YData: Introduction to Data Science

Class 17: Text manipulation

Overview

Basic text manipulation

Regular expressions

If there is time:

- Introduction to statistical inference

Announcement: Homework 7

Homework 7 has been posted!

It is due on Gradescope on Sunday March 31st at 11pm

Also keep working on your final projects

- A (very polished) draft of final project is due April 7th

Where we are and where we're going…

What we have covered:

- What is Data Science
- Basics of Python (data types, lists, etc.)
- Numerical computations (numpy)
- Data tables (pandas)
- Data visualization (matplotlib and seaborn)
- Functions and for loops

Today: Text manipulation

- Could be helpful for final project

The rest of the semester:

- Statistical analysis
- Machine Learning
- Ethics and conclusions


text
MaNiPuLaTiOn

Text manipulation

80% of a Data Scientists time is cleaning data

- Text manipulation is a big part of cleaning data

20% of a Data Scientists time is complaining about cleaning data

Python has many string methods that are useful for manipulating text and cleaning data!

Text manipulation: capitalization

Some of the simplest string methods involve changing capitalization.

Changing capitalization can be useful when joining DataFrames

- i.e., if they Index values are the same, but the values have different capitalization
    - For example, joining different countries, but in one DataFrame the country names are capitalized and in the other they are not

Text manipulation: capitalization

Python strings have a number of methods to change the capitalization of words including:

- .capitalize(): Converts the first character to upper case
- .lower(): Converts a string into lower case
- .upper(): Converts a string into upper case
- .title(): Converts the first character of each word to upper case
- .swapcase(): Swaps cases, lower case becomes upper case and vice versa

Let's explore this in Jupyter!

Text manipulation: string padding

Often we want to remove extra spaces (called "white space") from the front or end of a string

Conversely, sometimes we want to add extra spaces to make a set of strings the same length

- This is known as "string padding"

Python strings have a number of methods that can pad/trim strings including:

- .strip(): Returns a trimmed version of the string (i.e., with no leading or trailing white space)
  - Also, .rstrip() and .lstrip(): Returns a right/left trim version of the string
- .center(num): Returns a centered string (with equal padding on both sides)
  - Also .ljust(num) and .rjust(num): Returns a right justified version of the string
- .zfill(num): Fills the string with a specified number of 0 values at the beginning

Let's explore this in Jupyter!

Text manipulation: checking string properties

There are also many functions to check properties of strings including:

- .isalnum(): Returns True if all characters in the string are alphanumeric
- .isalpha(): Returns True if all characters in the string are in the alphabet
- .isnumeric(): Returns True if all characters in the string are numeric

- .isspace(): Returns True if all characters in the string are whitespaces

- .islower(): Returns True if all characters in the string are lower case
- .isupper(): Returns True if all characters in the string are upper case
- .istitle(): Returns True if the string follows the rules of a title

Let's explore this in Jupyter!

Text manipulation: splitting and joining strings

There are several methods that can help us join strings that are contained into a list into a single string, or conversely, parse a single string into a list of strings. These include:

- .split(separator_string): Splits the string at the specified separator, and returns a list
- .splitlines(): Splits the string at line breaks and returns a list
- .join(a_list): Converts the elements of an iterable into a string

Let's explore this in Jupyter!

Text manipulation: finding and replacing substrings

Some methods for locating a substring within a larger string include:

- .count(substring): Returns the number of times a specified value occurs in a string
- .rfind(substring): Searches the string for a specified value and returns the last position of where it was found.
- .startswith(substring): Returns true if the string starts with the specified value
- .endswith(substring) : Returns true if the string ends with the specified value
- .replace(original_str, replacement_str): Replace a substring with a different string.

Let's explore this in Jupyter!

Text manipulation: filling in strings with values

There are a number of ways to fill in strings parts of a string with particular values.

Perhaps the most useful is to use "f strings", which have the following syntax such as:

- value_to_fill = "my_value"
- f"my string {value_to_fill} will be filled in"

Let's explore this in Jupyter!

Regular expressions!

Regular expressions

Regular expressions are string that allow you find more complex patterns in pieces of text

- They are powerful although can be a bit hard to read

```
[^ ]*?@[^ ]*?\.[^ ]*
```

To use regular expressions in Python we can import the re module

```
import re
```

We can check if a piece of text contains a particular substring by converting the output of re.match() method into a Boolean

```
bool(re.match("regular_expression", "piece_of_text"))
```

Regular expressions

[ ] means match anything in the range inside the braces

- "ch[io]mp" matches "chimp" and "chomp"

^ indicates the start of a string

- ^z matches words that start with a lower case z

$ indicates the end of a string

- z$ matches words that end with a lower case z

Note: if the ^ appears inside square braces it means not

- ^[^aeiou] matches words that don't start with a lower case vowel

The following are special regular expression characters that are reserved:

. * \ $ {} [] ^ ?

Let's explore this in Jupyter!

Regular expressions: wildcard characters

. (period) matches any single character

- bool(re.match("m.ss", "mess"))

* means match 0 or more of the preceding character

- bool(re.match("xy*z", ”xz"))

+ means match 1 or more of the preceding character

- bool(re.match("xy+z", "xz"))

will the following match?

- bool(re.match(".*a.*e", "pineapple"))

Example

```
phone_strings = [ "apple",
                  "219 733 8965",
                  "329-293-8753",
                  "Work: 579-499-7527",
                  "Home: 203.867.9305"]
```

The phone number can be matched with the regular expression:

```
".*([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
```

Let's explore this in Jupyter!

Escape sequences

In regular expressions a period (.) means any character

- So how can you detect if a period is in a string?

Escape sequences in R start with two slashes \\ and cause the next character to be treated literally rather than as a special character

- To match a period we use \\.
- To match a $ symbol we use \\$

Example

- bool(re.match(".*\\$100", "Joanna has $100 and Chris has $0"))

Character classes

Other special characters are also designated by using a double slash first

- \s space
- \n new line or also \r
- \t tab

Let's explore this in Jupyter!


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Statistical Inference

Inference

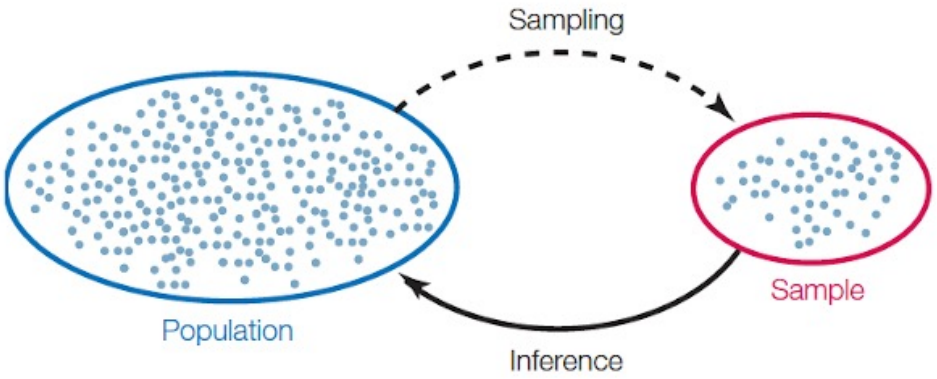
Statistical Inference: Making conclusions about a population based on data in a random sample

This usually involves using data in a sample to estimate the value of a fixed unknown number

- i.e., we estimate values of a "parameter"

Example:

- Estimating the average height of all humans on Earth from a random sample of 1,000 humans
  - Our estimate will vary from sample to sample



Terminology

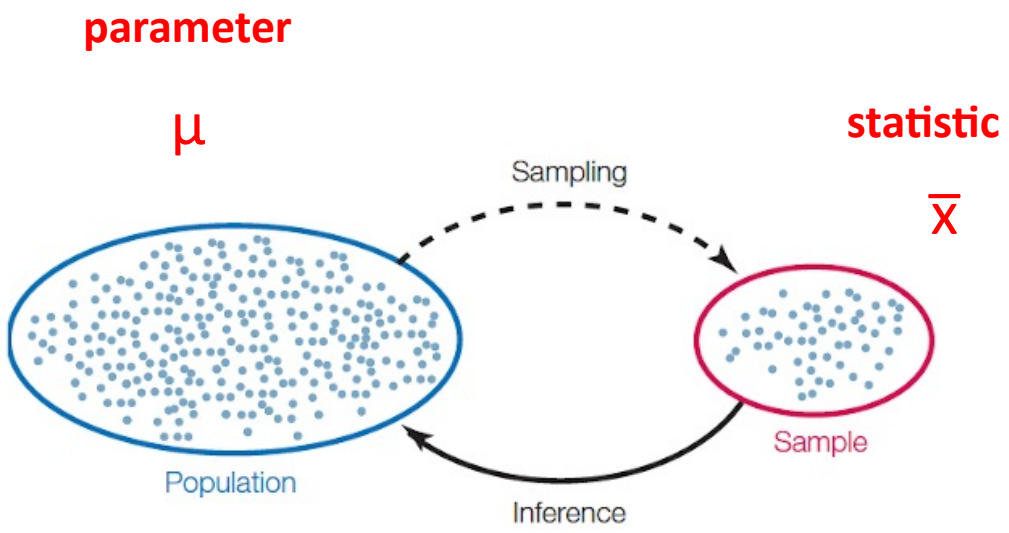
A parameter is number associated with the population

- e.g., population mean μ
- e.g., average height of all humans

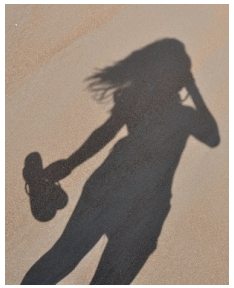
A **statistic** is number calculated from the sample

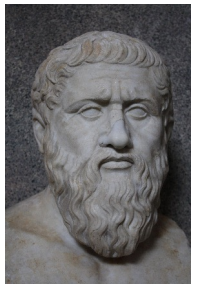
- e.g., sample mean $\bar{x}$
- e.g., average height of 1,000 people in our sample

A statistic can be used as an estimate of a parameter



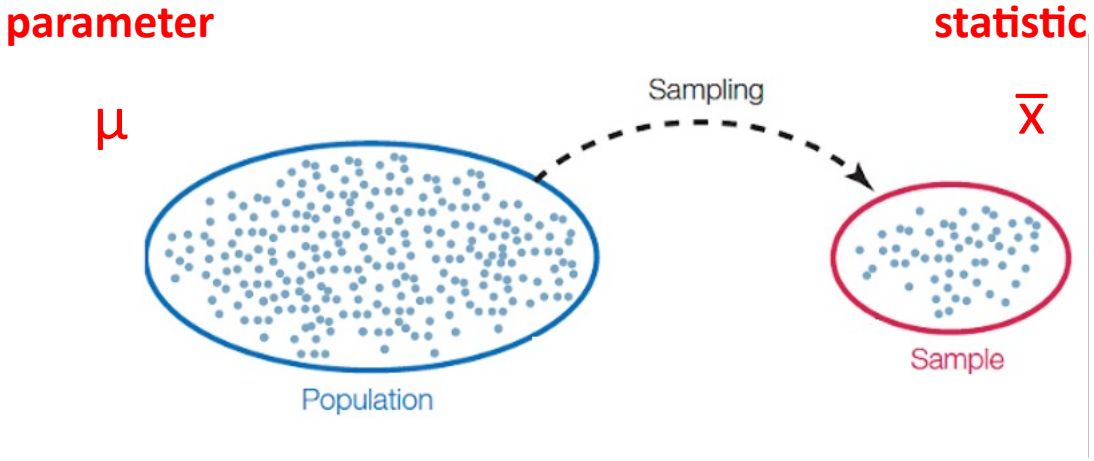
Examples of parameters and statistics





	Sample Statistic	Population Parameter
Mean	$\bar{x}$	μ
Proportion	$\hat{p}$	π
Standard deviation	s	σ
Correlation	r	ρ
Regression slope	b	β

Sampling



Simple random sample: each member in the population is equally likely to be in the sample

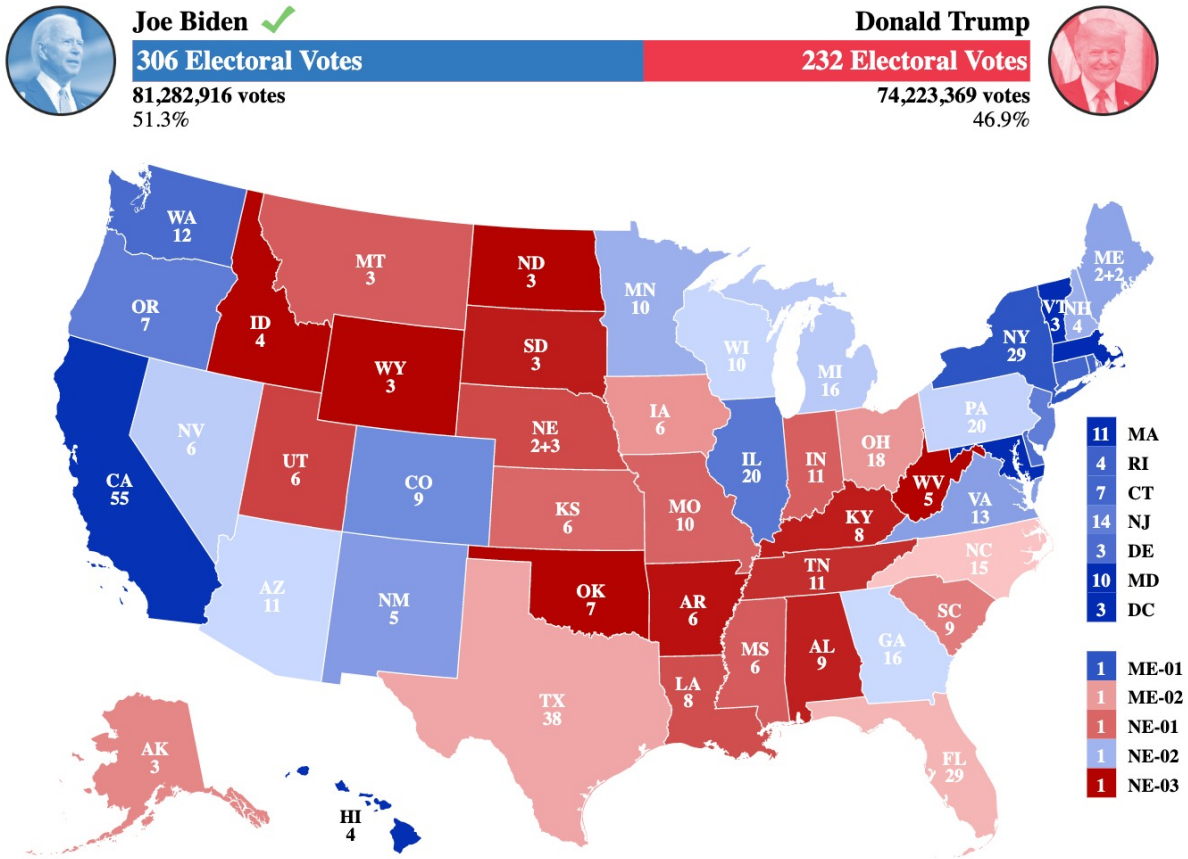
Allows for generalizations to the population!

- **No bias**: statistics (on average) equal parameter value

Why does this work?

- Soup analogy!

Example: The 2020 US Presidential Election



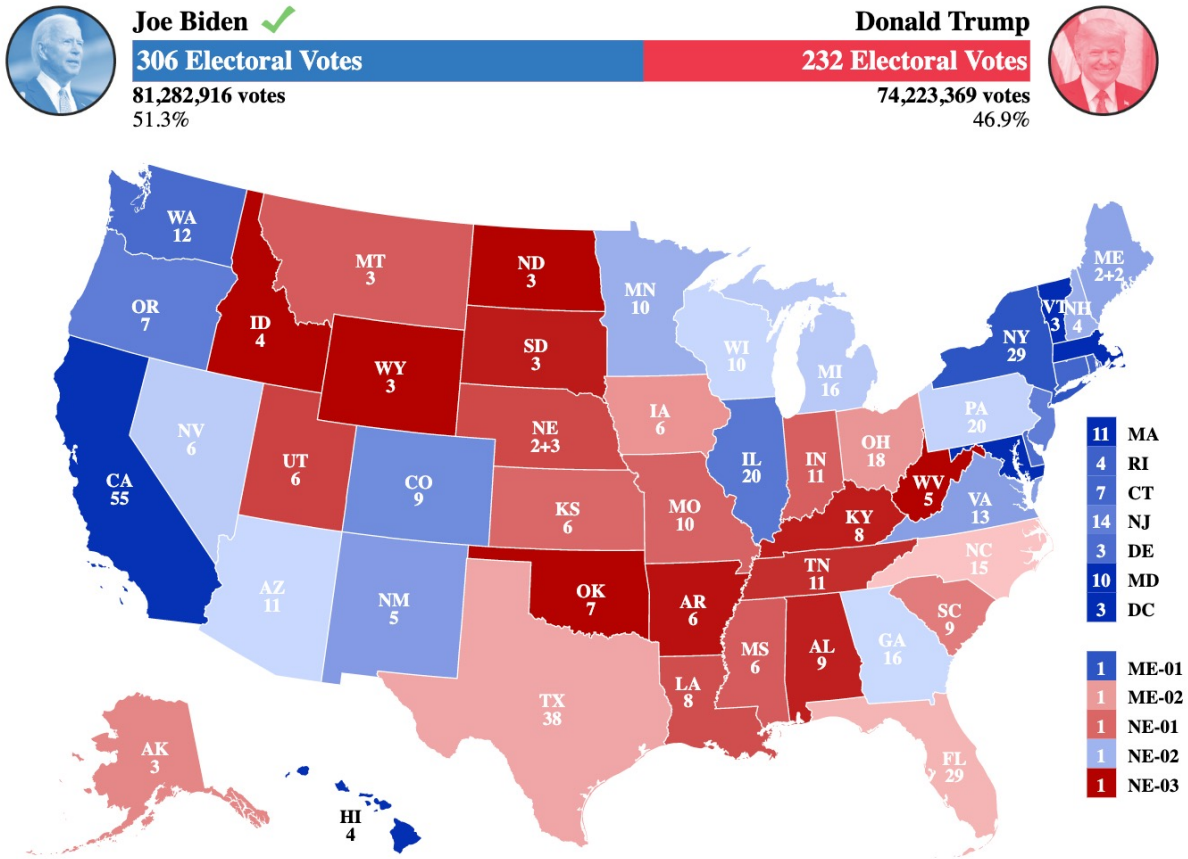
According to The Cook Political Report, the voting outcome in Georgia was

- Trump = 2,461,854
- Biden = 2,473,633

We can denote the proportion of the vote that Biden got using π_{Biden}

- Q: what is the value of π_{Biden} ?

Example: The 2020 US Presidential Election



If 1,000 voters were randomly sampled, we could denote the proportion in the sample that voted for Biden using: $\hat{p}_{Biden}$

Would we expect $\hat{p}_{Biden}$ to be equal to π_{Biden}?

If we repeated the process of sampling another 1,000 random voters, would we expect to get the same $\hat{p}_{Biden}$?

Let's explore this in Jupyter!

Sampling distributions

Probability distribution of a statistic

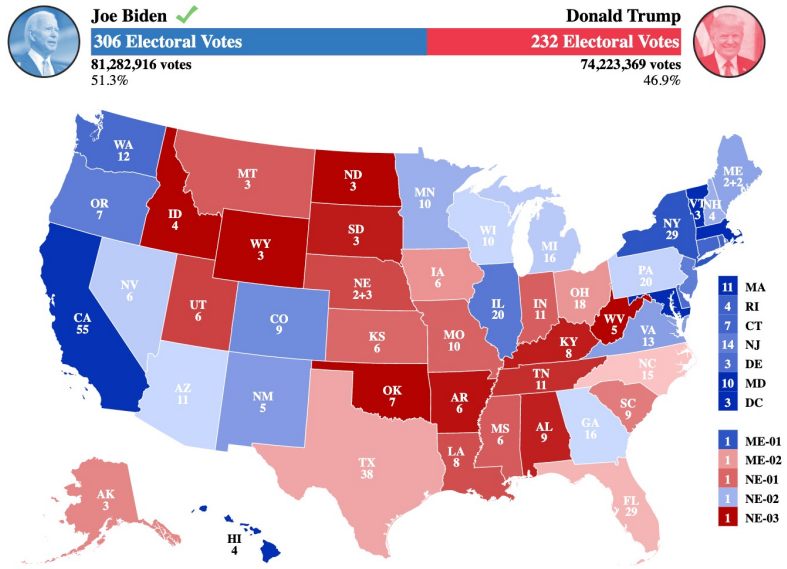
Values of a statistic vary because random samples vary

A **sampling distribution** is a probability distribution of *statistics*

- All possible values of the statistic and all the corresponding probabilities
- We can approximate a sampling distribution by a simulated statistics

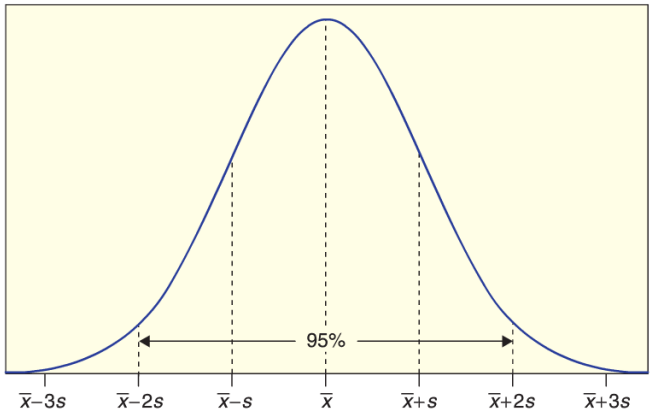
π_{Biden}

n = 1,000



$\hat{p}_{Biden}$

$\hat{p}_{Biden}$

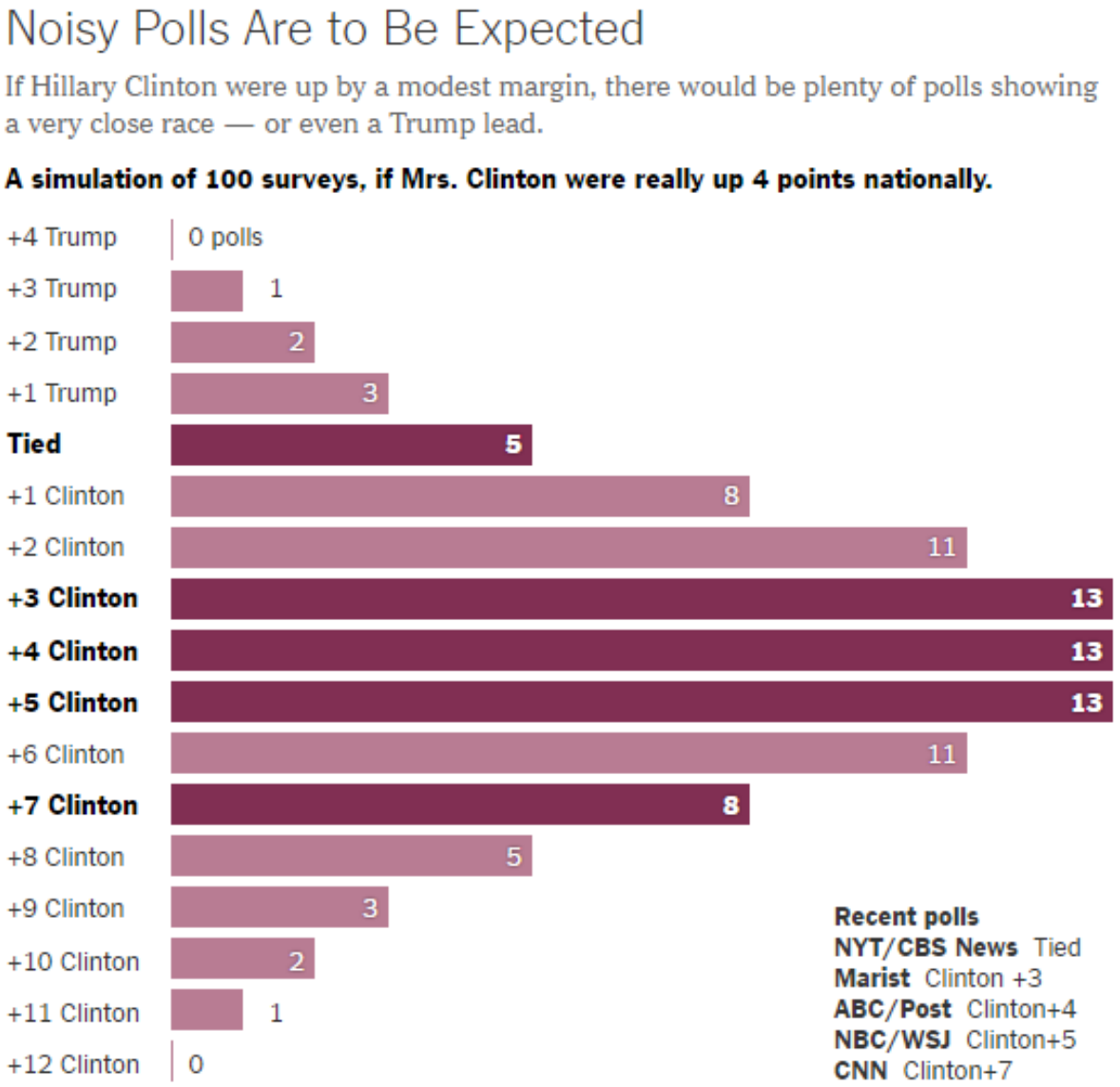


$\hat{p}_{Biden}$

Sampling distribution!

Let's explore this in Jupyter!

Confused by Contradictory Polls? Take a Step Back



What is this called?

What parameter are they trying to estimate?

NY Times 7/18/16