**Learning goals:**

- Gain familiarity with grouping and aggregation
- Understand that pivoting is an application of grouping
- Apply pandas knowledge to realistic data questions

# Intro to pandas, Part 2

---

**UC Berkeley Data 100 Summer 2019**
**Sam Lau**

(Slides adapted from Josh Hug and John DeNero)

# Announcements

For consistency with 61A and 61B, I will start using the term “method” everywhere to return to a function that belongs to a class.

- Example: `dog.bark()` indicates `bark` is a method of the `dog` class.
- Also correct to say `dog.bark()` indicates `bark` is a function of the `dog` class, but I’ll be using “method” instead.

# Announcements

- There is a live lecture Piazza thread: Ishaan will post soon.
- Lecture in 105 North Gate until further notice!
- Exam Conflict form link changed: http://bit.ly/su19-alt-final
  - Due Friday at 11:59pm
  - DSP exams will have a separate form, will send later
- Small group tutoring is starting next week; more info soon
- Project 1 released today! Due next Tues
- **HW1 deadline extended to tonight at 11:59pm.**
- **DSP students: I don't have access to your letters. Please email them to me.**

# Clarification on Indices

# Index for pandas vs. Index for Python

For a Python list, we say that:

```
mylist = ['hello', 'world', 'data', 100]
mylist[3]
```

Returns the element at index 3. All indexes are integers.

# Index for pandas vs. Index for Python

In pandas, the word “index” refers to the collection of labels for each row.

| State | Motto | Translation | Language | Date Adopted |
|---|---|---|---|---|
| **Alabama** | Audemus jura nostra defendere | We dare defend our rights! | Latin | 1923 |
| **Alaska** | North to the future | — | English | 1967 |
| **Arizona** | Ditat Deus | God enriches | Latin | 1863 |
| **Arkansas** | Regnat populus | The people rule | Latin | 1907 |
| **California** | Eureka (Εὕρηκα) | I have found it | Greek | 1849 |

# Index for pandas vs. Index for Python

Using .loc extracts values based on labels.

- `mottos.loc['California', 'Motto']`

Using .iloc extracts values based on positions.

- `mottos.iloc[2, 1]`
- Looks like Python list indexing!

**For clarity, I will always use “index” to refer to row labels.**

**I will use “position” when talking about `.iloc`.**

# Groupby (and `isin`)

# Last Time

We looked at the SSN (aka Baby Names) dataset

With slicing, we can answer questions like:

- What were the five most popular names in 2017?
- How many babies were born in California since 1950?
- How many males were born in California since 1950?

# Harder Questions

What about:

- What was the most popular male name during each year in the data?
- What are the three states with the most babies born?

We can answer them now but it’s a pain!

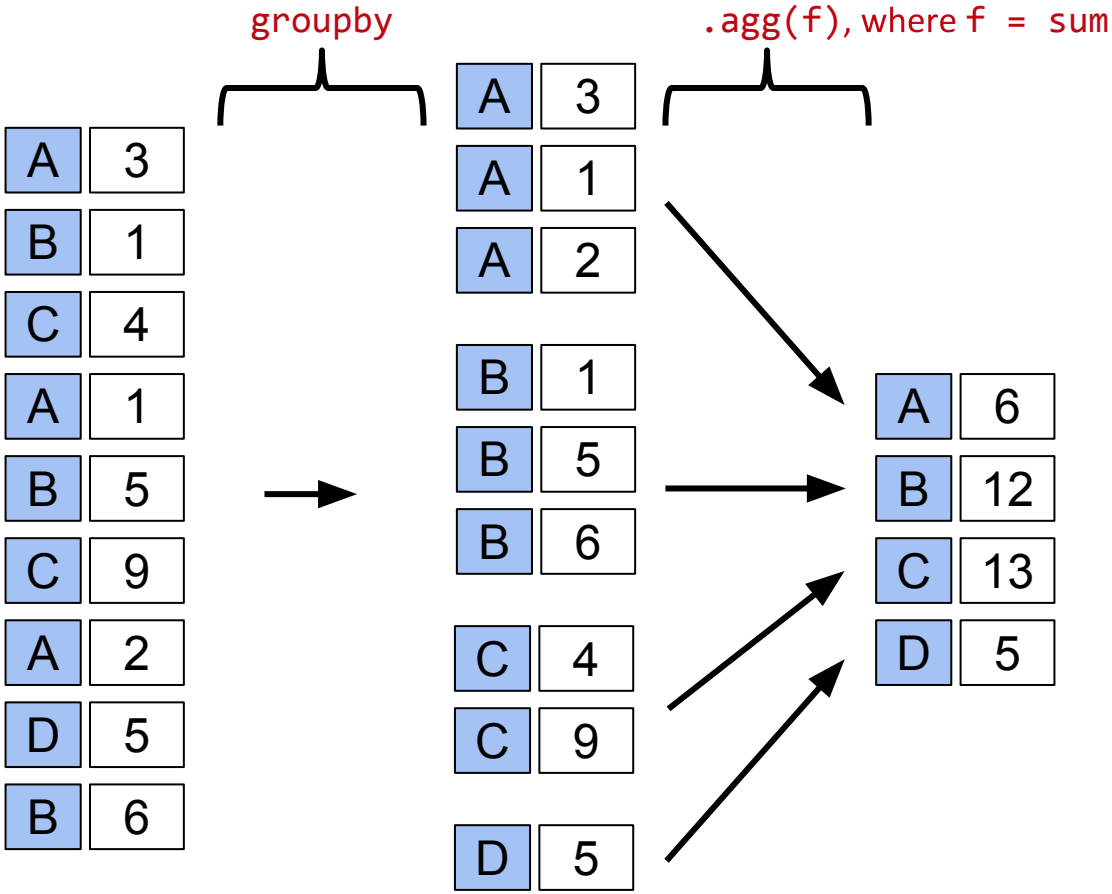
# groupby

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

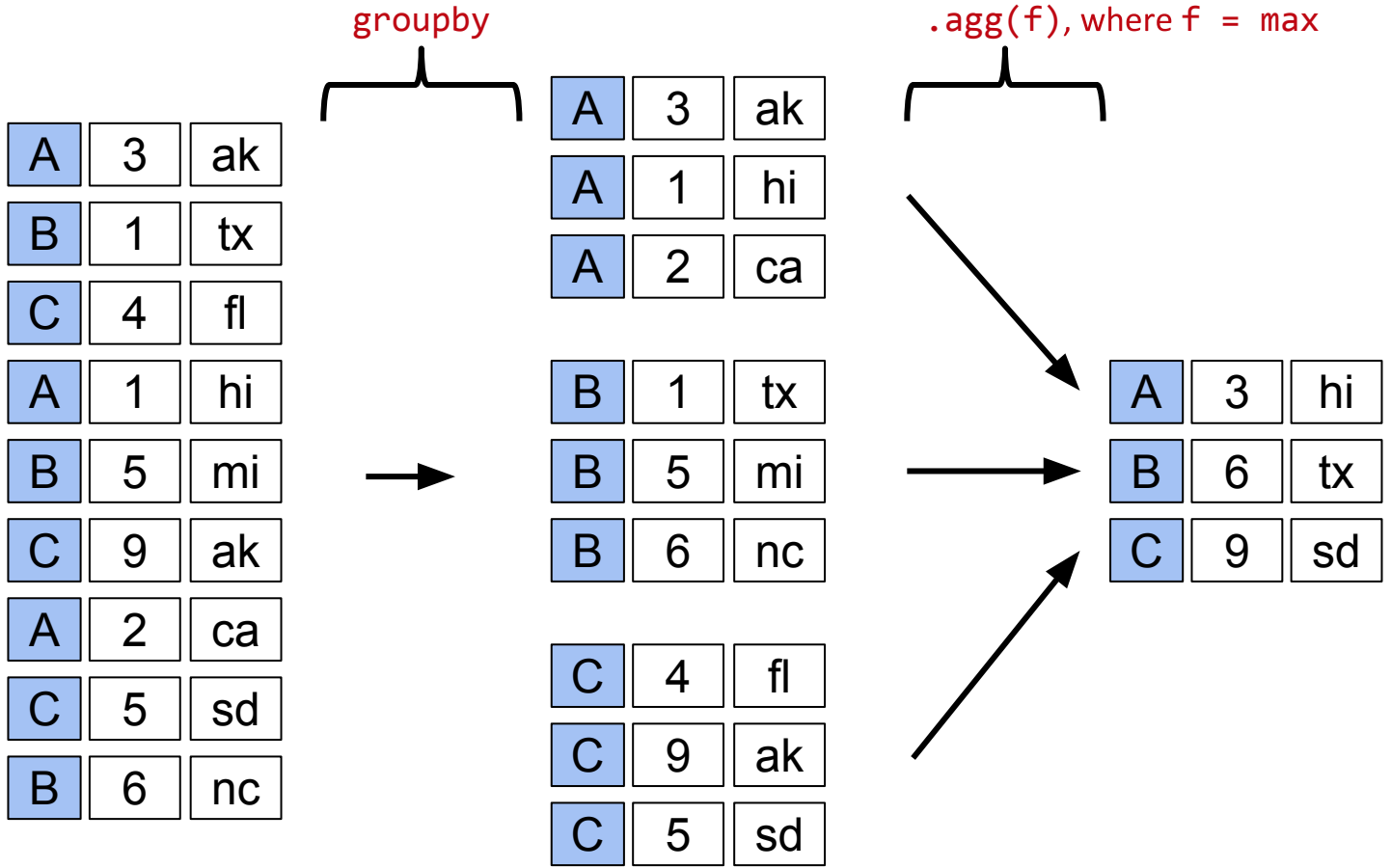
- What was the average share of the vote across all U.S. elections for each political **party**?
- What was the size of the average class in **each department** at Berkeley in **each term?**

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

# Series groupby/agg Summary



# DataFrame groupby/agg Summary



# groupby demo

# 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')}
```

# groupby Key Concepts

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

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

```
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')
```

```
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 |

# 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 `MultiIndex`

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 |

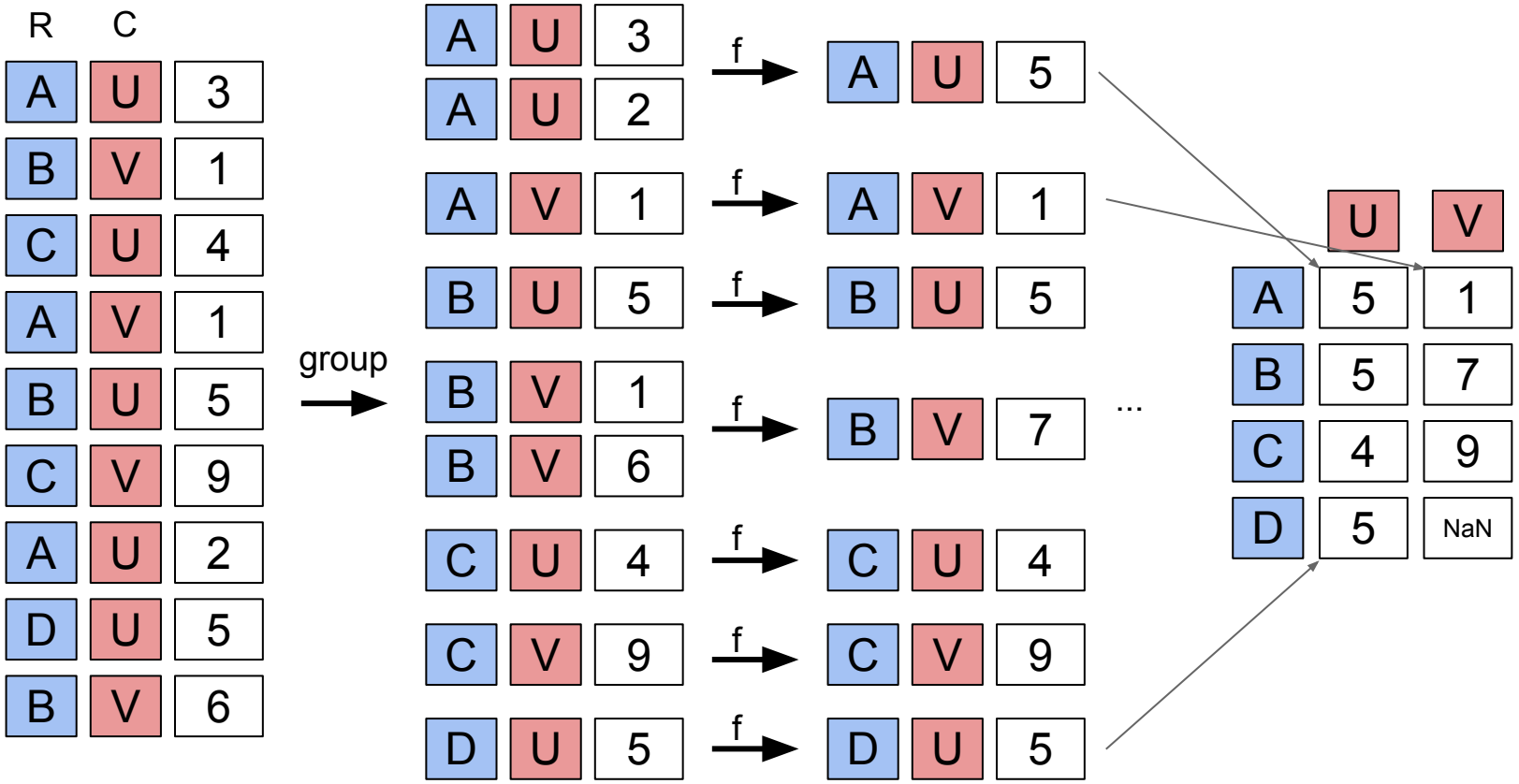
# Pivot Tables

You've already seen pivot tables in data 8.

- Most basic usage: Working with data that has been classified according to two variables.

**Pivot tables are like grouping by two columns**, then "rotating" one of the grouped columns to become the column labels.

# Pivot Tables

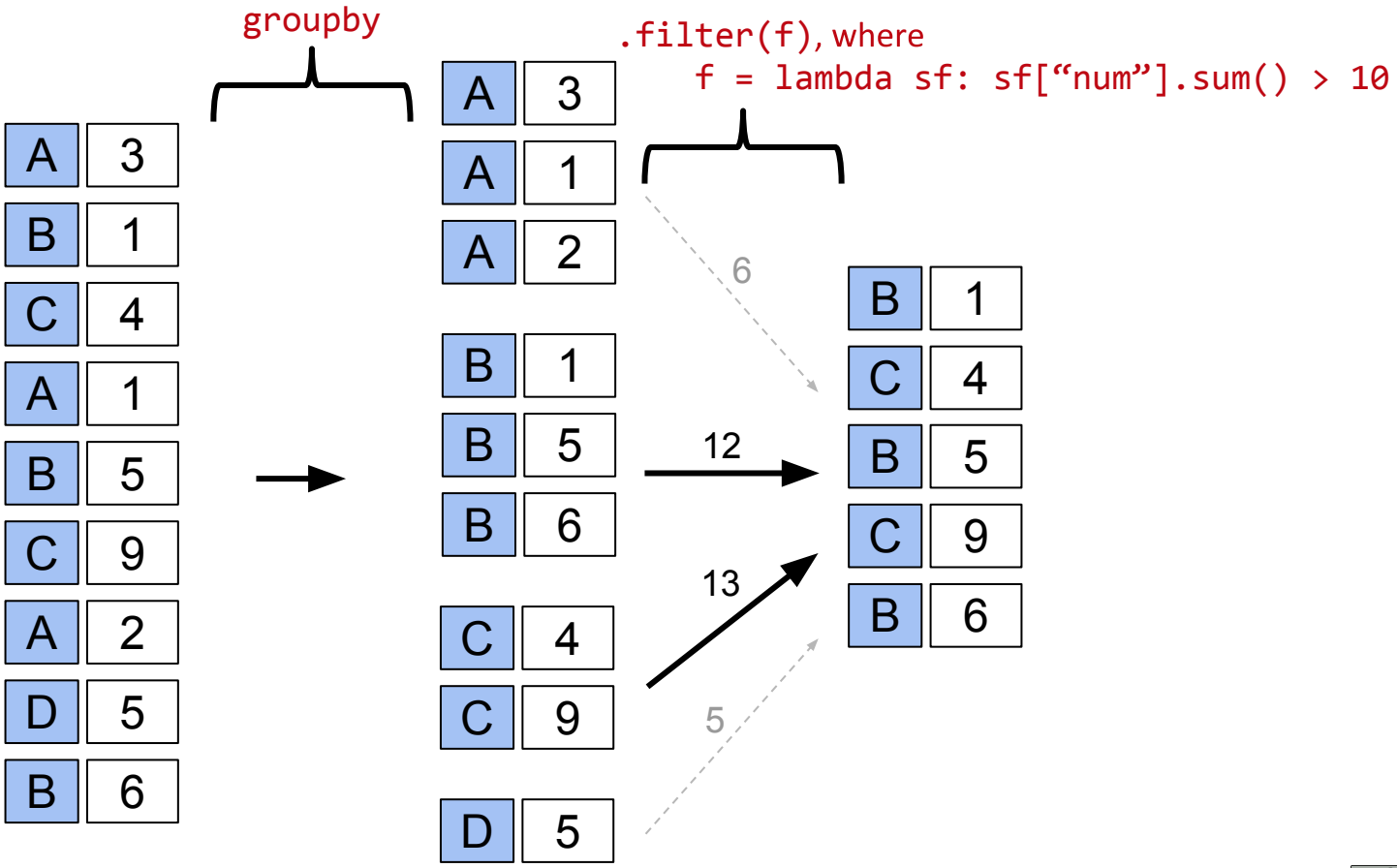


# Filtering by Group

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 slicing, 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"])]`

# Break!
# Fill out Attendance:
# http://bit.ly/at-d100

# Baby Names Case Study Q2

# Baby Names

Let's try solving another real world problem using the baby names dataset: **What were the most popular M/F names in CA for every year?**

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

**(demo)**

# Recap

1. Group by state and year
2. Aggregate using max

1. `df.groupby`
2. `grouped.agg(np.max)`

# How do I know when I need to group?

Do I need to count the number of times each value appears?

- Probably use `.value_counts()` here.

Do I need to aggregate values together?

**Do I find myself wanting to loop through the unique values in a column?**

# Practice Exercises on Enrollment Data

# Spring Enrollment Data

Suppose we have a DataFrame called `df` that contains all Spring offerings of courses in several departments offered at Berkeley between 2012 and 2018.

- Warmup: How would you find all offerings of this class? For reference, our course number is C100.

| | Term | Subject | Number | Title | Enrollment Cnt | Instructor |
|---|---|---|---|---|---|---|
| **0** | 2018 Spring | Computer Science | 9A | Programmers' Matlab | 8 | Carol Marshall; Paul Hilfinger |
| **1** | 2018 Spring | Computer Science | 9C | Programmers' C | 11 | Carol Marshall; Paul Hilfinger |
| **2** | 2018 Spring | Computer Science | 9E | Productive Unix Use | 36 | Carol Marshall; Paul Hilfinger |
| **3** | 2018 Spring | Computer Science | 9F | Programmers' C++ | 33 | Carol Marshall; Paul Hilfinger |
| **4** | 2018 Spring | Computer Science | 9G | Programmers' Java | 11 | Carol Marshall; Paul Hilfinger |

# Spring Enrollment Data

Suppose we have a DataFrame called `df` that contains all Spring offerings of courses in several departments offered at Berkeley between 2012 and 2018.

- Warmup: How would you find all offerings of this class? For reference, our course number is C100.

| | Term | Subject | Number | Title | Enrollment Cnt | Instructor |
|---|---|---|---|---|---|---|
| **20** | 2018 Spring | Computer Science | C100 | Princ&Tech Data Sci | 610 | Fernando Perez; Joseph Gonzalez |
| **387** | 2017 Spring | Computer Science | C100 | Princ&Tech Data Sci | 67 | Bin Yu; Deborah Nolan; Joseph Gonzalez; Joseph... |
| **670** | 2017 Spring | Statistics | C100 | Princ&Tech Data Sci | 28 | Bin Yu; Deborah Nolan; Joseph Gonzalez; Joseph... |

# Spring Enrollment Data

**Challenge 2:** Create a series where each row correspond to one subject (e.g. English), and each value corresponds to the average number of students for courses in that subject. For example, your series might have a row saying that the average number of students in a Computer Science class is 88.

# Spring Enrollment Data

**Challenge 3:** Create a multi-indexed series where each row corresponds to one subject (e.g. English) offered during one semester (e.g. Spring 2017), and each value corresponds to the maximum number of students for courses in that subject during that semester.

For example, you might have a row saying that the maximum number of students in a computer science course during Spring 2012 was 575.

# Spring Enrollment Data

**Challenge 4:** Try to compute the size of the largest class ever taught by each instructor. This challenge is stated more vaguely on purpose. You'll have to decide what the data structure looks like. Your result should be sorted in decreasing order of class size.

# Baby Names Case Study Q3

# Baby Names

Let's try solving another real world problem using the baby names dataset: **Can we deduce a person's birth sex from the last letter of their name?**

As with the previous case study, we'll use some stuff that we haven't formally learned (e.g. visualization)!

# Recap

1. Get last letter of each name.
2. Group by last letter
3. Visualize distributions

1. `Series.str`
2. `df.groupby`
3. `df.plot`

# Summary

groupby: one of the most useful (and complicated!) methods for aggregating and summarizing values.

Making a pivot table is essentially doing a groupby on two columns plus a “rotation”.

In the following lectures, we use these pandas tools to start asking real-world data questions!