

Python, Pandas, and a Bit of SH

Network Security and Measurement

Python

A Programming Language

- Dynamically-typed and interpreted programming language
- Need to know:
 - Versions 2 and 3 are incompatible, use python3
 - Indentation is part of the syntax (there are less braces here)
 - There are types, but you don't write them in code
- Executing the python binary without arguments gives you a REPL!
- Docs: <https://docs.python.org/3.7/>

Python Code

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 Pandas

Pandas

- Build 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

- `pd.read_csv(FILENAME, ...)`
- There are options to configure the format, such as:
 - To name and select columns,
 - Specify the column separator, ...
 - Or, to read data in chunks.

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

	port	count
0	80	1131992
1	8080	878982

Table Operations

- 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)
```


More About Performance

- Most interesting: processing time and memory usage
- You can time operations yourself
- Two links on the topic
 - <https://medium.com/bigdatarepublic/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 in Pandas

Matplotlib

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

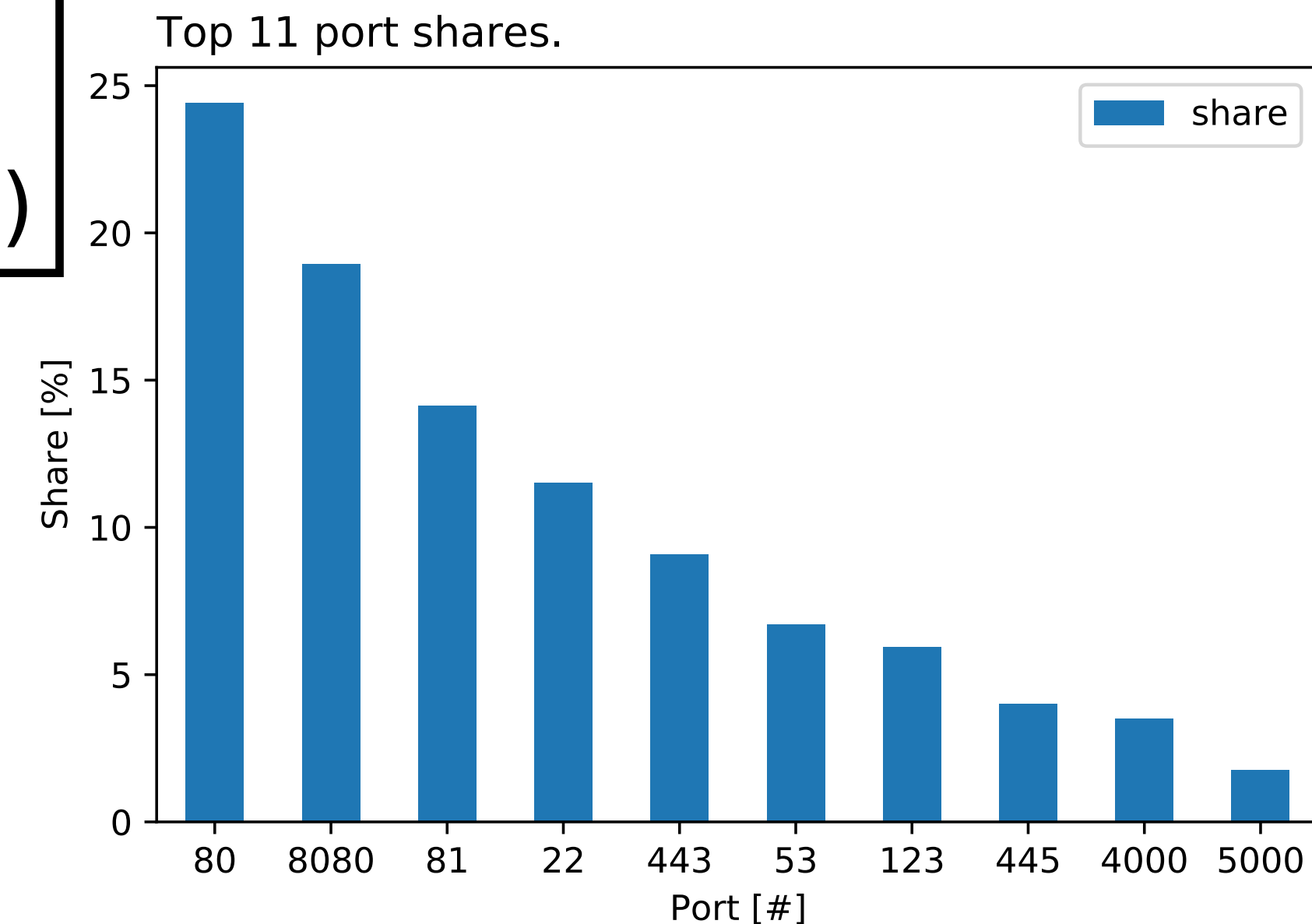
```
ax = df.plot.bar(x='port', y='share', rot=0)
```

- Plots can be configured a lot!

```
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)
```



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|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
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.**