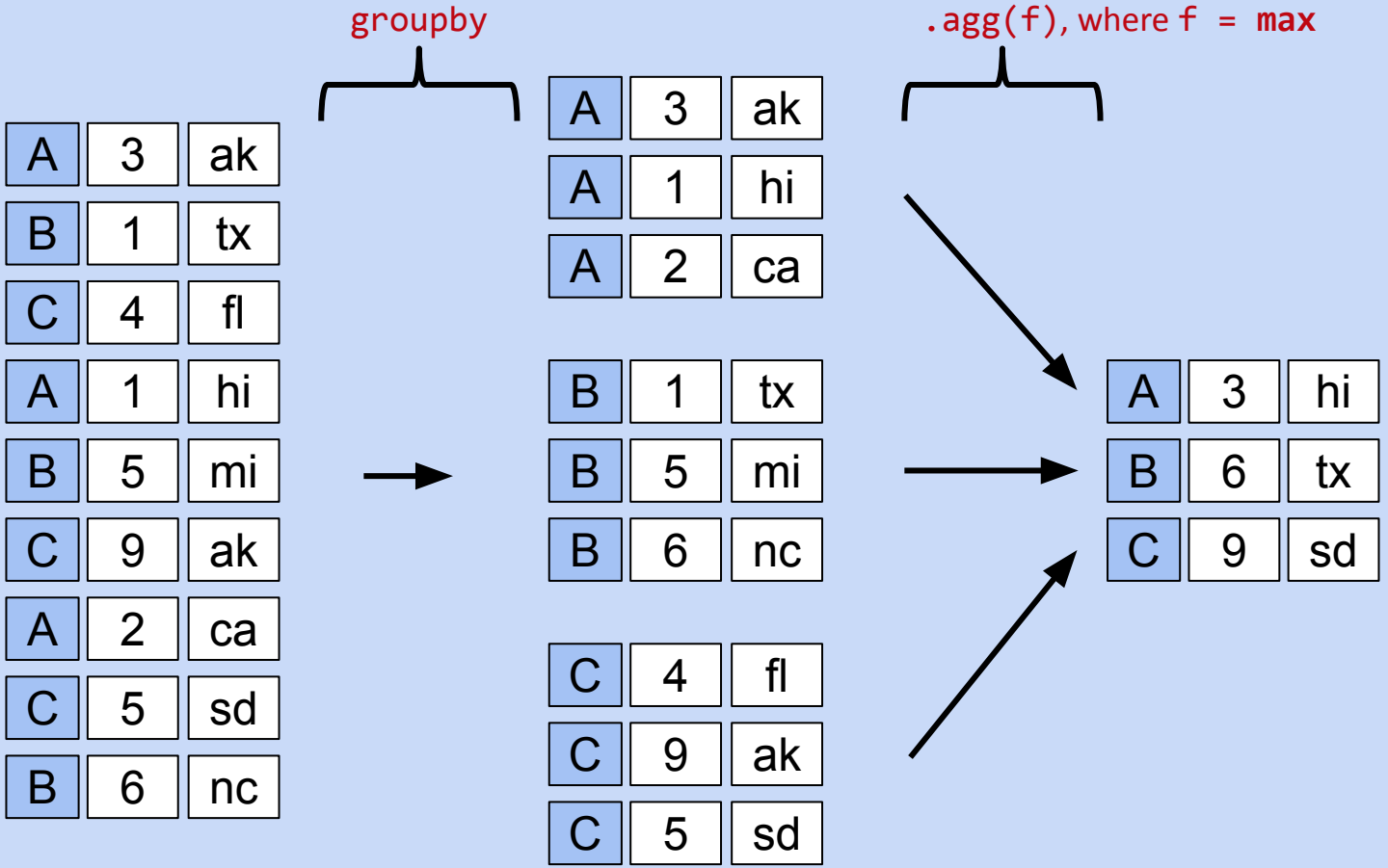
What goes in the ?? for row A?

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

CC BY NC SA

Answer

2173124



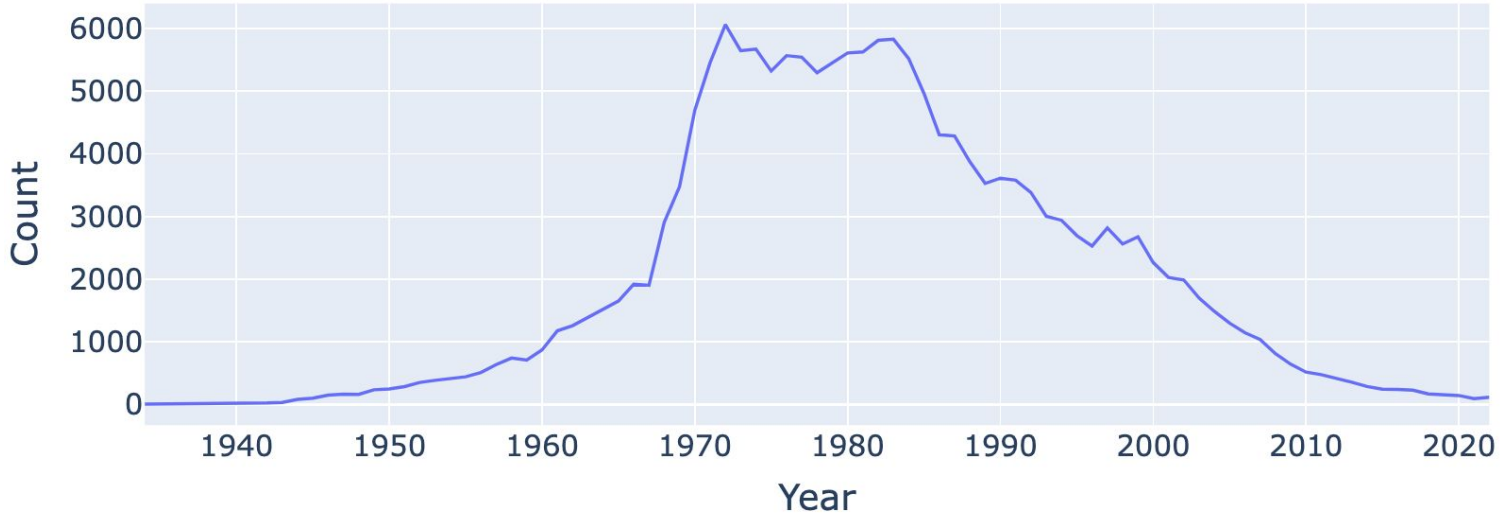
Case Study: Name "Popularity"

2173124

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

```
f_babynames = babynames[babynames["Sex"]=="F"]
f_babynames = f_babynames.sort_values(["Year"])
jenn_counts_ser = f_babynames[f_babynames["Name"]=="Jennifer"]["Count"]
```

Number of Jennifers Born in California Per Year.



CC BY NC SA

What Is "Popularity"?

2173124

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).
- The RTP is the ratio of babies born with a given name in 2022 to the *maximum* number of babies born with that name in *any* year.

Example for "Jennifer":

- In 1972, we hit peak Jennifer. 6,065 Jennifers were born.
- In 2022, there were only 114 Jennifers.
- RTP is 114 / 6065 = 0.018796372629843364.

CC BY NC SA

Calculating RTP

2173124

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

```
6065
```

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

```
114
```

Remember: `f_babynames` is sorted by year.
`.iloc[-1]` means "grab the latest year"

```
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
```


CC BY NC SA

Calculating RTP Using `.groupby()`

2173124

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

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

Name	Year	Count
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

This is the "**pandas**-ification" of logic you saw in Data 8.
Here, **`collect = rtp`**.

```
# Data 8 Tables code, not pandas
f_babynames.group("Name", ratio_to_peak)
```

Much of the logic you learned in Data 8 will serve you well in Data 100. Now, we'll implement it using **`pandas`**.

CC BY NC SA

slido

2173124

Are there any rows for which Year is not 1.0?

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

CC BY NC SA

Answer

2173124

In the rows shown, note the Year is 1.0 for every value.

Are there any rows for which Year is **not** 1.0?

A. Yes, names that appeared for the first time
B. Yes, names that did not appear in 2022.
C. Yes, names whose peak Count was in 2022
D. No, every row has a Year value of 1.0.

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

Name	Year	Count
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

CC BY NC SA

A Note on Nuisance Columns

At least as of the time of this slide creation (January 2025), executing our agg call results in a `TypeError`.

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

```
Cell In[110], line 5, in ratio_to_peak(series)
      1 def ratio_to_peak(series):
      2     """
      3     Compute the RTP for a Series containing the counts per year for a single name
      4     """
----> 5     return series.iloc[-1] / np.max(series)

TypeError: unsupported operand type(s) for /: 'str' and 'str'
```


CC BY NC SA

A Note on Nuisance Columns

2173124

Below, we explicitly select the column(s) we want to apply our aggregation function to **BEFORE** calling `agg`. This avoids the warning (and can prevent unintentional loss of data).

```
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
...	...
Zyanya	0.466667
Zyla	1.000000
Zylah	1.000000
Zyra	1.000000
Zyrah	0.833333

13782 rows × 1 columns

CC BY NC SA

Renaming Columns After Grouping

2173124

By default, `.groupby` will not rename any aggregated columns (the column is still named "Count", even though it now represents the RTP.

For better readability, we may wish to 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
...	...

CC BY NC SA

Some Data Science Payoff

2173124

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

CC BY NC SA

Some Data Science Payoff

2173124

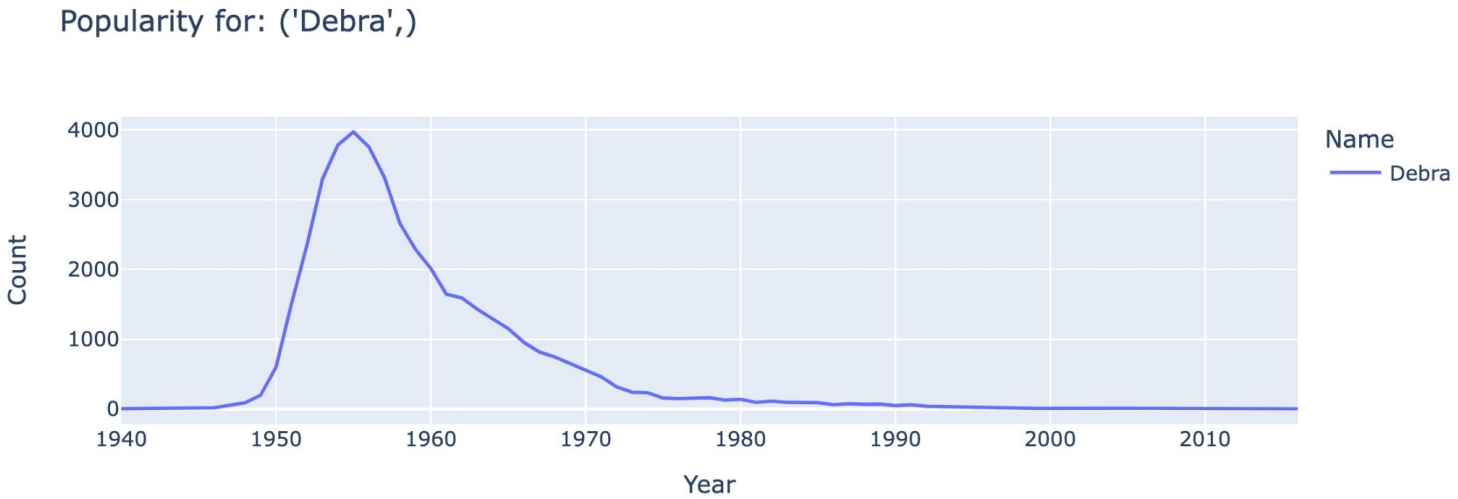
By sorting `rtp_table` we can see the names whose popularity has decreased the most.

```
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")
```



We'll learn about plotting in week 4.

Some Data Science Payoff

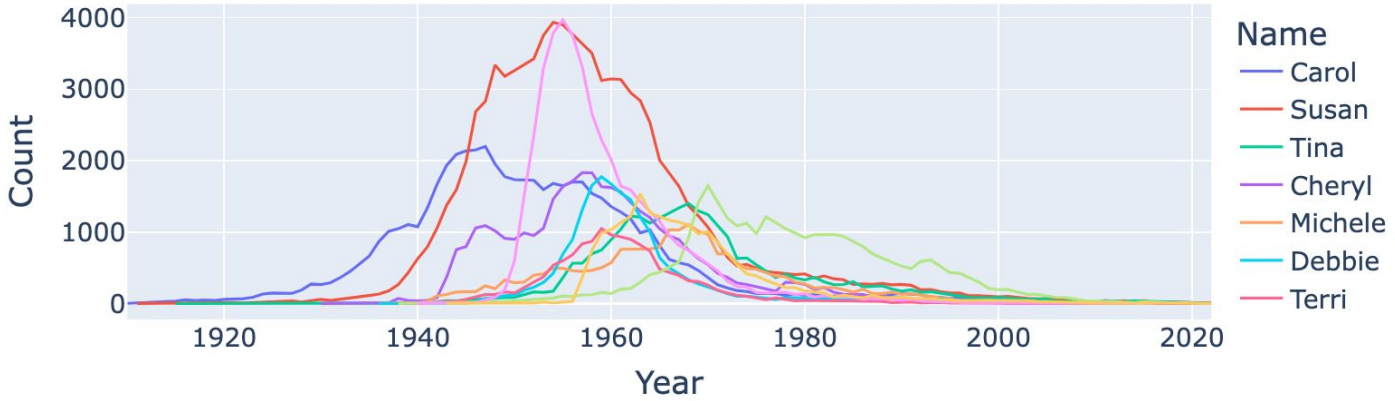
2173124

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

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

```
ndex(['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")
```



Raw `GroupBy` Objects and Other Methods

2173124

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).sum()
df.groupby(col).min()
df.groupby(col).max()
```

```
df.groupby(col).first()
df.groupby(col).last()
df.groupby(col).size()
df.groupby(col).count()
```

```
df.groupby(col).filter()
```

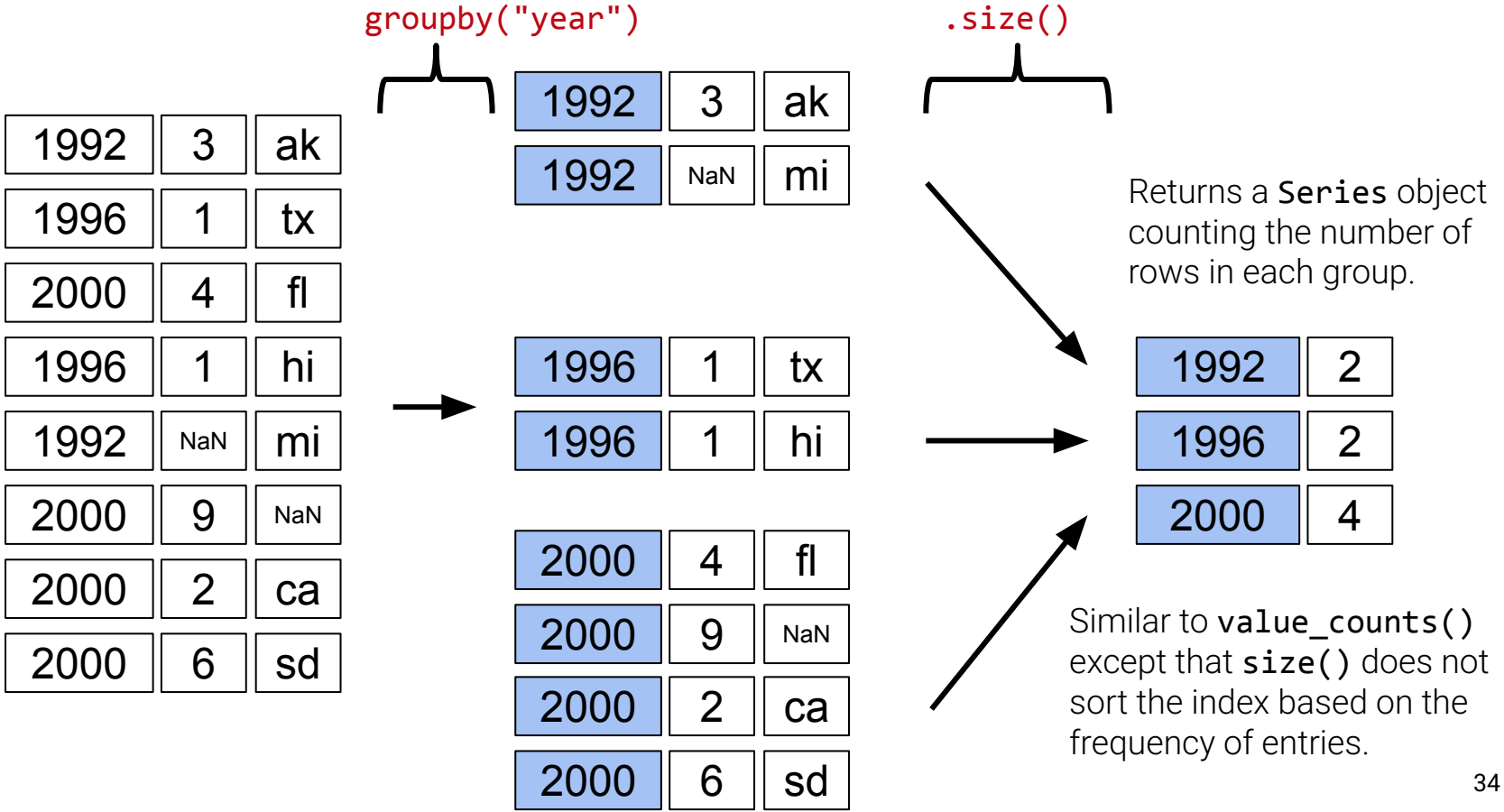
🤨What's the difference?

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

CC BY NC SA

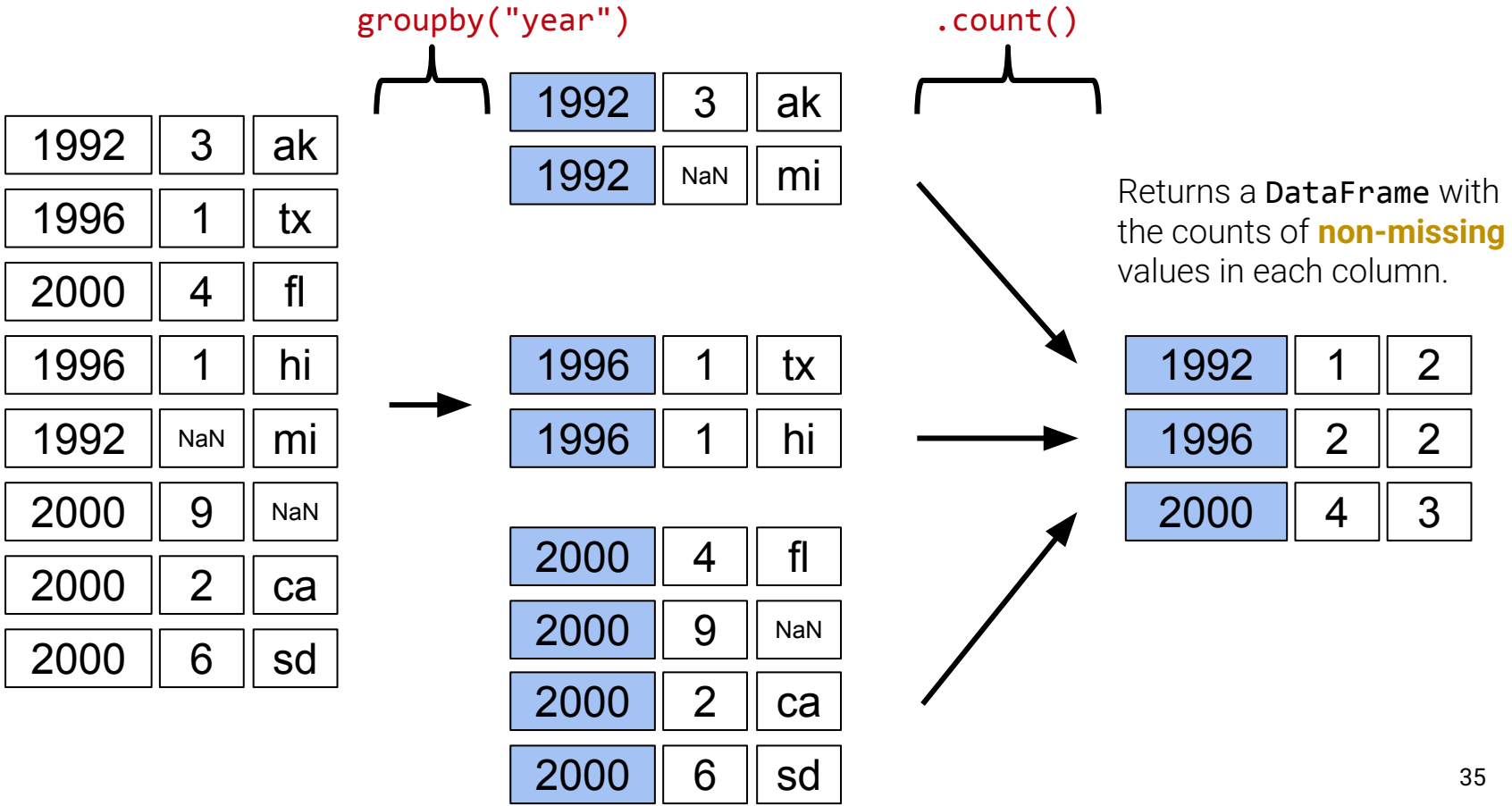
groupby.size() and groupby.count()

2173124



`groupby.size()` and `groupby.count()`

2173124



Returns a `DataFrame` with the counts of **non-missing** values in each column.

CC BY NC SA

Filtering by Group

2173124

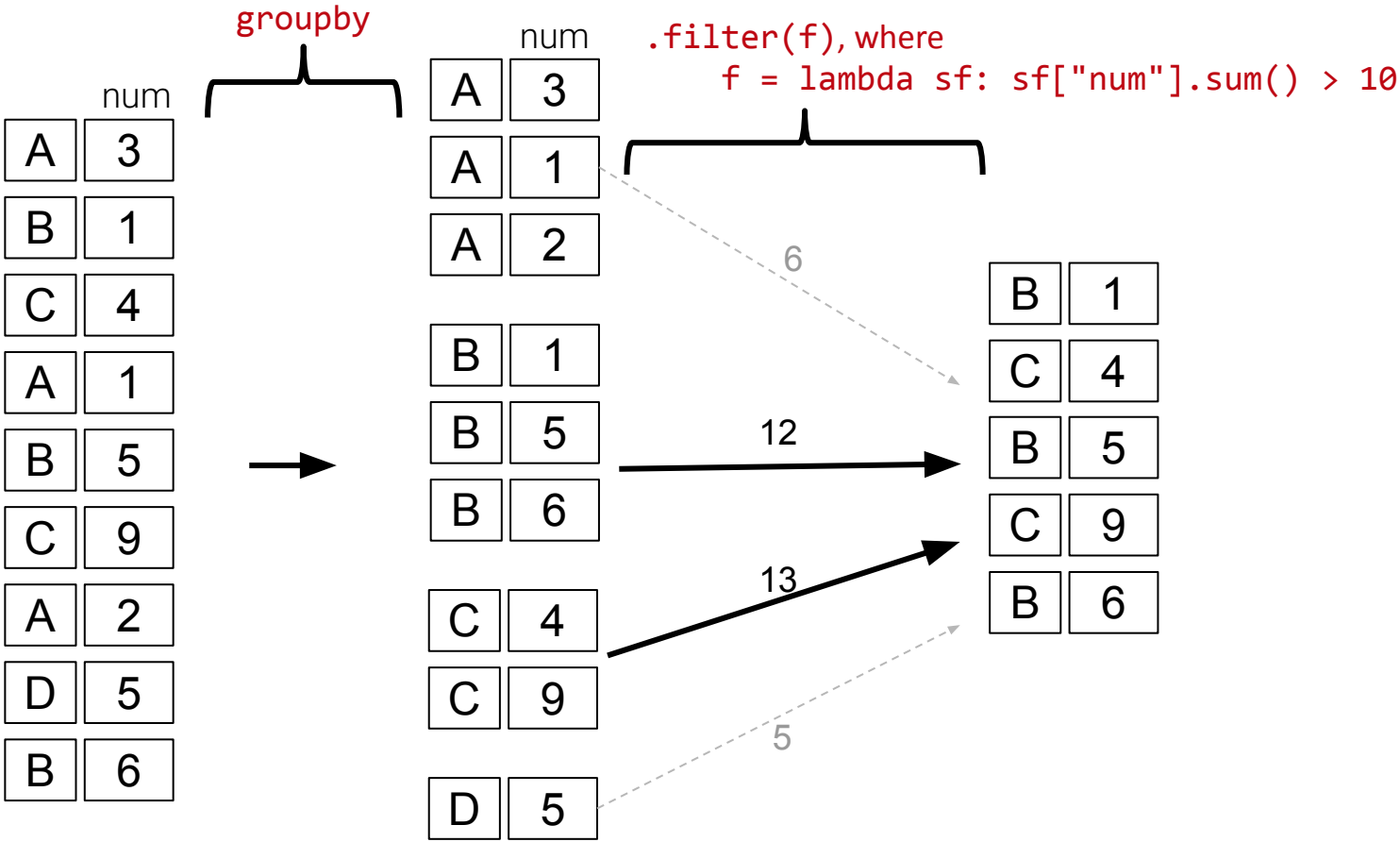
Another common use for groups is to filter data.

- `groupby.filter` takes an argument `func`.
- `func` is a function that:
  - Takes a `DataFrame` as input.
  - Returns either `True` or `False`.
- `filter` applies `func` to each group/sub-`DataFrame`:
  - If `func` returns **True** for a group, then all rows belonging to the group are **preserved**.
  - If `func` returns **False** for a group, then all rows belonging to that group are **filtered out**.
- Notes:
  - Filtering is done per group, not per row. Different from boolean filtering.
  - Unlike `agg()`, the column we grouped on does NOT become the index!

CC BY NC SA

groupby.filter()

2173124



Filtering Elections Dataset

2173124

Going back to the `elections` dataset.

Let's keep only election year results where the max '%' 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

CC BY NC SA

2173124

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.

Party	Year	Candidate	Popular vote	Result	%
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

CC BY NC SA

Attempt #1

2173124

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

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

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

Problem with Attempt #1

2173124

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

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

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

Every column is calculated independently! Among Democrats:

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

CC BY NC SA

Attempt #2: Motivation

2173124

- We want to preserve entire rows, so we need an aggregate function that does that.

Party	Year	Candidate	Popular vote	Result	%
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

CC BY NC SA

Raw `GroupBy` Objects and Other Methods

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:

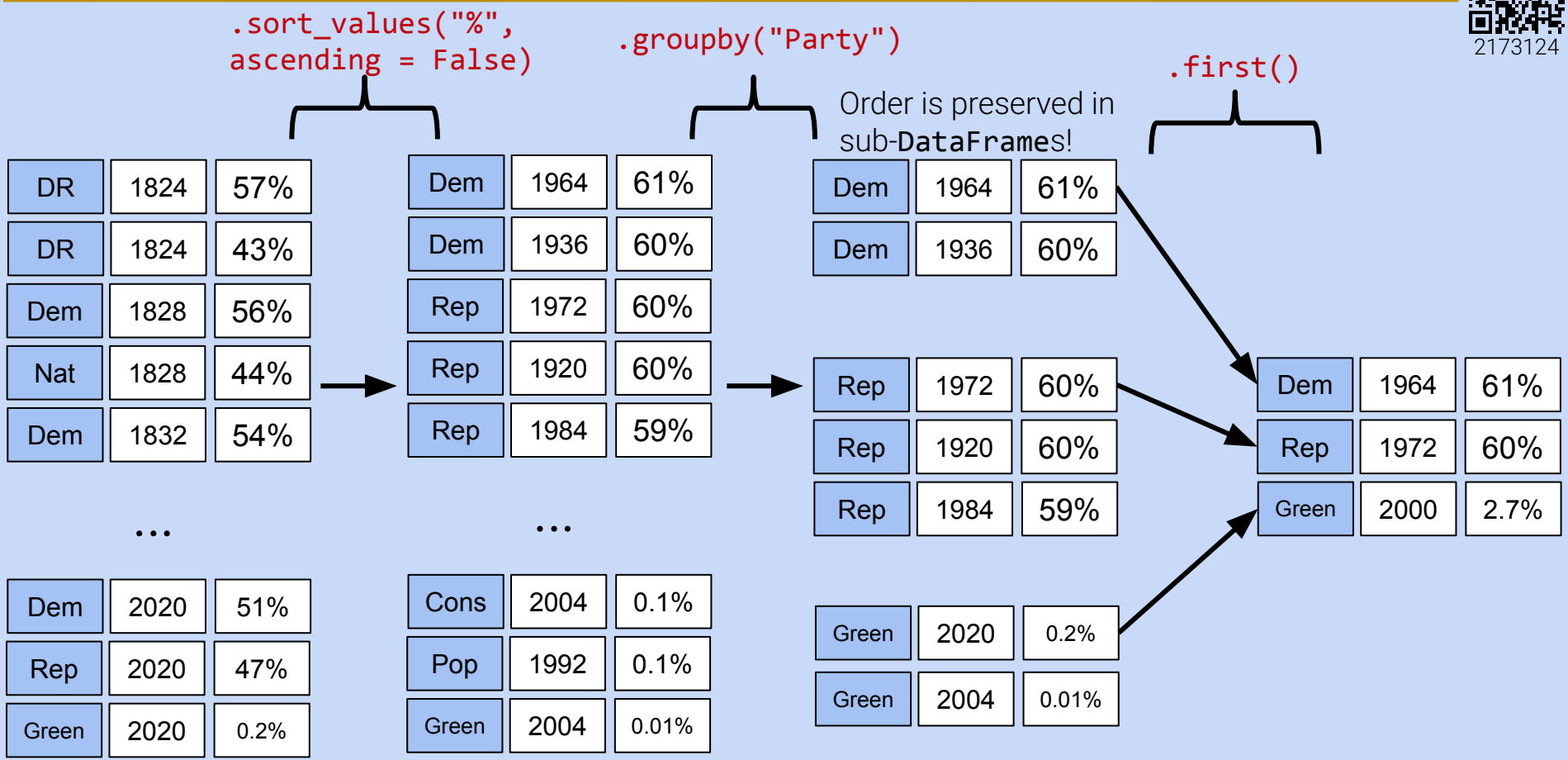
```
df.groupby(col).mean()
df.groupby(col).sum()
df.groupby(col).min()
df.groupby(col).max()
```

```
df.groupby(col).first()
df.groupby(col).last()
df.groupby(col).size()
df.groupby(col).count()
```

```
df.groupby(col).filter()
```


CC BY NC SA

Attempt #2: Solution



Attempt #2: Solution

2173124

- First sort the **`DataFrame`** so that rows are in descending order of `%`.
- Then group by Party and take the first item of each sub-**`DataFrame`**.
- Note: Lab will give you a chance to try this out if you didn't quite follow during lecture.

```
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`

Party	Year	Candidate	Popular vote	Result	%
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

CC BY NC SA

groupby Puzzle - Alternate Approaches

2173124

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")
```

In Pandas, there's more than one way to get to the same answer.

CC BY NC SA

More on `DataFrameGroupby` Object

2173124

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

CC BY NC SA

2173124

Pivot Tables

Lecture 4, Data 100 Spring 2025

- Custom Sorts
- Grouping
- **Pivot Tables**
- Joining Tables

Grouping by Multiple Columns

2173124

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

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

		Count
Year	**Sex**	
1910	**F**	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.

CC BY NC SA

Pivot Tables

A more natural approach is to use our Data 8 brains and create a pivot table.

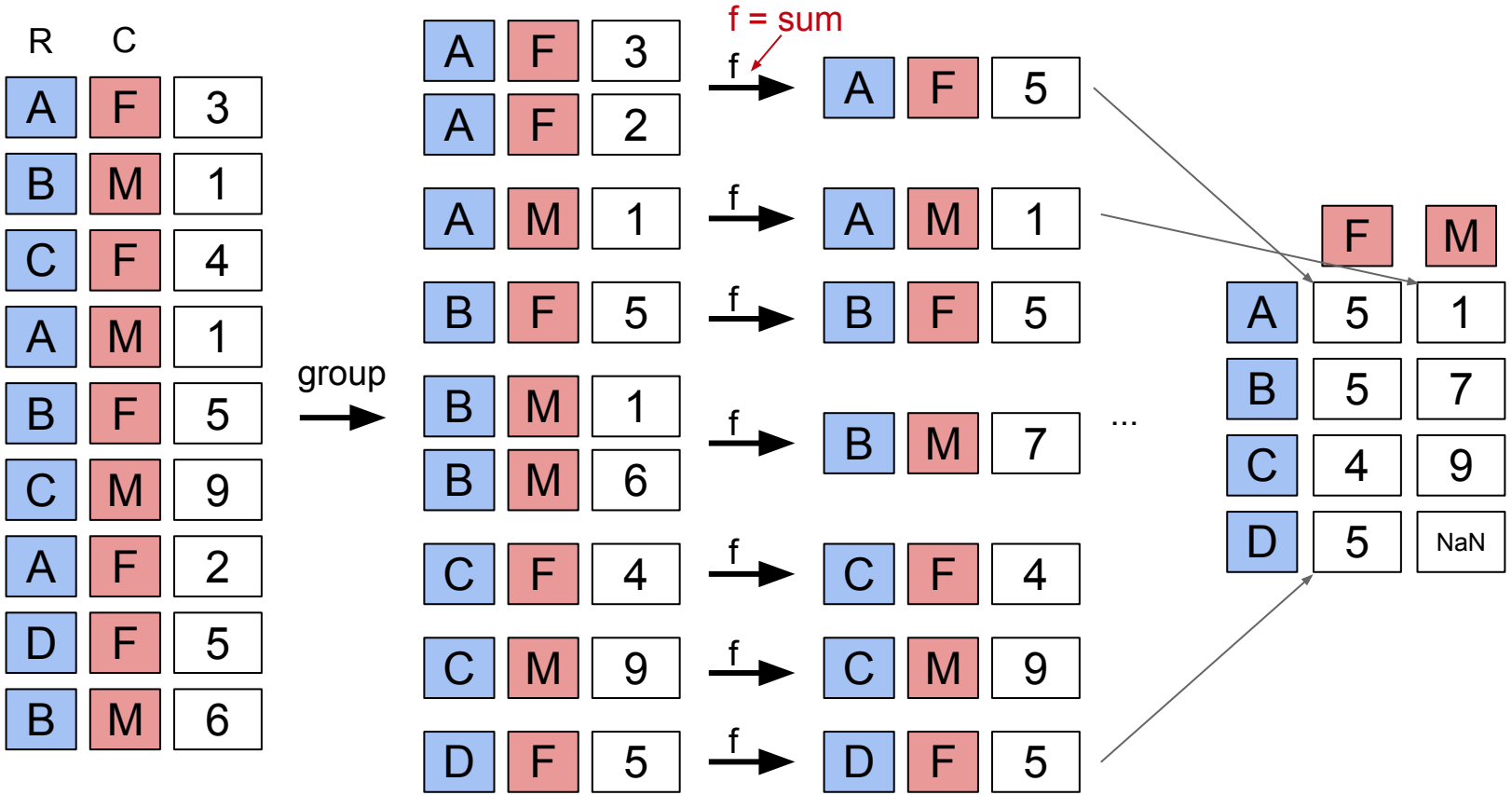
```
babynames_pivot = babynames.pivot_table(
    index = "Year",      # rows (turned into index)
    columns = "Sex",     # column values
    values = ["Count"], # field(s) to process in each group
    aggfunc = np.sum,    # group operation
)
babynames_pivot.head(6)
```

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

CC BY NC SA

Pivot Table Mechanics

2173124



Pivot Tables with Multiple Values

2173124

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

CC BY NC SA

2173124

Join Tables

Lecture 4, Data 100 Spring 2025

- Custom Sorts
- Grouping
- Pivot Tables
- **Joining Tables**

CC BY NC SA

Joining Tables

2173124

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 solve this problem, we'll have to join tables.

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

CC BY NC SA

Creating Table 1: Babynames in 2022

2173124

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

CC BY NC SA

Creating Table 2: Presidents with First Names

2173124

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

CC BY NC SA

Joining Our Tables: Two Options

2173124

```
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")
```

	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

CC BY NC SA

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

LECTURE 4

Pandas, Part III

Content credit: Acknowledgments

CC BY NC SA