LECTURE 3

Pandas, Part III

Advanced Pandas (More on Grouping, Aggregation, Pivot Tables Merging)

New Syntax / Concept Summary

Today we'll cover:

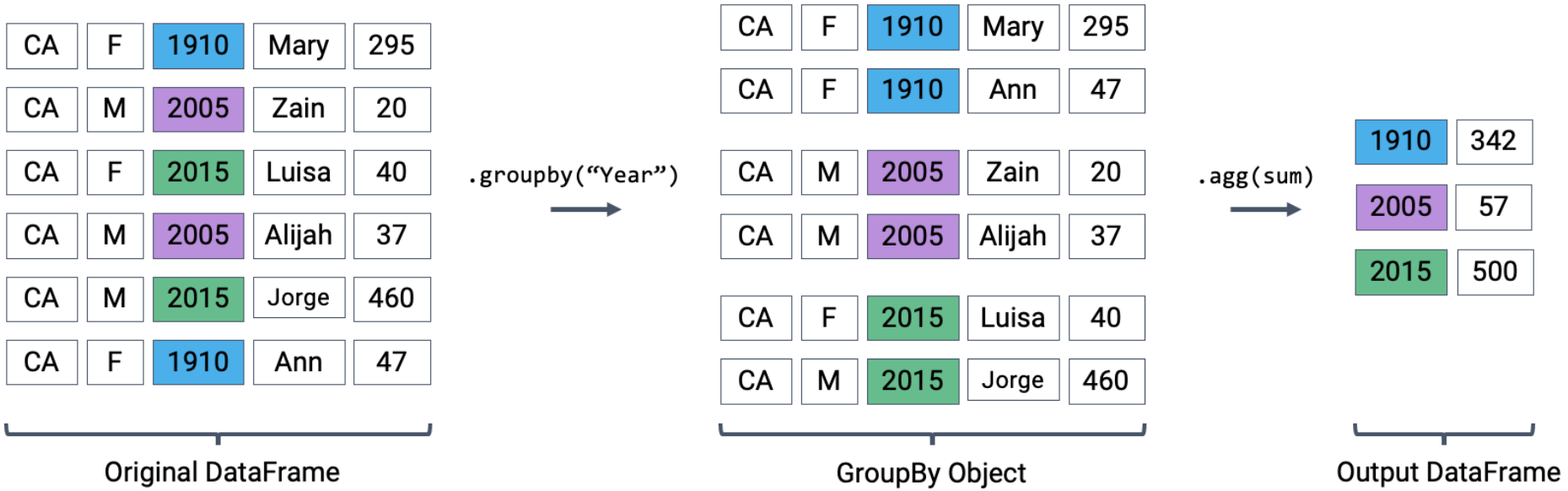
- Groupby: Output of .groupby("Name") is a DataFrameGroupBy object. Condense back into a DataFrame or Series with:
  - groupby.size
  - groupby.filter
  - and more...
- Pivot tables: An alternate way to group by exactly two columns.
- Joining tables using pd.merge.
- Exploratory data analysis

Groupby Review

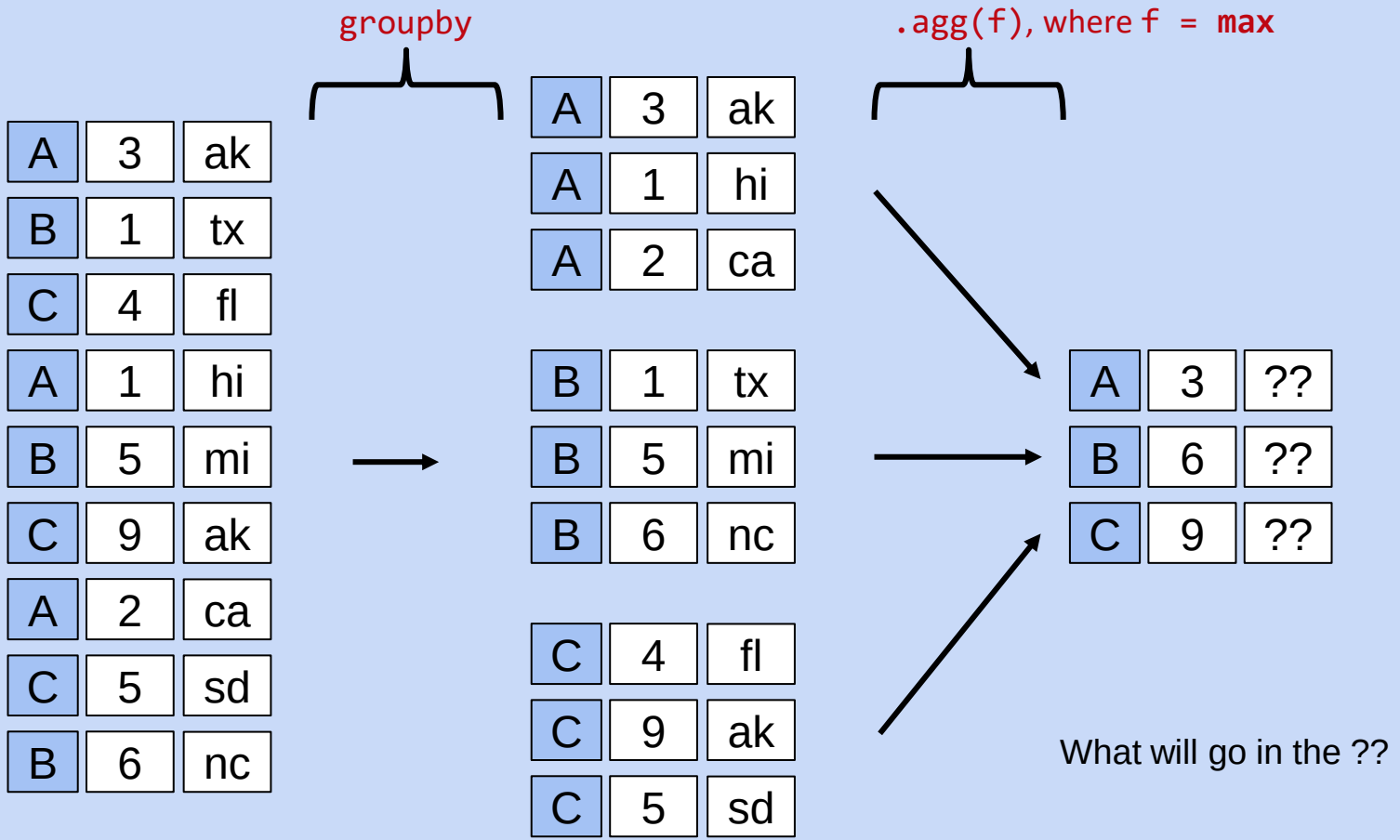
Lecture 04,

- **Groupby Review**
- Other DataFrameGroupBy Features
- Groupby and PivotTables
- A Quick Look at Joining Tables

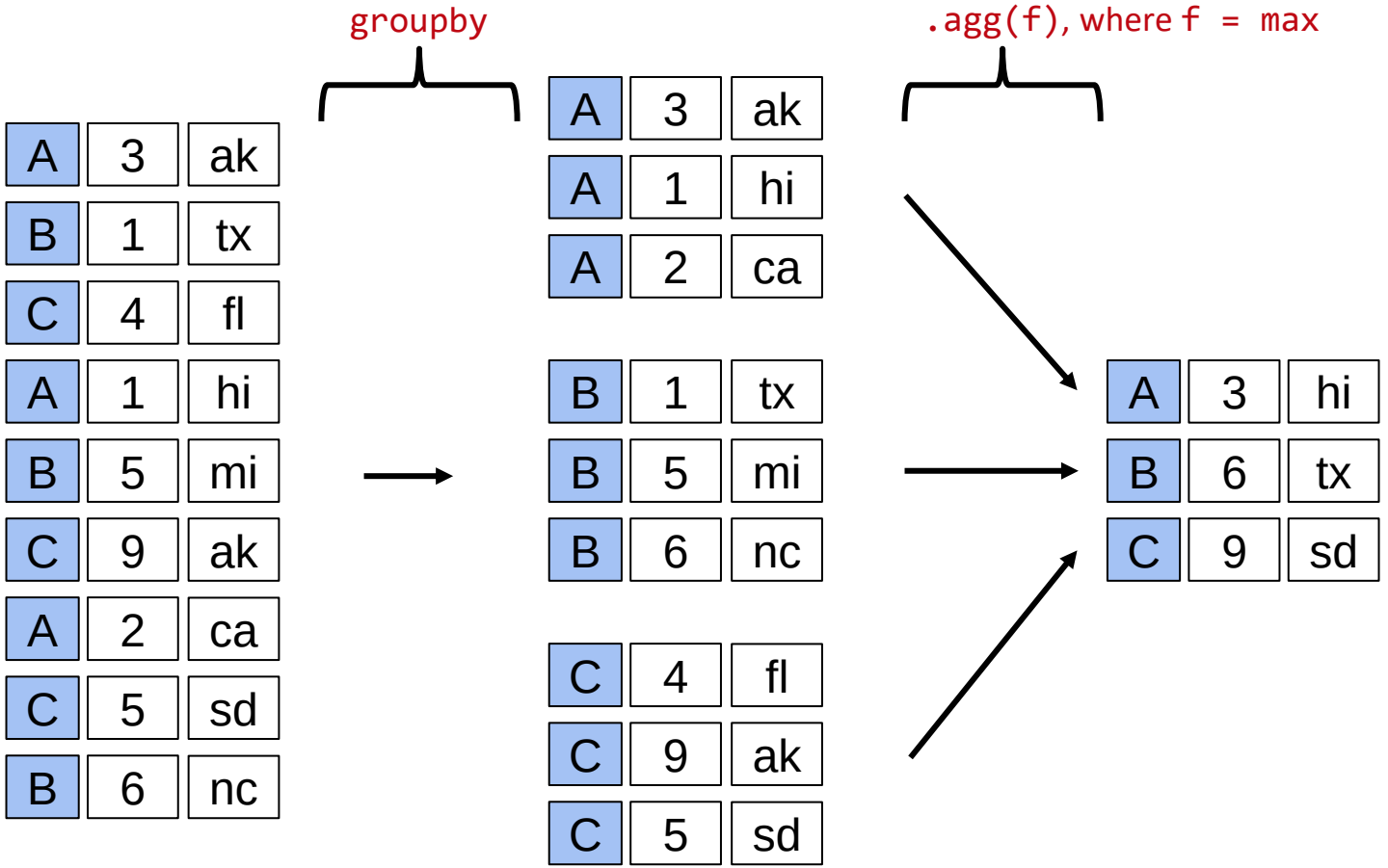
Groupby Review



Groupby Review Question



Answer


groupby
.agg(f), where f = max
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
A 3 ak
A 1 hi
A 2 ca
B 1 tx
B 5 mi
B 6 nc
C 4 fl
C 9 ak
C 5 sd
A 3 hi
B 6 tx
C 9 sd

Question

	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

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

Groupby Puzzle #1

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

```
elections.groupby("Party").agg(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

Groupby Puzzle #1

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

```
elections.groupby("Party").agg(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 latest candidate name: Woodrow Wilson
- Highest % of vote: 61.34

Groupby Puzzle #1

Very hard puzzle: Try to write code that returns the table below.

- Each row shows the best result (in %) by each party.
  - For example: Best Democratic result ever was Johnson's 1964 win.

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

Groupby Puzzle #1

Very hard puzzle: Try to write code that returns the table below.

- First sort the DataFrame so that rows are in descending order of %.
- Then group by Party and take the first item of each series.
- 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").agg(lambda x : x.iloc[0])
```

	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

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!

Groupby Puzzle #1 - Alternate Approaches

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")
```

Other DataFrameGroupBy Features

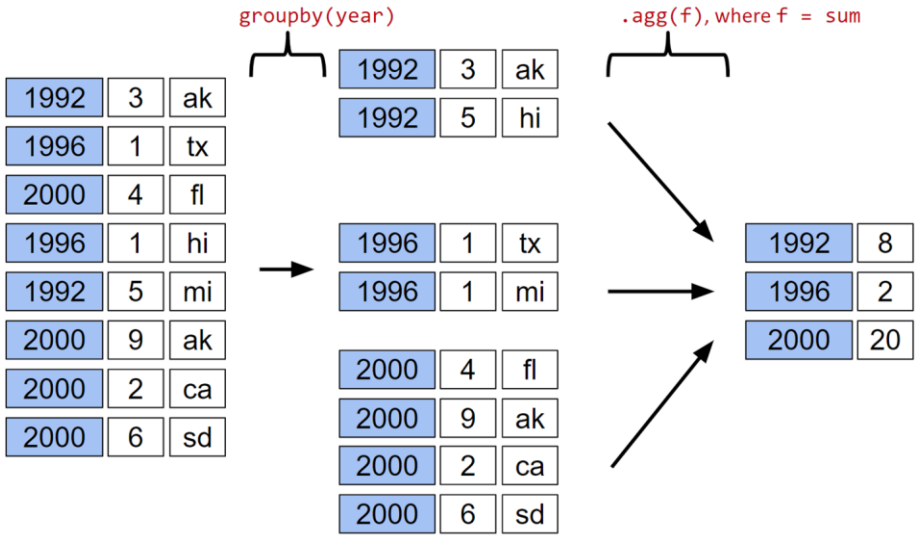
Lecture 04,

- Groupby Review
- **Other DataFrameGroupBy Features**
- Groupby and PivotTables
- A Quick Look at Joining Tables

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)`:
  - Organizes all rows with the same year into a subframe for that year.
  - Creates a new dataframe with one row representing each subframe year.
  - All integer rows in each subframe are combined using the sum function.



Raw groupby Objects

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:

- `agg`: Creates a new DataFrame with one aggregated row per subframe.
- `max`: Creates a new DataFrame aggregated using the max function.
- `size`: Creates a new Series with the size of each subframe.
- `filter`: Creates a copy of the original DataFrame, but keeping only rows from subframes that obey the provided condition.

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

More on DataFrameGroupby Object

We can look into DataFrameGroupby objects in following ways:

```
grouped_by_year = elections.groupby("Year")
grouped_by_year.groups.keys()
```

```
dict_keys([1824, 1828, 1832, 1836, 1840, 1844, 1848, 1852, 1856, 1860, 1864, 1868, 1872, 1876, 188
0, 1884, 1888, 1892, 1896, 1900, 1904, 1908, 1912, 1916, 1920, 1924, 1928, 1932, 1936, 1940, 1944,
1948, 1952, 1956, 1960, 1964, 1968, 1972, 1976, 1980, 1984, 1988, 1992, 1996, 2000, 2004, 2008, 201
2, 2016, 2020])
```

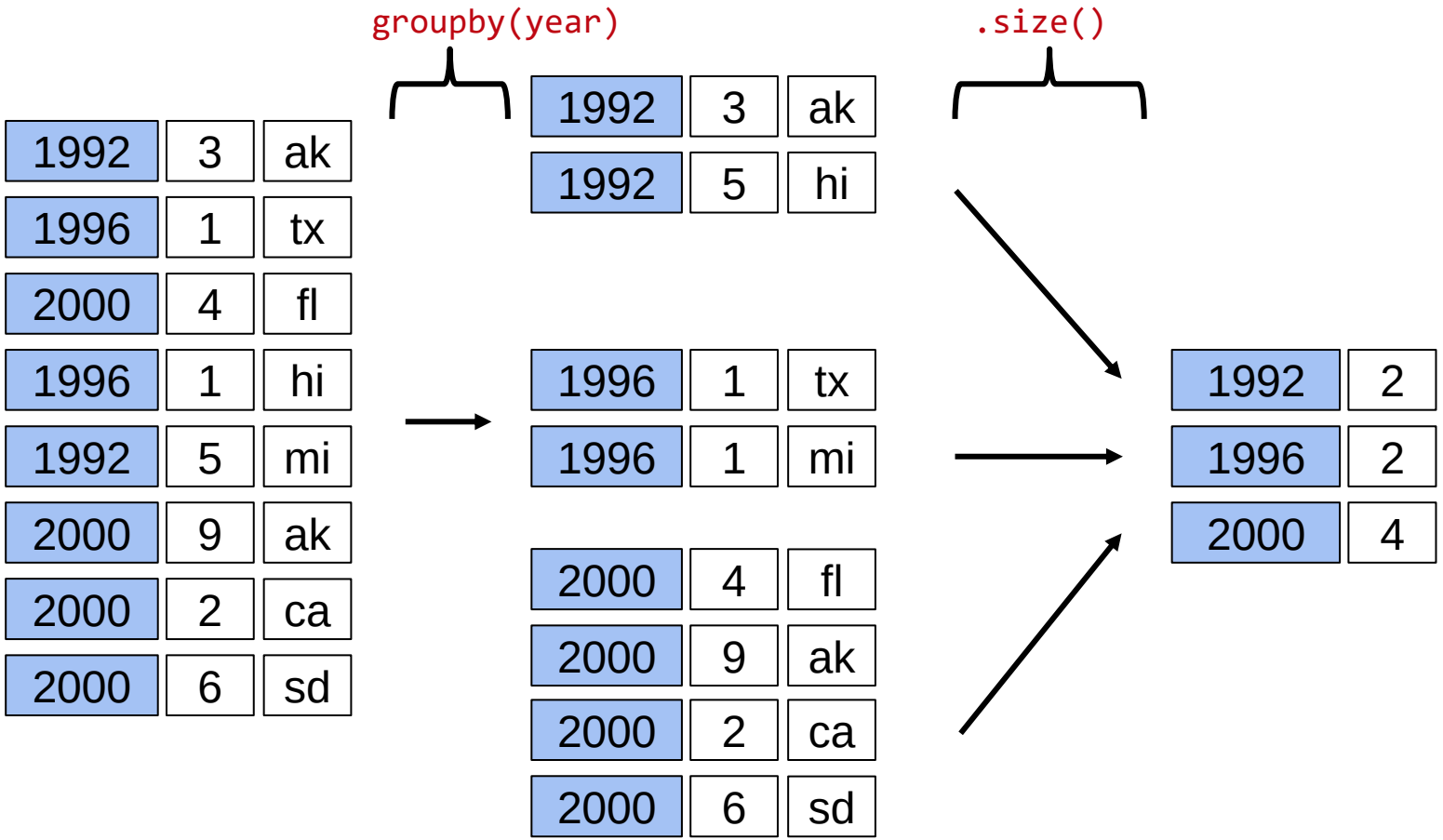
```
grouped_by_year.groups[2020]
```

```
Int64Index([178, 179, 180, 181], dtype='int64')
```

```
grouped_by_year.get_group[2020]
```

	Year	Candidate	Party	Popular vote	Result	%
178	2020	Joseph Biden	Democratic	81268924	win	51.311515
179	2020	Donald Trump	Republican	74216154	loss	46.858542
180	2020	Jo Jorgensen	Libertarian	1865724	loss	1.177979
181	2020	Howard Hawkins	Green	405035	loss	0.255731

groupby.size()

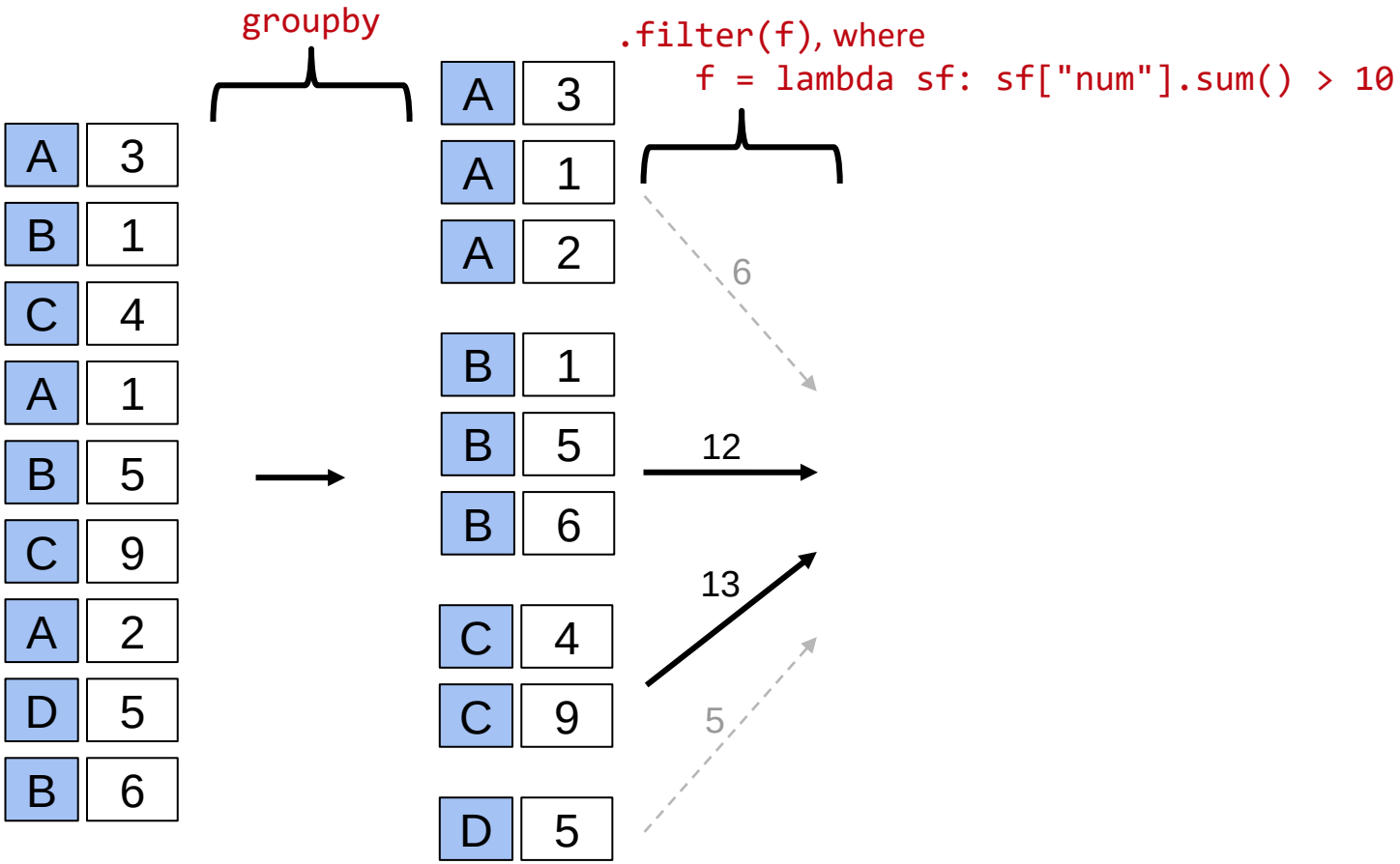


Filtering by Group

Another common use for groups is to filter data.

- `groupby.filter` takes an argument `f`.
- `f` is a function that:
  - Takes a DataFrame as input.
  - Returns either True or False.
- For each group g, f is applied to the subframe comprised of the rows from the original dataframe corresponding to that group.

groupby.filter



Filtering 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)
         .set_index("Year")
         .sort_index()
```

Year	Candidate	Party	Popular vote	Result	%
1860	Abraham Lincoln	Republican	1855993	win	39.699408
1860	John Bell	Constitutional Union	590901	loss	12.639283
1860	John C. Breckinridge	Southern Democratic	848019	loss	18.138998
1860	Stephen A. Douglas	Northern Democratic	1380202	loss	29.522311
1912	Eugene V. Debs	Socialist	901551	loss	6.004354
1912	Eugene W. Chafin	Prohibition	208156	loss	1.386325
1912	Theodore Roosevelt	Progressive	4122721	loss	27.457433
1912	William Taft	Republican	3486242	loss	23.218466
1912	Woodrow Wilson	Democratic	6296284	win	41.933422
1968	George Wallace	American Independent	9901118	loss	13.571218
1968	Hubert Humphrey	Democratic	31271839	loss	42.863537
1968	Richard Nixon	Republican	31783783	win	43.565246
1992	Andre Marrou	Libertarian	290087	loss	0.278516
1992	Bill Clinton	Democratic	44909806	win	43.118485
1992	Bo Gritz	Populist	106152	loss	0.101918
1992	George H. W. Bush	Republican	39104550	loss	37.544784
1992	Ross Perot	Independent	19743821	loss	18.956298

Groupby and PivotTables

Lecture 04,

- Groupby Review
- Other DataFrameGroupBy Features
- **Groupby and PivotTables**
- A Quick Look at Joining Tables

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:

Example:

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

		Count
Year	**Sex**	
1880	**F**	90994
	M	110490
1881	**F**	91953
	M	100737
1882	**F**	107847
	M	113686

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

Pivot Tables

A more natural approach is to use our 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)
```

	Count	
Sex	F	M
Year		
1880	90994	110490
1881	91953	100737
1882	107847	113686
1883	112319	104625
1884	129019	114442
1885	133055	107799

groupby(["Year", "Sex"]) vs. pivot_table

The pivot table more naturally represents our data.

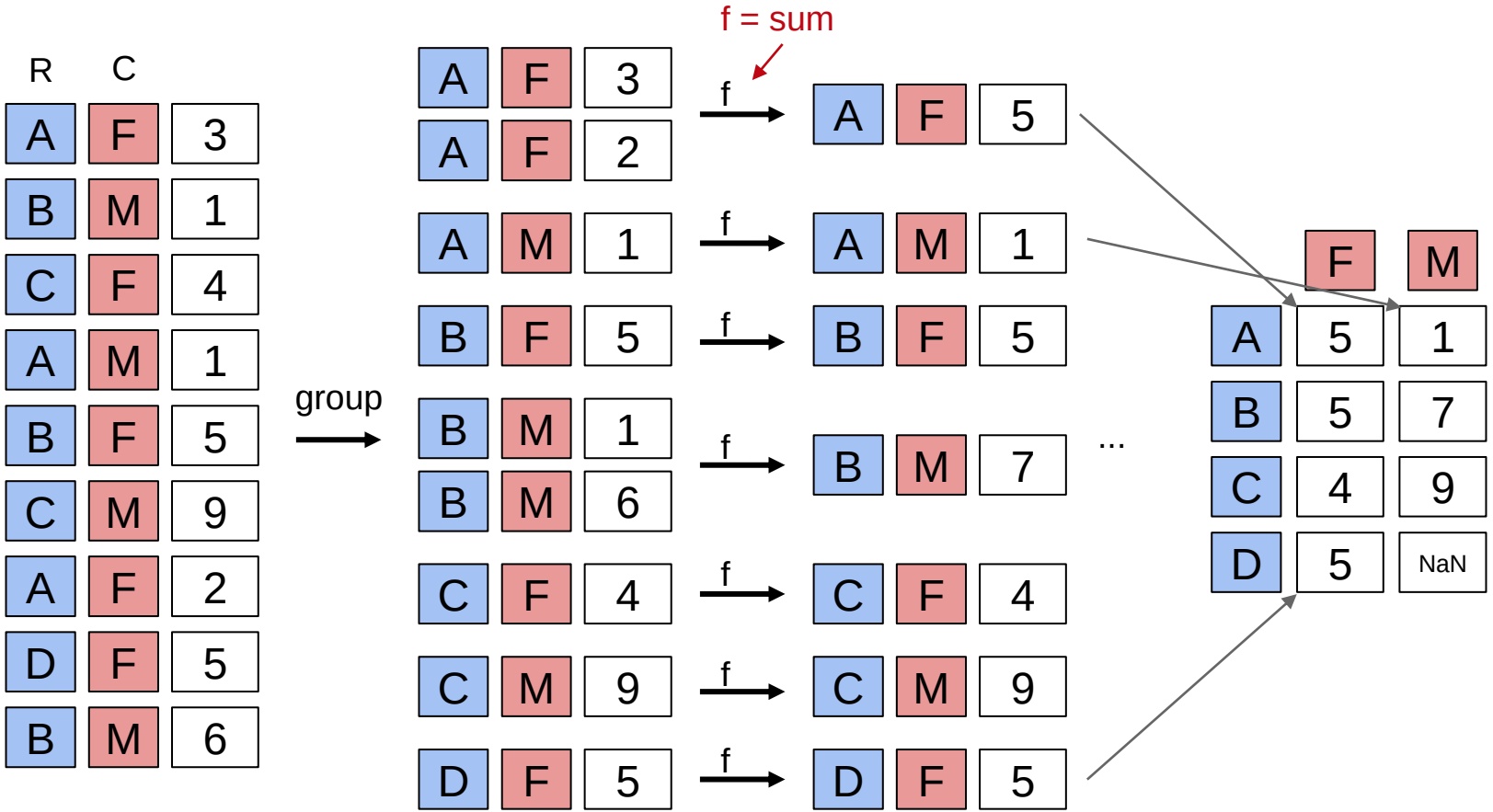
groupby output

		Count
Year	**Sex**	
1880	**F**	90994
	M	110490
1881	**F**	91953
	M	100737
1882	**F**	107847
	M	113686

pivot_table output

	Count	
Sex	**F**	**M**
Year		
1880	90994	110490
1881	91953	100737
1882	107847	113686
1883	112319	104625
1884	129019	114442
1885	133055	107799

Pivot Table Mechanics



Pivot Tables

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

A Quick Look at Joining Tables

Lecture 04,

- Groupby Review
- Other DataFrameGroupBy Features
- Groupby and PivotTables
- **A Quick Look at Joining Tables**

Joining Tables

Suppose want to know the 2020 popularity of presidential candidate's names.

- Example: Dwight Eisenhower's name Dwight is not popular today, with only 5 babies born with this name in California in 2020.

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

- This will be almost exactly like Table.join from data 8 (http://data8.org/datascience/_autosummary/datascience.tables.Table.join.html)

Creating Table 1: Babynames in 2020

Let's set aside names from 2020 first:

```
babynames_2020 = babynames[babynames["Year"] == 2020]
babynames_2020
```

	Name	Sex	Count	Year
0	Olivia	F	17641	2020
1	Emma	F	15656	2020
2	Ava	F	13160	2020
3	Charlotte	F	13065	2020
4	Sophia	F	13036	2020
...	...	...	...	...
31448	Zykell	M	5	2020
31449	Zylus	M	5	2020
31450	Zymari	M	5	2020
31451	Zyn	M	5	2020
31452	Zyran	M	5	2020

31453 rows × 4 columns

Creating Table 2: Presidents with First Names

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

- You'll have a chance to write this code again on lab, so don't worry about the details too much.

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

	Year	Candidate	Party	Popular vote	Result	%	First Name
...	...	...	...	...	...	...	...
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

Joining Our Tables

```
merged = pd.merge(left = elections, right = babynames_2020,
                  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	2020	Andrew	867
1	1828	Andrew Jackson	Democratic	642806	win	56.203927	Andrew	CA	M	2020	Andrew	867
2	1832	Andrew Jackson	Democratic	702735	win	54.574789	Andrew	CA	M	2020	Andrew	867
3	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878	John	CA	M	2020	John	617
4	1828	John Quincy Adams	National Republican	500897	loss	43.796073	John	CA	M	2020	John	617

THANK YOU

usman.nazir@lums.edu.pk

usmanweb.github.io

Slides and content credits: Narges Norouzi, Lisa Yan, Josh Hug