LECTURE 4

Pandas, Part IV

Advanced Pandas (More on Grouping and Merging)

CSCI 3022, Fall 2023 @ CU Boulder

Maribeth Oscamou

Announcements

- Today's last day for Getting to Know You Meetings (see link in Piazza)
- HW 3 Released Tonight
- Nb 3 released tonight

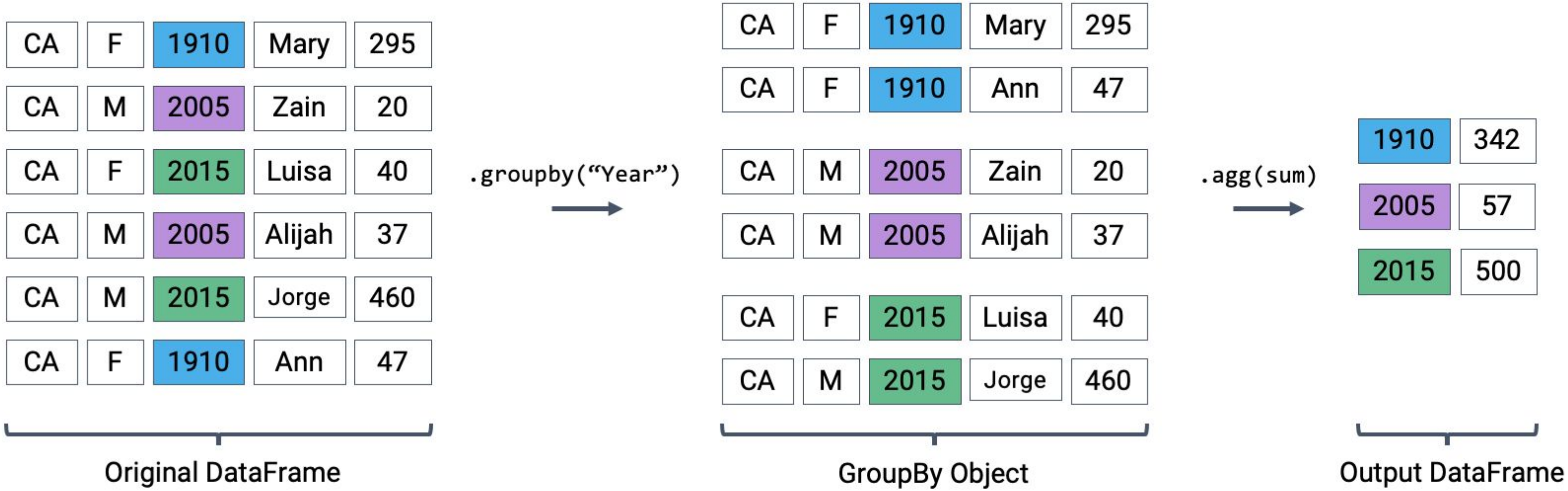
Today's Roadmap

- Pandas, Part IV
  - Groupby Review
  - Demo
  - Joining Tables
  - More on Groupby

Revisiting `groupby.agg`

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

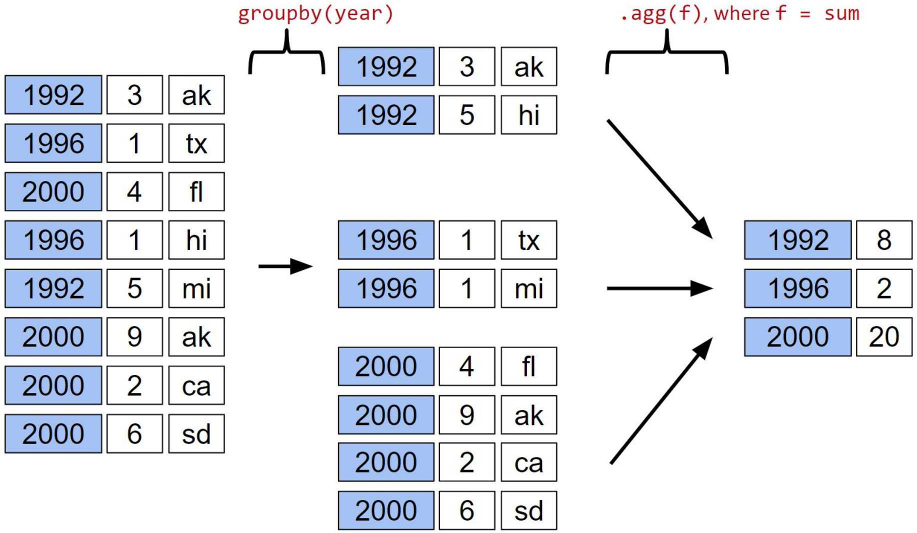
`babynames.groupby("Year")[["Count"]].agg(sum)` computes the total number of babies born in each year.



Revisiting `groupby.agg`

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`.



Aggregation Functions

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()
```

iClicker Poll

Which of the following code computes the total number of babies in the babynames dataset with each name and returns the exact output shown here? (Select all that apply)

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

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

C). `babynames.groupby("Name")[["Count"]].sum()`

D). `babynames.groupby("Name").sum(numeric_only=True)`

E). `babynames.groupby(["Name","Year"]).agg(sum)`

	Count
Name	
Aadan	18
Aadarsh	6
Aaden	647
Aadhav	27
Aadhini	6
...	...
Zymir	5
Zyon	133
Zyra	103
Zyrah	21
Zyrus	5

`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

Review: Problem with Attempt #1

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: Motivation

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

	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

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).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()
```

Attempt #2: Solution

DR	1824	57%
DR	1824	43%
Dem	1828	56%
Nat	1828	44%
Dem	1832	54%
...		
Dem	2020	51%
Rep	2020	47%
Green	2020	0.2%

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

Dem	1964	61%
Dem	1936	60%
Rep	1972	60%
Rep	1920	60%
Rep	1984	59%
...		
Cons	2004	0.1%
Pop	1992	0.1%
Green	2004	0.01%

```
.groupby("Party")
```

Order is preserved in sub-`DataFrame`s!

Dem	1964	61%
Dem	1936	60%

Rep	1972	60%
Rep	1920	60%
Rep	1984	59%

Green	2020	0.2%
Green	2004	0.01%

```
.first()
```

Dem	1964	61%
Rep	1972	60%
Green	2000	2.7%

Attempt #2: Solution

- 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`.

```
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

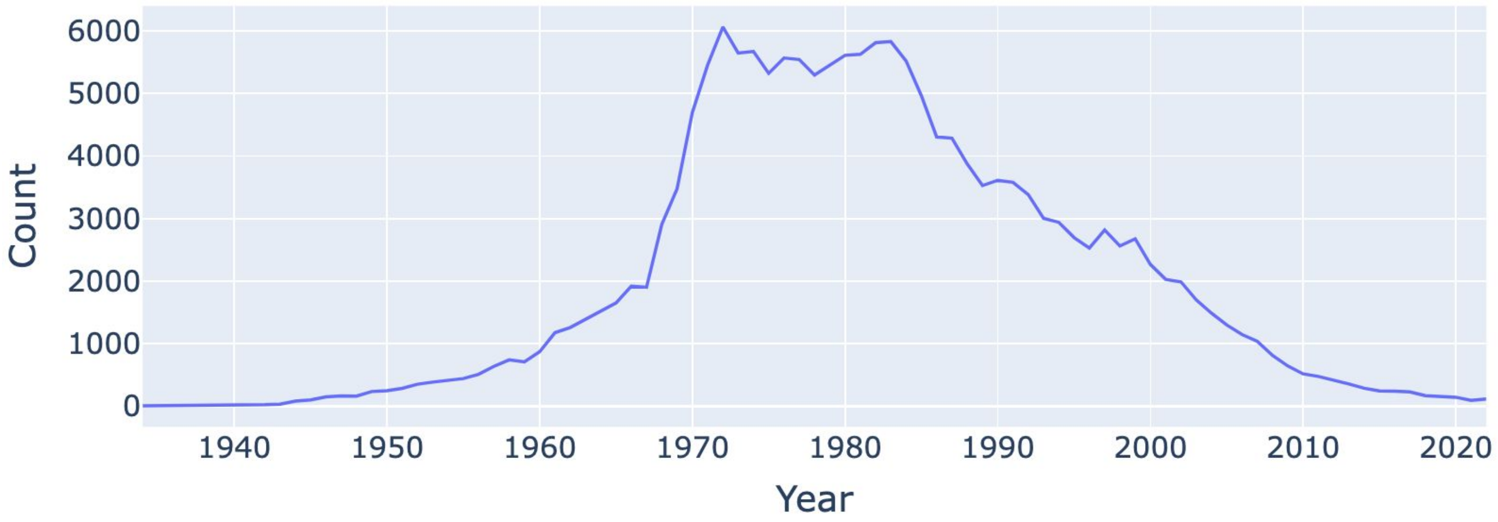
Demo

- Pandas, Part IV
  - Groupby Review
  - **Demo**
  - Joining Tables
  - More on Groupby

DEMO: Putting Things Into Practice

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

Example: Number of Jennifers Born in California Per Year.

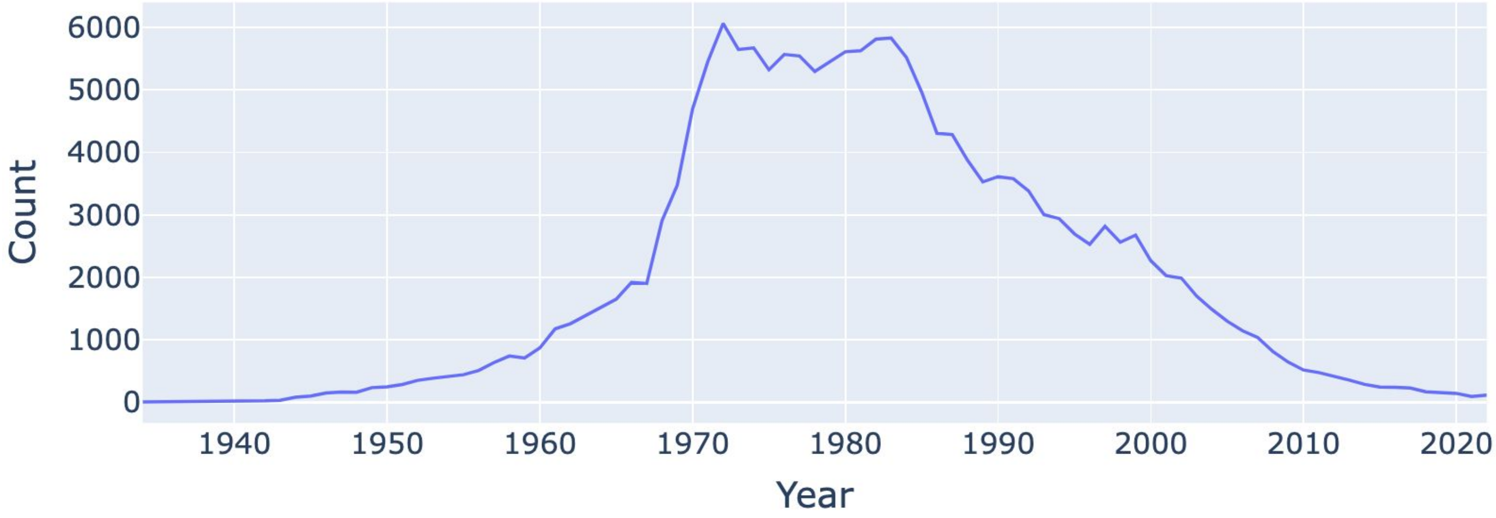


DEMO: Putting Things Into Practice

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

Number of Jennifers Born in California Per Year.



What Is "Popularity"?

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.

Calculating RTP

```
max_jenn = max(f_babynames[f_babynames["Name"] == "Jennifer"]["Count"])
```

```
6065
```

```
curr_jenn = f_babynames[f_babynames["Name"] == "Jennifer"]["Count"].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)
```

```
jenn_counts_ser = f_babynames[f_babynames["Name"] == "Jennifer"]["Count"]
ratio_to_peak(jenn_counts_ser)
0.018796372629843364
```

Calculating RTP Using `.groupby()`

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

A Note on Nuisance Columns

At least as of the time of this slide creation (August 2023), 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'
```

A Note on Nuisance Columns

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

Renaming Columns After Grouping

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"})
```

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

→

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

Some Data Science Payoff

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

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

	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

Some Data Science Payoff

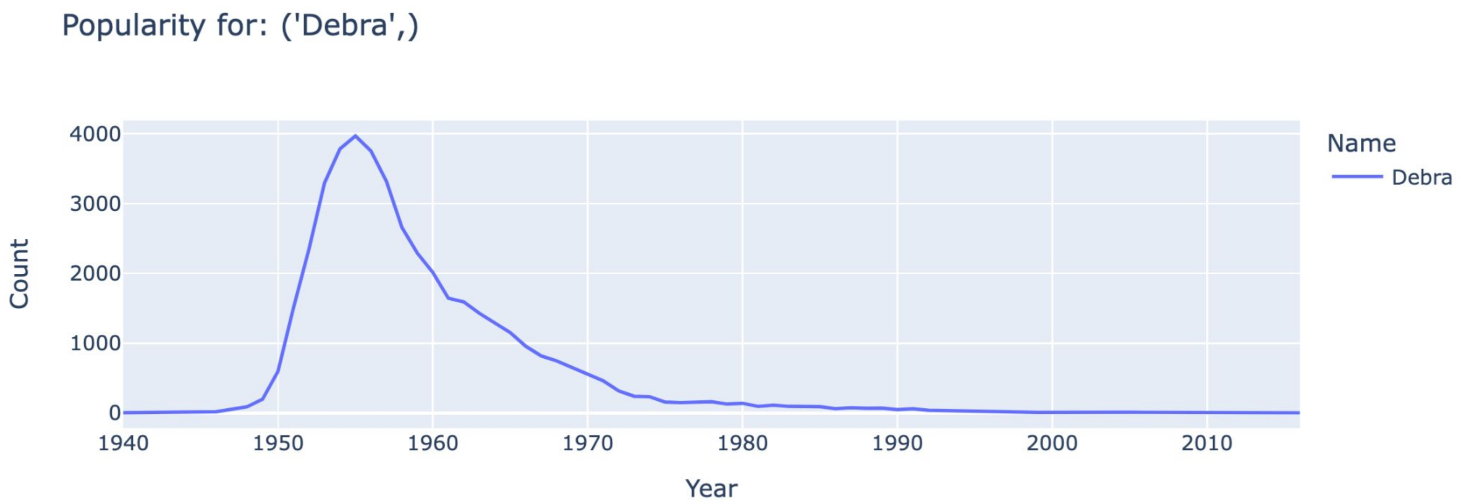
By sorting **`rtp_table`** we can see the names whose popularity has decreased the most.

```
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.

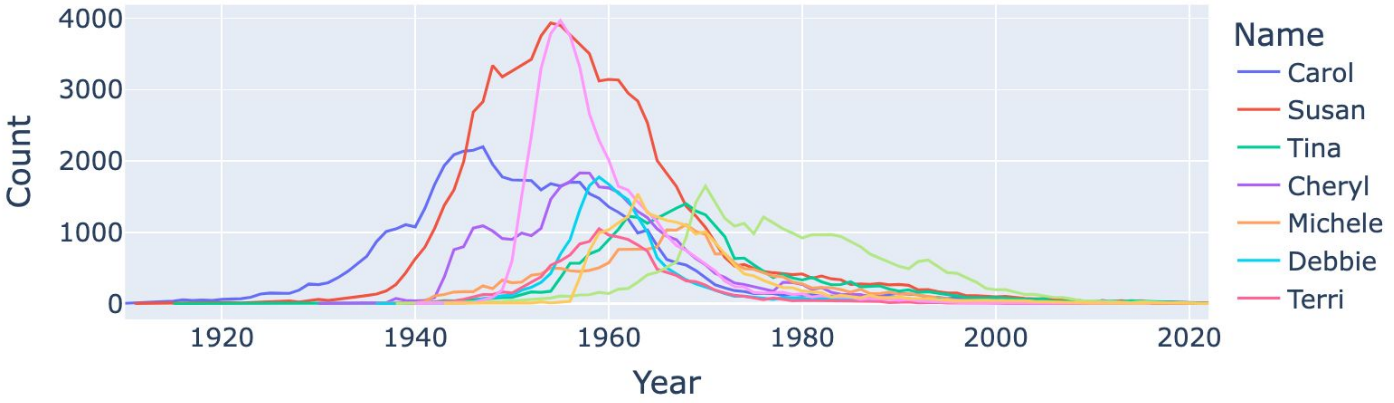
Some Data Science Payoff

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')
```

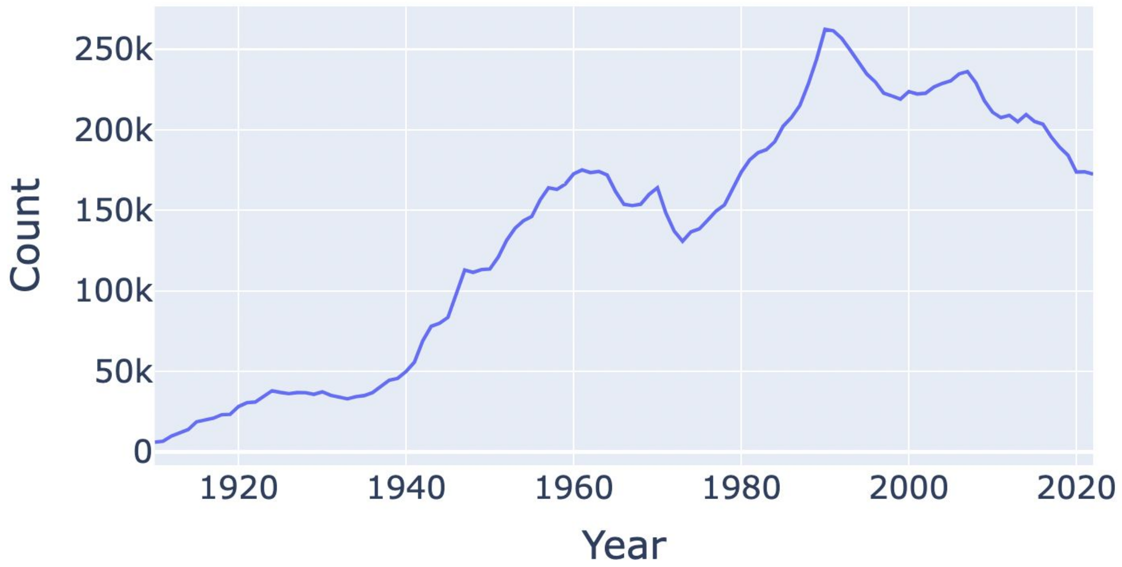
```
px.line(f_babynames[f_babynames["Name"].isin(top10)],
                        x = "Year", y = "Count", color = "Name")
```



Plotting Birth Counts

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

```
puzzle2 = f_babynames.groupby("Year")[["Count"]].agg(sum)
px.line(puzzle2, y = "Count")
```



A Word of Warning!

We made an enormous assumption when we decided to use this dataset to estimate the birth rate.

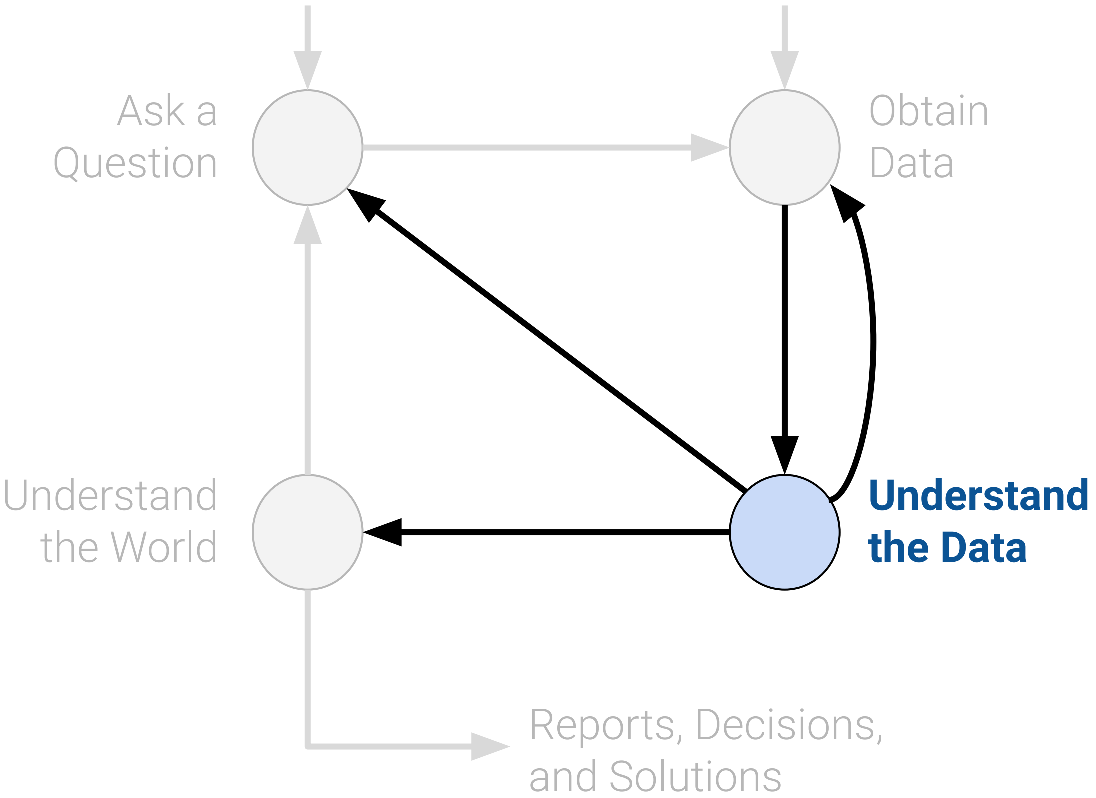
- According to https://lao.ca.gov/LAOEconTax/Article/Detail/691, the true number of babies born in California in 2020 was 421,275 but our plot shows 173,763 babies.
- What happened?

From Lecture 1: Exploratory Data Analysis and Visualization

- How is our data organized and what does it contain?
- Do we already have relevant data?
- What are the biases, anomalies, or other issues with the data?
- How do we transform the data to enable effective analysis?

Bottom line: Blindly using tools is dangerous!

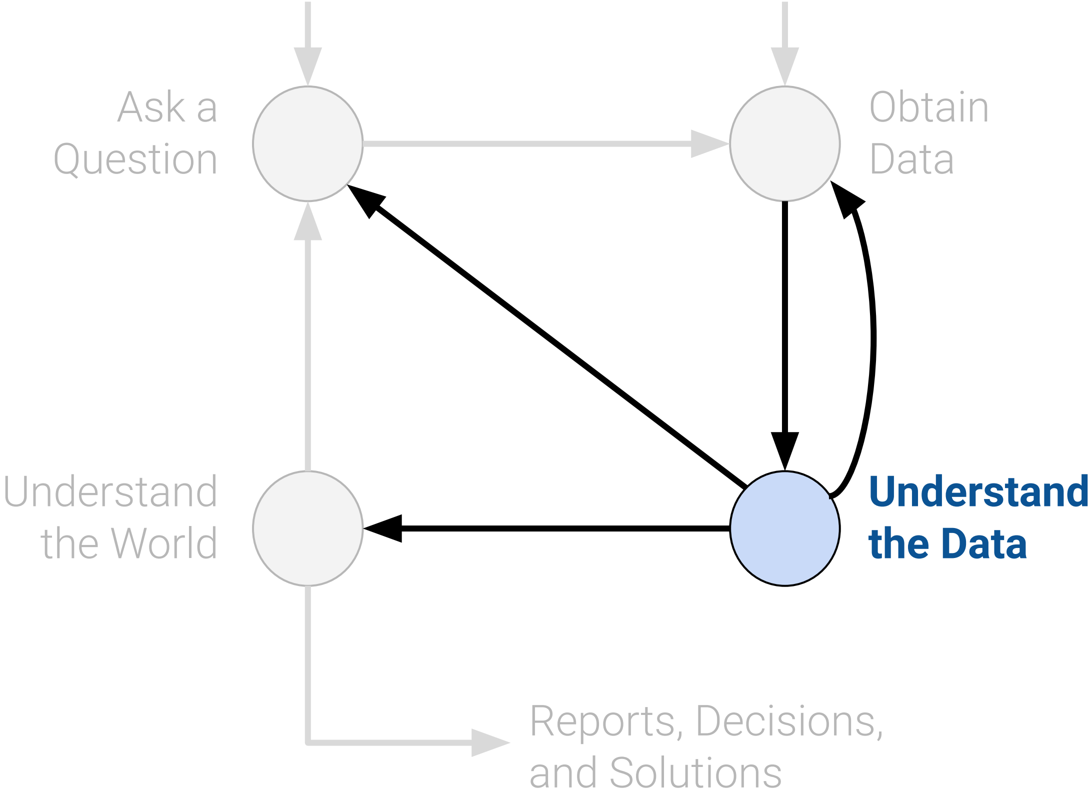
Lisa will cover EDA next week.



From Lecture 1: Exploratory Data Analysis and Visualization

What are the biases, anomalies, or other issues with the data?

- **We only used names for babies who are female at birth.**
- Not all babies register for social security.
- The database does not include names of popularity less than 5 per year



Joining Tables

- Pandas, Part IV
  - Groupby Review
  - Demo
  - **Joining Tables**
  - More on Groupby

Joining Tables

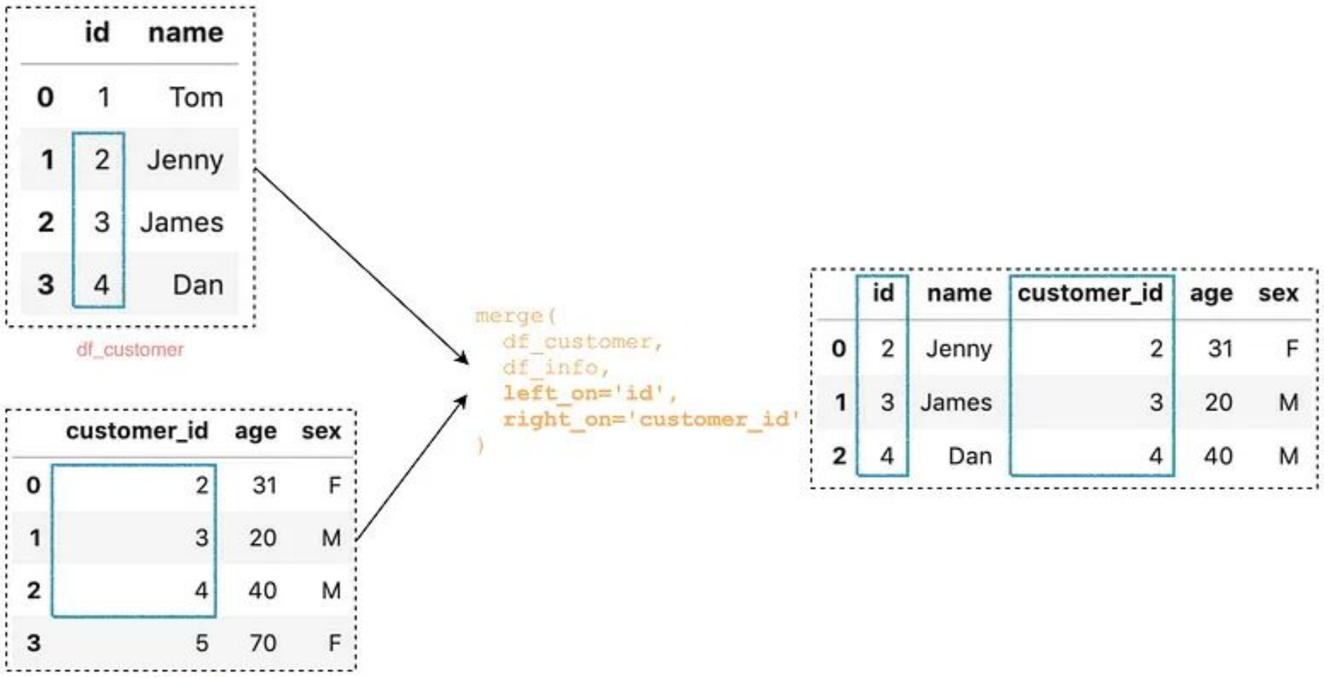
Suppose want to know the popularity of presidential candidate's names in 2022.

- 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.

Joining Tables

```
pd.merge(df_customer,df_info_2,left_on='id',right_on='customer_id')
```



	id	name
0	1	Tom
1	2	Jenny
2	3	James
3	4	Dan

df_customer

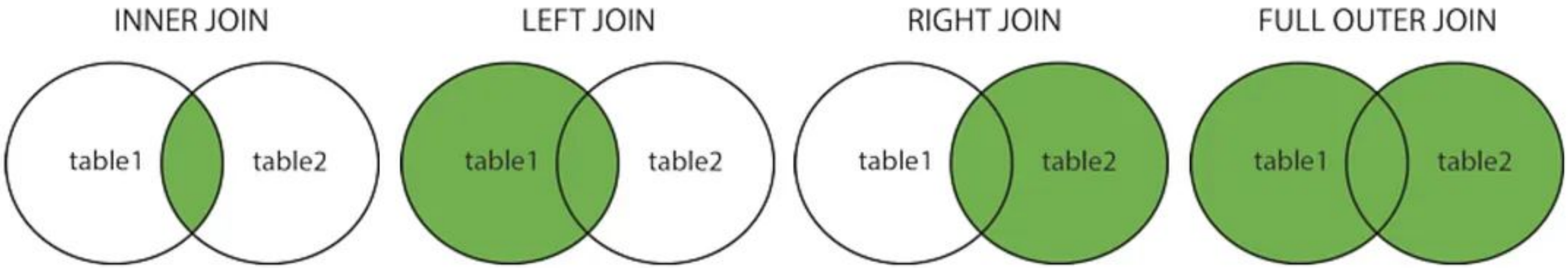
	customer_id	age	sex
0	2	31	F
1	3	20	M
2	4	40	M
3	5	70	F

df_info_2

	id	name	customer_id	age	sex
0	2	Jenny	2	31	F
1	3	James	3	20	M
2	4	Dan	4	40	M

The default setting is Inner Join (so it will only keep the rows that have matching keys in both dataframes).

Joining Tables: Types of Joins



- `inner`: the default join type in Pandas `merge()` function and it produces records that have matching values in both DataFrames
- `left`: produces all records from the left DataFrame and the matched records from the right DataFrame
- `right`: produces all records from the right DataFrame and the matched records from the left DataFrame
- `outer`: produces all records when there is a match in either left or right DataFrame

Joining Tables

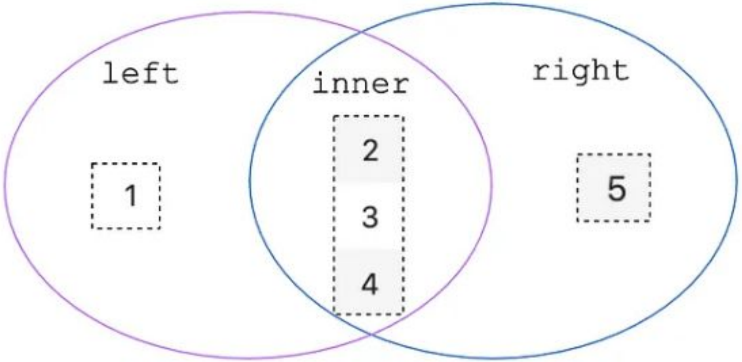
	id	name
0	1	Tom
1	2	Jenny
2	3	James
3	4	Dan

df_customer

	id	age	sex
0	2	31	F
1	3	20	M
2	4	40	M
3	5	70	F

df_info

```
merge(df_customer, df_info, on='id', how=?)
```



Creating Table 1: Babynames in 2022

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

```
m_babynames_2022 = babynames.query('Sex=="M" and Year==2022')
m_babynames_2022
```

	State	Sex	Year	Name	Count
404545	CA	M	2022	Liam	2610
404546	CA	M	2022	Noah	2497
404547	CA	M	2022	Mateo	2371
404548	CA	M	2022	Sebastian	2086
404549	CA	M	2022	Julian	1620
404550	CA	M	2022	Oliver	1617
404551	CA	M	2022	Santiago	1547
404552	CA	M	2022	Benjamin	1524
404553	CA	M	2022	Elijah	1438
404554	CA	M	2022	Ezekiel	1398

Creating Table 2: Presidents with First Names

To join our table, we'll also need to set aside the first names of each candidate (in the code below you should determine what should go in place of the ?).

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

	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
...	...	...	...	...	...	...	...
177	2016	Jill Stein	Green	1457226	loss	1.073699	Jill
178	2020	Joseph Biden	Democratic	81268924	win	51.311515	Joseph
179	2020	Donald Trump	Republican	74216154	loss	46.858542	Donald
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979	Jo
181	2020	Howard Hawkins	Green	405035	loss	0.255731	Howard

182 rows × 7 columns

Joining Our Tables

```python
merged = pd.merge(left = elections, right = m_babynames_2022,
                  left_on = "First Name", right_on = "Name")
```

	Year_x	Candidate	Party	Popular vote	Result	%	First Name	State	Sex	Year_y	Name	Count
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122	Andrew	CA	M	2022	Andrew	741
1	1828	Andrew Jackson	Democratic	642806	win	56.203927	Andrew	CA	M	2022	Andrew	741
2	1832	Andrew Jackson	Democratic	702735	win	54.574789	Andrew	CA	M	2022	Andrew	741
3	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878	John	CA	M	2022	John	490
4	1828	John Quincy Adams	National Republican	500897	loss	43.796073	John	CA	M	2022	John	490
...	...	...	...	...	...	...	...	...	...	...	...	...
136	2016	Darrell Castle	Constitution	203091	loss	0.149640	Darrell	CA	M	2022	Darrell	5

More on Groupby

- Pandas, Part IV
  - Groupby Review
  - Demo
  - Joining Tables
  - **More on Groupby**

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
```

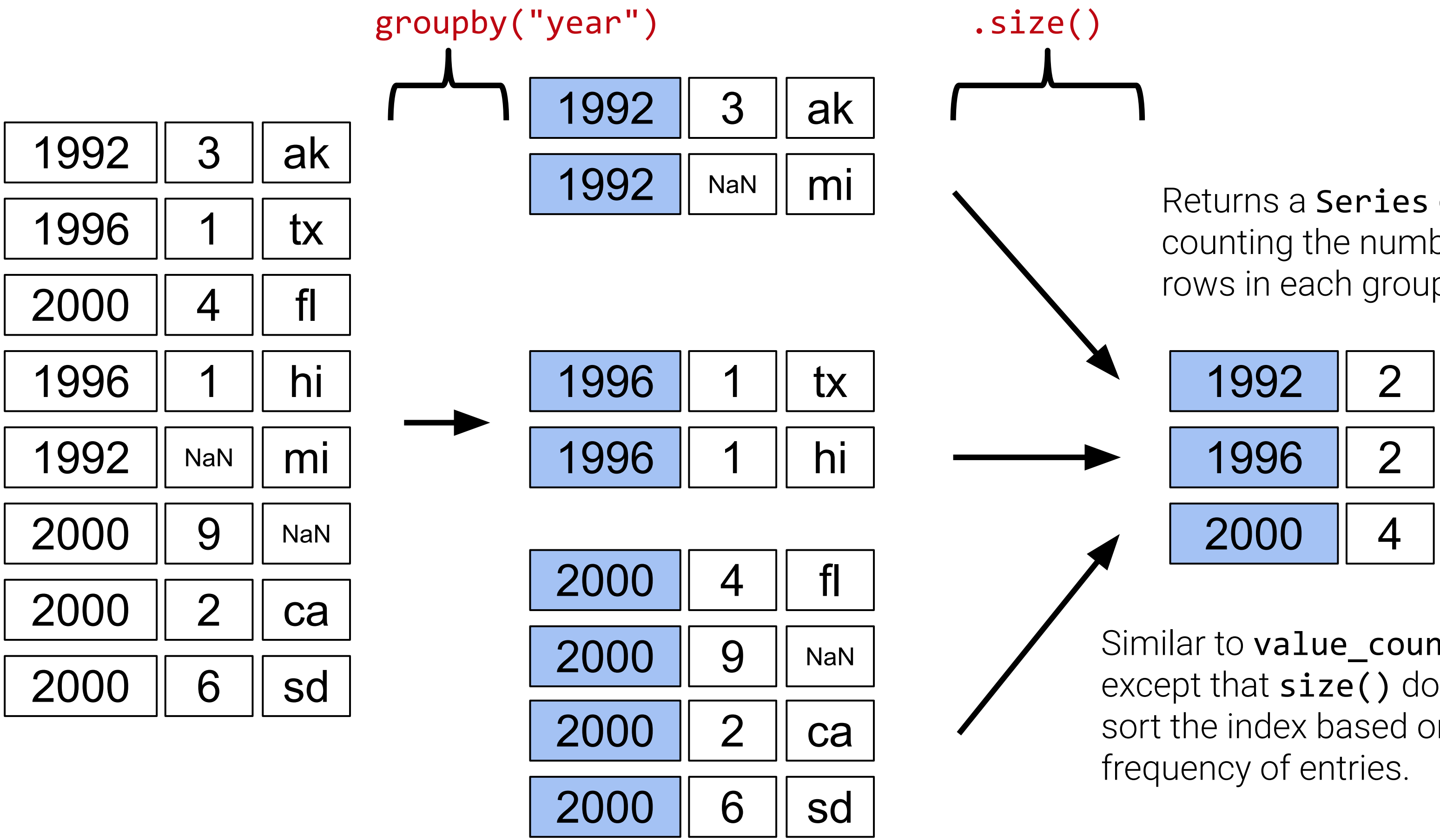
Given a **`DataFrameGroupBy`** object, can use various functions to generate **`DataFrames`** (or **`Series`**). **`agg`** is only one choice:

```
df.groupby(col).mean()     df.groupby(col).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 https://pandas.pydata.org/docs/reference/groupby.html for a list of **`DataFrameGroupBy`** methods.

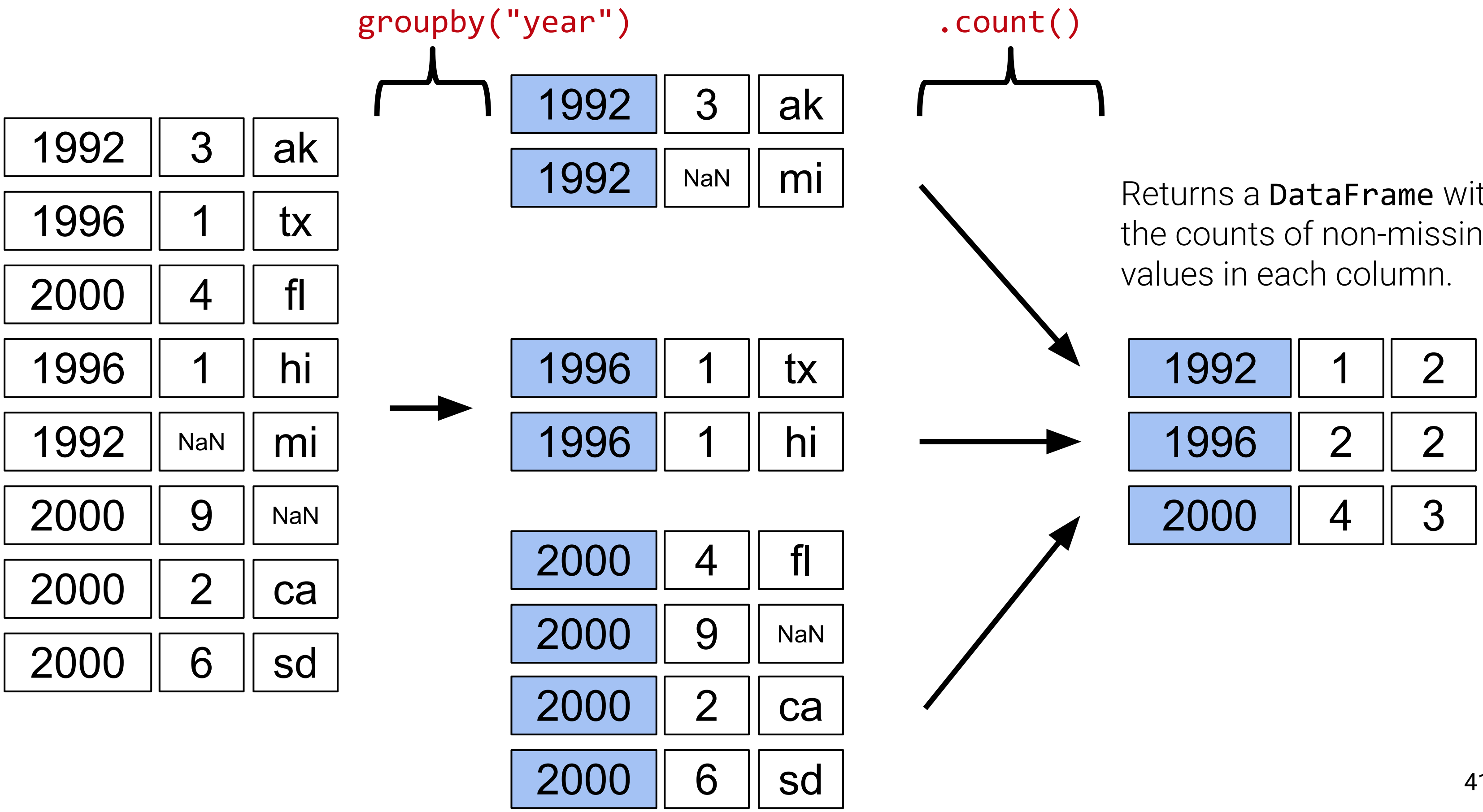
`groupby.size()` and `groupby.count()`



Returns a `Series` object counting the number of rows in each group.

Similar to `value_counts()` except that `size()` does not sort the index based on the frequency of entries.

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



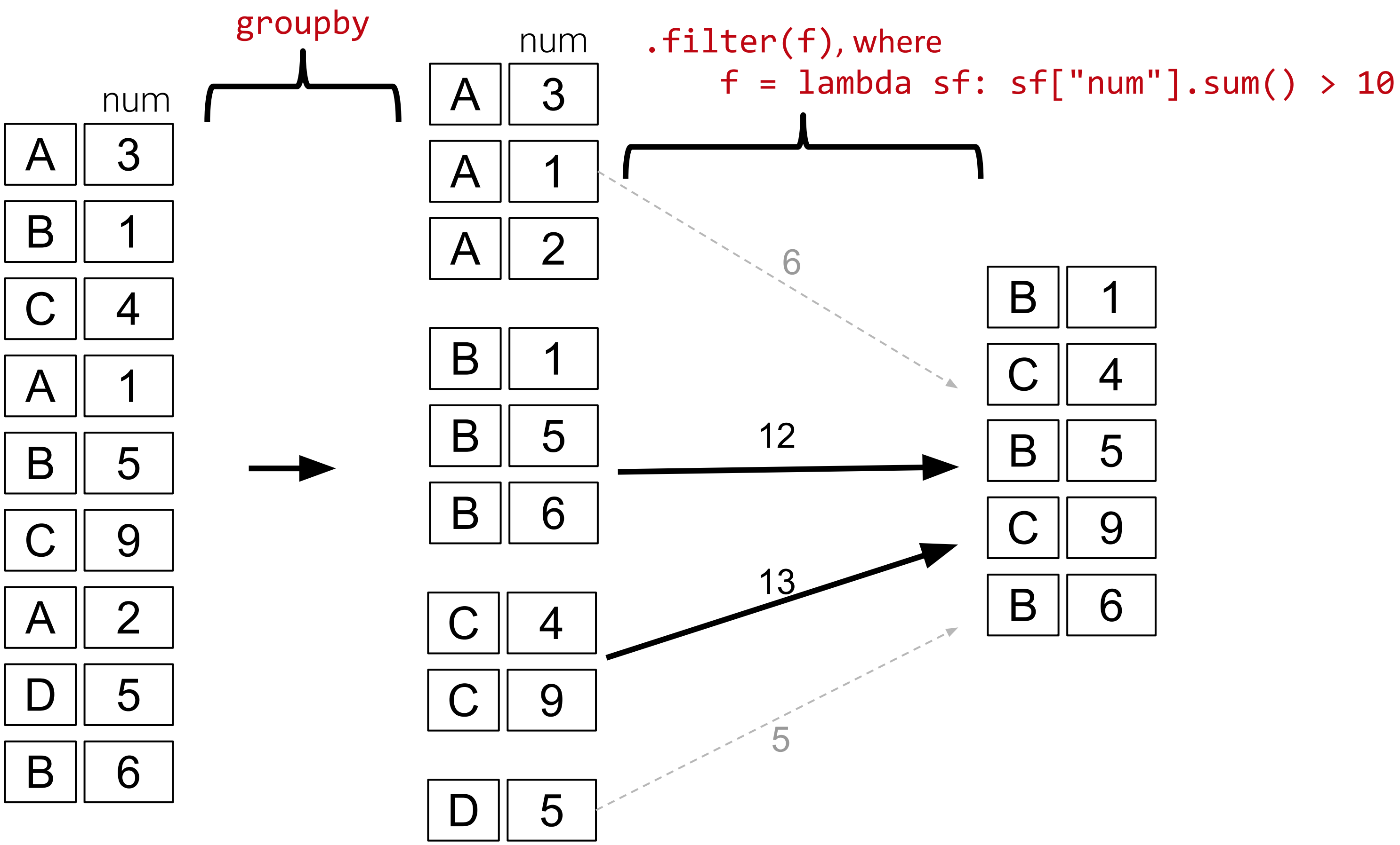
Returns a `DataFrame` with the counts of non-missing values in each column.

Filtering by Group

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!

groupby.filter()



Filtering Elections Dataset

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

There's More Than One Way to Find the Best Result by Party

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

- Each approach has different tradeoffs in terms of readability, performance, memory consumption, complexity, etc.
- Takes a very long time to understand these tradeoffs!
- If you find your current solution to be particularly convoluted or hard to read, maybe try finding another way!

More on `DataFrameGroupby` Object

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], 'Citiz
ens': [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], 'Democratic-Republican': [0,
1], 'Dixiecrat': [103], 'Farmer-Labor': [78], 'Free Soil': [15, 18], 'Green': [149, 155, 156, 165, 170, 177, 181],
'Greenback': [35], 'Independent': [121, 130, 143, 161, 167, 174], 'Liberal Republican': [31], 'Libertarian': [125,
128, 132, 138, 139, 146, 153, 159, 163, 169, 175, 180], 'National Democratic': [50], 'National Republican': [3, 5],
'National Union': [27], 'Natural Law': [148], 'New Alliance': [136], 'Northern 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, 65, 69, 72, 79, 80, 84, 87, 90, 96, 98,
104, 106, 109, 112, 113, 117, 120, 122, 131, 133, 135, 142, 145, 152, 157, 166, 171, 173, 179], 'Socialist': [58, 6
2, 66, 71, 76, 85, 88, 92, 95, 102], 'Southern Democratic': [25], 'States' Rights': [110], 'Taxpayers': [147], 'Uni
on': [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

Grouping by Multiple Columns

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.

Just Finished...

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

LECTURE 5

Pandas IV

Content credit: [Acknowledgments]