

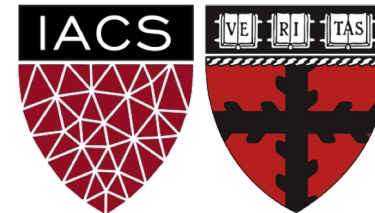
Lecture 3: Data II

How to get it, methods to
parse it, and ways to explore it.

Harvard IACS

CS109A

Pavlos Protopapas, Kevin Rader, and Chris Tanner



ANNOUNCEMENTS

- **Homework 0** isn't graded for accuracy. If your questions were surface-level / clarifying questions, you're in good shape.
- **Homework 1** is graded for accuracy
 - it'll be released **today (due in a week)**
- **Study Break** this Thurs @ 8:30pm and Fri @ 10:15am
- After lecture, please update your Zoom to the latest version

Background

- So far, we've learned:

Lecture 1	What is Data Science?
Lectures 1 & 2	The Data Science Process
Lecture 2	Data: types, formats, issues, etc.
Lecture 2	Regular Expressions (briefly)
This lecture	How to get data and parse web data + PANDAS
Future lectures	How to model data

Background

- The Data Science Process:

Ask an interesting question

Get the Data

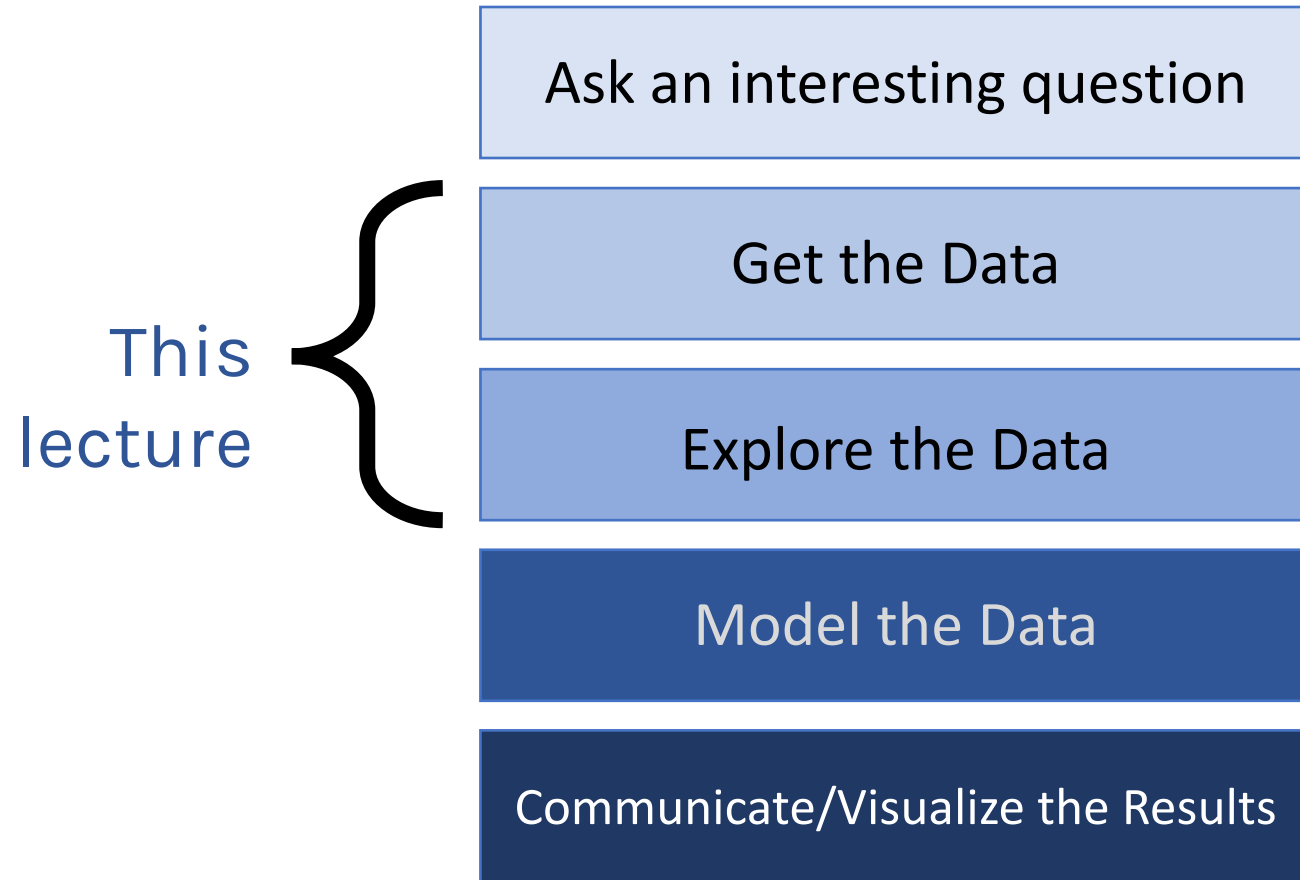
Explore the Data

Model the Data

Communicate/Visualize the Results

Background




- The Data Science Process:



Learning Objectives

- Understand different ways to obtain it
- Be able to extract any web content of interest
- Be able to do basic PANDAS commands to store and explore data
- Feel comfortable using online resources to help with these libraries (**Requests**, **BeautifulSoup**, and **PANDAS**)

Agenda

-  How to get web data?
-  How to parse basic elements using BeautifulSoup
-  Getting started with PANDAS

What are common sources for data?

(For Data Science and computation purposes.)

Data can come from:

- You curate it
- Someone else provides it, all pre-packaged for you (e.g., files)
- Someone else provides an API
- Someone else has available content, and you try to take it (web scraping)

Web scraping

- Using programs to get data from online
- Often much faster than manually copying data!
- Transfer the data into a form that is compatible with your code
- Legal and moral issues ([per Lecture 2](#))

Why scrape the web?

- Vast source of information; can combine with multiple datasets
- Companies have not provided APIs
- Automate tasks
- Keep up with sites / real-time data
- Fun!

Web scraping tips:

- Be careful and polite
- Give proper credit
- Care about media law / obey licenses / privacy
- Don't be evil (no spam, overloading sites, etc)

Robots.txt

- Specified by web site owner
- Gives instructions to web robots (e.g., your code)
- Located at the top-level directory of the web server
 - E.g., <http://google.com/robots.txt>

Web Servers

- A server maintains a long-running process (also called a daemon), which listens on a pre-specified port
- It responds to requests, which is sent using a protocol called HTTP (HTTPS is secure)
- Our browser sends these requests and downloads the content, then displays it
- 2– request was successful, 4– client error, often `page not found`; 5– server error (often that your request was incorrectly formed)

HTML

- Tags are denoted by angled brackets
- Almost all tags are in pairs e.g., `<p>Hello</p>`
- Some tags do not have a closing tag e.g., `
`

Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ttle</title>
  </head>
  <body>
    <h1>Body Title</h1>
    <p>Body Content</p>
  </body>
</html>
```

HTML

- **<html>**, indicates the start of an html page
- **<body>**, contains the items on the actual webpage
(text, links, images, etc)
- **<p>**, the paragraph tag. Can contain text and links
- **<a>**, the link tag. Contains a link url, and possibly a description of the link
- **<input>**, a form input tag. Used for text boxes, and other user input
- **<form>**, a form start tag, to indicate the start of a form
- ****, an image tag containing the link to an image

How to Web scrape:

1. **Get** the webpage content

- **Requests** (Python library) **gets** a webpage for you

2. **Parse** the webpage content

- (e.g., find all the text or all the links on a page)
- **BeautifulSoup** (Python library) helps you **parse** the webpage.
- Documentation: <http://crummy.com/software/BeautifulSoup>

The Big Picture Recap

Data Sources

Files, APIs, Webpages (via **Requests**)

Data Parsing

Regular Expressions, BeautifulSoup

Data Structures/Storage

Traditional lists/dictionaries, PANDAS

Models

Linear Regression, Logistic Regression, kNN, etc

BeautifulSoup only concerns webpage data

1. **Get** the webpage content

Requests (Python library) **gets** a webpage for you

```
page = requests.get(url)
```

```
page.status_code
```

```
page.content
```

1. **Get** the webpage content

Requests (Python library) **gets** a webpage for you

```
page = requests.get(url)
page.status_code
page.content
```

Gets the status from the webpage request.

200 means success.

404 means page not found.

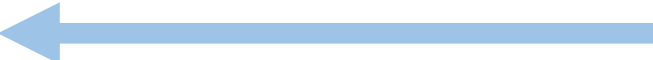
1. **Get** the webpage content

Requests (Python library) **gets** a webpage for you

```
page = requests.get(url)
```

```
page.status_code
```

```
page.content
```



Returns the content of the response, in bytes.

2. Parse the webpage content

BeautifulSoup (Python library) helps you **parse** a webpage

```
soup = BeautifulSoup(page.content, "html.parser")  
soup.title  
soup.title.text
```

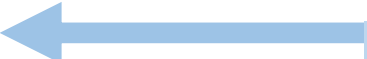
2. Parse the webpage content

BeautifulSoup (Python library) helps you **parse** a webpage

```
soup = BeautifulSoup(page.content, "html.parser")
```

```
soup.title
```

```
soup.title.text
```



Returns the full context, including the title tag. e.g.,

```
<title data-rh="true">The New York Times  
– Breaking News</title>
```

2. Parse the webpage content

BeautifulSoup (Python library) helps you **parse** a webpage

```
soup = BeautifulSoup(page.content, "html.parser")
```

```
soup.title
```

```
soup.title.text
```

Returns the text part of the title tag. e.g.,

The New York Times – Breaking News

BeautifulSoup

- Helps make messy HTML digestible
- Provides functions for quickly accessing certain sections of HTML content

Example

```
import bs4
## get bs4 object
soup = bs4.BeautifulSoup(source)
## all a tags
soup.findAll('a')
## first a
soup.find('a')
## get all links in the page
link_list = [l.get('href') for l in soup.findAll('a')]
```

HTML is a tree

- You don't have to access the HTML as a tree, though;
- Can immediately search for tags/content of interest (a la previous slide)

Example

```
tree = bs4.BeautifulSoup(source)

## get html root node
root_node = tree.html

## get head from root using contents
head = root_node.contents[0]

## get body from root
body = root_node.contents[1]

## could directly access body
tree.body
```

Exercise 1 time!

PANDAS



Kung Fu Panda is property of DreamWorks and Paramount Pictures

What / Why?

- Pandas is an *open-source* Python library (anyone can contribute)
- Allows for high-performance, easy-to-use data structures and data analysis
- Unlike NumPy library which provides multi-dimensional arrays, Pandas provides 2D table object called **DataFrame** (akin to a spreadsheet with column names and row labels).
- Used by *a lot* of people

How

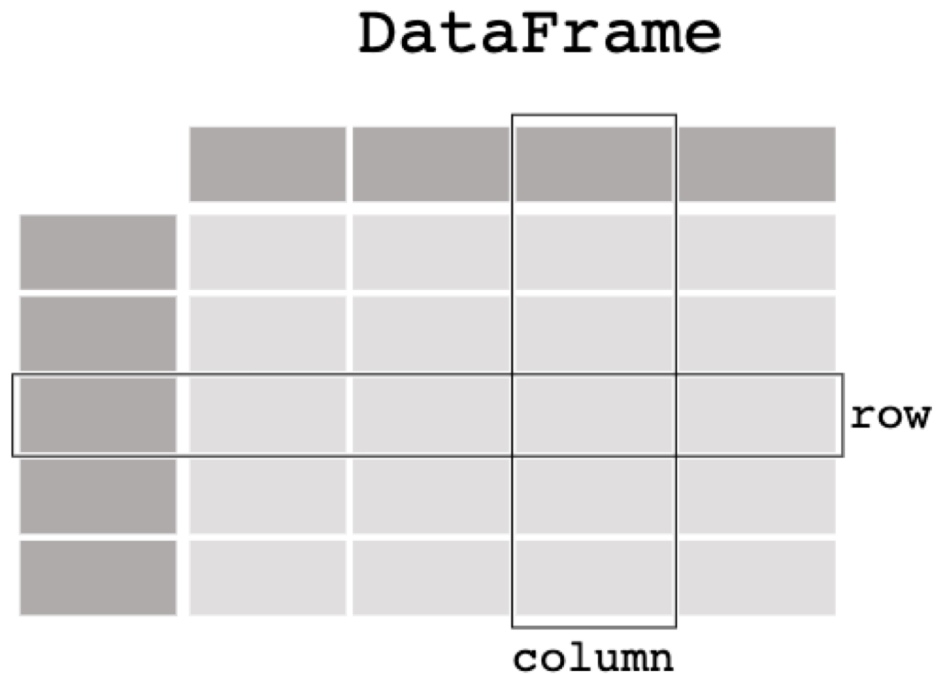
- import **pandas** library (convenient to rename it)
- Use **read_csv()** function

```
import pandas as pd  
dataframe = pd.read_csv("yourfile.csv")
```

Store and Explore Data: PANDAS

What it looks like

pandas data table representation



Visit https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/01_table_oriented.html for a more in-depth walkthrough

Example

- Say we have the following, tiny DataFrame of just 3 rows and 3 columns

```
>>> df2
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9
```

```
df2['a']
```

selects column **a**

```
>>> df2['a']
0    1
1    4
2    7
```

```
df2['a'] == 4
```

returns a Boolean list representing which rows of column **a** equal 4:
[False, True, False]

```
df2['a'].min()
```

returns **1** because that's the minimum value in the **a** column

```
df2[['a', 'c']]
```

selects columns **a** and **c**

```
>>> df2[['a', 'c']]
   a  c
0  1  3
1  4  6
2  7  9
```


Example continued

```
>>> df2
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9
```

`df2['a'].unique()` returns all distinct values of the `a` column once

`df2.loc[2]`

returns a Series
representing the
row w/ the label `2`

```
>>> df2.loc[2]
a    7
b    8
c    9
```

`df2.loc[df2['a'] == 4]`

`.loc` returns all rows that were passed-in

`[False, True, False]`

```
>>> df2.loc[df2['a'] == 4]
   a  b  c
1  4  5  6
```

Example continued

```
>>> df2
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9
```

`df2.iloc[2]` returns a Series representing the row at index 2 (NOT the row labelled 2. Though, they are often the same, as seen here)

`df2.sort_values(by=['c'])` returns the DataFrame with rows shuffled such that now they are in ascending order according to column `c`. In this example, `df2` would remain the same, as the values were already sorted

Common PANDAS functions

- **High-level viewing:**
 - `head()` – first N observations
 - `tail()` – last N observations
 - `describe()` – statistics of the quantitative data
 - `dtypes` – the data types of the columns
 - `columns` – names of the columns
 - `shape` – the # of (rows, columns)

Common PANDAS functions

- **Accessing/processing:**
 - `df["column_name"]`
 - `df.column_name`
 - `.max()`, `.min()`, `.idxmax()`, `.idxmin()`
 - `<dataframe> <conditional statement>`
 - `.loc[]` – label-based accessing
 - `.iloc[]` – index-based accessing
 - `.sort_values()`
 - `.isnull()`, `.notnull()`

Common Panda functions

- **Grouping/Splitting/Aggregating:**
 - `groupby()`, `.get_groups()`
 - `.merge()`
 - `.concat()`
 - `.aggegate()`
 - `.append()`

Why?

- EDA encompasses the “*explore data*” part of the data science process
- EDA is crucial but often overlooked:
 - If your data is bad, your results will be bad
 - Conversely, understanding your data well can help you create smart, appropriate models

What?

1. Store data in data structure(s) that will be convenient for exploring/processing
(Memory is fast. Storage is slow)
2. Clean/format the data so that:
 - Each row represents a single object/observation/entry
 - Each column represents an attribute/property/feature of that entry
 - Values are numeric whenever possible
 - Columns contain atomic properties that cannot be further decomposed*

* Unlike food waste, which can be composted.
Please consider composting food scraps.

What? (continued)

3. Explore **global** properties: use histograms, scatter plots, and aggregation functions to summarize the data
4. Explore **group** properties: group like-items together to compare subsets of the data (are the comparison results reasonable/expected?)

This process transforms your data into a format which is easier to work with, gives you a basic overview of the data's properties, and likely generates several questions for you to follow-up in subsequent analysis.

We will address EDA
more and dive into
Advanced PANDAS
operations

Exercise 2 time!