CS334: Principles and Techniques of Data Science

Lecture 3

Mobin Javed
Ihsan Ayyub Qazi

Slides partially adapted from DS100 at UC Berkeley

Please download

Lecture Slides / Lecture 3 / hands-on.zip

from LMS

Plan for Today

- Finish case-study from last time: are first-borns more likely to attend Harvard?
- Introduce Jupyter notebooks
- Introduce Pandas, with emphasis on:
  - Key Data Structures (data frames, series, indices).
  - How to index into these structures.
  - How to read files to create these structures.
  - Other basic operations on these structures.
- Go over some important and handy iPython features and concepts:
  - Shell commands (e.g. !dir or !ls).
  - Portable vs. operating system specific code.
- Exercise: Solve a basic data science problem using Pandas.

Are first-borns more likely to attend Harvard?

Are First Borns More Likely to Attend Harvard?

- Harvard Philosopher Michael Sandel (among others) argues that birth order has a significant impact on work ethic and therefore life outcomes.

- Asked in class what fraction of students were first born, and found that 75-80% were consistently first born! P(F | H) = 75 to 80%

- What's going on?

Are First Borns More Likely to Attend Harvard?

- Harvard Philosopher Michael Sandel (among others) argues that birth order has a significant impact on work ethic and therefore life outcomes.

- Asked in class what fraction of students were first born, and found that 75-80% were consistently first born! P(F | H) = 75 to 80%

- What's going on?

Obvious question, what fraction of children are first born?

US Census Data for mothers with college age children is:

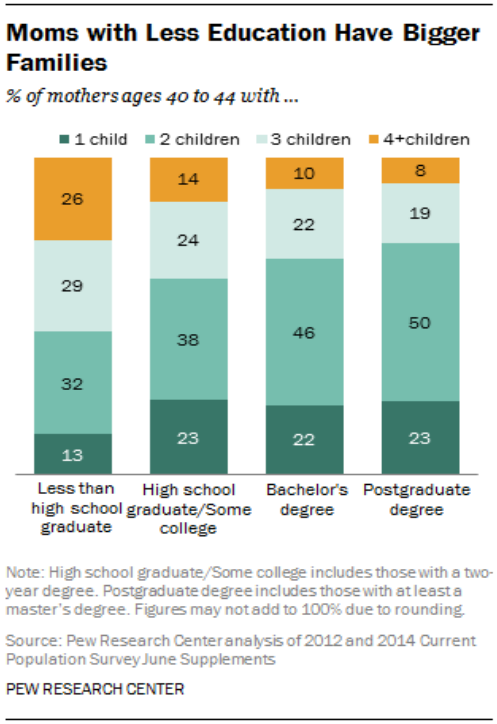
Number of Children			
1	2	3	4
21%	43%	23%	13%

Percent first born:

100/(21 + 43*2 + 23*3 + 13*4)=44%

Are First Borns More Likely to Attend Harvard?

According to Pew Research Center



How might this impact the proportion of first born at Harvard?

http://www.pewsocialtrends.org/2015/05/07/family-size-among-mothers/

Are First Borns More Likely to Attend Harvard?

- A 2012 article walks through the math regarding Sandel's argument in great detail. Sandel measures P(F | H), but what he cares about is a different quantity.

- Sandel wants to show that being a firstborn conveys an advantage over NOT being firstborn. In other words, he wants to show that:

  $$r = P(H \mid F) / P(H \mid F^c) \text{ is greater than } 1$$

  where

  P(H | F): Probability of getting into Harvard given you are firstborn

  $P(H \mid F^c)$: Probability of getting into Harvard given you are NOT firstborn

Recall Baye's Rule

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

$$P(A|B)\,P(B) = P(A \cap B) = P(B|A)\,P(A)$$

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

Find the chance first born is at Harvard

- We want to find the chance you are a Harvard student (H) given you are first born (F) and
- Compare it to the chance you are a Harvard student (H) given you are not first born (F^c)

$$P(H \mid F) = \frac{P(H \cap F)}{P(F)}$$

$$P(H \mid F^c) = \frac{P(H \cap F^c)}{P(F^c)}$$

Find the chance first born is at Harvard

- Recall

$$P(H \cap F) = P(H)P(F \mid H)$$

which implies Bayes Rule: $$P(H \mid F) = \frac{P(H)P(F \mid H)}{P(F)}$$

- Substituting using Bayes rule, the ratio becomes:

$$\frac{P(H \mid F)}{P(H \mid F^c)} = \frac{P(F \mid H)}{P(F^{c} \mid H)} \times \frac{1 - P(F)}{P(F)}$$

Sandel measured P(F|H) = 3/4. We also need P(F), which is determined by the fertility rate of Harvard mothers

Find the chance first born is at Harvard

- In a population of ƛN students, there are exactly N first-borns (where ƛ is the fertility rate)

- Giving P(F) = 1/ ƛ

Sandel measured P(F|H) = 3/4. We also need P(F), which is determined by the fertility rate of Harvard mothers

$$P(F) = \frac{1}{fertility}$$

$$r = \frac{P(H \mid F)}{P(H \mid F^c)} = \frac{3/4}{1/4} \times \frac{1 - P(F)}{P(F)} = 3(fertility - 1)$$

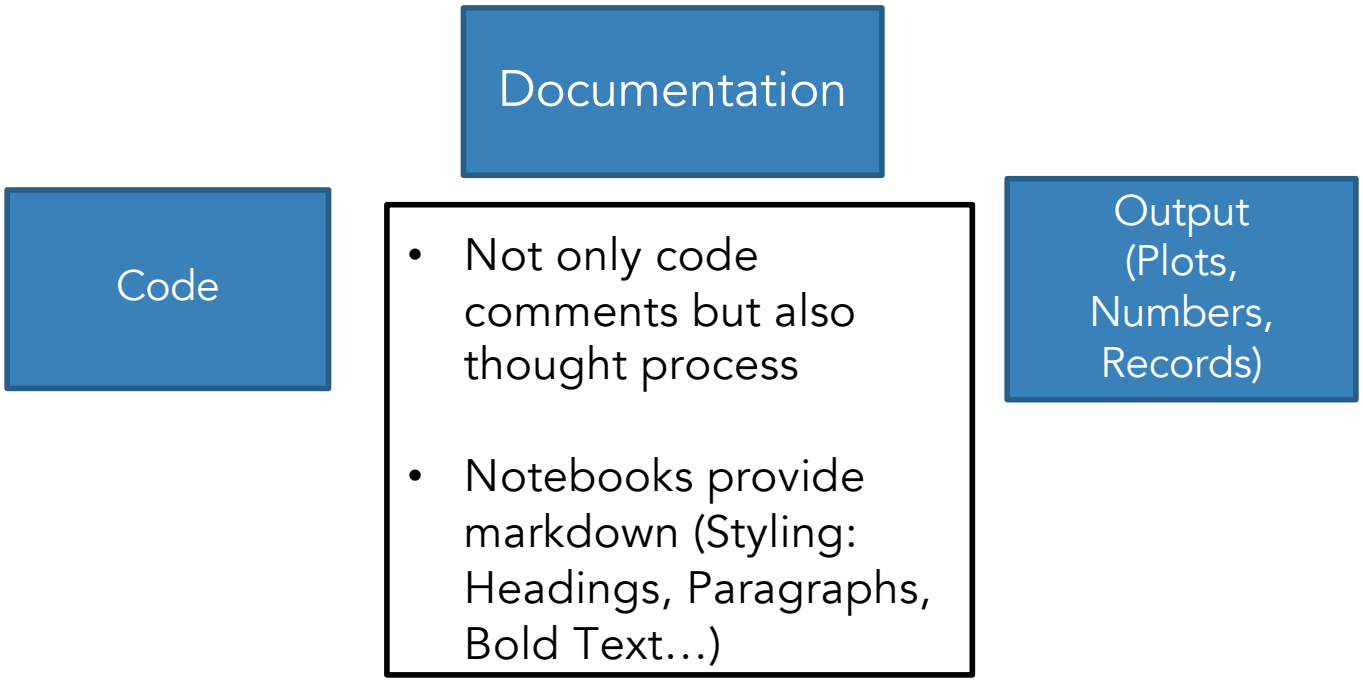
If we plug in the US value for 1994 we get: r = 3.9

If we plug in the US highest educated value from Pew we get: r = 3.2

Jupyter Notebooks

Why Notebooks?

Allows a data scientist to tell a narrative



Example Notebook

ipython_notebook_tutorial.pynb

Pandas Data Structures:
Data Frames, Series, and Indices

Pandas Data Structures

There are three fundamental data structures in pandas:

- Data Frame: 2D data tabular data.
- Series: 1D data. I usually think of it as columnar data.
- Index: A sequence of row labels.

Data Frame

	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Series

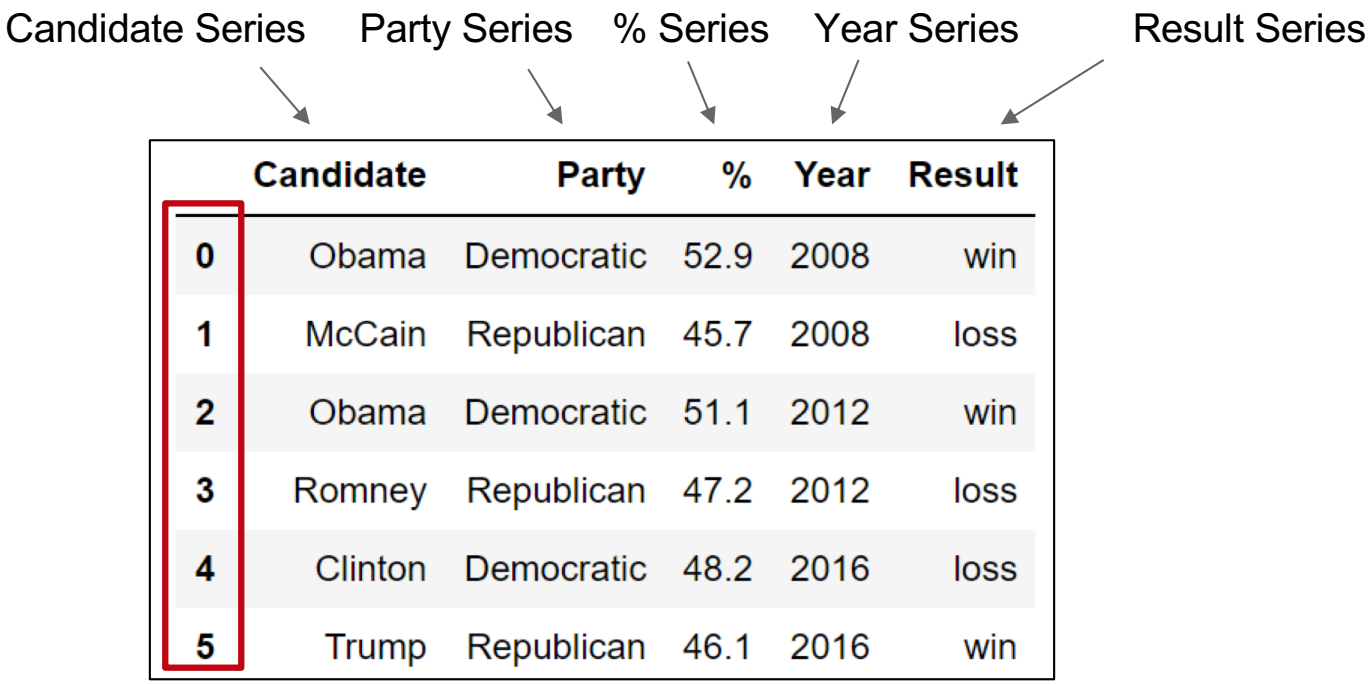
```
0      Obama
1     McCain
2      Obama
3     Romney
4    Clinton
5      Trump
Name: Candidate, dtype: object
```

Index

The Relationship Between Data Frames, Series, and Indices

We can think of a Data Frame as a collection of Series that all share the same Index.

- Candidate, Party, %, Year, and Result Series all share an index from 0 to 5.



	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Non-native English speaker note: The plural of “series” is “series”. Sorry.

Indices Are Not Necessarily Row Numbers

Indices (a.k.a. row labels) can also:

- Be non-numeric.
- Have a name, e.g. “State”.

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

Indices

The row labels that constitute an index do not have to be unique.

- Left: The index values are all unique and numeric, acting as a row number.
- Right: The index values are named and non-unique.

	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Year	Candidate	Party	%	Result
2008	Obama	Democratic	52.9	win
2008	McCain	Republican	45.7	loss
2012	Obama	Democratic	51.1	win
2012	Romney	Republican	47.2	loss
2016	Clinton	Democratic	48.2	loss
2016	Trump	Republican	46.1	win

Column Names Must Be Unique!

Column names in Pandas are always unique!

- Example: Can't have two columns named "Candidate".

	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Hands On Exercise

Let’s experiment with reading csv files and playing around with indices.

- See 02-pandas-basics.ipynb.

Indexing with The [] Operator

Indexing by Column Names Using [] Operator

Given a dataframe, it is common to extract a Series or a collection of Series. This process is also known as “Column Selection” or sometimes “indexing by column”.

```
elections[["Candidate", "Party"]].head(6)
```

	Candidate	Party
0	Reagan	Republican
1	Carter	Democratic
2	Anderson	Independent
3	Reagan	Republican
4	Mondale	Democratic
5	Bush	Republican

- Column name argument to [] yields Series.
- List argument to [] yields a Data Frame.

```
elections["Candidate"].head(6)
```

```
0       Reagan
1       Carter
2    Anderson
3       Reagan
4      Mondale
5         Bush
Name: Candidate, dtype: object
```

Indexing by Column Names Using [] Operator

Given a dataframe, it is common to extract a Series or a collection of Series. This process is also known as “Column Selection” or sometimes “indexing by column”.

```
elections[["Candidate"]].head(6)
```

	Candidate
0	Reagan
1	Carter
2	Anderson
3	Reagan
4	Mondale
5	Bush

- Column name argument to [] yields Series.
- List argument (**even of one name**) to [] yields a Data Frame.

```
elections["Candidate"].head(6)
```

```
0       Reagan
1       Carter
2    Anderson
3       Reagan
4      Mondale
5         Bush
Name: Candidate, dtype: object
```

Indexing by Row Slices Using [] Operator

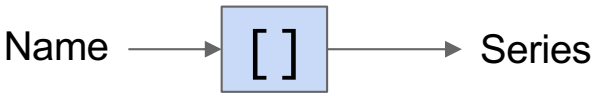
We can also index by row numbers using the [] operator.

- Numeric slice argument to [] yields rows.
- Example: [0:3] yields rows 0 to 2.

```
elections[0:3]
```

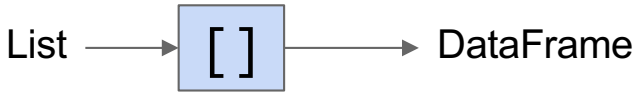
	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
1	Carter	Democratic	41.0	1980	loss
2	Anderson	Independent	6.6	1980	loss

[] Summary



Single Column Selection

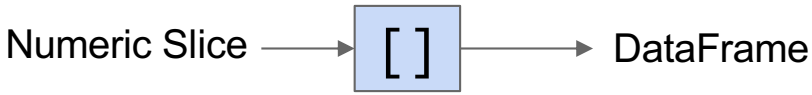
```
elections["Candidate"].head(6)
0       Reagan
1       Carter
2    Anderson
3       Reagan
4      Mondale
5         Bush
Name: Candidate, dtype: object
```



Multiple Column Selection

```
elections[["Candidate"]].head(6)
```

	Candidate
0	Reagan
1	Carter
2	Anderson
3	Reagan
4	Mondale
5	Bush



(Multiple) Row Selection

```
elections[0:3]
```

	Candidate	Party	%	Year	Result
0	Reagan	Republican	50.7	1980	win
1	Carter	Democratic	41.0	1980	loss
2	Anderson	Independent	6.6	1980	loss

Note: Row Selection Requires Slicing!!

`elections[0]` will not work unless the elections data frame has a column whose name is the numeric zero.

- Note: It is actually possible or columns to have names that non-String types, e.g. numeric, datetime etc.

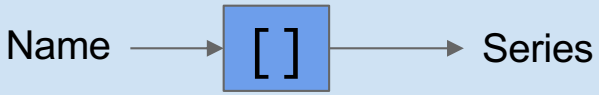
Question

```
weird = pd.DataFrame({1:["topdog","botdog"], "1":["topcat","botcat"]})
weird
```

	1	1
0	topdog	topcat
1	botdog	botcat

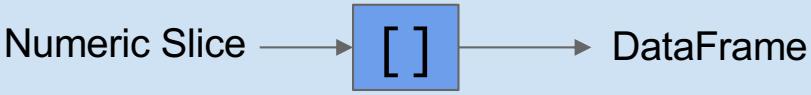
Try to predict the output of the following:

- `weird[1]`
- `weird[“1”]`
- `weird[1:]`



Single Column Selection

Multiple Column Selection



(Multiple) Row Selection

Boolean Array Selection

Boolean Array Input

Yet another input type supported by [] is the boolean array.

```
elections[[False, False, False, False, False,
           False, False, True, False, False,
           True, False, False, False, True,
           False, False, False, False, False,
           False, False, True]]
```

	Candidate	Party	%	Year	Result
7	Clinton	Democratic	43.0	1992	win
10	Clinton	Democratic	49.2	1996	win
14	Bush	Republican	47.9	2000	win
22	Trump	Republican	46.1	2016	win

Boolean Array Input

Yet another input type supported by [] is the boolean array. Useful because boolean arrays can be generated by using logical operators on Series.

Length 23 Series where every entry is "Republican", "Democrat" or "Independent."

Length 23 Series where every entry is either "True" or "False", where "True" occurs for every independent candidate.

```
elections[elections['Party'] == 'Independent']
```

	Candidate	Party	%	Year	Result
2	Anderson	Independent	6.6	1980	loss
9	Perot	Independent	18.9	1992	loss
12	Perot	Independent	8.4	1996	loss

Boolean Array Input

Boolean Series can be combined using the & operator, allowing filtering of results by multiple criteria.

```
elections[(elections['Result'] == 'win')
          & (elections['%'] < 50)]
```

	Candidate	Party	%	Year	Result
7	Clinton	Democratic	43.0	1992	win
10	Clinton	Democratic	49.2	1996	win
14	Bush	Republican	47.9	2000	win
22	Trump	Republican	46.1	2016	win

Indexing with loc and iloc

Loc and iloc

Loc and iloc are alternate ways to index into a DataFrame.

- They take a lot of getting used to! Documentation and ideas behind them are quite complex.
- I'll go over common usages (see docs for weirder ones).

Documentation:

- loc: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html
- iloc: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.iloc.html
- More general docs on indexing and selecting: Link

Loc

Loc does two things:

- Access values by labels.
- Access values using a boolean array (a la Boolean Array Selection).

Loc with Lists

The most basic use of loc is to provide a list of row and column labels, which returns a DataFrame.

```
elections.loc[[0, 1, 2, 3, 4], ['Candidate','Party', 'Year']]
```

	Candidate	Party	Year
0	Reagan	Republican	1980
1	Carter	Democratic	1980
2	Anderson	Independent	1980
3	Reagan	Republican	1984
4	Mondale	Democratic	1984

Loc with Slices

Loc is also commonly used with slices.

- Slicing works with all label types, not just numeric labels.
- Slices with loc are **inclusive**, not **exclusive.**

```
elections.loc[0:4, 'Candidate':'Year']
```

	Candidate	Party	Year
0	Reagan	Republican	1980
1	Carter	Democratic	1980
2	Anderson	Independent	1980
3	Reagan	Republican	1984
4	Mondale	Democratic	1984

Loc with Single Values for Column Label

If we provide only a single label as column argument, we get a Series.

```
elections.loc[0:4, 'Candidate']
```

```
0       Reagan
1       Carter
2    Anderson
3       Reagan
4      Mondale
Name: Candidate, dtype: object
```

Loc with Single Values for Column Label

As before with the [] operator, if we provide a list of only one label as an argument, we get back a dataframe.

```
elections.loc[0:4, 'Candidate']
```

```
0       Reagan
1       Carter
2    Anderson
3       Reagan
4      Mondale
Name: Candidate, dtype: object
```

```
elections.loc[0:4, ['Candidate']]
```

	Candidate
0	Reagan
1	Carter
2	Anderson
3	Reagan
4	Mondale

Loc with Single Values for Row Label

If we provide only a single row label, we get a Series.

- Instead of the Series being one of the columns from the original data frame, it is a Series made up of the values from the requested row.
- The index of this Series is the names of the columns from the data frame.
- Putting the single row label in a list yields a dataframe version.

```
elections.loc[0, 'Candidate':'Year']
```

```
Candidate       Reagan
Party       Republican
%                 50.7
Year              1980
Name: 0, dtype: object
```

```
elections.loc[[0], 'Candidate':'Year']
```

	Candidate	Party	%	Year
0	Reagan	Republican	50.7	1980

Loc Supports Boolean Arrays

Loc supports Boolean Arrays exactly as you'd expect.

```
elections.loc[(elections['Result'] == 'win') & (elections['%'] < 50), 'Candidate':'%']
```

	Candidate	Party	%
7	Clinton	Democratic	43.0
10	Clinton	Democratic	49.2
14	Bush	Republican	47.9
22	Trump	Republican	46.1

iloc: Integer-Based Indexing for Selection by Position

In contrast to loc, iloc doesn't think about labels at all. Instead, it returns the items that appear in the numerical positions specified.

```
elections.iloc[0:3, 0:3]
```

	Candidate	Party	%
0	Reagan	Republican	50.7
1	Carter	Democratic	41.0
2	Anderson	Independent	6.6

```
mottos.iloc[0:3, 0:3]
```

State	Motto	Translation	Language
Alabama	Audemus jura nostra defendere	We dare defend our rights!	Latin
Alaska	North to the future	—	English
Arizona	Ditat Deus	God enriches	Latin

Advantages of loc:

- Harder to make mistakes.
- Easier to read code.
- Not vulnerable to changes to the ordering of rows/cols in raw data files.

Nonetheless, iloc can be more convenient. Use iloc judiciously.

Handy Properties and Utility Functions for Series and DataFrames

`head`, `size`, `shape`, and `describe`

`head`: Displays only the top few rows.

`size`: Gives the total number of data points.

`shape`: Gives the size of the data in rows and columns.

`describe`: Provides a summary of the data.

`index` and `columns`

`index`: Returns the index (a.k.a. row labels).

`columns`: Returns the labels for the columns.

The `sort_values` Method

One incredibly useful method for DataFrames is `sort_values`, which creates a copy of a DataFrame sorted by a specific column.

```
elections.sort_values('%', ascending=False)
```

	Candidate	Party	%	Year	Result
3	Reagan	Republican	58.8	1984	win
5	Bush	Republican	53.4	1988	win
17	Obama	Democratic	52.9	2008	win
19	Obama	Democratic	51.1	2012	win
0	Reagan	Republican	50.7	1980	win

The `sort_values` Method

We can also use `sort_values` on a Series, which returns a copy with with the values in order.

```
mottos['Language'].sort_values().head(5)
```

```
State
Washington          Chinook Jargon
Wyoming                    English
New Jersey                 English
New Hampshire              English
Nevada                     English
Name: Language, dtype: object
```

The `value_counts` Method

Series also has the function `value_counts`, which creates a new Series showing the counts of every value.

```
elections['Party'].value_counts()
```

```
Democratic     10
Republican     10
Independent     3
Name: Party, dtype: int64
```

The unique Method

Another handy method for Series is unique, which returns all unique values as an array.

```
mottos['Language'].unique()
```

```
array(['Latin', 'English', 'Greek', 'Hawaiian', 'Italian', 'French',
       'Spanish', 'Chinook Jargon'], dtype=object)
```

Baby Names Case Study Q1

Baby Names

Let's try solving a real world problem using the baby names dataset: What was the most popular name in California in 2017?

Along the way, we'll see some examples of what it's like to deal with real data, and will also explore some fancy iPython features.

See 02-case-study.ipynb.