

Python, Polars, and a Bit of SH

Network Security and Measurement

Python

A Programming Language

- Dynamically-typed and interpreted programming language
- Need to know:
 - Indentation is part of the syntax (there are less braces here)
 - There are types, you don't need to write them, but you can annotate your code!
- Executing the python binary without arguments gives you a REPL!
- Docs: <https://docs.python.org/3.10/>

PythonCode

Count occurrences of random numbers

```
import random

from collections import defaultdict

lst = []

for i in range(0, 10000):
    lst.append(random.randint(1, 100))

lst = [x for x in lst if x % 2 == 0]

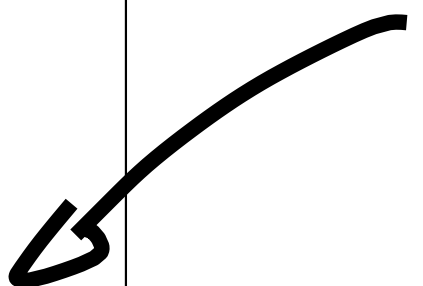
def dict_initializer():
    return 0

dct = defaultdict(dict_initializer)
for elem in lst:
    dct[elem] += 1

lst = list(dct.items())
lst.sort(key=lambda x: x[1], reverse=True)

for num, count in lst[:3]:
    print(f'{num}: {count}')
```

Generate some numbers



Filter odd numbers



Count occurrences



Sort from most to least



Print top three



Running Python

- Jupyter Notebooks
 - Interactive environments for incremental development
 - There is a HTML document in shared-data of the example
- Virtual environments: isolation for projects

```
# Create a virtual environment
python3 -m venv my_env
source my_env/bin/activate

# Install requirements
pip install matplotlib pandas

# Do your thing
python my_script.py

# Leave the environment
deactivate
```


Data Analysis with Polars

Polars

- Built around DataFrames and Series
- Allows efficient processing of large datasets, written in Rust
- Frequently used libraries have an import name by convention, here: pl

```
import polars as pl
```

- Docs: <https://docs.pola.rs/#example>

Read CSV into DataFrame

- `pd.read_csv(FILENAME, ...)`
- Options configure the format, such as:
 - Name and select columns
 - Specify the column separator
 - Read data in chunks (etc.)

shape: (10, 4)

	port	count
---	---	---
i64	i64	i64
0	80	1131992
1	8080	878982

```
df = pl.read_csv('shared-data/ports.csv', separator='|')
print(df)
```

Table Operations

- Columns can be selected via the `select` function which returns a Series
- Both types offer a variety of operations, such as sum, mean, etc.
- Use table and series operations when possible, they are performant!
- Avoid simple iterations of rows or columns with for loops
- Polars, internally, determines the most efficient way to execute a command using its query optimizer

```
total = df.select('count').sum()
df = df.with_columns(share=(pl.col('count')
    / total * 100).round(2))
```

shape: (10, 4)

	port	count	share
---	---	---	---
i64	i64	i64	f64
0	80	1131992	24.4

Pandas

- You can also use Pandas if you are already familiar with it
- Also built around DataFrames and Series
- Allows efficient processing of large datasets
- Frequently used libraries have an import name by convention, here: pd

```
import pandas as pd
```

- Docs: <https://pandas.pydata.org/docs/>

Read CSV into DataFrame (Pandas)

- `pd.read_csv(FILENAME, ...)`
- Options configure the format, such as:
 - Name and select columns
 - Specify the column separator
 - Read data in chunks (etc.)

```
df = pd.read_csv('shared-data/ports.csv', sep='|', index_col=0)
print(df)
```

	port	count
0	80	1131992
1	8080	878982

Table Operations (Pandas)

- Columns can be selected via the name which returns a Series
- Both types offer a variety of operations, such as sum, mean, etc.
- Use table and series operations when possible, they are performant!
- Avoid simple iterations of rows or columns with for loops

```
total = df['count'].sum()  
df['share'] = (df['count'] / total * 100).round(2)
```

Polars vs. Pandas

- Better performance for large data sets, more efficient memory usage
- Pandas has a broad ecosystem integration, e.g., for data visualization
- Consider data processing and data visualization as two separate steps
- Easy transformation between the two via Polars API or common file formats

```
df_pl = pl.from_pandas(df_pd)
df_pd = df_pl.to_pandas()
```

- Further reading: <https://www.statology.org/pandas-vs-polars-performance-benchmarks-for-common-data-operations/>

Diving Deeper into Performance

- Important characteristics: processing time & memory usage
- When in doubt, time it yourself
- Further reading:
 - <https://medium.com/bigdataarepublic/advanced-pandas-optimize-speed-and-memory-a654b53be6c2>
 - <https://stackoverflow.com/questions/52673285/performance-of-pandas-apply-vs-np-vectorize-to-create-new-column-from-existing-c/52674448#52674448>

Plotting with Pandas

Matplotlib

- DataFrames can plot via matplotlib (import matplotlib.pyplot as plt)

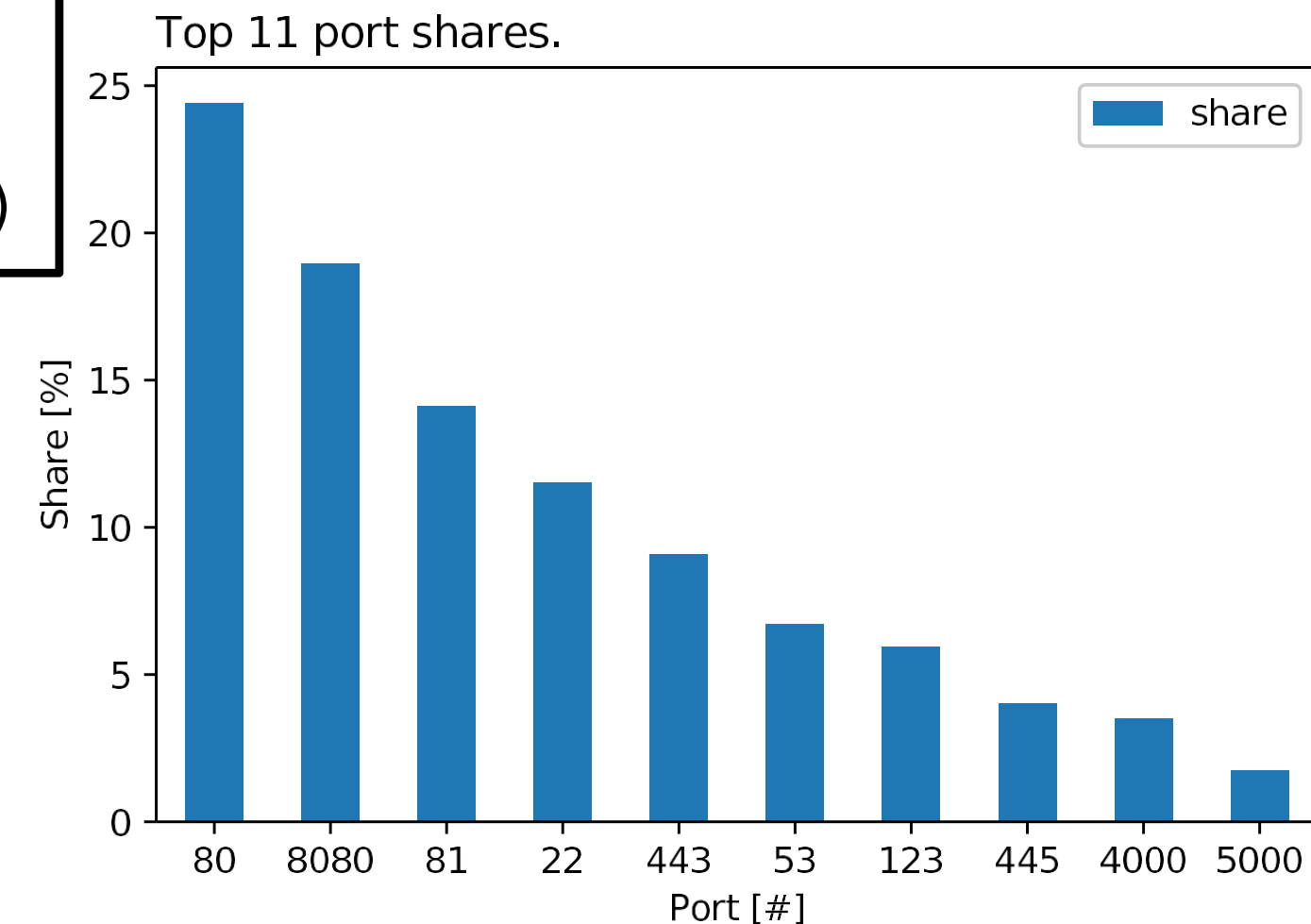
```
ax = df.plot.bar(x='port', y='share', rot=0)
```

- Configure your plot (always name the axis)

```
ax.set_xlabel('Port [#]')  
ax.set_ylabel('Share [%]')  
ax.set_title('Top 11 port shares.', loc='left')
```

- Show plots or save them

```
plt.show()  
fig = ax.get_figure()  
fig.savefig("ports.pdf", bbox_inches='tight')  
plt.close(fig)
```



Helpful DataFrame Functions (Polars)

- **df.sort()** – sorting a dataframe after a column
- **df.group_by()** – Group DataFrame using a mapper or by a Series of columns
- **df.filter()** – Filter values of a DataFrame
- **df.value_counts()** - Return a Series with the frequency of each distinct row in df.
- **df.group_by()** - Group DataFrame using a mapper or by a Series of columns.
 - Often combined with "aggregator" functions like mean(), sum(), etc.
 - Check the polars docs for detailed descriptions and examples:
<https://docs.pola.rs/api/python/stable/reference/index.html>

```
df.filter(pl.col('share') > 10).group_by('share').len().  
sort('len', descending=True)
```

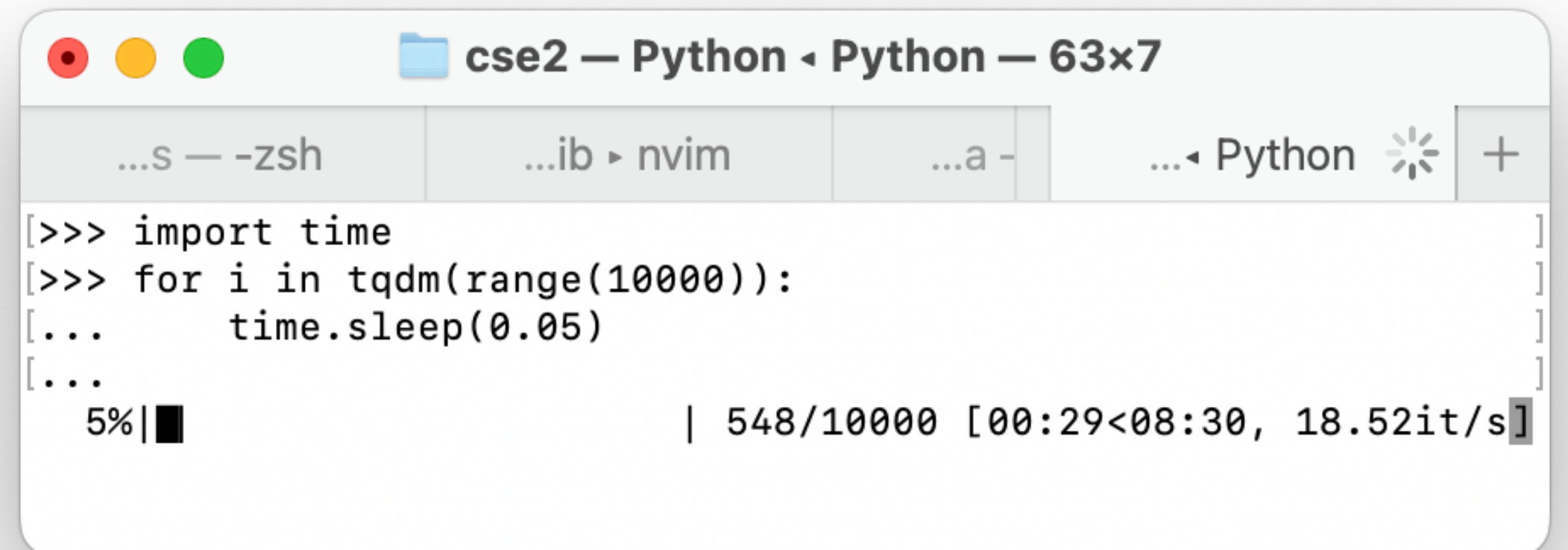
Helpful DataFrame Functions (Pandas)

- **df.value_counts()** - Return a Series containing the frequency of each distinct row in the DataFrame.
- **df.groupby()** - Group DataFrame using a mapper or by a Series of columns.
 - Often combined with "aggregator" functions like `mean()`, `sum()`, etc.
 - You might need to restore your index after operations (or use `as_index=False`)
 - You can also group timestamps by frequency with `pd.Grouper()`
- Check the pandas docs for detailed descriptions and examples:
<https://pandas.pydata.org/pandas-docs/stable/reference/index.html#api>

An Easy-to-use Progress Bar

- Some measurements run for minutes, hours or days
- The tqdm module shows a progress bar for iterations
- Simply wrap your iterable, such as a range:

```
from tqdm import tqdm
for i in tqdm(range(10000)) :
    ...
```



The screenshot shows a terminal window titled "cse2 — Python — Python — 63x7". The terminal contains the following Python code:

```
[>>> import time
[>>> for i in tqdm(range(10000)):
[...     time.sleep(0.05)
[...     ]
```

Below the code, a tqdm progress bar is displayed. It shows "5%" completion with a black bar. To the right of the bar, the text "| 548/10000 [00:29<08:30, 18.52it/s]" indicates the current progress, estimated time remaining, and speed.

- Check <https://tqdm.github.io/> for details

Shell Tools

Thanks to Marcin and Clemens from FU Berlin for the slides!

UNIX tools are great!

Some key advantages:

chaining using pipe

processing of files larger than your RAM

easily parallelizable

How do you find how UNIX tools work?

1. `<tool> --help`

```
> $ cat --help
Usage: cat [OPTION]... [FILE]...
Concatenate FILE(s) to standard output.

With no FILE, or when FILE is -, read standard input.

-A, --show-all          equivalent to -vET
-b, --number-nonblank    number nonempty output lines, overrides -n
-e                      equivalent to -vE
.....
```

2. `man <tool>`

CAT(1)	User Commands	CAT(1)
NAME	cat - concatenate files and print on the standard output	
DESCRIPTION	Concatenate FILE(s) to standard output. With no FILE, or when FILE is -, read standard input. -A, --show-all equivalent to -vET -b, --number-nonblank number nonempty output lines, overrides -n	

3. Search Google

UNIX: What does my data look like?

- zcat / cat
- less
- wc
- gzip / gunzip
- grep
- cut
- sort
- uniq

zcat / cat [OPTION]... [FILE]...

cat takes a list of files and pastes their content to stdout.
zcat is for compressed files.

```
> $ ls

another_cool_test_file  cool_test_file

> $ cat cool_test_file another_cool_test_file

This is line number 1.
Some hacky lines in between.
Some hacky lines in between.
This is the last line of this wonderful file.
More lines.
Unix is user-friendly. It's just picky about who its friends are.
End of Story
```

less is more

less is used to navigate and view large files easily.

- **page down / up** to navigate
- **q** to quit
- **/** to search
- **j,k** to go up and down line by line
- **G** to go to bottom
- **gg** to go to top
- **h** for help

wc [OPTION]... [FILE]...

wc counts lines, words and bytes of a file.

```
> $ cat cool_test_file

This is line number 1.
Some hacky lines in between.
Some hacky lines in between.
This is the last line of this wonderful file.

> $ wc cool_test_file

 4   24 127 cool_test_file
```


gzip / gunzip [OPTION]... [FILE]..

gzip and **gunzip** compresses and decompresses files. The data for provided on your assignment is compressed. You can either use **zcat** or **gunzip**.

```
> $ ls  
  
another_cool_test_file  cool_test_file  
  
> $ gzip another_cool_test_file  
  
> $ ls  
  
another_cool_test_file.gz  cool_test_file
```

grep [OPTION]... PATTERNS [FILE]...

grep finds lines that contain a specific pattern.

```
> $ cat cool_test_file

This is line number 1.
Some hacky lines in between.
Some hacky lines in between.
This is the last line of this wonderful file.

> $ grep "is" cool_test_file

This is line number 1.
This is the last line of this wonderful file.
```

cut OPTION... [FILE]...

cut is used to select specific parts of lines, e.g., columns.

*Select columns **1,4** from file **sample_updates**. The delimiter is **'|'**.*

```
> $ cat sample_updates
```

```
U|A|1579046400|2001:200:0:fe00::6249:0
U|W|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
```

```
> $ cut -f 1,4 -d '|' sample_updates
```

```
U|2001:200:0:fe00::6249:0
U|202.249.2.185
U|202.249.2.185
U|202.249.2.185
U|202.249.2.185
U|202.249.2.185
U|202.249.2.185
U|202.249.2.185
U|202.249.2.185
U|202.249.2.185
U|202.249.2.185
U|202.249.2.185
```

sort [OPTION]... [FILE]...

sort sorts

Select file `sample_updates2` by `column3`. Columns are separated by '|'.
Sort numbers.

```
> $ cat sample_updates2
```

```
U|W|1579046447|2001:200:0:fe00::12a9:0
U|W|1579046453|2001:200:0:fe00::12a9:0
U|A|1579046400|2001:200:0:fe00::6249:0
U|A|1579046400|2001:200:0:fe00::6249:0
U|A|1579046418|202.249.2.169
U|A|1579046402|202.249.2.185
U|A|1579046447|202.249.2.185
U|A|1579046447|202.249.2.185
U|A|1579046447|2001:200:0:fe00::12a9:0
U|A|1579046400|2001:200:0:fe00::6249:0
U|A|1579046447|202.249.2.185
U|W|1579046400|202.249.2.185
```

```
> $ sort -n -k 3,3 -t '|' sample_updates2
```

```
U|A|1579046400|2001:200:0:fe00::6249:0
U|A|1579046400|2001:200:0:fe00::6249:0
U|A|1579046400|2001:200:0:fe00::6249:0
U|W|1579046400|202.249.2.185
U|A|1579046402|202.249.2.185
U|A|1579046418|202.249.2.169
U|A|1579046447|2001:200:0:fe00::12a9:0
U|A|1579046447|202.249.2.185
U|A|1579046447|202.249.2.185
U|A|1579046447|202.249.2.185
U|W|1579046447|2001:200:0:fe00::12a9:0
U|W|1579046453|2001:200:0:fe00::12a9:0
```


uniq [OPTION]... [INPUT [OUTPUT]]

uniq removes duplicate rows from a file.

uniq -c also counts how often each line occurred.

```
> $ cat sample_updates
```

```
U|A|1579046400|2001:200:0:fe00::6249:0
U|W|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
```

```
> $ uniq sample_updates
```

```
U|A|1579046400|2001:200:0:fe00::6249:0
U|W|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
```

```
> $ uniq -c sample_updates
```

```
1 U|A|1579046400 2001:200:0:fe00::6249:0
1 U|W|1579046400|202.249.2.185
8 U|A|1579046400|202.249.2.185
```

The output of these tools goes to stdout.
Let's save the results!

> and >>

> overwrites an existing file or creates a new file with the supplied text.

>> appends to a file if it exists. Otherwise like >.

```
> $ cat sample_updates
```

```
U|A|1579046400|2001:200:0:fe00::6249:0
U|W|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
```

```
> $ uniq sample_updates > test_output
```

```
> $ cat test_output
```

```
U|A|1579046400|2001:200:0:fe00::6249:0
U|W|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
```

```
> $ uniq sample_updates >> test_output
```

```
> $ cat test_output
```

```
U|A|1579046400|2001:200:0:fe00::6249:0
U|W|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
U|A|1579046400|2001:200:0:fe00::6249:0
U|W|1579046400|202.249.2.185
U|A|1579046400|202.249.2.185
```

Let's chain these tools!



The mighty Pipe

The mighty pipe **takes stdout** from one program and **pipes it into stdin** of another program.

How often does each IP occur in this file?

```
> $ cat sample_updates2
```

```
U|W|1579046447|2001:200:0:fe00::12a9:0
U|W|1579046453|2001:200:0:fe00::12a9:0
U|A|1579046400|2001:200:0:fe00::6249:0
U|A|1579046400|2001:200:0:fe00::6249:0
U|A|1579046418|202.249.2.169
U|A|1579046402|202.249.2.185
U|A|1579046447|202.249.2.185
U|A|1579046447|202.249.2.185
U|A|1579046447|2001:200:0:fe00::12a9:0
U|A|1579046400|2001:200:0:fe00::6249:0
U|A|1579046447|202.249.2.185
U|W|1579046400|202.249.2.185
```

```
> $ cat sample_updates2 | cut -d '|' -f 4 | sort | uniq -c
```

```
3 2001:200:0:fe00::12a9:0
3 2001:200:0:fe00::6249:0
1 202.249.2.169
5 202.249.2.185
```

BREATHE!

You always need basic domain knowledge.

The best data science stack does not exist.

Data is too large to fit into memory.

Use UNIX tools to prep!

Data is noisy and from multiple sensors.

Data does not lie but
is sometimes biased.

Awesome! Let's look at a few more tools

Capturing Network Traffic

- **tshark** [-i INTERFACE] [-f FILTER] [-T fields -e FIELD_NAME -e ANOTHER_FIELD_NAME]
 - Captures traffic and prints field values in the shell
 - The outbound interface on mobi8 is **bond0**
- **tcpdump** [-i INTERFACE] [-w DST_FILE] [-r SRC_FILE] [FILTER]
 - Captures traffic
 - Writes and reads pcap files

DNS Lookups

- **dig** [OPTIONS] [@server] [name] [type] [class] [queryopt...]
 - Queries DNS records
 - Look for the ";; ANSWER SECTION:"
- **drill** [OPTIONS] name [@server] [type] [class]
 - *"The name drill is a pun on dig. With drill you should be able get even more information than with dig."*
 - Specifically designed to be used with DNSSEC, enabled with "-D"
 - From the man page: -S shows tree of signatures, -DT performs a trace

**Data processing and data visualization are
two separate steps.**