Python,
Web Scraping APIs and HTML

MAS Departmental Disclaimers:

For students trying to take or audit from outside the MAS program.

Taking or auditing 400 courses is simply is not permitted because this is a self-supporting program. Sorry, unfortunately, you will NOT be able to take any of the 400 level Stats courses.

There are NO exceptions that can be made by the department. These classes were designed specifically for students who applied directly to the program.

The students of this program are also not allowed to audit or enroll in classes outside of the program as it was created for working professionals.

If you would like to apply for the program, you are welcome to do so: https://master.stat.ucla.edu/admissions/

Information is found here: https://master.stat.ucla.edu/ And here: https://master.stat.ucla.edu/faq/

Today, we are going to get a Python environment running in Docker

We are going to do this in a slightly different fashion then we have in previous weeks. This iteration will use a **`Dockerfile`** and .yml file to create the environment while mounting your local directory

First, sync your class repo to get the most recent week-4 files and directories

Now change your directory to `week-4/jupyter_build/`, in either terminal (mac) or powershell (windows). Run

```
docker-compose up -d
```

This will take a few minutes to download the image and install the python libraries necessary. We will let it run and continue our class discussion.

So how is this different from what we've run before?

Docker compose necessitates a `Dockerfile` and a `.yml` file to define our environment.

```
docker-compose up -d
```

We're using docker compose instead of the following commands:

```
docker build -t dockerjupyter <path_to_context>

docker run -d --rm -p 8888:8888: -v ..:/home/dev/ \
   --name jupyter dockerjupyter
```

The `Dockerfile` defines the environment and app

The file builds upon `Ubuntu`, installs some things with apt and pip, and last defines startup behavior to run `jupyter notebooks`

The `docker-compose.yml` configures how we want to build and run the container — it can be thought of as a `docker run` command in a `.yml` file

Now we can see how we can define the same environment for ourselves and anyone else who uses our code from this repository

The `Dockerfile` defines the environment and app

Set the python version

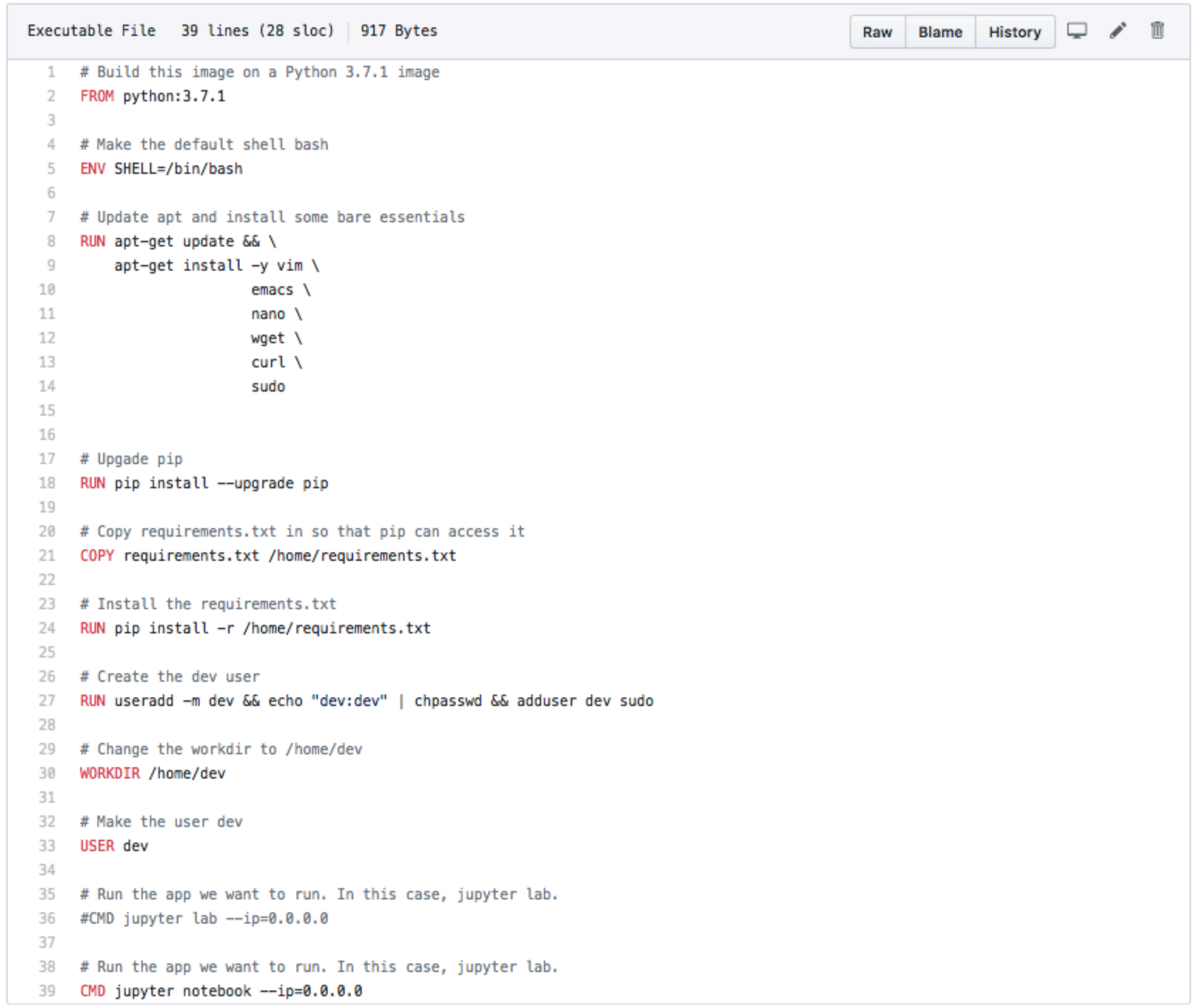
Make default shell bash

Install some things with apt and sudo

Install some more with requirements.txt

Create user and set directory

Run the jupyter notebook app



```
# Build this image on a Python 3.7.1 image
FROM python:3.7.1

# Make the default shell bash
ENV SHELL=/bin/bash

# Update apt and install some bare essentials
RUN apt-get update && \
    apt-get install -y vim \
                    emacs \
                    nano \
                    wget \
                    curl \
                    sudo


# Upgade pip
RUN pip install --upgrade pip

# Copy requirements.txt in so that pip can access it
COPY requirements.txt /home/requirements.txt

# Install the requirements.txt
RUN pip install -r /home/requirements.txt

# Create the dev user
RUN useradd -m dev && echo "dev:dev" | chpasswd && adduser dev sudo

# Change the workdir to /home/dev
WORKDIR /home/dev

# Make the user dev
USER dev

# Run the app we want to run. In this case, jupyter lab.
#CMD jupyter lab --ip=0.0.0.0

# Run the app we want to run. In this case, jupyter lab.
CMD jupyter notebook --ip=0.0.0.0
```

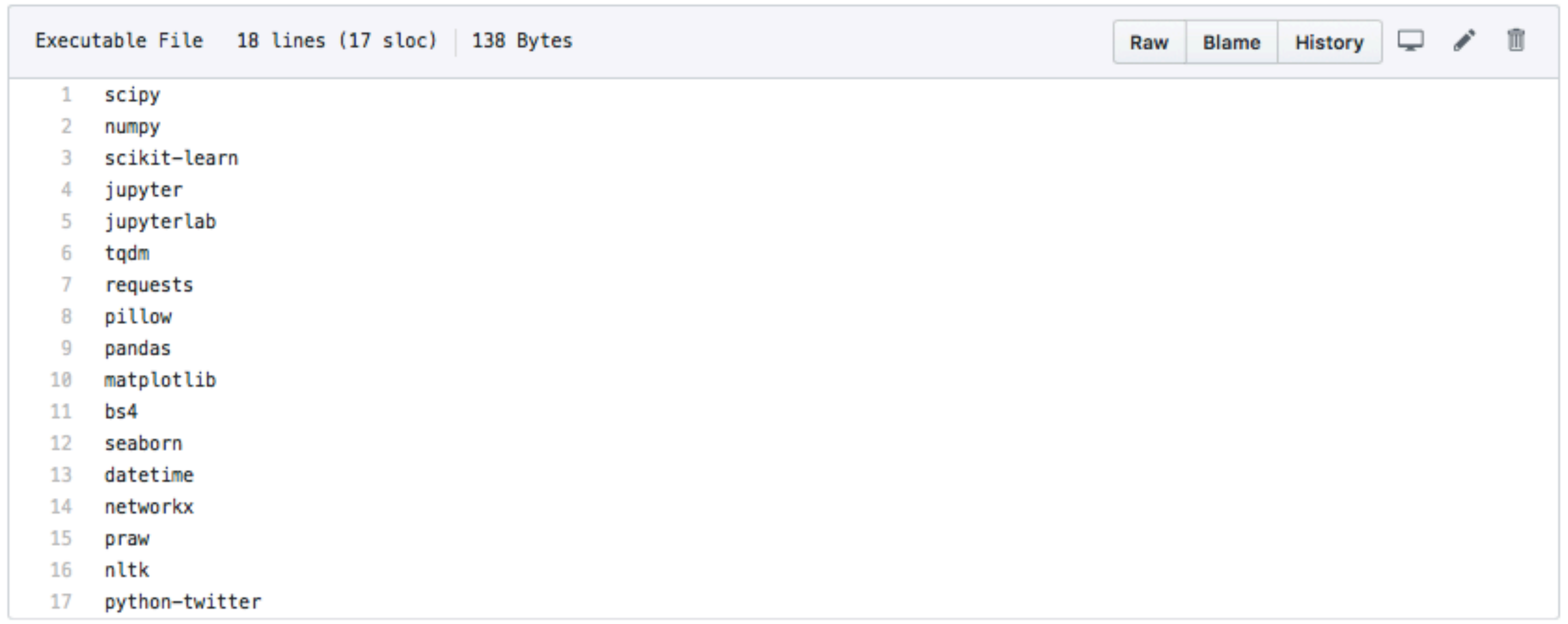
The `requirements.txt` is a list of python libraries to install with `pip`

Can add as many as you'd like here,

but note that you will need to rebuild

the image to ensure this changes are

incorporated



```
scipy
numpy
scikit-learn
jupyter
jupyterlab
tqdm
requests
pillow
pandas
matplotlib
bs4
seaborn
datetime
networkx
praw
nltk
python-twitter
```

The `docker-compose.yml` configures how we want to build and run the container

Compose file format version indicates

sets the version of Docker engine and

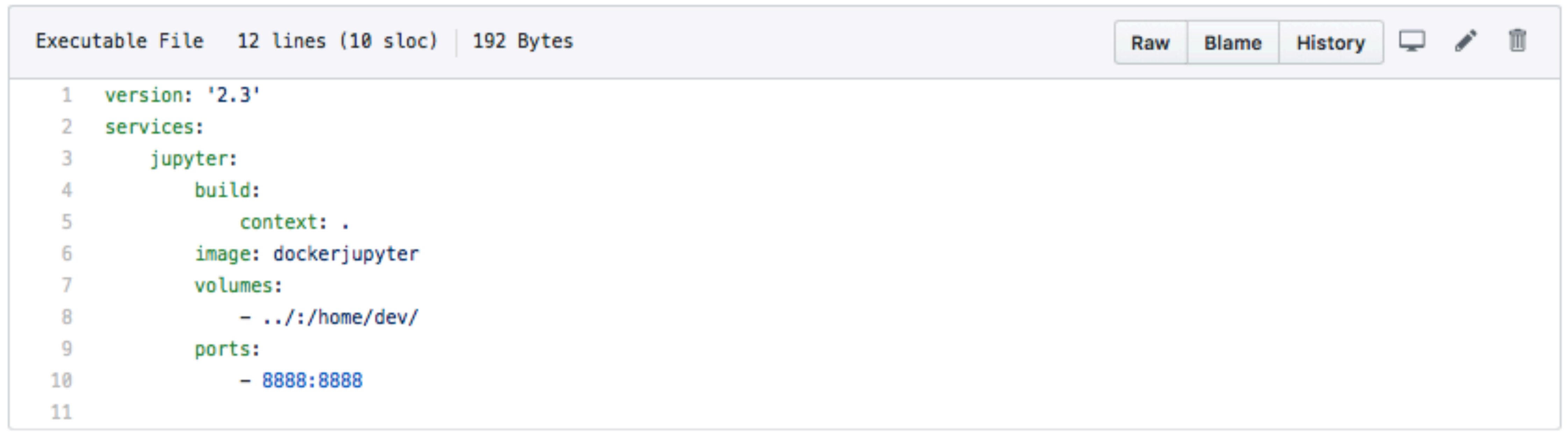
defines the compatibility of images

Gets the image were interested

in running (dockerjupyter)

Mounts the local volume as the current directory

Allocates the ports for the running container



```
version: '2.3'
services:
    jupyter:
        build:
            context: .
        image: dockerjupyter
        volumes:
            - ../:/home/dev/
        ports:
            - 8888:8888

```

Changing the docker environment is as simple as changing one of the files we’ve just seen.

Add Python libraries by adding to the `requirements.txt`

  Add versions by adding `<package> == <version_number>`

Add new `apt` functionality by adding packages in `Dockerfile`

Change the startup behavior through `CMD` line in `Dockerfile`

Changes files showing up in container and alter the volumes in `docker-compose.yml`

For any changes to be seen you must run

```
docker-compose up —build -d
```

Now change your directory to **`week-4/jupyter_build/`**, in either terminal (mac) or powershell (windows). Run

```
docker-compose up -d
```

This will take a few minutes to download the image and install the python libraries necessary. We will let it run and continue our class discussion….(10 minutes later)….once complete, run:

```
docker exec jupyter_build_jupyter_1 jupyter notebook list
```

Copy/Paste the link into your browser to see Jupyter (might need to use your IP instead of 0.0.0.0 for windows)

When you are done run **`docker-compose down -v`** to close

Jupyter notebooks are interactive programming environments that can be used for writing, executing, and documenting code.

Jupyter Notebooks are composed of "cells" which you can create, move, and delete. These cells can be different formats like "Markdown", "code", or "raw" (useful for preserving code but avoiding accidentally running it).

You can double click on Markdown cells to edit them.

Code cells run with the "Run" button in the toolbar or pressing Shift+Enter.

Jupyter Notebooks are nice because they can keep your code, documentation, and results all in one file. (We will also look at Jupyter Lab at some point which is more like RStudio in some sense with an interactive console)

Using Chrome’s Inspect tool

The official list of nominees and winners of the 90th Academy Awards (in 2018) is available on the Academy of Motion Picture Arts and Sciences' website. By ocular inspection, this data is at least semi-structured: there are groups of nominees, each group has a title, each nominee has the name of the movie. But this is not structured in the way that we typically analyze data, a table with rows (observations) and columns (features).

This exercise assumes you're using Chrome, if you're using Safari (instructions) or Mozilla (instructions), there are slightly different workflows. If you right click on the "Actor in a Leading Role" and select "Inspect", a new window should appear on the side of your browser window. Make sure you have the "Elements" tab clicked.

Using Chrome’s Inspect tool

You should see a nested list of collapsable elements. This is a very complex webpage, like many are these days, with many layers of tags. However if you mouse over the elements, things highlight in the primary browser window. You can see how this is really useful for matching specific HTML codes with what they produce. You can expand an element to see its sub-elements, sub-sub-elements, and so on.

Using Chrome’s Inspect tool

You should discover that the `<div class="view-grouping">` elements correspond to the nominee categories. The `"view-grouping"` is an attribute for the `div` tag. Expanding this element reveals a `<div class="view-grouping-content">` element, which then has seven "children": an `<h3>` (heading 3) tag for the name of the category, an obtuse `<div>` class for the winner, another `<h3>` tag for the nominees, and then four (or more) of the same obtuse `<div>` classes for the other nominees.

Expanding any one of the obtuse `<div>` elements reveals still more child tags that ultimately end with the name of the actor `<h4 class="field-content">Gary Oldman</h4>` and movie `<span class="field-content">Darkest Hour</span>`.

Forms of Structured Data

There are three primary forms of structured data you will encounter on the web: HTML, XML, and JSON.

HTML

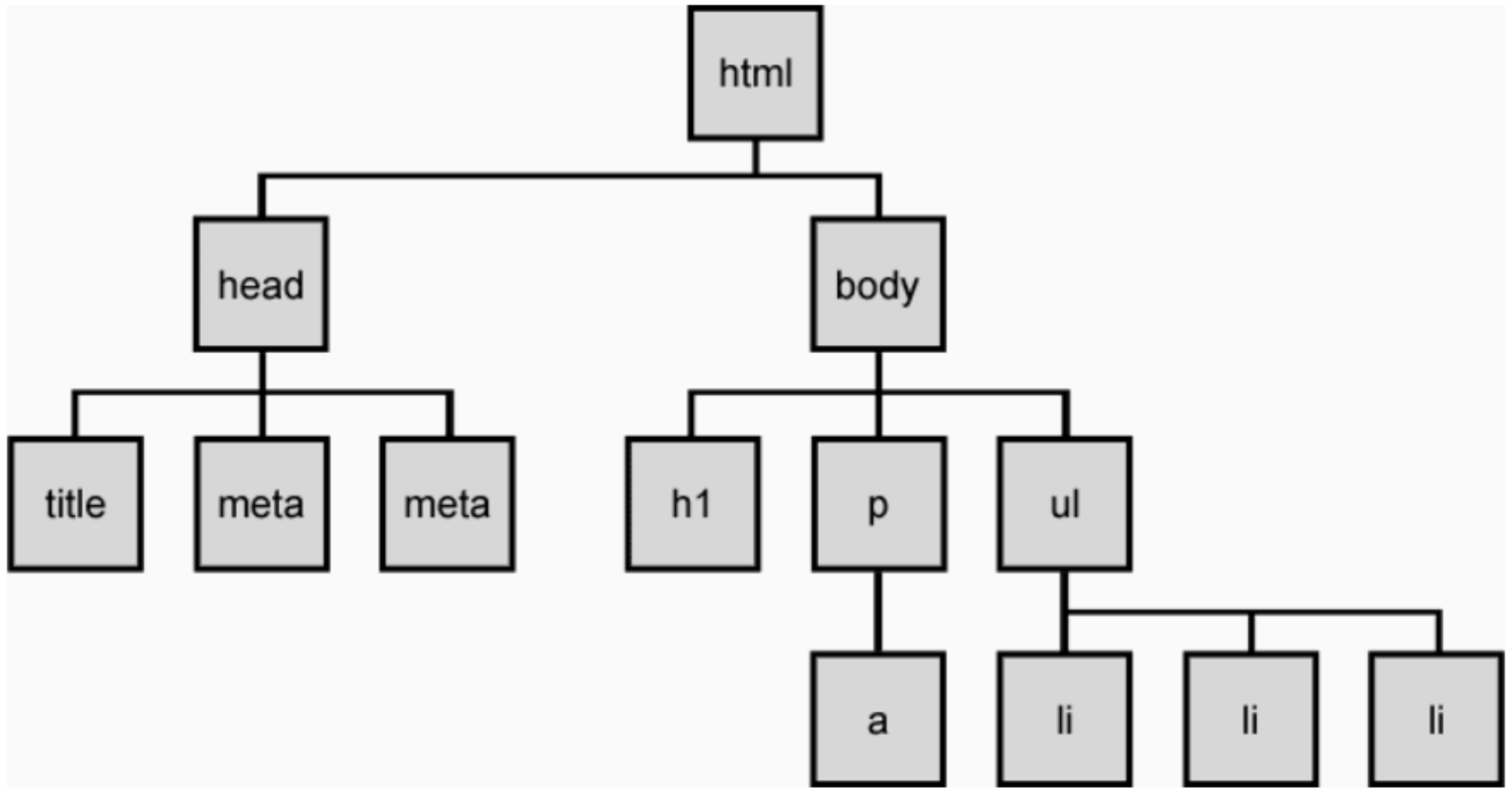
The Oscars example is an example of HyperText Markup Language (HTML), a markup standard that allows web browsers to render documents transmitted by a web server into webpages.

HTML is distinct from—but depends strongly on—the HyperText Transfer Protocol (HTTP) to move data from servers to clients.

HTML works with other standards like Cascading Style Sheets (CSS) and JavaScript to create the dynamic and interactive websites.

HTML

HTML has a tree-like structure with the "root" node being an `<html>` element with other elements like `<body>`s and `<head>`s as children, and those children having other children live the `<div>`, `<a>`, and other elements. Simple example of an HTML tree.



From Chris Bail's "Screen-Scraping" https://cbail.github.io/SICSS_Screenscraping_in_R.html

XML

The Extensible Markup Language (XML) is a related markup language for representing data structures.

XML is a robust and open—though verbose—standard for representing structured data.

The design goals of XML emphasize simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures such as those used in web services. (https://en.wikipedia.org/wiki/XML)

JSON

JavaScript Object Notation (JSON) is probably the most popular data markup language and is especially ubiquitous when retreiving data from the application programming interfaces (APIs) of popular platforms like The New York Times, Reddit, Wikipedia, *etc*.

HTML in the wild can be incredibly complicated and XML is verbose and has other ideosyncracies (which is probably why we won't see it after this class).

JSON is attractive for programmers using Python and R because it can represent a mix of different data types.

Python Data Types (a digression)

We need to make a brief digression into Python's fundamental data structures in order to understand the contemporary attraction to JSON.

Python has a few fundamental data types for representing collections of information:

Lists: This is a basic ordered data structure that can contain strings, ints, and floats

Dictionaries: This is an unordered data structure containing key-value pairs, like a phonebook.

Python Lists - Basic Example

```
# Make classrooms as lists with student names as strings
classroom0 = ['Alice','Bob','Carol','Dave']
classroom1 = ['Eve','Frank','Grace','Harold']
classroom2 = ['Isabel','Jack','Katy','Lloyd']
classroom3 = [‘Maria','Nate','Olivia','Philip']

# Make schools that contain classrooms
school0 = [classroom0,classroom1]
school1 = [classroom2,classroom3]

# Make a school district that contains schools
school_district = [school0,school1,school2]

# Inspect one school's enrollments
school_district

Output:
[[['Alice', 'Bob', 'Carol', 'Dave'], ['Eve', 'Frank', 'Grace', 'Harold']],
 [['Isabel', 'Jack', 'Katy', 'Lloyd'], ['Maria', 'Nate', 'Olivia', ‘Philip’]]]


#note that lists can contain other lists and list of lists…
```

Python Lists - Basic Example

```
# You can access list elements by their position. In this case, we could access the first name in
classroom0 by requesting the item at the 0th index
classroom0[0]

Output:
'Alice'

#note that python is 0 indexed in comparison to R’s 1; this may take awhile for some of you to
remember and get comfortable with
```

Python Dictionaries - Basic Example

In this example `mountain_west` is a dictionary of dictionaries where the outer dictionary is keyed by the name of the state and the inner dictionary has different statistical information.

```
# Make a dictionary of states containing dictionaries
mountain_west = {'Colorado': {'Abbreviation': 'CO',
                              'Area': 269601,
                              'Capital': 'Denver',
                              'Established': '1876-08-01',
                              'Largest city': 'Denver',
                              'Population': 5540545,
                              'Representatives': 7},
                 'Idaho': {'Abbreviation': 'ID',
                              'Area': 216443,
                              'Capital': 'Boise',
                              'Established': '1890-07-03',
                              'Largest city': 'Boise',
                              'Population': 1683140,
                              'Representatives': 2}}

#Note that dictionaries can contain other dictionaries.
```

Python Dictionaries - Basic Example

```
# what are the key values
mountain_west.keys()

Output:
dict_keys(['Colorado', ‘Idaho’])

#dictionary values
mountain_west.values()

Output:
dict_values([{'Abbreviation': 'CO', 'Area': 269601, 'Capital': 'Denver', 'Established': '1876-08-01',
'Largest city': 'Denver', 'Population': 5540545, 'Representatives': 7}, {'Abbreviation': 'ID', 'Area':
216443, 'Capital': 'Boise', 'Established': '1890-07-03', 'Largest city': 'Boise', 'Population':
1683140, 'Representatives': 2}])

#You access dictionary values by their keys. Statistics about Colorado
mountain_west[‘Colorado']

Output:{'Abbreviation': ‘CO',
'Area': 269601,
'Capital': ‘Denver',
'Established': ‘1876-08-01',
'Largest city': ‘Denver',
'Population': 5540545,
‘Representatives': 7}
```

Python List and Dictionaries - Basic Example

```
# Nested data structures do not need to be the same type
school_district_dict = {'School 1':{'Classroom A':['Alice','Bob','Carol','Dave'],
                     'Classroom B':['Eve','Frank','Grace','Harold']
                                  },
                         'School 2':{'Classroom A':['Isabel','Jack','Katy','Lloyd'],
                                     'Classroom B':['Maria','Nate','Olivia','Philip']
                                  },
                         'School 3':{'Classroom A':['Quinn','Rachel','Steve','Terry','Ursula'],
                                     'Classroom B':['Violet','Walter','Xavier','Yves','Zoe']
                                  }
                         }

# access the 3rd member of "Classroom B" in "School 2"
school_district_dict['School 2']['Classroom B’][2]

Output:
‘Olivia’
```

A list can take just about any other data structure as a value. Dictionaries are great because they label your data for you but don't preserve order and need unique keys, so they're bad at storing repeated data like a time series.

Forms of Structured Data

We’ve now seen the 3 types of structured data that we will encounter while web scraping.

Let’s take a look at some different scraping Python Jupyter notebooks

First, we’ll start by looking at Wikipedia and Reddit. We’ll finish looking at some movie data and the oscars data again.

Ethics of Web Scraping

The phrase "data scraping" is colloquial and popular but has pejorative connotations.

Data is valuable: other people invested time in collecting, organizing, and sharing it.

When you show up with a scraper you built after maybe a dozen hours demanding data, you rarely pay the costs of labor, hosting, *etc*. that went into making the data available.

There are *very* good rationales for making many kinds of data more available: reproducibility of scientific results, sharing publicly-funded and/or close-to-zero marginal cost resources, transparency and accountability in democratic institutions, remixing for innovative new analyses, *etc*.

Ethics of Web Scraping

But data breaches have become eponymous (Target in 2013, Equifax in 2017, Facebook in 2018, *etc*.) because they violate other values like privacy.

In the context of data scraping, there are really four ethical "areas of difficulty”:

- **Informed consent**: does the data scraper obtain consent from every person whose data is being retrieved?
- **Informational risk**: can the data scraper inflict economic, social, *etc*. harm on individuals by disclosing data?
- **Privacy**: does the data scraper know which information a person intended to be private or public?
- **Decision-making under uncertainty**: does the data scraper know all the ways the data could be (mis)used?

Ethical and Legal Risks

- **Copyright infringement**: compiling data that someone else can claim ownership over
- **Trespass**: over-aggressive scraping shuts down someone else's property
- **Computer Fraud & Abuse Act**: misrepresenting yourself to access a system is “hacking”

I cannot provide legal advice, we will revisit these concerns throughout the course through best practices for avoiding infringement, staggering data collection, simulating human requests, securing data, and protecting privacy

Ethical Web Scraping

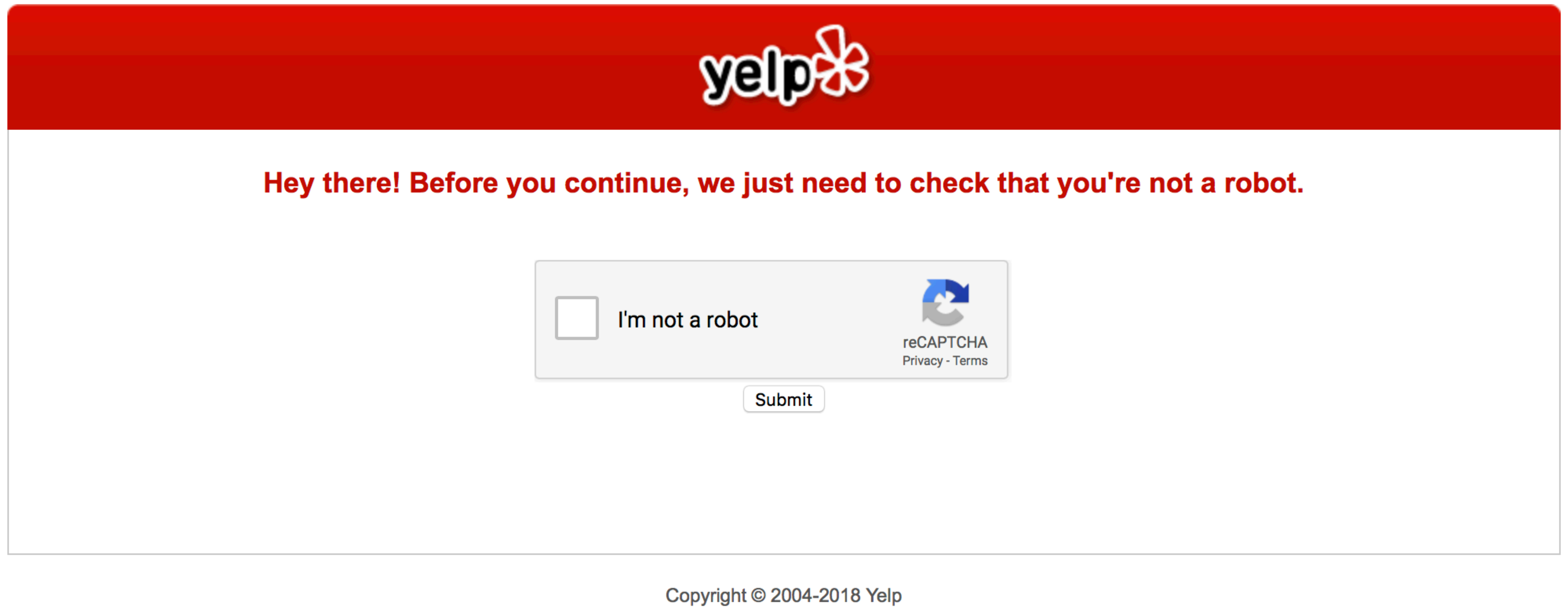
James Densmore has a nice summary of practices for ethical web scraping:

- If you have a public API that provides the data I'm looking for, I'll use it and avoid scraping all together.
- I will always provide a User Agent string that makes my intentions clear and provides a way for you to contact me with questions or concerns.
- I will request data at a reasonable rate. I will strive to never be confused for a DDoS attack.
- I will only save the data I absolutely need from your page. If all I need it OpenGraph meta-data, that's all I'll keep.
- I will respect any content I do keep. I'll never pass it off as my own.
- I will look for ways to return value to you. Maybe I can drive some (real) traffic to your site or credit you in an article or post.
- I will respond in a timely fashion to your outreach and work with you towards a resolution.
- I will scrape for the purpose of creating new value from the data, not to duplicate it.

Ethical Web Scraping

Other things you should do:

- Reading the Terms of Service and Privacy Policies for the site's rules on scraping.
- Inspecting the robots.txt file for rules about what pages can be scraped, indexed, *etc*
- Be gentle on smaller websites by running during off-peak hours and spacing out requests
- Identify yourself by name and email in your User-Agent strings


Hey there! Before you continue, we just need to check that you're not a robot.

Copyright © 2004-2018 Yelp

Robot.txt files?

Provides a sitemap to the robot to get other pages, it allows all kinds of User-agents, and disallows crawling of pages in specific directories (ads, polls, tests).

This is CNNs

```
print(requests.get('https://www.cnn.com/robots.txt').text)
```

When scraping websites its worthwhile (and recommended to include a user-agent) so that the webmaster can get in contact

```
contact_header = {'User-Agent':'Python research tool by Nate Langholz, langholz@g.ucla.edu’}
```

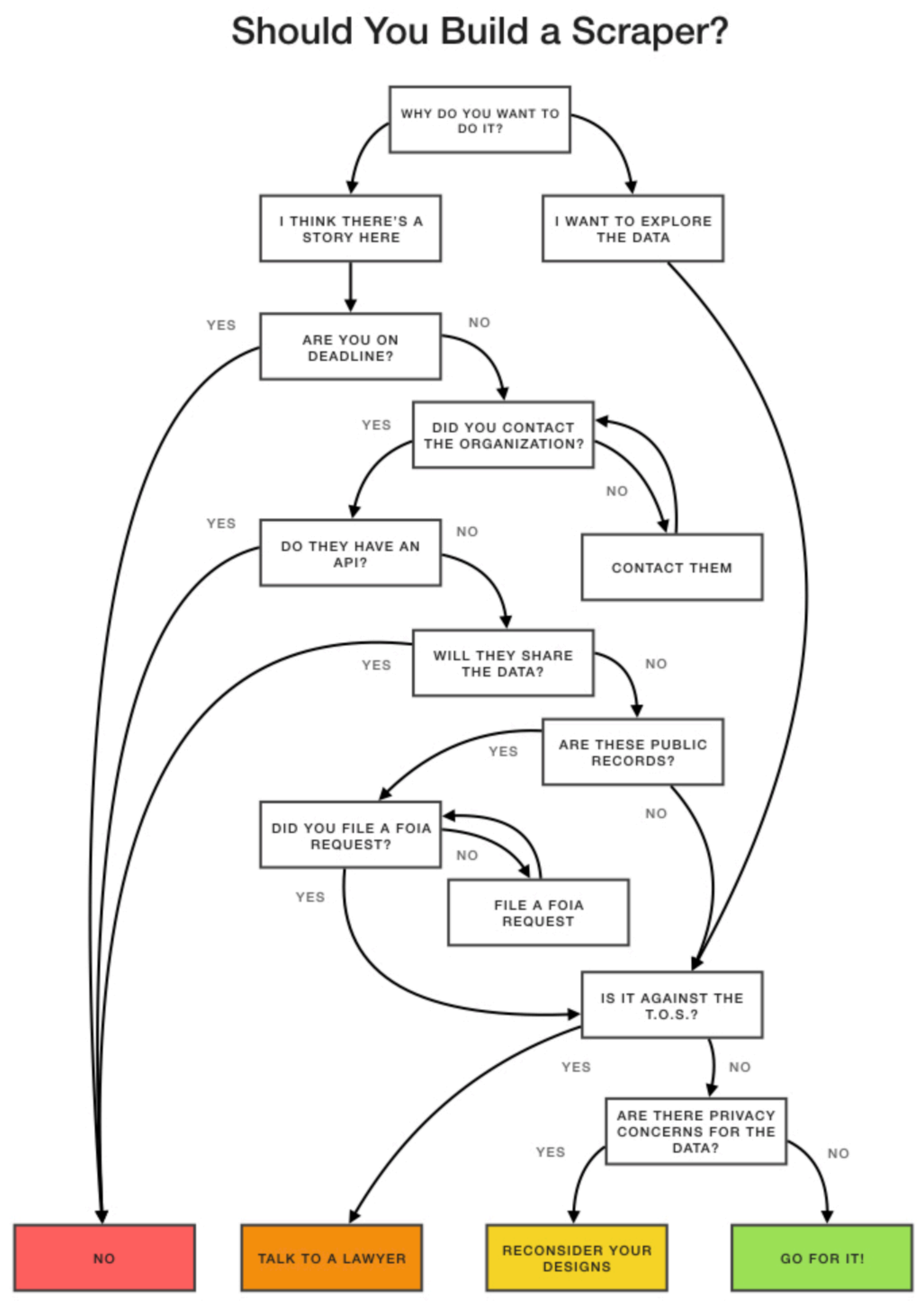
```
request = requests.get('https://www.cnn.com',headers=contact_header)
```

Should you create a scaper?

Sophie Chou made this nice decision flow-chart of whether to build a scraper or not from a NICAR panel in 2016

There are some ethical justifications for violating a site's terms of service to scrape data:

- Obtaining data for the public interest from official statements, government reports, etc.
- Conducting audit studies (as long as these are responsibly designed and pre-cleared)
- The data is unavailable from APIs, FOIA requests, and other reports


Should You Build a Scraper?
WHY DO YOU WANT TO DO IT?
I THINK THERE'S A STORY HERE
I WANT TO EXPLORE THE DATA
ARE YOU ON DEADLINE?
YES
NO
DID YOU CONTACT THE ORGANIZATION?
YES
NO
CONTACT THEM
DO THEY HAVE AN API?
YES
NO
WILL THEY SHARE THE DATA?
YES
NO
ARE THESE PUBLIC RECORDS?
YES
NO
DID YOU FILE A FOIA REQUEST?
NO
YES
FILE A FOIA REQUEST
IS IT AGAINST THE T.O.S.?
YES
NO
ARE THERE PRIVACY CONCERNS FOR THE DATA?
YES
NO
NO
TALK TO A LAWYER
RECONSIDER YOUR DESIGNS
GO FOR IT!

Adverse consequences of web scraping include:

- Compromising the privacy and integrity of individual users' data
- Damaging a web server with too many requests
- Denying access to the web service to other authorized users
- Infringing on copyrighted material
- Damaging the business value of a web site

Computer Fraud and Abuse Act

The Computer Fraud and Abuse Act was passed in 1984, in large part due to the 1983 film WarGames starring Matthew Broderick. A plain reading of the text of the law (18 U.S.C. § 1030) criminalizes just about any form of web scraping:

- Whoever intentionally accesses a computer without authorization or exceeds authorized access, and thereby obtains… information from any protected computer;
- knowingly causes the transmission of a program, information, code, or command, and as a result of such conduct, intentionally causes damage without authorization, to a protected computer;
- the term “exceeds authorized access” means to access a computer with authorization and to use such access to obtain or alter information in the computer that the accesser is not entitled so to obtain or alter;
- the term “damage” means any impairment to the integrity or availability of data, a program, a system, or information;
- the term “protected computer” means a computer which is used in or affecting interstate or foreign commerce or communication, including a computer located outside the United States that is used in a manner that affects interstate or foreign commerce or communication of the United States

Violators can be fined and jailed under a misdemeanor charge for up to 1 year for the first violation and jailed up to 10 years under a felony charge for repeated violations.

Computer Fraud and Abuse Act

This law has a chilling effect on many forms of research, journalism, and other forms of protected speech. The CFAA has been used by federal prosecutors to bring federal felony charges against programmers, journalists, and activists. In 2011, programmer and hacktivist Aaron Swartz (who contributed to the development of RSS, Markdown, Creative Commons, and Reddit) was arrested and charged with violating the CFAA for downloading several million PDFs from JSTOR over MIT's network. The decision to prosecute was unusual. Facing 35 years of imprisonment and over $1 million in fines under the CFAA, Swartz committed suicide on January 11, 2013.

Computer Fraud and Abuse Act

In 2016, four computer science researchers and the publisher of *The Intercept* who all use scraping techniques to run experiments to measure bias and discrimination in web content filed suit with the ACLU against the U.S. Government: *Sandvig v. Sessions*. Their research involves creating multiple fake accounts, providing inaccurate information to websites, using automated tools to record publicly-available data, and other scraping techniques. In March 2018, the D.C. Circuit Court ruled two of the plantiffs have standing to sue and the case is currently being prepared for trial.