DATA SCIENCE SMORGASBORD

Unix Shell - Python - CAREER

Ryan R. Rosario

Statistics 404

February 28, 2017

```
ok> echo Hello, World!
Hello, World!
```

whoami

Ryan R. Rosario

Your TA for Stats 404.

- Education
  - Ph.D. Candidate in Statistics
    - Machine Learning & Natural Language Processing
    - Applying the Bootstrap to Small Texts
    - My Last Hurrah: **March 14 (Ph.D. Defense)**
  - M.S. Computer Science
  - M.S. Statistics
  - B.S. Mathematics & Statistics
- Resides in Mammoth Lakes, CA
- From Thousand Oaks, CA

`whoami`: EXPERIENCE

Since this is a Professional Masters audience, I'll highlight some professional experiences.

Facebook
Quantitative Engineer
"Machine Learning Engineer"
More about this in a bit…

Springboard
Data Science Intensive and Career Track Mentor
Oversee student projects and learning.
My alumni now work at Amazon, WeWork, AirBnb, Pandora etc.

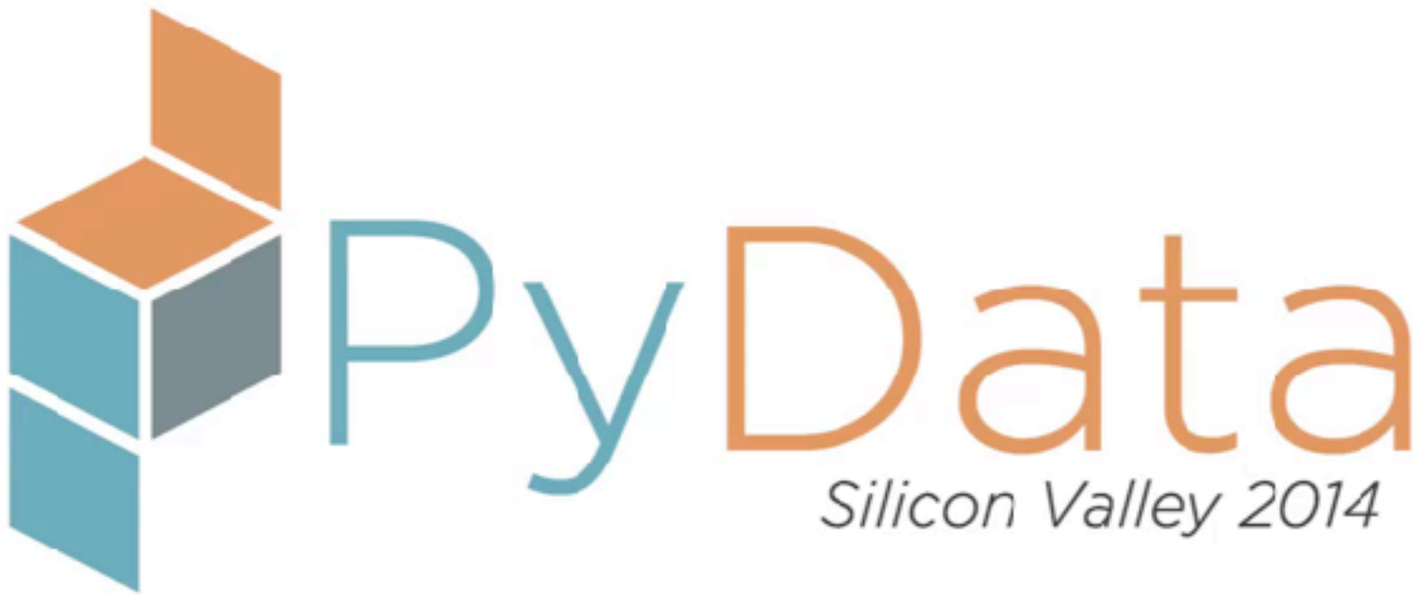
WHAT I DID AT FACEBOOK

My research interest in Natural Language Processing and Machine Learning.

- Developed the company-wide sentiment analysis system.
- Developed a system for topic identification and topic summarization.
- Worked on automated tagging of IT tickets.
- Worked with HR's People Analytics team using classical statistics to analyze inequities in performance review text and leveling.

WHAT I DID AT FACEBOOK

You can learn more about what I did in this PyData 2014 Silicon Valley talk:



https://www.youtube.com/watch?v=y3ZTKFZ-1QQ

HOW DATA SCIENCE WORKED AT FACEBOOK

The biggest Data Science and Statistics related teams when I was there:

Core Data Science Academic, most have Ph.D.s. Research is big focus. Algorithm Development. Created their own algorithmic products.	**AI Research (FAIR)** Led by Yann LeCun. Academic and research oriented, but heavy on engineering. Deep learning, robots, AR, computer vision.
Analytics The biggest Data Science org. Lots of SQL, data tools, create metrics, consulting with product teams. Much less engineering.	**Quantitative Engineering** Machine learning and NLP solutions to business problems. Heavier on software engineering.

MY PHILOSOPHY

In an undergraduate course, and some graduate courses, I would spend more time on **syntax** and going **step-by-step** in great detail.

In most graduate courses, and professional courses, I focus on **exposure** to (hopefully) new concepts via solving a Data Science problem. I try to give enough so you can learn more later. Please feel free to ask questions. I also want to include *career relevant* material (like in the previous slides).

This talk is in the latter group.

THE SMORGASBORD

I intend to cover a lot that I do not believe has been covered in your program yet. This is meant to be a survey for exposure:

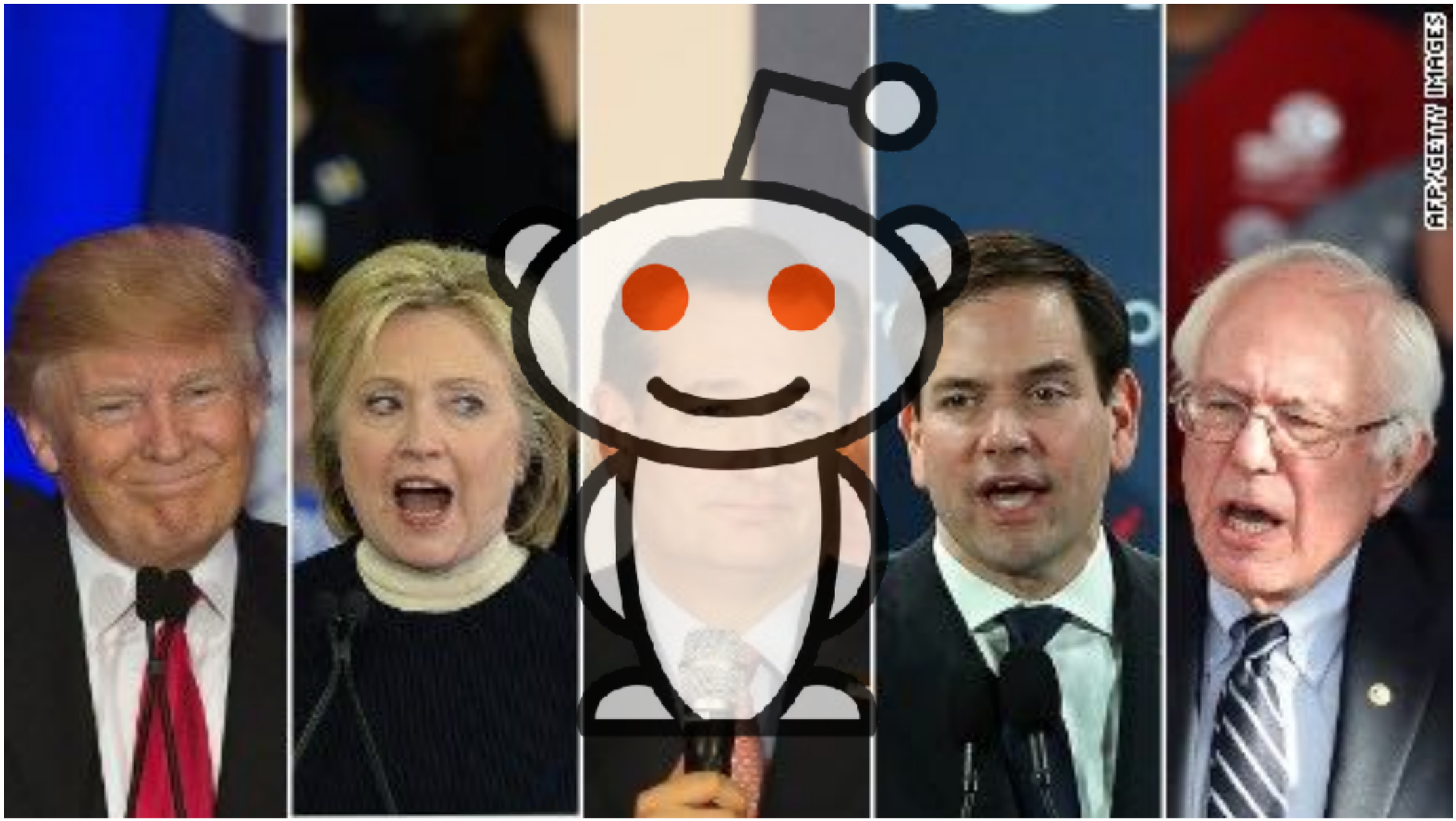
1. Several GNU command-line tools for processing data.
2. Python Introduction
   - ★ Basics, Pandas, Scikit-learn, Jupyter Notebook
3. PyData Demo with a Regular Expressions Detour
   - ★ Statistics and Data Science are all about patterns
4. General Career Tips

THE SMORGASBORD

That's right...

(Though I love them both)

Examining 2016 Candidate Sentiment



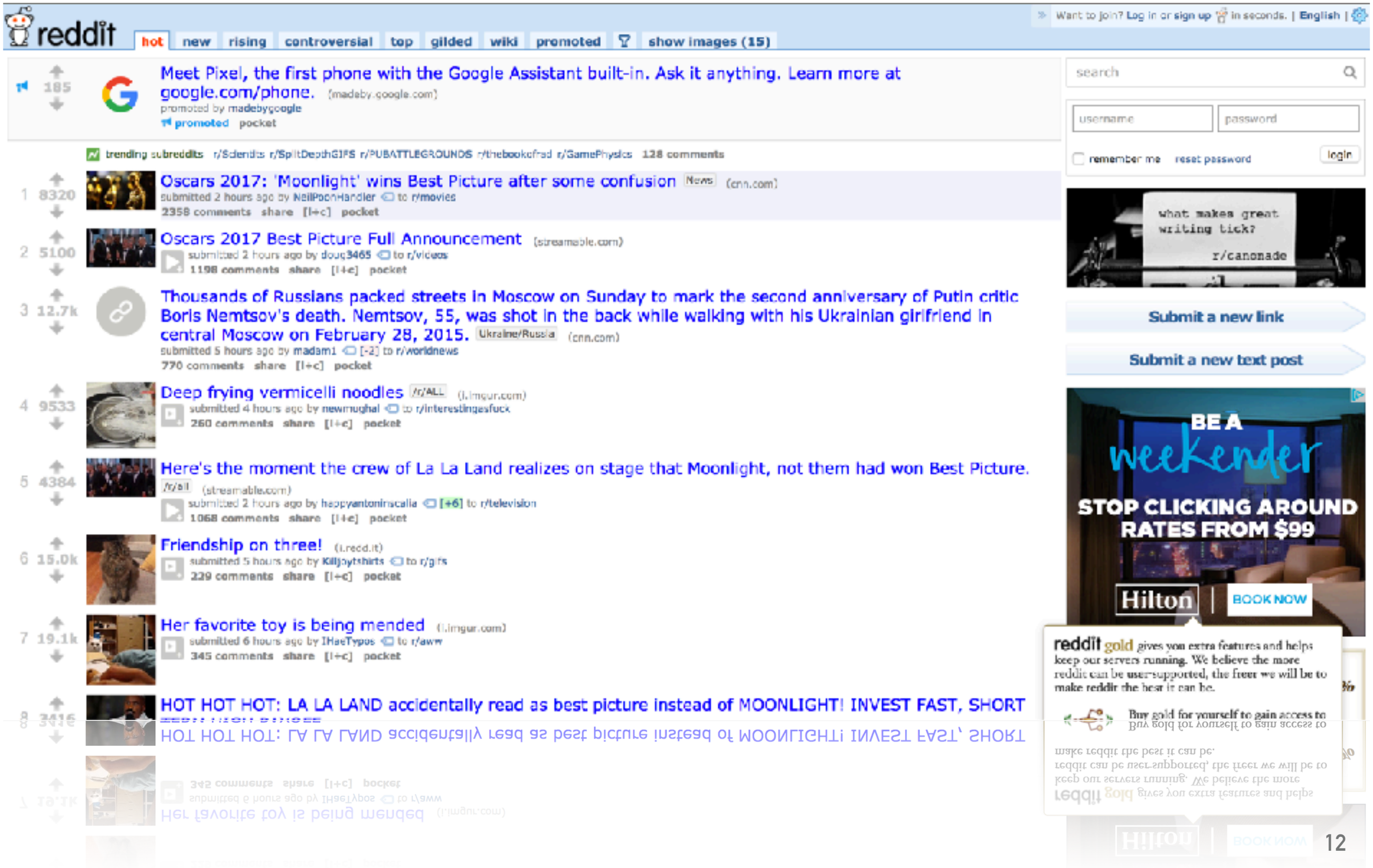
Our Motivation in this Lecture

REDDIT POLITICS

Reddit (**`reddit.com`**) is a social news site.

- Users submit links or text content (their own words).
- Other users comment on the submission.
- Users vote submissions and comments **up** (good) or **down** (bad).

REDDIT


reddit
hot new rising controversial top gilded wiki promoted show images (15)
Want to join? Log in or sign up in seconds. | English
Meet Pixel, the first phone with the Google Assistant built-in. Ask it anything. Learn more at google.com/phone. (madeby.google.com)
promoted by madebygoogle
promoted pocket
trending subreddits r/Scientics r/SplitDepthGIFS r/PUBATTLEGROUNDS r/thebookofrad r/GamePhysics 128 comments
1 8320 Oscars 2017: 'Moonlight' wins Best Picture after some confusion News (cnn.com)
submitted 2 hours ago by NeilPoonHandler to r/movies
2358 comments share [l+c] pocket
2 5100 Oscars 2017 Best Picture Full Announcement (streamable.com)
submitted 2 hours ago by doug3465 to r/videos
1198 comments share [l+c] pocket
3 12.7k Thousands of Russians packed streets in Moscow on Sunday to mark the second anniversary of Putin critic Boris Nemtsov's death. Nemtsov, 55, was shot in the back while walking with his Ukrainian girlfriend in central Moscow on February 28, 2015. Ukraine/Russia (cnn.com)
submitted 5 hours ago by madam1 [-2] to r/worldnews
770 comments share [l+c] pocket
4 9533 Deep frying vermicelli noodles /r/ALL (i.imgur.com)
submitted 4 hours ago by newmughal to r/interestingasfuck
260 comments share [l+c] pocket
5 4384 Here's the moment the crew of La La Land realizes on stage that Moonlight, not them had won Best Picture.
/r/all (streamable.com)
submitted 2 hours ago by happyantoniacalia [+6] to r/television
1068 comments share [l+c] pocket
6 15.0k Friendship on three! (i.redd.it)
submitted 5 hours ago by Killjoytshirts to r/gifs
229 comments share [l+c] pocket
7 19.1k Her favorite toy is being mended (i.imgur.com)
submitted 6 hours ago by IHaeTypos to r/aww
345 comments share [l+c] pocket
8 3416 HOT HOT HOT: LA LA LAND accidentally read as best picture instead of MOONLIGHT! INVEST FAST, SHORT
search
username password
remember me reset password login
what makes great writing tick?
r/canonade
Submit a new link
Submit a new text post
BE A weekender
STOP CLICKING AROUND
RATES FROM $99
Hilton | BOOK NOW
reddit gold gives you extra features and helps keep our servers running. We believe the more reddit can be user-supported, the freer we will be to make reddit the best it can be.
Buy gold for yourself to gain access to

REDDIT POLITICS

Reddit has forums for every possible topic imaginable including politics and the political candidates.

Welcome to /r/Politics! Please read the wiki before participating.

7 — Innocent civilians go undercover in Fulton County Jail to gain a deeper understanding of the criminal justice system. 60 Days In: Atlanta premieres Thursday at 9/8c on A&E. (www.aetv.com)
promoted by AEN_Official
promoted comment share pocket

You are not a subscribed member of this community. Please subscribe to enable voting.

28.9k — Sean Spicer's leak search of his staffers' phones was immediately leaked, and Twitter is loving it dailycaller.com
submitted 10 hours ago by Elloaks Foreign
2475 comments share [l+c] pocket

4384 — Lawmaker: 'I got confirmation' my Russia-Ukraine plan was delivered to White House, despite what Trump lawyer says
businessinsider.com
submitted 7 hours ago by benwsapp
198 comments share [l+c] pocket

5852 — Nominee for Navy Secretary to withdraw politico.com
submitted 8 hours ago by Loadback bernieBot [-2]
454 comments share [l+c] pocket

2334 — Twitter Goes Wild After President Trump Gets Caught Parroting Fox News sfgate.com
submitted 5 hours ago by bluetexan62
134 comments share [l+c] pocket

username password remember me reset password login

SUBMIT A NEW LINK

welcome to r/politics

REDDIT POLITICS

Jason Baumgartner has collected every Reddit submission and comment posted on reddit since 2005.

https://pushshift.io/

The data are separated by month, with each comment file containing about **35GB** of space.

I combined all of 2015 through January 2017 into one file, of size **761GB**.

DON'T JUST DO IT, SHARE IT

All of the materials for this lecture will be posted on my blog.

I will send an email once this is done!

TOO MUCH INFORMATION (TMI)

Please note that you will likely need to make some modifications to how you read the data to work with the constraints of your laptop. This demo was prepared on a server with:

- 32 cores
- 288GB RAM
- 11TB of Disk Space

NOT JASON... IT'S JSON... "J-SAHN"

The data are in a form called JSON, for JavaScript Object Notation. It follows a simple hierarchical format that is easy to read and use with modern languages.


```
{
  "score_hidden": false,
  "name": "t1_cnas8zv",
  "link_id": "t3_2qyr1a",
  "body": "Most of us have some family members like this. *Most* of my family is like this. ",
  "downs": 0,
  "created_utc": "1420070400",
  "score": 14,
  "author": "YoungModern",
  "distinguished": null,
  "id": "cnas8zv",
  "archived": false,
  "parent_id": "t3_2qyr1a",
  "subreddit": "exmormon",
  "author_flair_css_class": null,
  "author_flair_text": null,
  "gilded": 0,
  "retrieved_on": 1425124282,
  "ups": 14,
  "controversiality": 0,
  "subreddit_id": "t5_2r0gj",
  "edited": false
}
```

NOT JASON... IT'S JSON... "J-SAHN"

JSON is a pretty important format for anyone working with computer-generated data that comes from an API.

It comes from JavaScript/ECMAScript and is defined as ECMAScript Standard 404 (an ironic number).

For more info: http://json.org

Aside: XML, and particularly HTML are two other formats you should become familiar with, but we won't deal with those here.

NOT JASON... IT'S JSON... "J-SAHN"

So... I have **761GB** of data. What do I do with it?

For general purpose munging, I might want to throw it in Hadoop or Spark, but we don't need that here.

First, we extract all comments that involve the most interesting 2016 political candidates.

A MOTLEY CREW

Hillary Clinton
/r/hillaryclinton

Bernie Sanders
/r/sandersforpresident

Donald Trump
/r/The_Donald

Free-for-All
/r/politics

NOT JASON... IT'S JSON... "J-SAHN"

`jq` is a command-line tool for querying JSON files. It's very useful, though slightly esoteric for this talk, but I want to share how I worked with this data.

You can get `jq` here if you're interested:

`https://stedolan.github.io/jq/`

NOT JASON... IT'S JSON... "J-SAHN"

Don't worry about the syntax. Some of it we will see in a bit.

```
cat ../comments.json |  jq -r 'select(
    .subreddit=="politics" or
    .subreddit=="The_Donald" or
    .subreddit=="SandersForPresident" or
    .subreddit=="hillaryclinton") |
    [.subreddit, .created_utc, .author, .author_flair_text,
    .score, .controversiality, .body] |
    @tsv' > political_comments.tsv
```

This process takes **8 hrs** on one core, and takes our **761GB** file down to **9GB**.

OUR DATA

The resulting data:

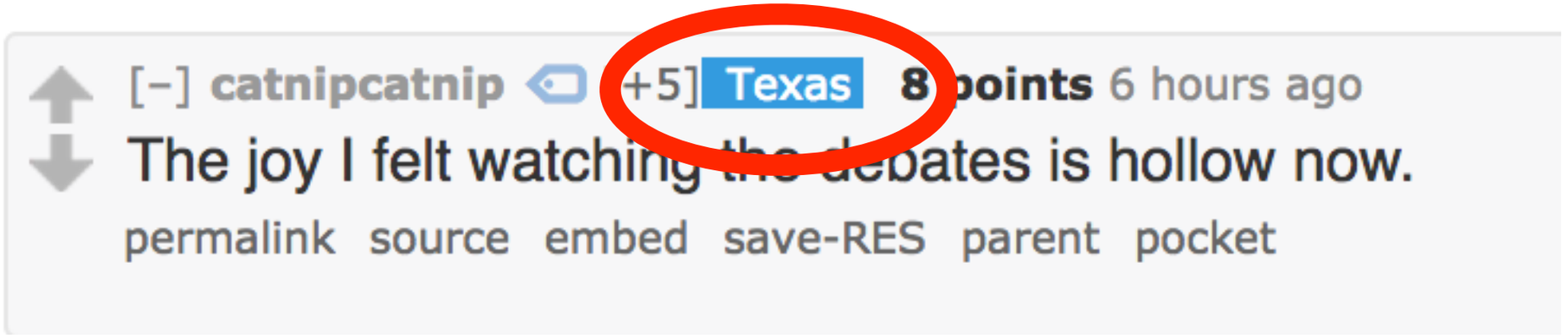
Description	Comment Count
Hillary Clinton	1,343,874
Bernie Sanders	4,374,876
Donald Trump	11,652,277
Politics	23,713,934
Total Comments	**41,084,961**

OUR DATA

Field	Name	Type
1	Subreddit Name	String
2	Timestamp	Integer
3	Username	String
4	Flair	String
5	Score	Integer
6	Controversial Flag	0/1
7	Text	String

OUR DATA

"Flair" is an optional piece of text someone displays next to their name as a "badge." Usually only highly involved members use flair and they *usually* give some kind of information about the poster.



Reddit also marks certain comments (and submissions) as *controversial*. This usually means that the item has been upvoted and downvoted a large number of times.

PART I

The Command-Line

TRIMMING THE FAT

Unix (common variants are Linux and MacBSD included in MacOS X) provides a rich syntax and set of GNU tools that allow us to slice and dice data and chain together data cleansing steps into a pipeline.

We will see that in a moment…

A SHELL GAME

If you are on a Mac, you access the command-line through the **Terminal** app in `Applications>Utilities`.

If you are on Windows 10 (Anniversary Edition and later), you can use a native Bash shell.

On prior versions of Windows, you can use Cygwin.

DISCLAIMER

In this command snippers that follow, you should copy each line into the shell individually, not all together as it may not work.

STRAY CAT STRUT

We can display a file to the screen using the **`cat`** command.

More importantly, we can start a *pipeline* using **`cat`** to turn a file into a stream.

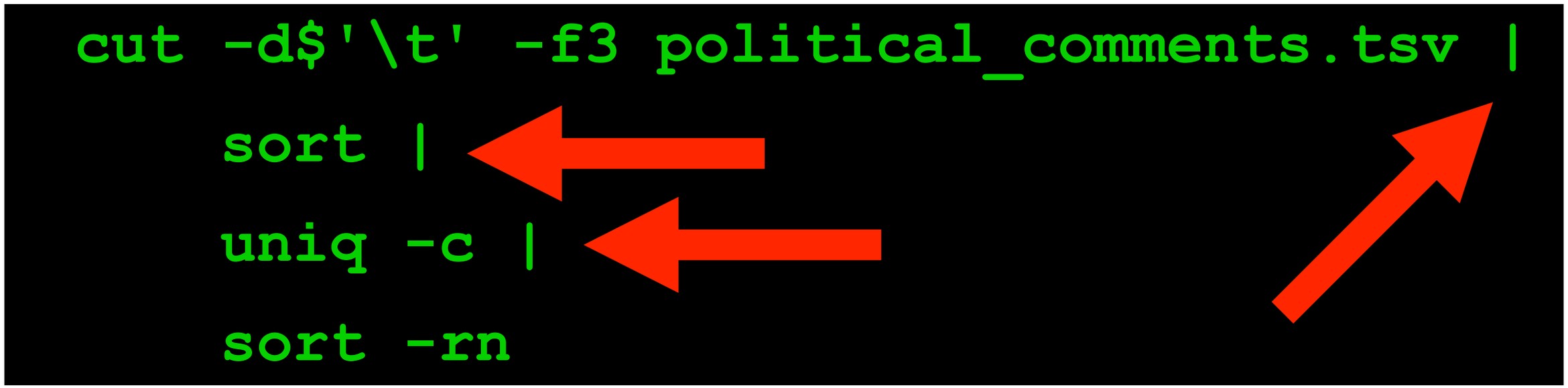
```
cat filename
```

We can also stack multiple files sequentially using **`cat`**.

```
cat filename1 filename2 filename3
```

DATA PLUMBING: THE PIPE (|)

In our field always group data operations into a sequence of steps. The *pipe* operator allows us to do this.



`c1 | c2`

"take the output of the command before the pipe (c1), and use it as the input for the command on the right (c2)"

TRIMMING THE FAT

BEFORE we load anything into R or Python, there's a lot we can do on the command-line to clean up the data and filter it.

How can we do that?

First note that the data is **tab-delimited**.

TRIMMING THE FAT: `cut`

We can look at individual columns, or ranges of columns using the **cut** command. We specify a delimiter with **-d** and a field or range with **-f**.

Note: -d only accepts a single character, so we need the $ sign to specify we want to treat `\t` as a single character.

```
cut -d$'\t' -f3 political_comments.tsv
```

TRIMMING THE FAT: `cut`

Some more examples of **`cut`**:

```
cut -d$'\n' -f1,4 file.txt
cut -d' ' -f1-4 file.ssv
cut -f1,3 --output-delimiter=" " file.ssv
    > new file.ssv
```

TRIMMING THE FAT: `cut`

So, now we have a list of usernames that made comments. We want to look for anomalies here, but just having a list is not helpful. Let's instead count how many times each user comments

```
cut -d$'\t' -f3 political_comments.tsv |
    sort |
    uniq -c |
    sort -rn |
    head
```

TRIMMING THE FAT: `cut`

By using this pipeline, we notice some interesting things:

```
4091175 [deleted]
 268495 AutoModerator
  52951 PoliticsModeratorBot
  34793 TrumpTrain-bot
  26685 fastmandan
  23730 GonzoNation
  22715 JumpingJazzJam
  17940 CarmineFields
  17384 PEPE_Price_bot
  16419 escalation
```

TRIMMING THE FAT: `cut`

By using this pipeline, we notice some interesting things:

```
4091175 [deleted]
 268495 AutoModerator
  52951 PoliticsModeratorBot
  34793 TrumpTrain-bot
  26685 fastmandan
  23730 GonzoNation
  22715 JumpingJazzJam
  17940 CarmineFields
  17384 PEPE_Price_bot
  16419 escalation
```

TRIMMING THE FAT: `cut`

We see three important things:

- lots of anonymous comments: **`[deleted]`**
- warnings, bans, etc. by **`automoderator`**
- lots of comments by **`bot`**s

We will come back to this in a moment…

READING FROM THE PIPE

Now let's look closer at this pipeline:

```
cut -d$'\t' -f3 political_comments.tsv |
    sort |
    uniq -c |
    sort -rn
```

MAKING ORDER OUT OF CHAOS: `sort` and `uniq`

We can sort data in *ascending order* using the **sort** command. Simple!

We can use a few arguments to change how it works:

`-r`	Reverse: Sort in descending order
`-n`	Treat input as numeric
`-k`	Specify a sort key using delimiter `-t`
`-m`	Merge two already-sorted files

MAKING ORDER OUT OF CHAOS: `sort` and `uniq`

Once the file is sorted, we can find the unique rows using `uniq`.

The only interesting argument is `-c` which counts the number of times each unique value occurs, as in our previous example.

MAKING `heads` AND `tails` OF THE SITUATION

Finally, we can get a quick glimpse of the output using `head` or `tail`.

- `head` displays the top 10 lines.
- `tail` displays the *last* 10 lines.

We can display a different number of lines. For example, to display 100 lines, add `-100`.

MAKING `heads` AND `tails` OF THE SITUATION

There is one interesting use of tail that you may use if you write software, apps etc.

```
tail -f
```

This command **streams** data from files as the data is written. Use `tail -f` to listen to a socket, watch a log file populate with errors etc.

OK, back to this . . .

ALL DATA SCIENTISTS DO IS COUNT THINGS

Earlier we saw that we have a large number of weird comments. But how significant is this?

wc DOESN'T ONLY MEAN "WATER CLOSET"

We can determine the percentage of the comments made by these weird users. We have the numerator, let's get the denominator.

```
wc -l political_comments.tsv
```

We see there are 41,084,961 comments so

User	Count	Percentage
[deleted]	4,091,175	10%
AutoModerator	268,495	0.6%

`wc` DOESN'T ONLY MEAN "WATER CLOSET"

`wc` stands for **word count** and we can count other things in files or streams as well

`-l`	line counts (we used this)
`-w`	word counts
`-m`	character counts
`-c`	byte counts

DOMO ARIGATO MR. ROBOTO

But what about those bots? How many of those do we have? We can't really know, but if **bot** is present in special parts of the username, we can assume it's a bot. Right??

```
4091175 [deleted]
 268495 AutoModerator
  52951 PoliticsModeratorBot
  34793 TrumpTrain-bot
  26685 fastmandan
  23730 GonzoNation
  22715 JumpingJazzJam
```

A BRIEF REDIRECTION

Before we move any further, let's save the list of usernames so that we don't have to recompute it each time we want to query it. We are going to use this again, but this time we won't do a count (no `-c`)

```
cut -d$'\t' -f3 political_comments.tsv |
    sort |
    uniq > usernames.dat
```

A BRIEF REDIRECTION

The **>** takes the output from the left, and writes it to the file (or stream) listed on the right. This overwrites an existing file, or creates a new one if it doesn't exist.

We can also append new data to an existing file using **>>**.

There are other ways to output streams to other streams we won't cover. Google "input output redirection" and the **`tee`** command.

FINDING THE DIAMOND IN THE ROUGH: `grep`

We do a naive search for bots by searching for the word `bot`. To search input for a particular term, we can use `grep`. The flag **`-i`** means "case doesn't matter."

```
cat usernames.dat | grep -i bot
```

We could just do the following, but I want to show the usage of cat.

```
grep -i bot usernames.dat
```

FINDING THE DIAMOND IN THE ROUGH: `grep`

But there is a problem…

TrumpTrain-bot	PEPE_Price_bot	trumpcoatbot	robotzor
bottomlines	IsometricRobot	RobotJINI	gifs-bot
robottonic	DrRubotnik	Deplorables_bot	MAGA_bot
RobotCowboy	BrakeSaboteur	Marco_Robotio	autowikibot
hotbotnica	rabot101	Mrrobot13377	r0botdevil

FINDING THE DIAMOND IN THE ROUGH: `grep`

We have a lot of false positives. We could train a classifier to mine the proper *pattern*, but let's just use our intelligence.

We see a pattern

- true bots have usernames that **end** in bot.
- capitalization does not matter

FINDING THE DIAMOND IN THE ROUGH: `grep`

So how do we use that information to filter out bots? We can use `-v` to invert the search. Find all usernames that do not have `bot` at the end. We sort as it will come in useful later.

```
cat usernames.dat | grep -iv 'bot$' |
sort >goodusers.dat
```

We specify a *pattern* called a *regular expression*. The **$** means "find the word bot when it appears at the end of the line." Conversely, ^ matches at the beginning.

FINDING THE DIAMOND IN THE ROUGH: `grep`

We will remove the bots from the data in a bit but first…

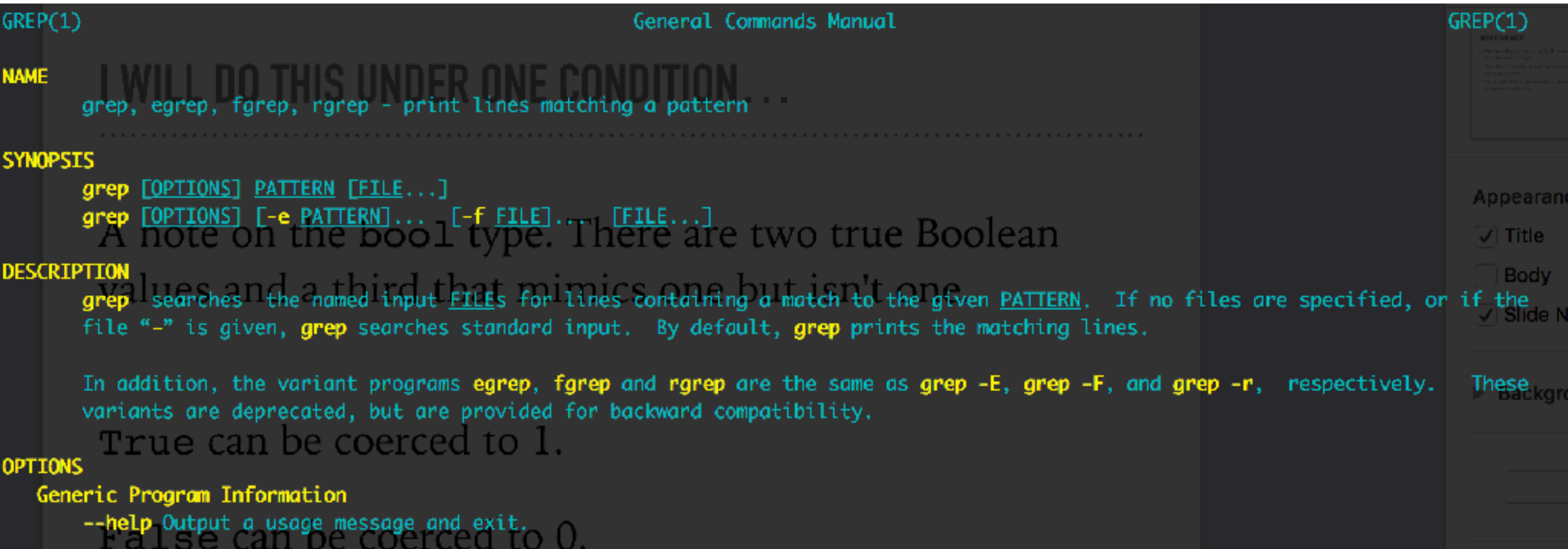
`grep` can do some very complex things. Here are some more arguments

- `-e` used multiple times allows multiple patterns.
- `-f` read patterns from a file, one line per pattern
- `-v` invert the match; show lines that don't match
- `-c` only count the number of lines that match
- `-o` print only the part that matches (regex)
- `-n` print the line number where match was found

A WINNING ARGUMENT

So where do all of these arguments come from? To see the documentation for a program or a command and its arguments, use `man` for manual.

```
man grep
```



PEOPLE ALL OVER THE WORLD... JOIN IN

OK, so if we want to remove bots from the Reddit comment file, we can use `grep -v` right? **NO!**

Unfortunately, `grep` considers *lines* not *fields* so if we used `grep -v` we would be filtering out comments whose text contains the pattern `bot$` as well.

This is a good job for the command `join`.

PEOPLE ALL OVER THE WORLD... JOIN IN

We will join the comments file `political_comments.tsv` with the list of good users.

The join key in `political_comments.tsv` is column 3.

The join key in `goodusers.dat` is column 1 (there is only one column).

PEOPLE ALL OVER THE WORLD... JOIN IN

But life is never *this* easy. In order to join two files on the command-line, both files must be sorted!

So we sort the huge `tsv` file on column 3 (username), using `\t` as the delimiter.

```
sort -k3 -t$'\t' political_comments.tsv >
          political_comments_sorted.tsv
```

The `goodusers.dat` file is already sorted.

PEOPLE ALL OVER THE WORLD... JOIN IN

Now we can join both files. We specify left join key (LHS file) with `-1` and the right join key (RHS file) with `-2` and the delimiter with `-t`.

```
join -1 3 -2 1 -t$'\t'
    candidate_comments_sorted.tsv
    goodusers.dat >candidate_comments_clean.tsv
```

MOVING ON...

There's more we can do on the command-line to clean the data up, but let's move on. We now have a clean file `political_comments_clean.tsv`.

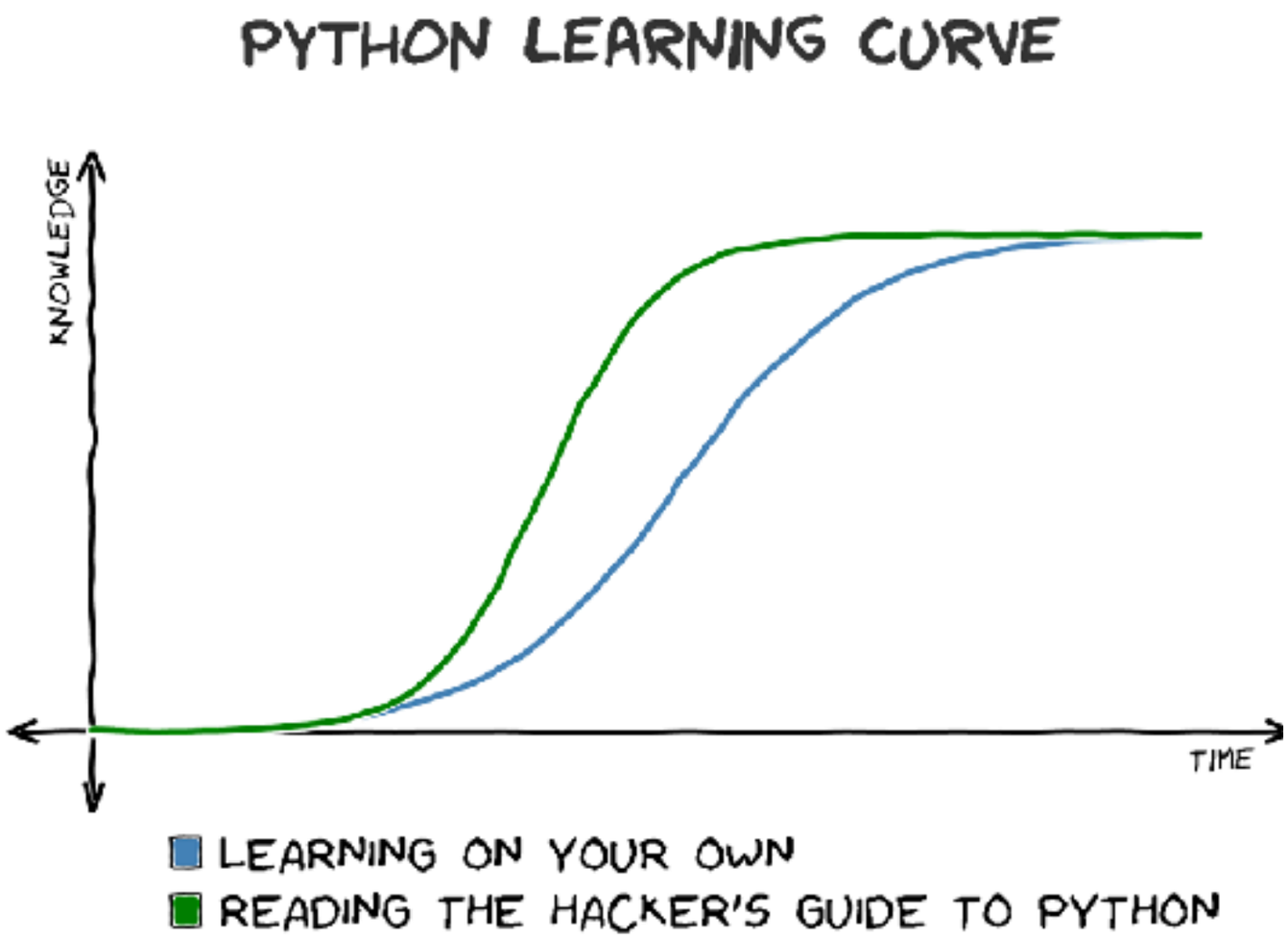
We could remove some nuisance characters using `tr`.

We can do more advanced cleaning with regular expressions using `sed` and `awk`.

OUR SAVIOR

Many of the tools we just used were written fully or partly by **Dr. Paul Eggert.**

He is a lecturer at UCLA in Computer Science!



PART II

Python

A SNAKE IN THE GRASS

Python is another critical language for Data Science.

- **strongly** and **dynamically** typed.
- can be used for **scripting**, or **full applications** and is a **great glue language.**
- code is written in `.py` files and *compiled* to byte-code in a `.pyc` which is then *interpreted* (CPython).
- good engineering may make use of object-oriented programming, but in Python is it not enforced.

PYTHON VS. R FOR DATA SCIENCE

Oh boy. A big moot point in this field…

Advantages of Python over R*

- Is a full, general purpose programming language.
- Natively more scalable for system development.
- A full technology ecosystem.
- Is *usually* faster, but this depends!
- More straightforward *general* syntax.

** - my opinion only.*

PYTHON VS. R FOR DATA SCIENCE

Advantages of R over Python*

- Is *data first*.
- Strong ecosystem for *statistics*.
- Rock solid visualization features that are easy to access.
- Early access to cutting-edge *statistical* methods.
- More straightforward *data* syntax.

** - my opinion only.*

PYTHON VS. R FOR DATA SCIENCE

Pick the one that solves your problem!

- Analysis? Lean toward **R.**
- System development and/or software? **Python.**

More considerations:

- RStudio apparently supports Python now.
- Jupyter (IPython) Notebook supports R.
- R can be called from Python.
- Both can call C and C++.

ANACONDA: NOT ONLY A NICKI MINAJ "SONG"

There are many distributions of Python, including the standard one at `python.org`, and the one that comes with Mac and Linux Systems.

Anaconda encourages use of clean Python environments. Multiple versions of Python are no problem, and package management is a breeze.

I use Python 3, and you should too…

`https://www.continuum.io/downloads`

DISCLAIMER

This will be a quick introduction. Even if the whole 3 hours was spent only on Python, we could only scratch the surface.

I will provide resources at the end of this section.

HELLO WORLD

Like most interpreted languages, Python provides an *interpreter*, a shell where you can type commands and code and see what they do.

You can access the interpreter with the command `python`, though a better interpreter is `ipython`.

Otherwise, you can run a script using

```
python scriptname.py
```

INDENTATIONS AREN'T JUST FOR PARAGRAPHS

For better or worse, Python uses indentation to group lines together, rather than braces `{}`.

Indentations must be consistent!

```
for (int i=0; i < 5; i++) {
for (int j=0; j < 5; j++) {
for (int k=0; k < 5; k++) {
dosomething();
}
dosomething2();
}}
```

Ugly

```
for i in range(5):
    for j in range(5):
        for k in range(5):
            dosomething()
        dosomething2()
```

Pretty!

HELLO WORLD

First we write the usual, stupid Hello World app.

```
print("Hello world!")
```

NO COMMENT

You should comment complex parts of your code. To comment a single line, use #.

```
print("hello world")     # prints hello world
```

To comment multiple lines, use """ or '''

```
def do_nothing():
    """
    This function does nothing.
    Call this in an infinite loop, and you get my behavior
    on the weekend.
    """
    pass
```

VARIABLE DECLARATION AND TYPES

In Python, there are no variable *type* declarations, just variable declarations.

```
x = 5

foo = "hello world"

shopping_list = ["eggs", "milk", "beer"]

index = {"name": "Ryan", "phone": "yeah right"}
```

VARIABLE DECLARATION AND TYPES

There are some unique variable declarations as well, such as multiple assignments and tuple packing.

```
a = b = c = 1  # literal must be last value.

x, y, z = 3, "hello", False
```

VARIABLE DECLARATION AND TYPES

You can delete a variable, but this very rarely used except with collections (`lists`, `dicts` etc.) and in high performance computing.

```
a = 10

del a
```

Like Java, Python uses *garbage collection*. The runtime determines which variables are no longer used and removes them from RAM without any user interaction.

VARIABLE DECLARATION AND TYPES: NUMERIC

There are three major numeric types in Python

1. `int`: integer
2. `float`: floating point decimal
3. `complex`: complex numbers, not used in statistics.

There are no precision modifiers (`int8`), sign modifiers (`uint8`) or `doubles`. BUT, in packages like `scipy` and `pandas`, there are.

VARIABLE DECLARATION AND TYPES: NUMERIC

```
a = 2               # int
b = 3.5             # float
c = float(2)        # float from an int
d = 2.              # float, explicitly defined
```

We also have the usual arithmetic operators (+, -, *, /):

```
a, b, c = 3, 2, 1.2
a + b     # 5
a * c     # 3.6  - implicit conversion among numeric types
a / b     # 1.5 in Python 3, 1 in Python 2!!
a % b     # 1 — modulus
a // b    # 1 - floor division
a ** 2    # Power, 9
```

VARIABLE DECLARATION AND TYPES: NUMERIC

Assignment (in-place) operators also work.

```
a, b, c = 3, 2, 1.2
a += b     # a is 5
a *= c     # 3.6  - implicit conversion among numeric types
a /= b     # a is 1.5 in Python 3, 1 in Python 2!!
a %= b     # a ix 1 — modulus
a //= b    # a is 1 - floor division
a **= 2    # Power, a is 9
```

These can be useful in low memory situations in `numpy`.

Oh, and just assume I reset `a = 3` each command.

VARIABLE DECLARATION AND TYPES: STRINGS

Strings are defined with single quotes, or double quotes.

```
foo = "Don't worry be happy"

bar = 'Don\'t worry be happy'     # escape the apostrophe

baz = "2.0"                       # a string
```

Special characters such as carriage returns, tabs and Unicode can also be embedded.

```
foo = "Don't worry be happy\nIt's just a test."

bar = "Don\'t worry be happy\\nIt\s just a test'
```

VARIABLE DECLARATION AND TYPES: STRINGS

Some arithmetic operators work with strings:

- + concatenates two strings

```
"hello" + " " + "world"    # "hello world"
```

- * when used with an int repeats a string n times

```
"yadda " * 3                    # "yadda yadda yadda "
```

- The % is used to insert values of variables into a string

```
my_name = "Ryan"
print("Hello, my name is %s." % my_name)
```

VARIABLE DECLARATION AND TYPES: STRING INTERPOLATION

The thing with the `%` is called **string interpolation**.

- `%s` to inject another string (concatenation)
- `%d` to inject an integer
- `%f` to inject a float

You can also specify by variable name, use a collection, specify the length, padding, and a ton of other things related to each data type. It can get quite complicated and unreadable. For more information:

`https://pyformat.info/`

VARIABLE DECLARATION AND TYPES: STRING INTERPOLATION

Python 3 recommends a different way to do string interpolation using the `format` function.

```
"Hello my name is {}.".format("Ryan")

"Hello my name is {my_name}.".format(my_name="Ryan")


firstname = "Ryan"

lastname = "Rosario"

"Hello my name is {} {}".format(firstname, lastname)
```

It is also possible to get complicated with data types by specifying the type and any other formatters in the braces.

VARIABLE DECLARATION AND TYPES: STRING INTERPOLATION

Two most common features you will probably use are getting the length of a string, and splitting on a character.

- Use function **`len()`** to get the length of a string.
- Use string method **`split()`** to split on a character.

```
mystr = "Hello my name is {}.".format("Ryan")
len(mystr)     # 22


mystr.split(' ')  #Returns a list:
                  # ["Hello", "my", "name", "is", "Ryan"]
```

VARIABLE DECLARATION AND TYPES: LISTS

Lists are ordered collections, like an array, but can contain mixed types.

```
mylist = []     # create an empty list
mylist2 = [1, 2, 3]
mylist3 = [1, 2, 'mixed']     # mixed datatypes
```

Lists are indexed starting at 0 using the brackets `[ ]`.

We add to a list using append, and can add two lists together.

```
mylist.append(1)     # [1] append has side effect.
mylist + mylist2     # [1, 1, 2, 3]
mylist2 * 3          # [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

VARIABLE DECLARATION AND TYPES: LISTS

There are two major ways to iterate over lists:

```
mylist = [1, 2, 3, "hello"]
for value in mylist:
    do_something(value)          # value is an entry.
```

Or by index:

```
for i in range(len(mylist)):    # in Python 2, use xrange()
    # range generates an iterator from 0 to len(mylist)
    # NOT INCLUDING i=len(mylist):    [0, len(mylist))
    do_something(mylist[i])
```

VARIABLE DECLARATION AND TYPES: LISTS

Oh, and you can use *negative* indices to access the list from the back.

```
mylist = [1, 2, 3, "hello"]
mylist[-1]    # "hello", the last entry
mylist[-3]    # 2
```

To *delete* an entry in the list, use `del`. This is one of the acceptable uses of `del`.

```
mylist = [1, 2, 3, "hello"]
del mylist[2]    # [1, 2, "hello"]
```

VARIABLE DECLARATION AND TYPES: THE API

To get a full list of methods that can be called on an object, `dir()`

```
dir("my string")

dir([1,2,3])
```

The `help()` function is also useful.

```
help(str)          # Getting help on a class / object type.

help("mystring".strip)  # Getting help on a method

                        # Note that strip() does not work.

import os

help(os)                # Getting help on a module.
```

VARIABLE DECLARATION AND TYPES: THE API

We can see the `len()` can be called on collections.

- On a `string`, gives the length of the string in characters.
- On a `list`, gives the number of entries.
- On a `dict`, gives the number of keys.
- On a `tuple`, gives the number of entries.

VARIABLE DECLARATION AND TYPES: DICTIONARIES

A **`dict`** (for dictionary) is an *unordered* key-value structure indexed by a key rather than an index. They are similar to hash tables in other languages.

```
mydict = {}     # create an empty dict
# Simple structure with one level:
mydict2 = {"firstname": "Joe", "lastname": "Bruin"}
# A dict with mixed types.
mydict3 = {"firstname": "Joe", "lastname": "Bruin",
           "age": 100,
           "friends": ["Josie Bruin", "Gene Block"]
          }
```

VARIABLE DECLARATION AND TYPES: DICTIONARIES

We extract a value by key rather than by index:

```
mydict3["first"]     # returns Joe.
mydict3[0]           # Raises an exception, there is no key "0".
```

Caveat: Keys must be hashable. To be safe, use simple datatypes.

To delete from a `dict`, use `del` with the key to delete.

```
del mydict3["first"]
```

VARIABLE DECLARATION AND TYPES: DICTIONARIES

There are two major ways to iterate over dictionaries.

By key:

```
for key in mydict:
    do_something(key, mydict[key])      # value is a key.
```

Or by key/value pair:

```
for key, value in mydict.items():  # use iteritems() in Python 2
    do_something(key, value)
```

VARIABLE DECLARATION AND TYPES: TUPLES

The simplest collection is the tuple. It is similar to a `pair` in C++ and maybe a `union` in C.

They are *immutable* (can't be changed after declaration) and may contain mixed types.

```
mytuple = (1, )     # a tuple with one element, WATCH THE COMMA!
mytuple2 = (1, 2, 'foo', False)
```

We access by index, and iterate the same as a list.

```
mytuple2[3]     # False
```

VARIABLE DECLARATION AND TYPES: YOUR OWN TYPES

There are several add-on collection types available in the `collections` package. Some good stuff there!

You can also create your own types by writing a `class`. We won't get into that in this quick survey of Python, but take a look at the references at the end of this section.

VARIABLE DECLARATION AND TYPES: NESTED TYPES

Collection datatypes can also be nested and contain other datatypes:

```
facebook_profile = {"name": "Ryan Rosario",
                    "age": None,
                    "major": ["Statistics", "Computer Science"],
                    "jobs": [
                        {"company": "Facebook",
                         "title": "Engineer"},
                        {"company": "UCLA",
                         "title": "Grad Student"}]}
# A list of lists.
[1, 2, [3, 4], 5]
# A group of FB profiles could be a list of dicts.
```

Now we will quickly go over the basic conditionals and control flow in Python.

Many of these are the same, or very similar to R and C, so this will be quicker than the previous section.

I WILL DO THIS UNDER ONE CONDITION...

A note on the `bool` type. There are two true Boolean values and a third that mimics one but isn't one.

`True` can be coerced to 1.

`False` can be coerced to 0.

`None` is of type `None`! Though, in conditionals it evaluates to `False`.

I WILL DO THIS UNDER ONE CONDITION...

Python of course supports `if/else` conditional statements.

1. Use `if` to start the statement
2. Use `elif` for alternate conditions (`else if`, `elsif` in other languages)
3. Use `else` to catch all other cases.
4. `if` is required; `elif` and `else` are not. Be careful!
5. These can be nested with indentation.

I WILL DO THIS UNDER ONE CONDITION...

Some examples:

```
# An exponential r.v. x cannot take a negative value:
x = -1
if x < 0:
    raise ValueError("Exponential r.v. cannot be negative")


# Mutual exclusion. Let y be a positive int.
if y % 2:    # Note we don't explicitly check the literal value.
    return "odd"
else:
    return "even"
```

I WILL DO THIS UNDER ONE CONDITION...

Some examples:

```
# Switch statement. Let x be a grade level starting at 0 for K.
if x == 0:
    return "Kindergarten":
elif 1 <= x <= 5:
    return "Primary"
elif 6 <= x <= 8:
    return "Middle"
elif 9 <= x <= 12:
    return "High"
else:
    return "College or higher"
```

I WILL DO THIS UNDER ONE CONDITION...

Boolean conditionals can be formed using the words `and`, `or`, and `not` as well as the usual operators.

The terms `is` and `is not` are also used for identity.

The term `in` is used to test collection membership.

I WILL DO THIS UNDER ONE CONDITION...

```
new_chardonnay = ["alcohol", "grapes", "black fly"]


if "black fly" in new_chardonnay:
    print("Isn't it ironic?")


if type(my_var) is str:          # checking identity
    print("It's a string!")
# Let prime_numbers be a list containing all the prime_numbers
# I know, we don't know all of them, just go with it :)
if x > 0 and x % 2 and x not in prime_numbers:
    print("We've found an odd number that's not prime!")
```

Caveat: `is` `in` does not work because `is` is for identity.

LOOP DA LOOP

We repeat actions using loops, just like in R and other languages.

There are only two types of loops: `for` and `while`.

There is no `repeat`, `do while` or explicit `foreach`.

LOOP DA LOOP

`for` works just like in other languages, and has `foreach` functionality built-in via its syntax. As soon as the condition is False, the loop body is skipped.

```
for value in <some_collection>:     # foreach example
    do_something(value)

for key, value in my_dictionary:    # foreach with two iterators.
    do_something2(key, value)

for i in range(33):     # Powers of 2 up to 2**32
    print(2 ** i)           # note the 33! Ends at 32
```

LOOP DA LOOP

`while` works just like in other languages. As soon as the condition is False, the loop body is skipped.

```
while some_condition:
    do_something()


i = 0                   # initialize counter variable
while i < 3:
    print("da ")        # da da da
    i += 1              # increment counter
```

LOOP DA LOOP

In many networking applications and others that wait for user input, it is common to see an explicit infinite loop.

```
import time

while True:
    inval = check_for_input()
    if inval:
        break
    …
    time.sleep(1)    # optional, wait a second or so.
```

LOOP DA LOOP

There are ways to manually control execution within a loop if certain situations arise.

- `break` cancels remaining execution within the loop and moves to the next block of code.
- `continue` cancels remaining execution of this iteration of the loop and moves to the next one.
- `pass` literally does nothing. When syntax is required, but you want to do nothing, use `pass`.

LOOP DA LOOP: WHAT `else`?

I struggled with whether or not I should teach this one.

Oddly enough, there is an **`else`** statement for loops in Python.

If we exhaust the loop without breaking out, the code in `else` runs.

One use case is when you iterate over an object, and reach the end without finding what you're looking for.

LOOP DA LOOP: WHAT `else`?

```
haystack = ['mouse', 'rat', 'hay', 'hay', 'hay', 'baby']


# Search for a needle in the haystack.
for thing in haystack:
    if thing == "needle":
        print("We found the needle in the haystack!")
        break
else:
    print("We didn't find the needle.")
```

CONJUNCTION JUNCTION WHAT'S YOUR FUNCTION?

Functions are very simple in Python compared to other languages.

- There are no return types.
- There are no parameter types.
- Don't have to worry about pass by value or pass by reference, Python has its own mechanics.
- Can return multiple values of different types.

CONJUNCTION JUNCTION WHAT'S YOUR FUNCTION?

Functions are defined with the keyword `def`, followed by a name, a set of parameters in parentheses and a colon.

```
def foo(myint1, myint2):
    added, multiplied = myint1 + myint2, myint1 * myint2
    return added, multiplied
```

Caveat: `foo` implies that the parameters should be `ints`. If they are not, an exception will be raised. Python does not check beforehand (strong and dynamic).

CONJUNCTION JUNCTION WHAT'S YOUR FUNCTION?

Calling a function is no different than in R or C.

```
foo(5, 6)                    # gives 11, 30
foo("hello", "world")        # an Exception is raised.
```

CONJUNCTION JUNCTION WHAT'S YOUR FUNCTION?

There are four types of function arguments in Python:

1. required arguments
2. keyword arguments
3. default arguments
4. variable length arguments

REQUIRED ARGUMENTS

Required arguments are the most common. They must be present, and must be given in the order they were defined.

If a value for the argument is not passed, an exception will be raised.

```
   def foo(a, b):
       return


foo(2)
TypeError: function() missing 1 required positional argument: 'b'
```

KEYWORD ARGUMENTS

Keyword arguments are similar in R. They can be specified out of order as long as the keyword is present.

```
def foo(a, b, c):
    return

foo(c, b=2, a=0)     # This works!!
```

Caveat: As soon as an parameter is out of order, all following parameters must be passed with keywords!

KEYWORD ARGUMENTS

Here is a function with 6 arguments.

Caveat: As soon as an parameter is out of order, all following parameters must be passed with keywords!

```
   def foo(a, b, c, d, e, f):
       return
foo(True, 5, 2, "car", 8.0, False)        # works, all required
foo(True, 5, 2, f=False, "car", 8.0)      # does not work.
foo(True, 5, 2, e="car", d=False, 8.0)    # does not work.
foo(True, 5, 2, f=False, d="car", e=8.0) # works
```

DEFAULT ARGUMENTS

Default arguments are similar to other languages, but must appear after required arguments.

```
def foo(first, last, age=21):
    print(
        "My name is {} {} and I am {}.".format(
            first, last, age))

foo("Ryan", "Rosario")   # "My name is Ryan Rosario and I am 21."
foo(last="Rosario", first="Ryan")   # This works!
```

VARIABLE LENGTH ARGUMENTS

Variable length arguments are a bit niche for everyday use, but appear everywhere in open-source code.

`*args`, `**kwargs`

The stars **do not** denote pointers.

- `*args` represents a number of unnamed arguments.
- `**kwargs` represents a number of named arguments.

They **always** appear last, and cannot easily be mixed with other types aside from required arguments.

VARIABLE LENGTH ARGUMENTS

Here is an example:

```
def buy(cc_number, *items, **metadata):
    for item in items:
        add_to_cart(item)
    charge_credit_card(cc_number)
    ship_items(items, metadata)

buy(1234, "apple", "beer", , ship_to="Ryan", method="USPS")
```

You could also just pass a collection instead of using variable length arguments.

CAVEAT #1

Python is neither *pass by value* nor *pass by reference*.

- **Immutable** objects (`int`, `str`, `tuple`) *mimic* "pass by value." Changing their value in a function does not change it outside the function.
- **Mutable** objects (everything else) *mimic* "pass by reference." Changing their value in a function ***does*** change the value outside the function.

This is very important and has bitten me many times.

CAVEAT #1

```
a = 10
def pass_by_value_sortof(x):
    x = 5
pass_by_value_sortof(a)    #  a is still 10
```

```
b = [1, 2, 3]
def pass_by_reference_sortof(x):
    x[0] = 1000
pass_by_reference_sortof(b)    #  b is now [1000, 2, 3]
```

CAVEAT #1

Wait what? Integers and strings are immutable?

You had to ask, *sigh*.

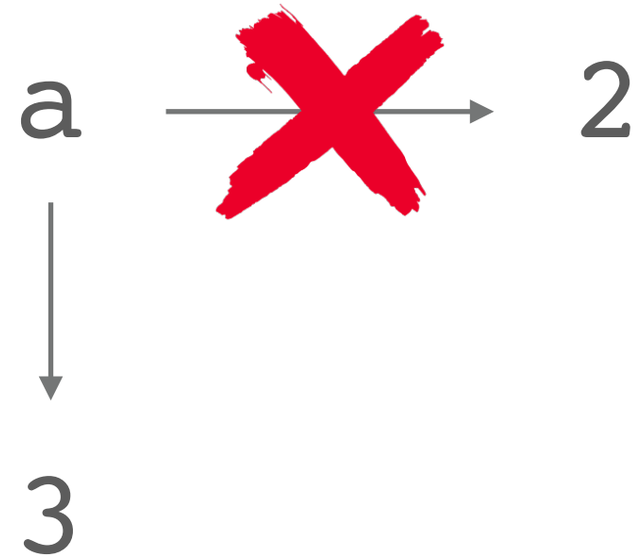
Python "variables" are just a binding between a name, and a value.

CAVEAT #1

When we say `a = 2`, what is actually happening is we define a *name* a, which points to a value in RAM, the number 2.

CAVEAT #1

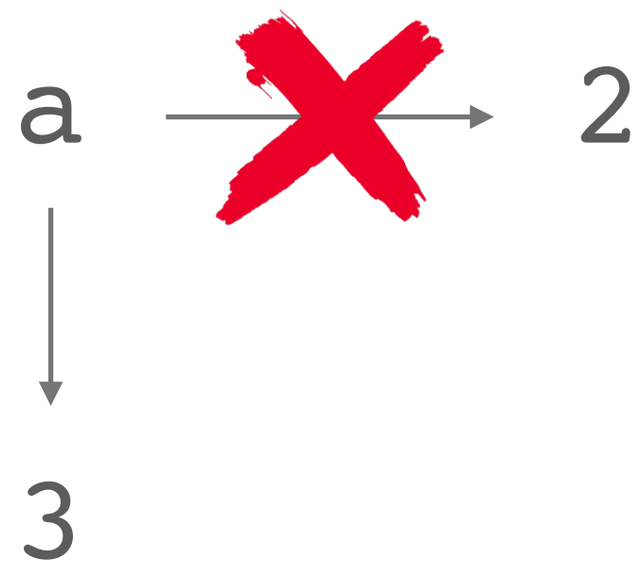
Now let's do something radical and "change" the value, say, `a = 3`.



CAVEAT #1

All the we have done is *rebind* `a` to point to the value 3 in RAM. The value 2 still exists in RAM, **it did not change.**

The garbage collector will see that nothing *refers* to that value of 2 in RAM, and it will delete it, eventually.



CAVEAT #1

Since the *value itself* did not change in RAM, we say that an `int` is immutable. This also applies to strings and tuples, though tuples cannot even be modified!

MODULES: THIS BUT NOT THAT

You will want to use external packages. There are several ways to import packages into Python for use.

```
import numpy
```

Import all of `numpy`. Access members with `numpy.<member>`

```
import numpy as np
```

Import all of `numpy`, but access access as `np.<member>`

MODULES: THIS BUT NOT THAT

You will want to use external packages. There are several ways to import packages into Python for use.

```
import sklearn.model_selection
```

Import only the `model_selection` submodule from `sklearn` (`scikit-learn`). Access with `sklearn.model_selection.<member>`

```
import sklearn.model_selection as ms
```

Access with `ms.<member>`

MODULES: THIS BUT NOT THAT

You will want to use external packages. There are several ways to import packages into Python for use.

```
from sklearn import cross_validation
```

Import only the `cross_validation` submodule from `sklearn` (`scikit-learn`). Access with `cross_validation.<member>`

```
from sklearn import cross_validation as cv
```

Access with `cv.<member>`

This syntax is sometimes used to import functions.

MODULES: THIS BUT NOT THAT

Don't ever do this!

```
from sklearn import *
```

This pollutes your namespace and may cause clashes with other modules you have imported.

WHAT'S THE CONTEXT? FILE I/O AND CONTEXT MANAGERS

As Statisticians, Data Scientists, whatever, we use a lot of files. We do the following operations on files:

- open them
- close them (this is important)
- read from them
- write to them

Context managers make sure that we close whatever we open, and only allow access within a `with` block.

IN WHAT CONTEXT? FILE I/O AND CONTEXT MANAGERS

Files, and other things we will talk about in a bit, can be opened in different modes.

- `r` open text file for reading
- `rb` open binary file for reading
- `w` open text file for writing (overwrite)
- `wb` open binary file for writing (overwrite)
- `a` open a text file for *appending* (no overwrite)
- `ab` open a binary file for *appending* (no overwrite)

IN WHAT CONTEXT? FILE I/O AND CONTEXT MANAGERS

There are several other modes, that open a file for reading and writing and their file pointer and overwrite behavior varies. These are left as an exercise.

`r+` `rb+` `w+` `wb+` `a+` `ab+`

There is also mode `x` for writing, creating a new file if it does not exist and raising an exception if it does exist.

IN WHAT CONTEXT? FILE I/O AND CONTEXT MANAGERS

An example of reading a text CSV file:

```
with open("data.csv", "r") as myfile:
    for line in myfile:
        fields = line.split(",")
        do_something(fields)
```

Note that we specify a file handle using `as`. The file is closed and not available outside of the `with` clause.

IN WHAT CONTEXT? FILE I/O AND CONTEXT MANAGERS

An example of writing a text CSV file from a list of lists:

```
lol = [["California", "Clinton], ["Nevada", "Clinton"],
       ["Texas", "Trump"], ["Arizona", "Trump"]]
with open("newdata.csv", "w") as myoutfile:
    for record in lol:
        myoutfile.write(','.join(record))
```

Note that we specify a file handle using `as`. The file is closed and not available outside of the `with` clause.

IN WHAT CONTEXT? FILE I/O AND CONTEXT MANAGERS

Files are not the only things that can be opened, closed, written to and read from. Context managers can be used for a variety of other things as well

- network connections
- database connections
- locks

And anything else that should be protected from leaking.

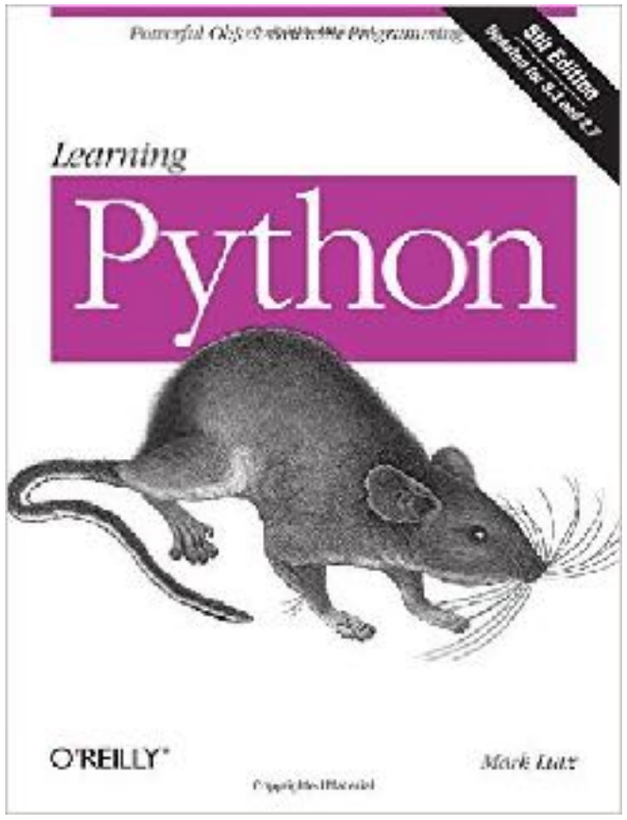
A PATH TO ENLIGHTENMENT

We covered the basics. There are several other important concepts:

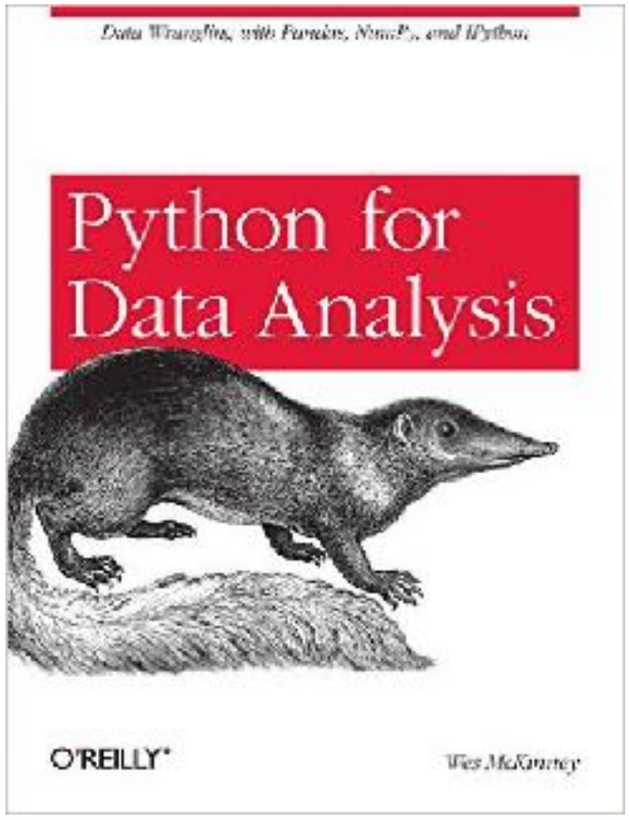
- classes
- `Unicode` and `bytes` (foreign characters)
- exceptions
- saving objects as JSON and `pickle` (like .Rdata)
- iterators and generators
- convenience functions (too many!)
- high performance computing (`Cython` and friends)

A PATH TO ENLIGHTENMENT

There are several great books that you may want to check out:



Learning Python
Mark Lutz



Python for Data Analysis
Wes McKinney

A PATH TO ENLIGHTENMENT

Dive into Python

http://www.diveintopython.net/

Learn Python the Hard Way

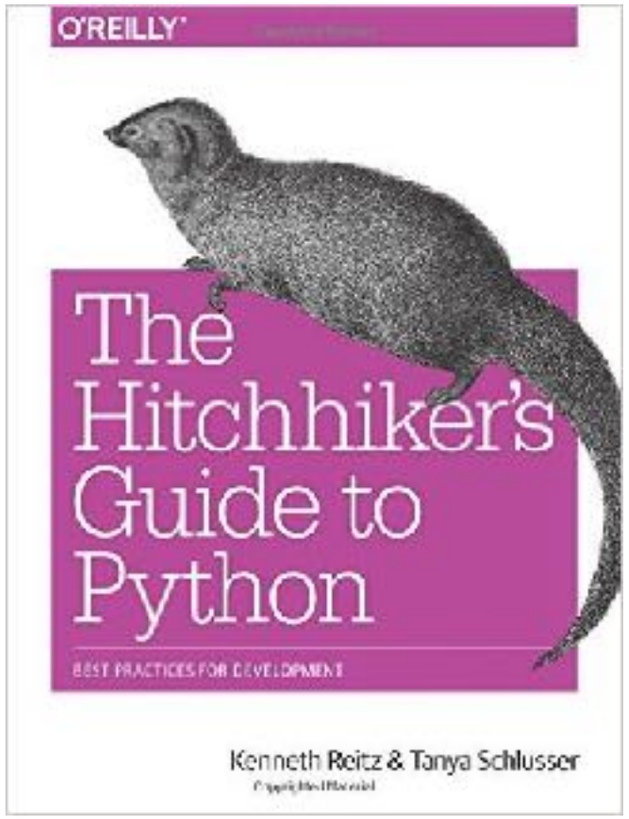
https://learnpythonthehardway.org/

The Python Documentation

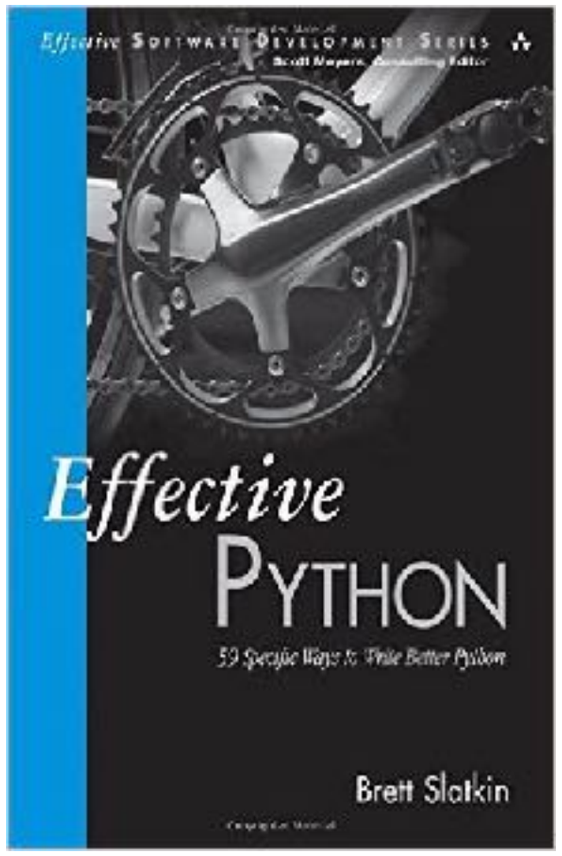
https://docs.python.org/

A PATH TO ENLIGHTENMENT

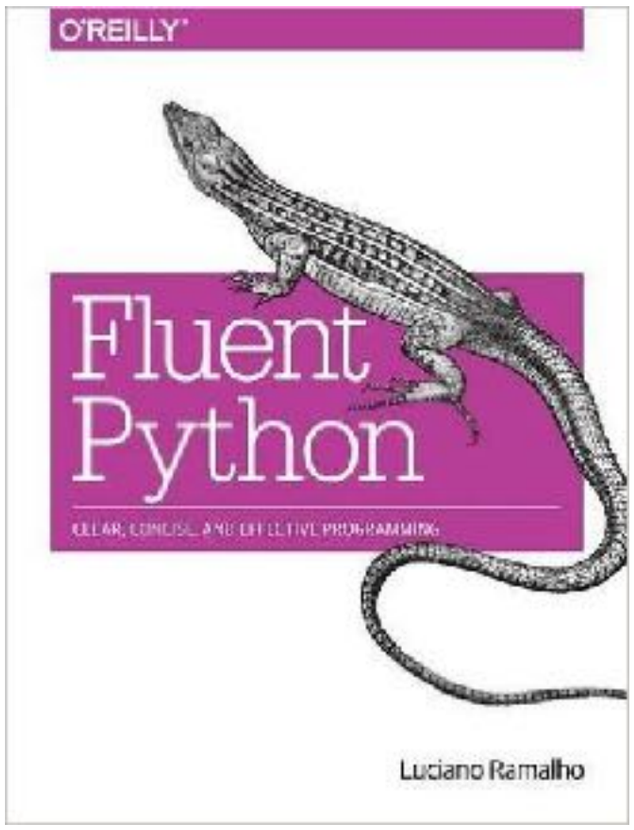
Once you're familiar, take your coding to the next level:



The Hitchhiker's Guide to Python
Reitz and Schlusser

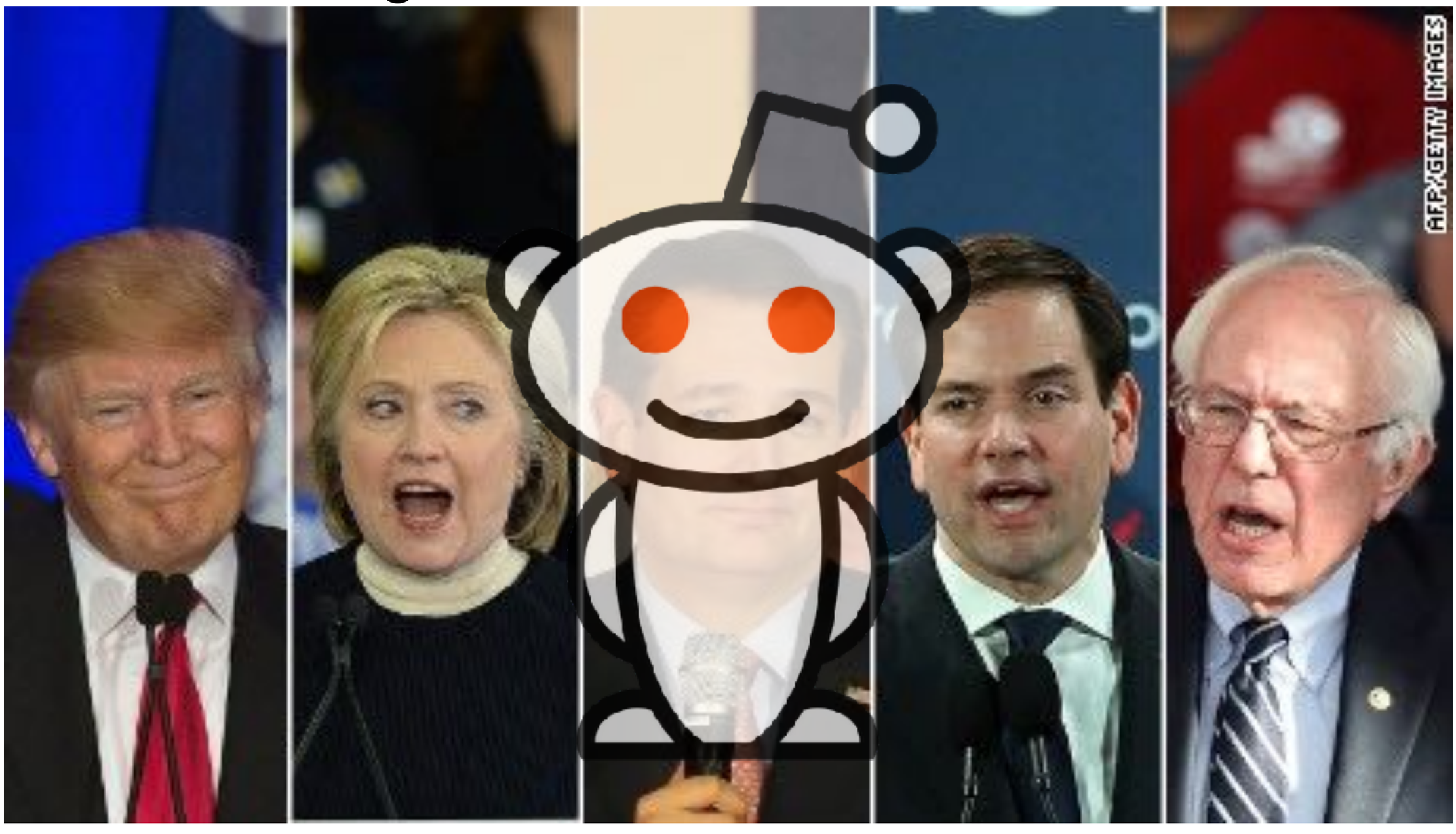


Effective Python
Brett Slatkin



Fluent Python
Luciano Ramalho

Examining 2016 Candidate Sentiment



PART III

PyData Demo

PYDATA

PyData is the data science stack for Python:

- `numpy` provides *N* dimensional arrays for native vectorized computation (like in R).
- `scipy` provides specific functionality more for pure science and engineering.
- `pandas` provides data frame support (like R) for data management and processing.
- `matplotlib` is a powerful visualization framework for Python. `seaborn` is a wrapper around matplotlib.
- `scikit-learn (sklearn)` for machine learning.

PYDATA

PyData is the data science stack for Python:

- `IPython` provides a much improved interpreter shell for Python including better formatting, more interactive features and magic commands.
- Jupyter Notebook provides a worksheet-like web interface for interactive data analysis, similar to RStudio and KnitR.
- There are many other packages as well.

PYDATA

For more information, check out the community website:

http://pydata.org/

But now it's demo time!

PYDATA

DEMO

HINDSIGHT IS 20/20

There are several things I wanted to get to, but I am only human, so I will instead provide resources since I am a perfectionist:

- Using this data in Spark.
- Python / High Performance Computing
- Scraping Websites
- A Proper Treatment of Regular Expressions

If you are interested in web scraping (you should be!), check out these resources

Excellent Tutorial by Jake Austwick

Covers requests, lxml, XPath, CSS Selectors

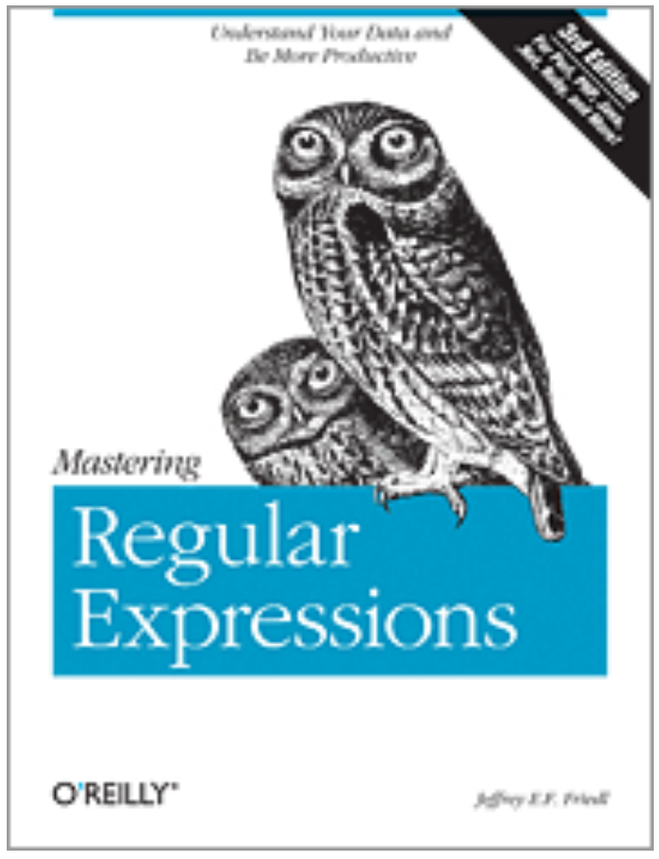
`http://jakeaustwick.me/python-web-scraping-resource/`

`BeautifulSoup` (HTML Parser) Tutorials

`http://web.stanford.edu/~zlotnick/TextAsData/Web_Scraping_with_Beautiful_Soup.html`

`http://www.pythonforbeginners.com/python-on-the-web/web-scraping-with-beautifulsoup/`

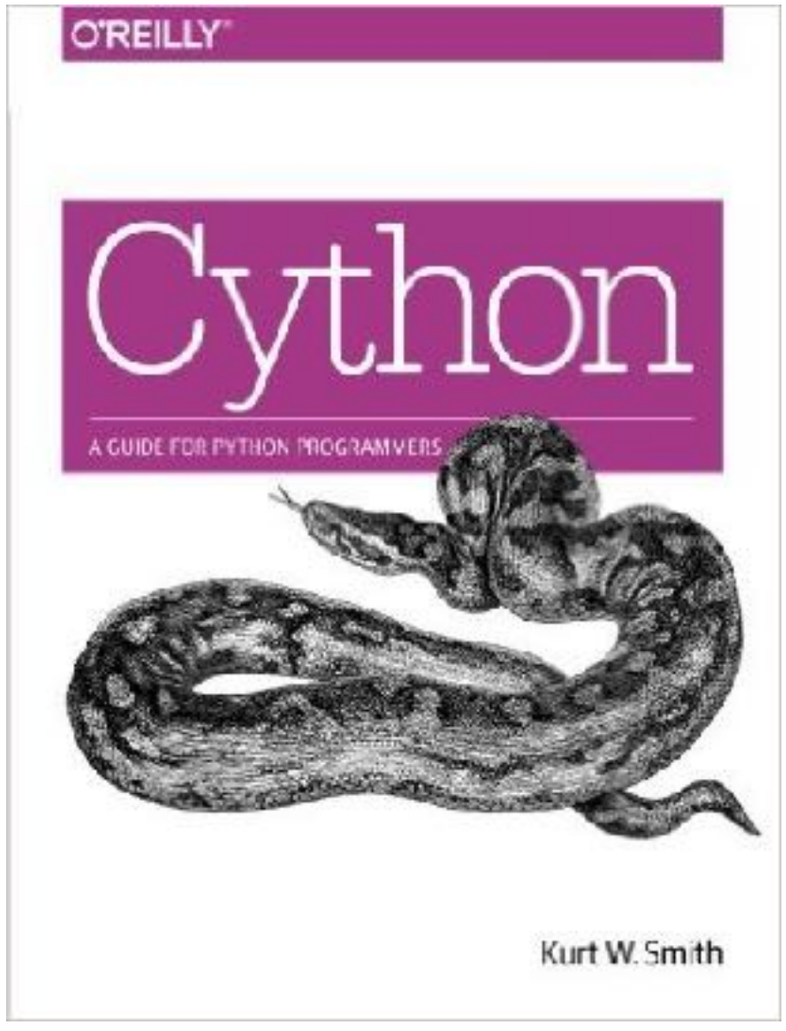
HINDSIGHT IS 20/20



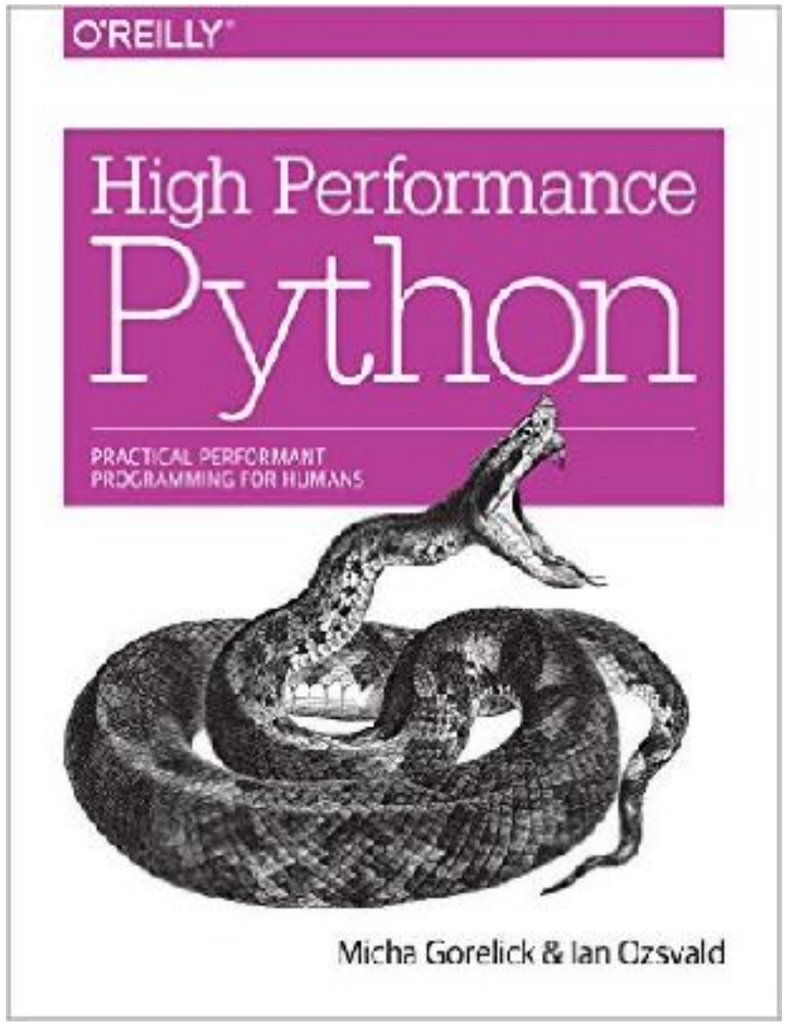
Mastering Regular Expressions
Jeff Friedl

Great website: http://www.regularexpressions.info

HINDSIGHT IS 20/20



Cython
Kurt Smith



High Performance Python
Gorelick and Ozsvald

HINDSIGHT IS 20/20

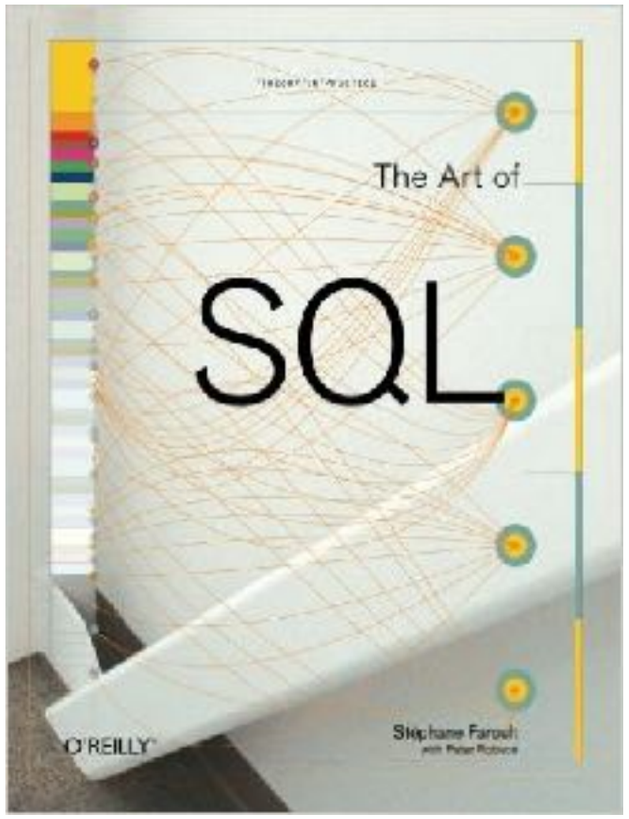
There is one other language I did not cover that is very important, but does not fit into this talk:

SQL

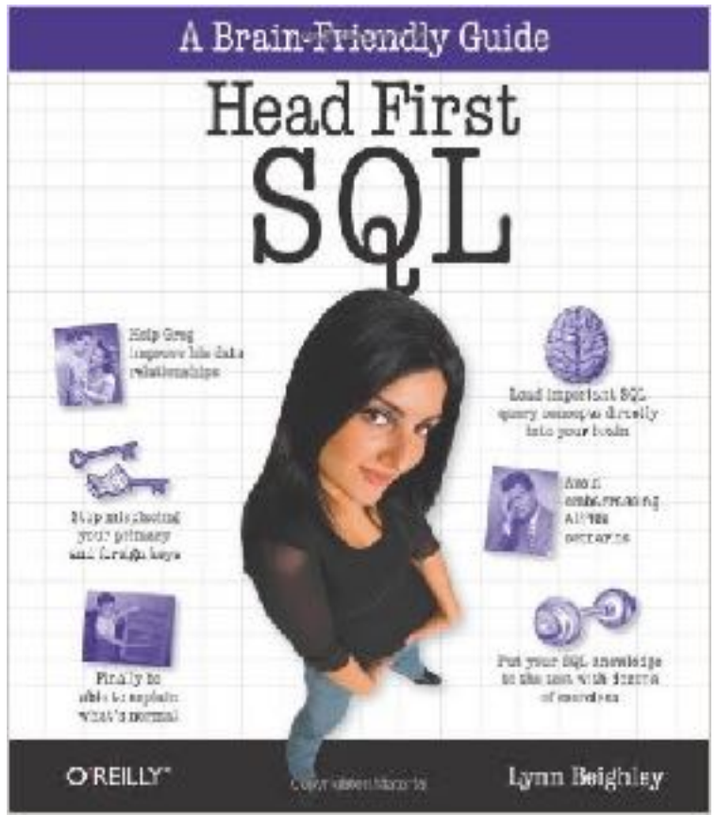
Learn it. You will need to know it. Every company has a database and you will need to pull data from it.

HINDSIGHT IS 20/20

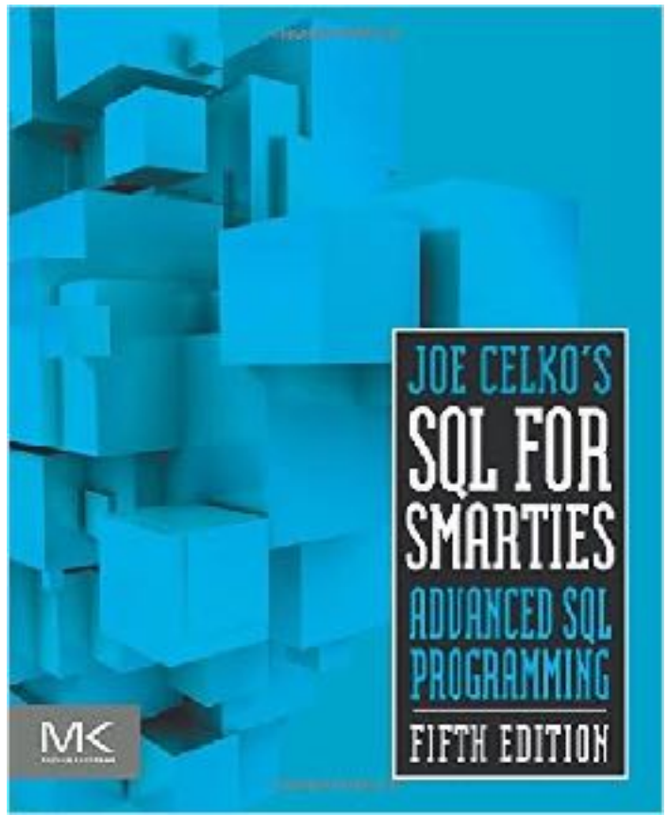
For SQL:



The Art of SQL
Faroult & Robson



Head First SQL
Lynn Beighley



SQL For Smarties
Joe Celko

PART IV
Data Science Career Tips

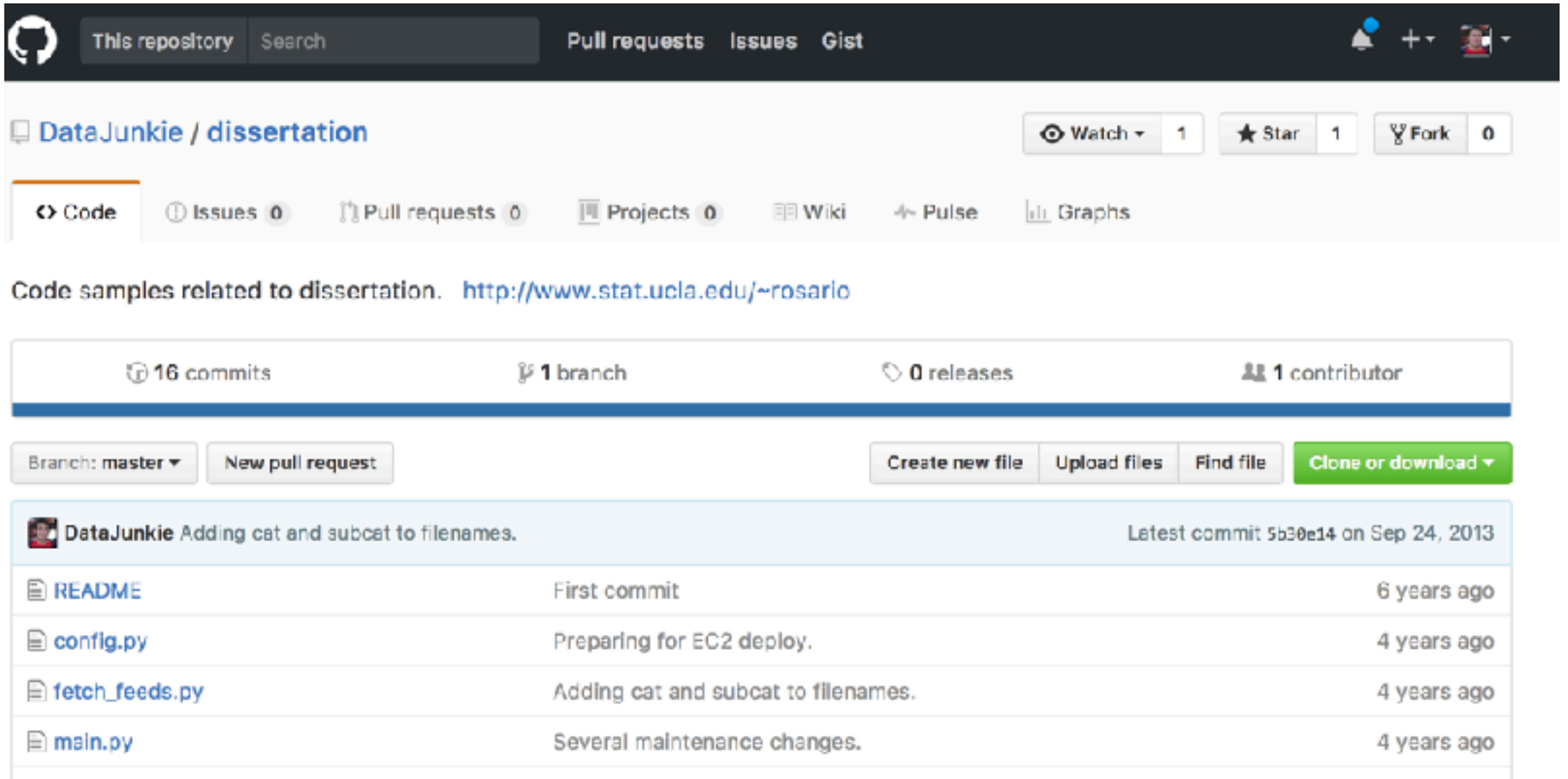
LET'S MEETUP

Los Angeles has several Data Science related meetups. Centralized as DataScience.LA

- R
- Machine Learning
- Data Science
- Python
- Hadoop
- Visualization

GITHUB: DON'T JUST DO, SHARE

Put your best, polished code on GitHub. You can also include Jupyter Notebooks and GitHub Pages (like blog posts) of your projects. **Employers see this.**



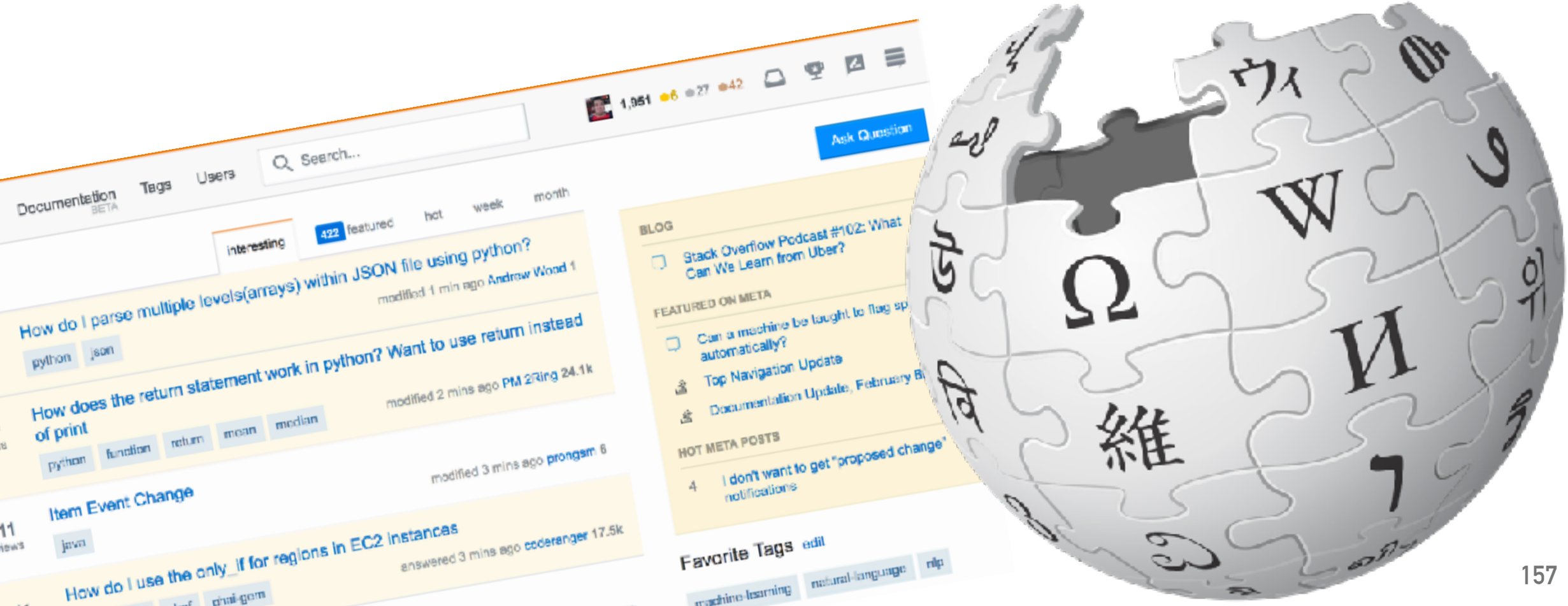
BLOGS AND TWITTER

Read, read, read. Learn what others are doing. Write comments. Tweet. Start your own blog.
This is how I got all of my jobs.

IS A STACK OVERFLOW GOOD NOW?

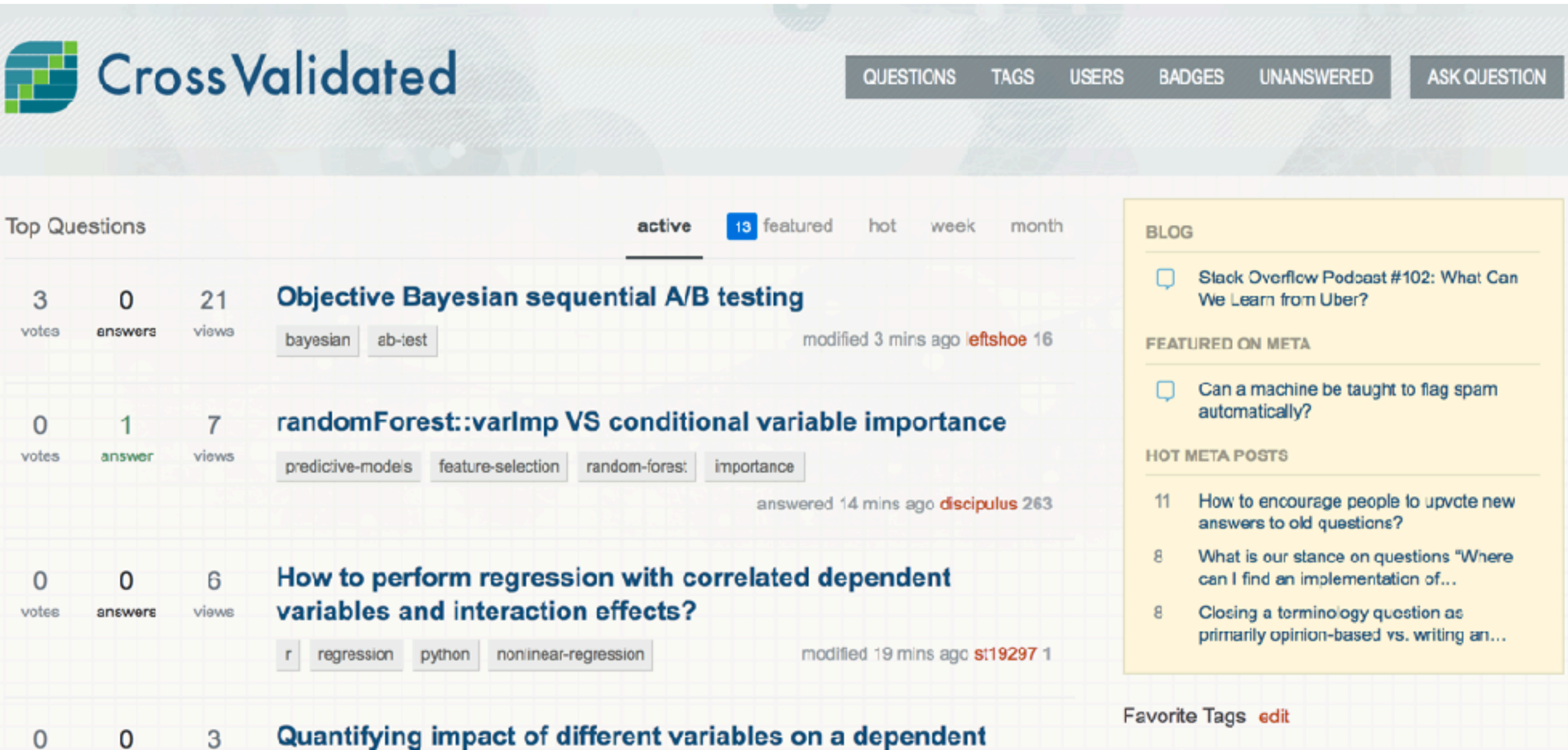
For some reason, I've heard some people say that they thought using StackOverflow was cheating. **In industry, we expect you to use Google, StackOverflow and Wikipedia.** I spent 95% of my day on these sites!

** Of course, in academia, Academic Integrity rules apply, so cite your sources.



CROSS-VALIDATION IS GOOD

There is a StackExchange for Statistics and Machine Learning. It's called CrossValidated.



IMPOSTER SYNDROME

Imposter syndrome is rampant in this field, and in many technical fields.

You are not an imposter, you are just learning.

Know what you don't know, but don't be intimidated!

SELF DOUBT IS GOOD, IT ENCOURAGES IMPROVEMENT

In this field, we analyze data. Many of us analyze ourselves and our work… to death. This is GREAT, but don't let it become **fear**.

- Ask yourself "why" with all of your analyses, conclusions, and assumptions.
- Always ask yourself if there is anything else you should do/ should have done (differently).
- Always ask yourself "what if…?"
- Always look for and fix holes in your work or thought process.

This yields ironclad arguments, or more work, which then yields ironclad arguments.

ALWAYS TELL A STORY

In this field, always practice telling a story with your data. Don't just give plots and numbers. You need the narrative to inspire the client.

This engages your audience, your employer, and shows your worth.

Communication is even more important than your data and technical skills.

Conclusion

MY HOPE

The goal of this talk was to introduce several tools, aside from the usual R and C, that are important in Applied Statistics and Data Science.

My hope is that the content and demos inspire you to learn more about GNU tools, the open-source software ecosystem, Python and other programming languages and tools.

CONTACT ME

Ryan R. Rosario
rosario@stat.ucla.edu
ryan@bytemining.com

Blog http://www.bytemining.com

Twitter
http://www.twitter.com/datajunkie
@DataJunkie

Bruin Forever!

Thank You!

Q&A