CS334: Principles and Techniques of Data Science

Lecture 4

Mobin Javed
Ihsan Ayyub Qazi

Slides partially adapted from DS100 at UC Berkeley

Goals For Today

Goals For Today

- Discuss aggregation operations:
  - Groupby
  - Pivot
- Operations for modifying groupby objects
  - Applying functions
  - Filter
- Case Study: Answer more questions on the baby names dataset

Groupby (and isin)

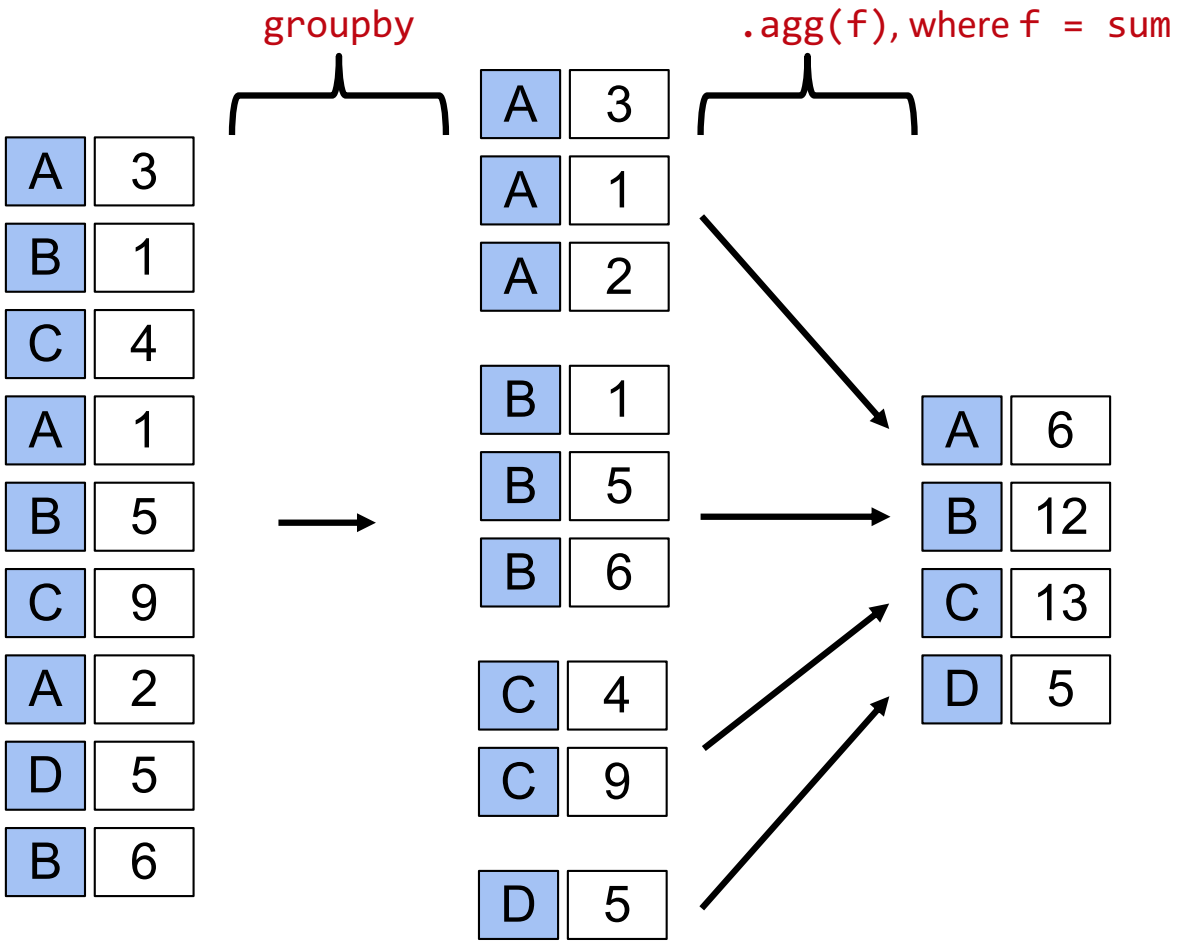
`groupby`

Often we want to perform aggregate analysis across data points that share some feature, for example:

- What was the average share of the vote across all U.S. elections for each political **party**?
- What was the size of the average class in **each department** at a given school in **each term?**
- Which **instructors** taught the largest classes at a given school and how large were they?

`groupby` is an incredibly powerful tool for these sorts of questions.

Series groupby/agg Summary



groupby Key Concepts

If we call groupby on a Series:

- The resulting output is a SeriesGroupBy object.
- The Series that are passed as arguments to groupby must share an index with the calling Series.

```
percent_grouped_by_party = df['%'].groupby(df['Party'])
percent_grouped_by_party.groups
```

```
{'Democratic': Int64Index([1, 4, 6, 7, 10, 13, 15, 17, 19, 21], dtype='int64'),
 'Independent': Int64Index([2, 9, 12], dtype='int64'),
 'Republican': Int64Index([0, 3, 5, 8, 11, 14, 16, 18, 20, 22], dtype='int64')}
```

SeriesGroupBy objects can then be aggregated back into a Series using an aggregation method.

```
percent_grouped_by_party.mean()
```

→

```
Party
Democratic      46.53
Independent     11.30
Republican      47.86
Name: %, dtype: float64
```

groupby Key Concepts

If we call `groupby` on a `DataFrame`:

- The resulting output is a `DataFrameGroupBy` object.

`DataFrameGroupBy` objects can then be aggregated back into a `DataFrame` or a `Series` using an aggregation method.

```
everything_grouped_by_party = df.groupby('Party')
```

```
{'Democratic': Int64Index([1, 4, 6, 7, 10, 13, 15, 17, 19, 21], dtype='int64'),
 'Independent': Int64Index([2, 9, 12], dtype='int64'),
 'Republican': Int64Index([0, 3, 5, 8, 11, 14, 16, 18, 20, 22], dtype='int64')}
```

```
everything_grouped_by_party.mean()
```

→

Party	%	Year
Democratic	46.53	1998.000000
Independent	11.30	1989.333333
Republican	47.86	1998.000000

groupby and agg

Most of the built-in handy aggregation methods are just shorthand for a universal aggregation method called agg.

- Example, `.mean()` is just `.agg(np.mean)`.

```
everything_grouped_by_party = df.groupby('Party')
```

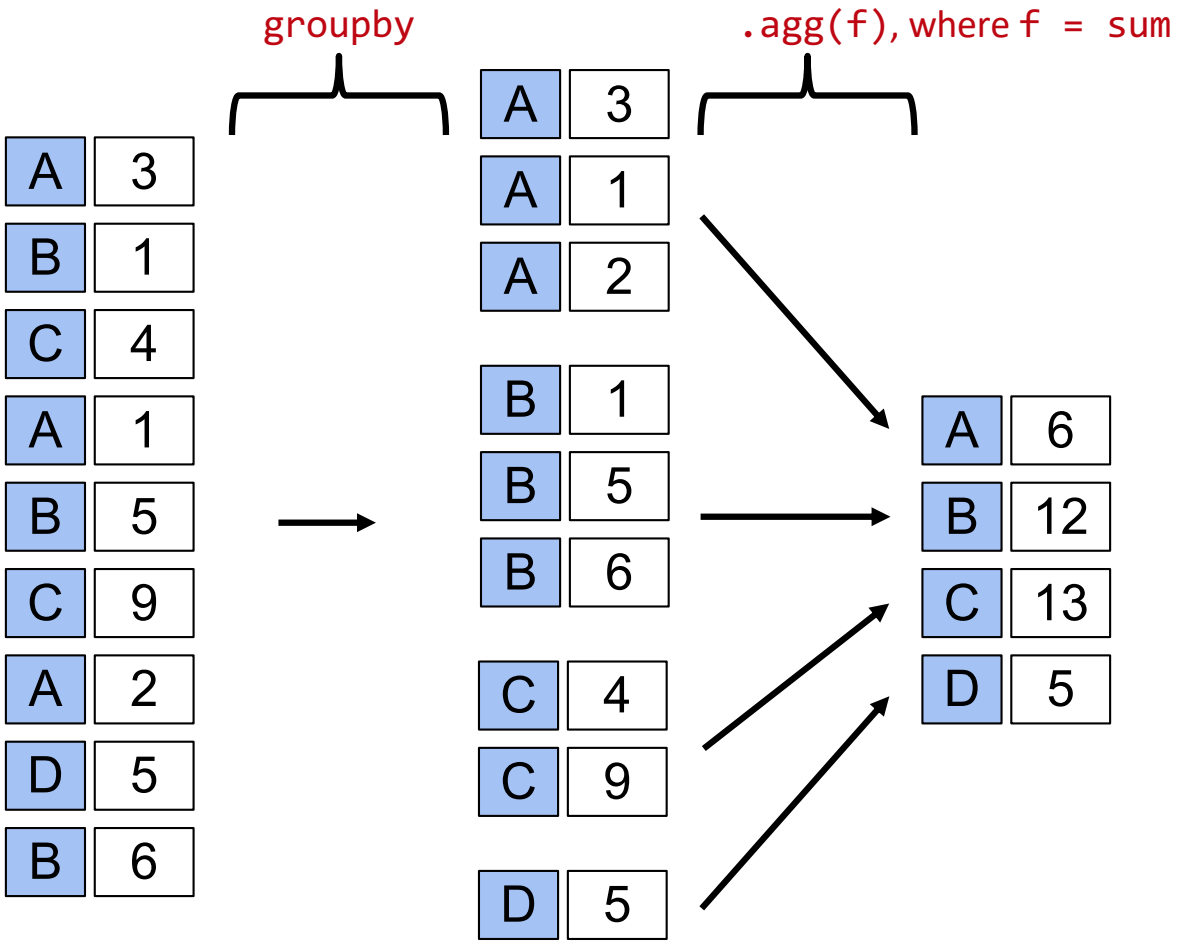
```
{'Democratic': Int64Index([1, 4, 6, 7, 10, 13, 15, 17, 19, 21], dtype='int64'),
 'Independent': Int64Index([2, 9, 12], dtype='int64'),
 'Republican': Int64Index([0, 3, 5, 8, 11, 14, 16, 18, 20, 22], dtype='int64')}
```

```
everything_grouped_by_party.mean()
```

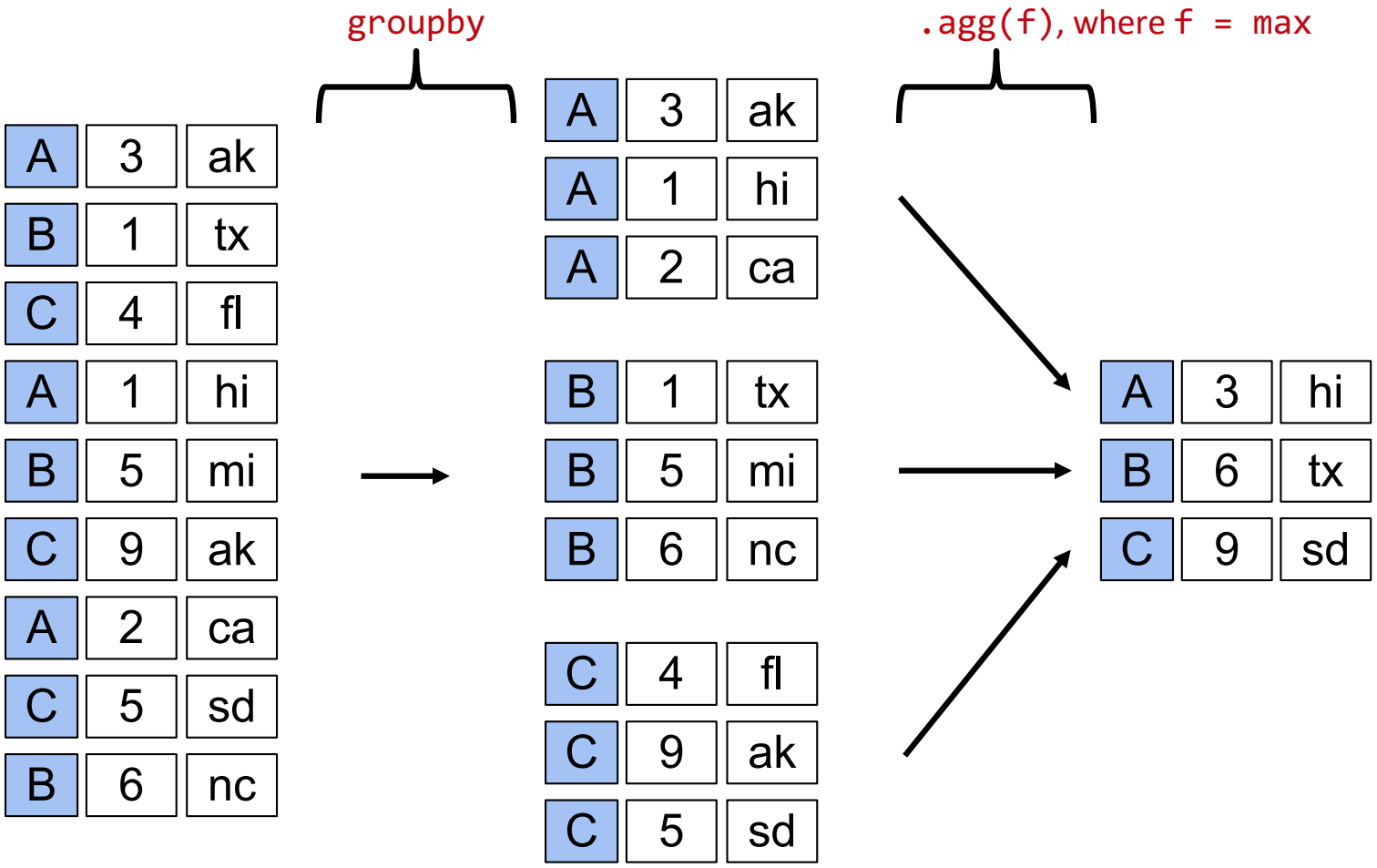
```
everything_grouped_by_party.agg(np.mean)
```

Party	%	Year
Democratic	46.53	1998.000000
Independent	11.30	1989.333333
Republican	47.86	1998.000000

Series groupby/agg Summary



DataFrame groupby/agg Summary



The `MultiIndex`

If we group a Series (or DataFrame) by multiple Series and then perform an aggregation operation, the resulting Series (or Dataframe) will have a `MultiIndex`.

```
everything_grouped_by_party_and_result = df.groupby([df['Party'], df['Result']])
everything_grouped_by_party_and_result.mean()
```

The resulting DataFrame has:

- Two columns "%" and "Year"
- A MultiIndex, where results of aggregate function are indexed by Party first, then Result.

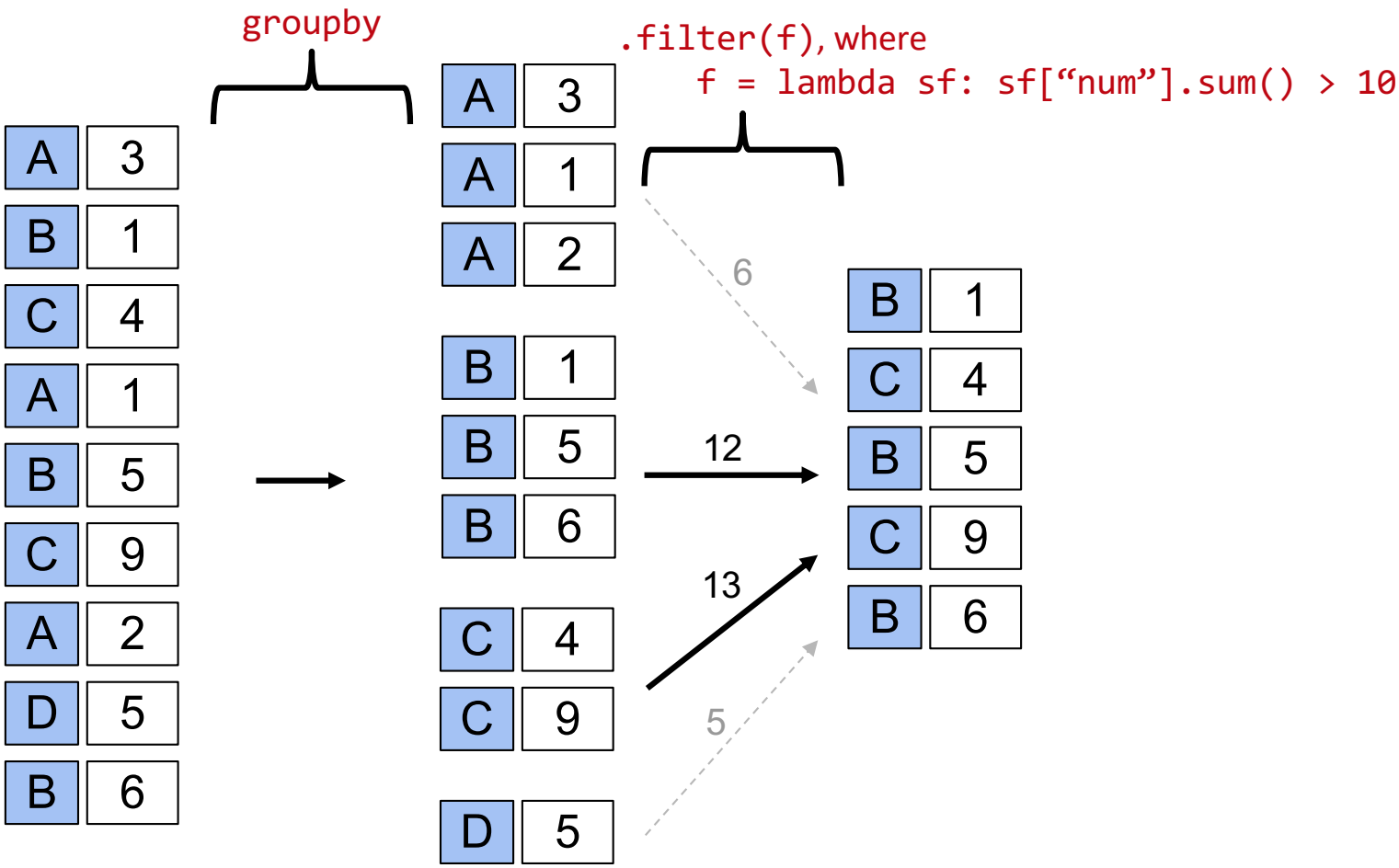
Party	Result	%	Year
Democratic	loss	44.850000	1995.333333
	win	49.050000	2002.000000
Independent	loss	11.300000	1989.333333
Republican	loss	42.750000	2002.000000
	win	51.266667	1995.333333

Filtering by Group

Another common use for groups is to filter data.

- `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.

Series groupby/filter Summary



isin

We saw last time how to build boolean arrays for filtering, e.g.

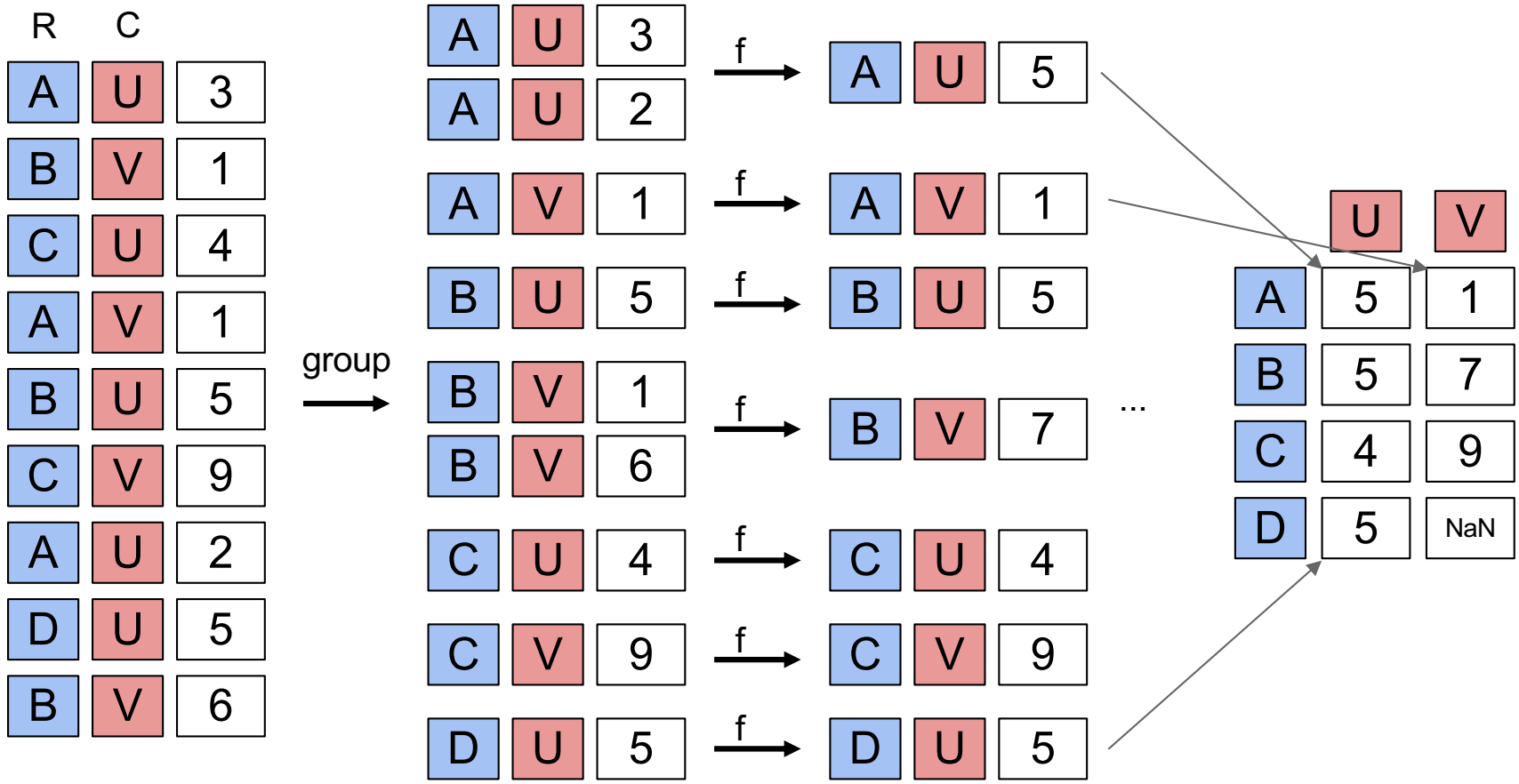
```
df["Party"] == "Democratic"
```

If we have a list of valid items, e.g. “Republican” or “Democratic”, we could use the | operator (| means or), but a better way is to use `isin`.

- Ugly:
```
df[(df["Party"] == "Democratic") | (df["Party"] == "Republican")]
```
- Better:
```
df[df["Party"].isin(["Republican", "Democratic"])]
```

A quick look at `pivot`

Pivot Tables



groupby Demo

See 03_groupby_basics.ipynb

Baby Names Case Study Q2

Baby Names

Let's try solving another real world problem using the baby names dataset: What was the most popular name in every state in every year and for every labeled gender?

- Spoiler alert, we will build a MultiIndexed DataFrame where the data column is "count".
- MultiIndex will be by state, year, and gender.

Head to case-study-exercise.ipynb.

- As with the previous case study, we'll use some stuff that we haven't formally learned (e.g. combining multiple dataframes that are stored across multiple files, %%time)!

Baby Names Case Study Q3