Network Security and Measurement

Python, Polars, and a Bit of SH

Timo Salomon | timo.salomon@haw-hamburg.de
Yue Xin | yue.xin@haw-hamburg.de

Python

Python

- Dynamically-typed and interpreted programming language
- Need to know:
  - Indentation is part of the syntax (less braces)
  - There are types, you don’t need to write them, but you can annotate your code!
- Executing python without arguments gives you a REPL!
- Docs: https://docs.python.org/3.12/

Python Code Example

Count occurrences of random numbers

```
import random
from collections import defaultdict

lst = []
for i in range(0, 10000):
    lst.append(random.randint(1, 100))          # generate some numbers

lst = [x for x in lst if x % 2 == 0]            # filter odd numbers

def dict_initializer():
    return 0

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

lst = list(dct.items())
lst.sort(key=lambda x: x[1], reverse=True)      # sort from most to least

for num, count in lst[:3]:
    print(f'{num}: {count}')                    # 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 polars
# 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, multi-threaded by default
  - Lazy execution and query optimization
- Immutable DataFrames: every operation returns a new object, no .copy() needed; never worry about corrupting your data
- Frequently used libraries have an import name by convention
- Docs: : https://docs.pola.rs/#example

```
import polars as pl
```

Performance

- Libraries are usually many orders of magnitude faster than your code!
- group_by: df.groupby('column_name').agg(pl.sum('value')).collect()

- **Input table: 100,000,000 rows x 9 columns (5 GB)**

DataFrames.jl	1.1.1	2021-05-15	9s
Polars	0.8.8	2021-06-30	11s
DuckDB	0.2.7	2021-06-15	14s
data.table	1.14.1	2021-06-30	15s
cuDF*	0.19.2	2021-05-31	17s
ClickHouse	21.3.2.5	2021-05-12	18s
spark	3.1.2	2021-05-31	34s
pandas	1.2.5	2021-06-30	70s
(py)datatable	1.0.0a0	2021-06-30	75s
dask	2021.04.1	2021-05-09	170s
dplyr	1.0.7	2021-06-20	175s
Arrow	4.0.1	2021-05-31	212s
Modin		see README	pending

Input table: 1,000,000,000 rows x 9 columns (50 GB)

Polars	0.8.8	2021-06-30	143s
data.table	1.14.1	2021-06-30	155s
DataFrames.jl	1.1.1	2021-05-15	200s
ClickHouse	21.3.2.5	2021-05-12	256s
cuDF*	0.19.2	2021-05-31	492s
spark	3.1.2	2021-05-31	568s
(py)datatable	1.0.0a0	2021-06-30	730s
dplyr	1.0.7	2021-06-20	internal error
pandas	1.2.5	2021-06-30	out of memory
dask	2021.04.1	2021-05-09	out of memory
Arrow	4.0.1	2021-05-31	internal error
DuckDB*	0.2.7	2021-06-15	out of memory
Modin		see README	pending

https://h2oai.github.io/db-benchmark/

Demonstration in Jupyterhub shared-data/intro

Import Data

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

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

```
shape: (10, 4)
```

	port	count
i64	i64	i64
0	80	1131992
1	8080	878982

Checkout your Data

```
df.dtypes
```

```
[String, Int64, String, Int64]
```

```
df.columns
```

```
['name', 'age', 'major', 'income']
```

```
df.head(2)
```

shape: (2, 4)

name	age	major	income
str	i64	str	i64
"Alice"	17	"IN"	3000
"Bob"	43	"BWL"	2000

```
df['major'].value_counts(sort=True)
```

shape: (2, 2)

major	count
str	u32
"BWL"	2
"IN"	1

Table Operations: Select, Filter

- Columns can be selected via the select function which returns a Series
- Both types offer a variety of operations, such as sum, mean, etc.
- Polars, internally, determines the most efficient way to execute a command using its query optimizer

df:

name	age	major	income
str	i64	str	i64
"Alice"	17	"IN"	3000
"Bob"	43	"BWL"	2000
"Christ"	33	"BWL"	3800

```
df.filter((pl.col('name')=='Alice') | (pl.col('age')<40)).select('name')
```

name	major
str	str
"Alice"	"IN"
"Christ"	"BWL"

Add / Replace Columns

```
new_df=df.with_columns((pl.col('income')/1000).alias('income_in_K'))
```

name	age	major	income	income_in_K
str	i64	str	i64	f64
"Alice"	17	"IN"	3000	3.0
"Bob"	43	"BWL"	2000	2.0
"Christ"	33	"BWL"	3800	3.8

if the same name in alias(), the old column will be replaced

```
new_df=df.with_columns((pl.col('income')/1000).alias('income'))
```

name	age	major	income
str	i64	str	f64
"Alice"	17	"IN"	3.0
"Bob"	43	"BWL"	2.0
"Christ"	33	"BWL"	3.8

group_by, agg

df:

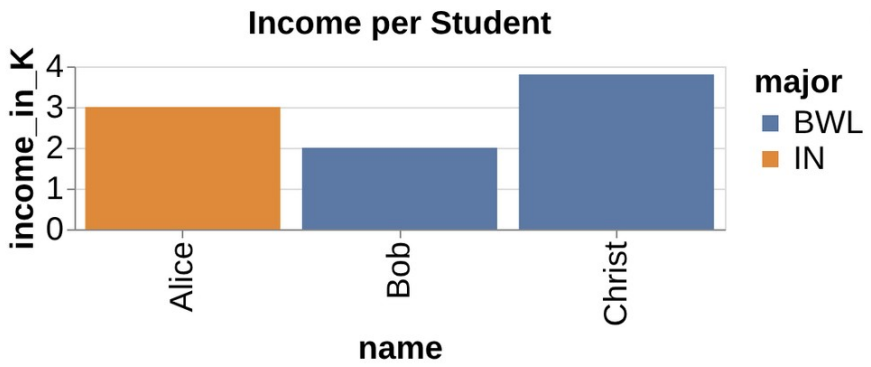
name	age	major	income
str	i64	str	i64
"Alice"	17	"IN"	3000
"Bob"	43	"BWL"	2000
"Christ"	33	"BWL"	3800

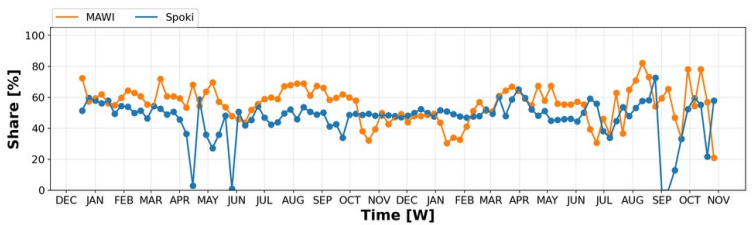
```
new_df.group_by("major").agg(
    pl.len().alias("count"),
    pl.col("age").mean().alias("avg_age"),
    pl.col("income").sum().alias("total_income"),
    pl.col("income").mean().alias("avg_income"),
)
```

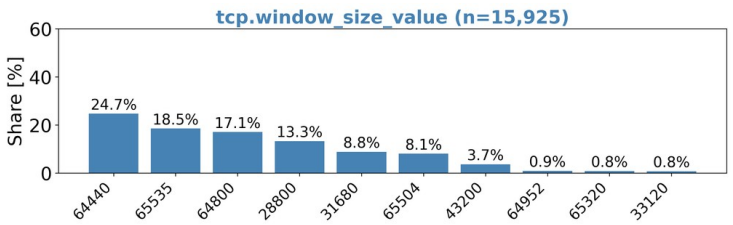
major	count	avg_age	total_income	avg_income
str	u32	f64	i64	f64
"IN"	1	17.0	3000	3000.0
"BWL"	2	38.0	5800	2900.0

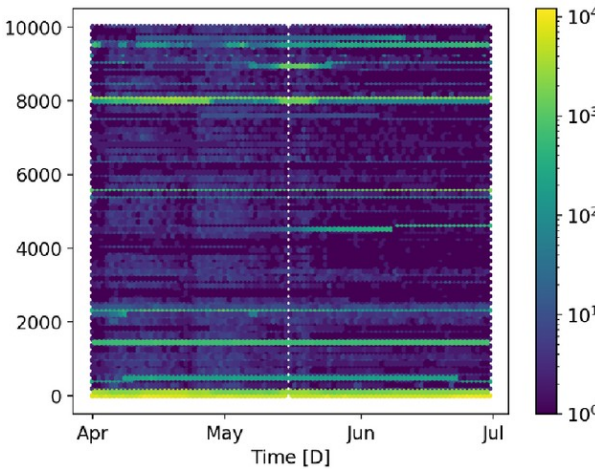
Plot

```python
import matplotlib.pyplot as plt
chart = new_df.plot.bar(
    x="name",
    y="income_in_K",
    color="major",
    tooltip=["name", "age", "major", "income_in_K"],
).properties(
    title="Income per Student",
    width=500,
    height=150,
)
```









Diving Deeper into Performance

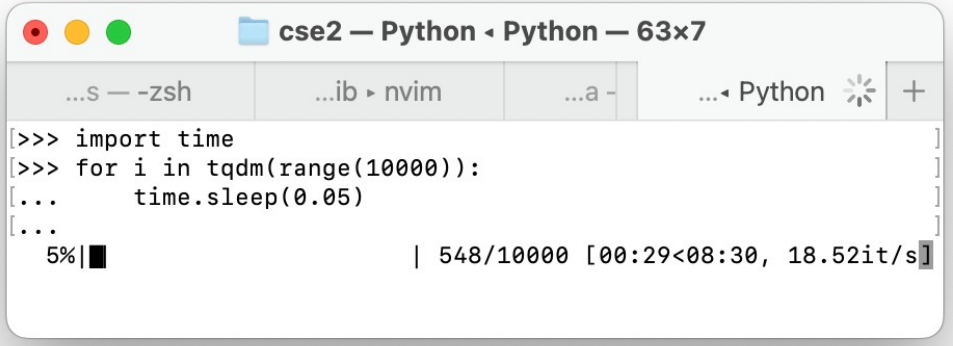
- Important characteristics: processing time & memory usage
- When in doubt, time it yourself
- Store aggregated data to reduce processing time
- Push common functionality into python modules
- Use libraries when possible, they are performant
- Avoid simple iterations of rows or columns with for loops

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)):
...
```

- Check https://tqdm.github.io/



Shell Tools

UNIX tools are great!
Some key advantages:
chaining using pipe processing of files larger than your RAM
easily parallelizable

How do you UNIX tools work?

1. `<tool> --help`
2. `man <tool>`
3. Search Google

Useful Tools

- zcat / cat — paste content of files to stdout
- less — navigate and view large files
- wc — count lines, words, bytes of a file
- gzip / gunzip — compress/uncompress
- grep — filter lines and search for patterns
- cut — select specific parts of lines, e.g., columns
- sort — sort numbers (also works with columns)
- uniq — remove duplicate rows

Work with output to stdout

- The output of these tools goes to stdout.
  - overwrites an existing file or creates a new file with the supplied text.
  - >> appends to a file if it exists. Otherwise like >.
- ‚|‘ The mighty pipe takes stdout from one program and pipes it into stdin of another program.

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

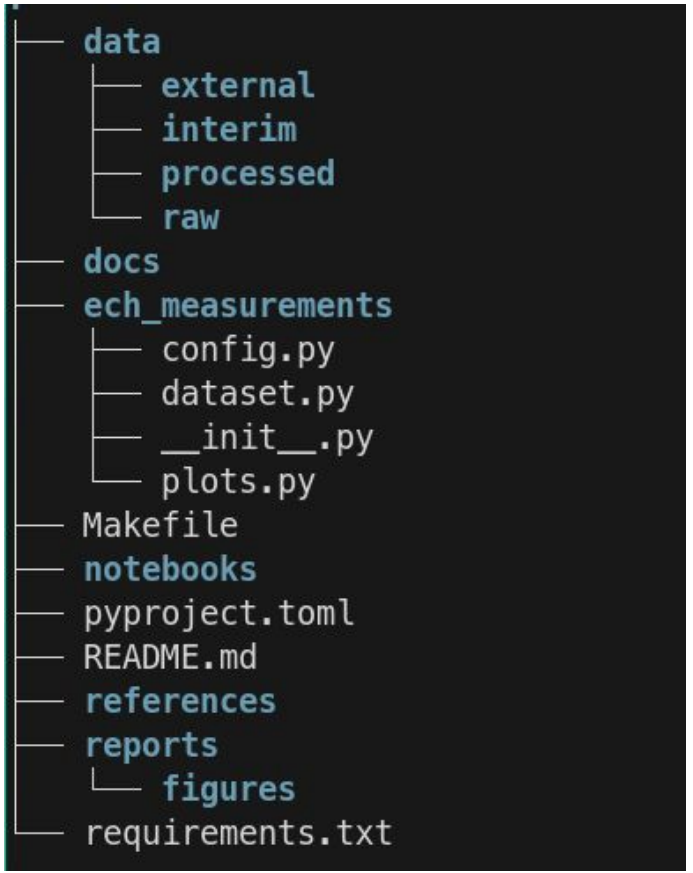
DNS Lookups

- 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 to get even more information than with dig."
  - Specifically designed to be used with DNSSEC, enabled with "-D"

Standardized Research Data Processing (SRDP)

Filesystem Hierarchy Standard



- Data sources should be marked with dates and original source
- Processed data can be stored
- Python modules for shared functions
- Makefile for running tasks and artifacts
- Jupyter notebooks for prototyping
- Export figures for reports

Cookiecutter

`pip install cookiecutter`

What is it?

- Creates a project folder structure from a template
- Fills in names, dates, author — via simple prompts
- Works with any language, not just Python
- Templates live on GitHub and are reusable

- Already installed on the jupyterhub
- Start a new project with ccds
- Answer prompts: project name, author, etc.
- Get a ready-to-code structure

Check: https://github.com/netd-tud/srdp.git and https://cookiecutter-data-science.drivendata.org

Example: https://github.com/netd-tud/artifacts-ipv6-sra-scanning.git

Data Science Remarks

- You always need basic domain knowledge
- Data is often to large to fit into memory
- Data may be noisy and from multiple sources
- Data does not lie, but is sometimes biased

- **Data collection, processing, and visualization are separate steps!**

Remarks to our Setup

- Jupyterhub at https://mobi8.inet.haw-hamburg.de/
- Your user is the HAW-Alias, e.g., www123
- Set your password on the first login
- „shared-data/“ contains common datasets for assignments
- You can exchange files via „pub/“
- The jupyterhub env is sometimes restrictive, e.g., when using packet sniffers such as tshark
  - shared-data/ssh-local.sh script to ssh into the system