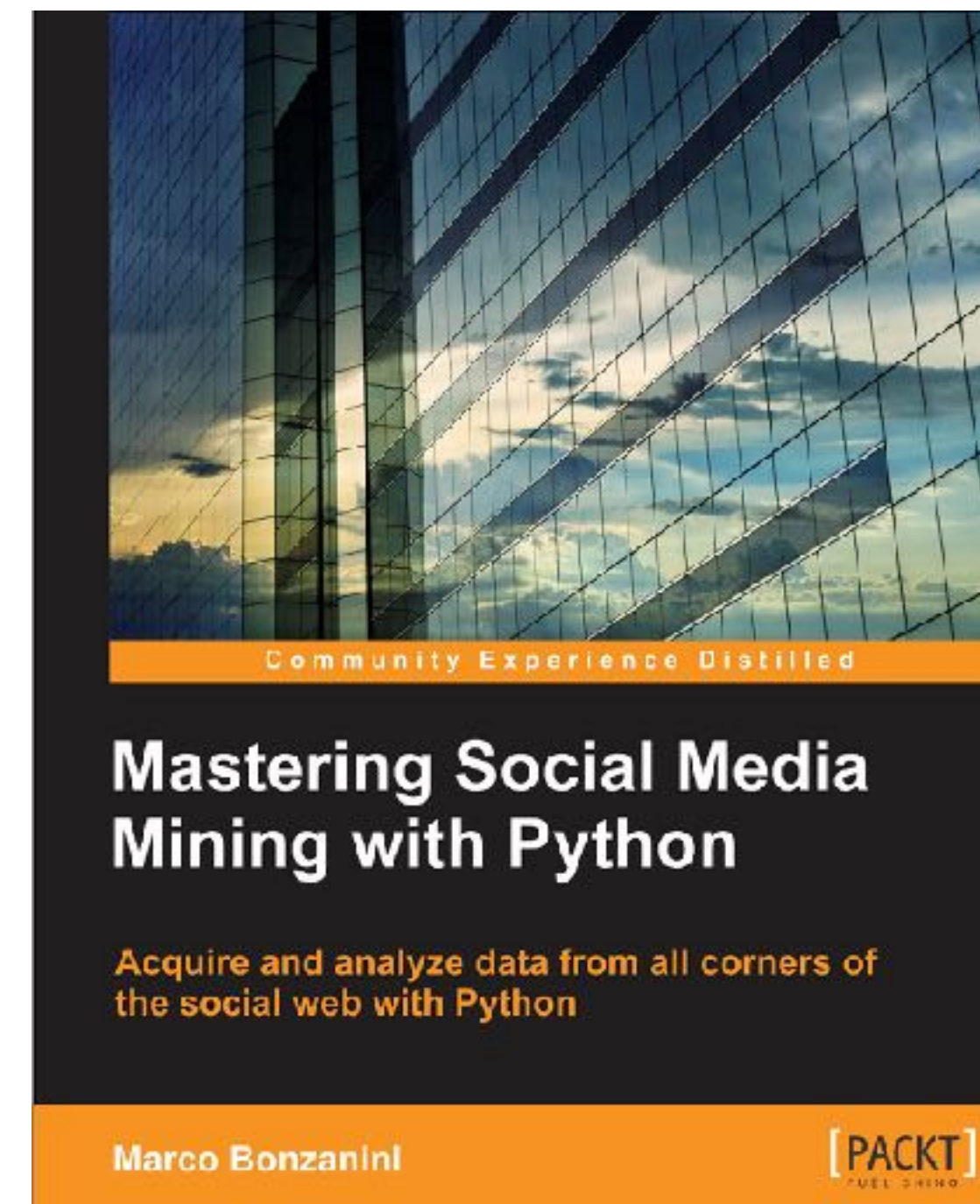


Building Data Pipelines in Python

Marco Bonzanini

QCon London 2017

Nice to meet you



R&D \neq Engineering

R&D \neq Engineering

R&D results in production = high value



Big Data Borat

@BigDataBorat



 Follow

In Data Science, 80% of time spent prepare data, 20% of time spent complain about need for prepare data.

RETWEETS

430

LIKES

173





Big Data Borat

@BigDataBorat



 Follow

In Data Science, 80% of time spent prepare data, 20% of time spent complain about need for prepare data.

RETWEETS

430

LIKES

173



Big Data Problems

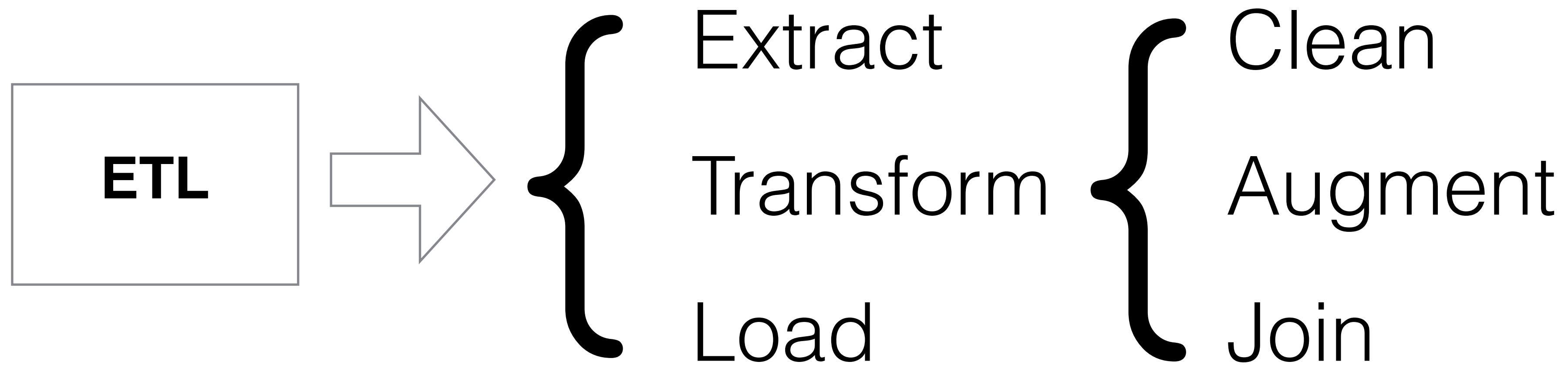
VS

Big Data Problems

Data Pipelines (from 30,000ft)



Data Pipelines (zooming in)



Good Data Pipelines

Easy to {
Reproduce
Productise

Towards Good Data Pipelines

Towards Good Data Pipelines (a)

Your Data is Dirty

unless proven otherwise

“It’s in the database, so it’s already good”

Towards Good Data Pipelines (b)

All Your Data is Important

unless proven otherwise

Towards Good Data Pipelines (b)

All Your Data is Important

unless proven otherwise

Keep it. Transform it. Don't overwrite it.

Towards Good Data Pipelines (c)

Pipelines vs Script Soups



Tasty, but not a pipeline

Anti-pattern: the script soup

```
$ ./do_something.sh
```

```
$ ./do_something_else.sh
```

```
$ ./extract_some_data.sh
```

```
$ ./join_some_other_data.sh
```

```
...
```


Script soups kill replicability

Anti-pattern: the master script

```
$ cat ./run_everything.sh  
./do_something.sh  
./do_something_else.sh  
./extract_some_data.sh  
./join_some_other_data.sh  
  
$ ./run_everything.sh
```

Towards Good Data Pipelines (d)

Break it Down

`setup.py` and `conda`

Towards Good Data Pipelines (e)

Automated Testing

i.e. why scientists don't write unit tests

Intermezzo



Let me rant about testing

(Unit) Testing

Unit tests in three easy steps:

- `import unittest`
- Write your tests
- Quit complaining about lack of time to write tests

Benefits of (unit) testing

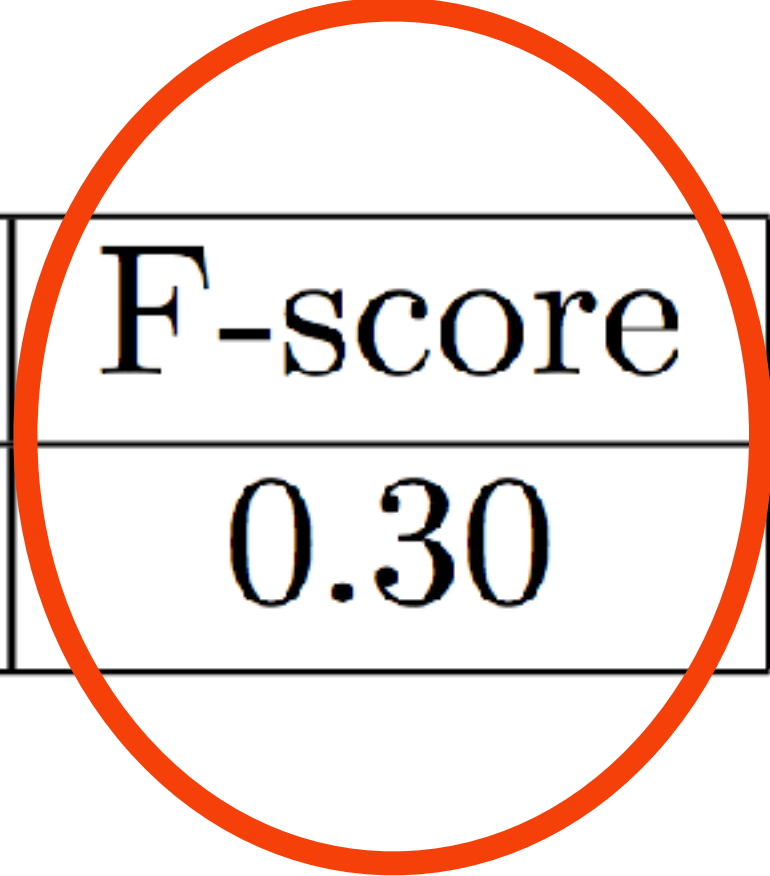
- Safety net for refactoring
- Safety net for lib upgrades
- Validate your assumptions
- Document code / communicate your intentions
- You're forced to think

Testing: not convinced yet?

Precision	Recall	F-score
0.34	0.31	0.30

Testing: not convinced yet?

Precision	Recall	F-score
0.34	0.31	0.30



Testing: not convinced yet?

Precision	Recall	F-score
0.34	0.31	0.30

```
f1 = fscore(p, r)
min_bound, max_bound = sorted([p, r])
assert min_bound <= f1 <= max_bound
```

Testing: I'm almost done

- Unit tests vs Defensive Programming
- Say no to tautologies
- Say no to vanity tests
- The Python ecosystem is rich:
py.test, nosetests, **hypothesis**, coverage.py, ...

</rant>

Towards Good Data Pipelines (f)

Orchestration

Don't re-invent the wheel

You need a workflow manager

Think:

GNU Make + Unix pipes + Steroids

Intro to Luigi

- Task dependency management
- Error control, checkpoints, failure recovery
- Minimal boilerplate
- Dependency graph visualisation

```
$ pip install luigi
```



Luigi Task: unit of execution

```
class MyTask(luigi.Task):  
    def requires(self):  
        return [SomeTask()]  
  
    def output(self):  
        return luigi.LocalTarget(...)  
  
    def run(self):  
        mylib.run()
```


Luigi Target: output of a task

```
class MyTarget (luigi.Target) :  
    def exists (self) :  
        ... # return bool
```

Great off the shelf support
local file system, S3, Elasticsearch, RDBMS
(also via `luigi.contrib`)

Intro to Airflow

- Like Luigi, just younger
- Nicer (?) GUI
- Scheduling
- Apache Project



Towards Good Data Pipelines (g)

When things go wrong

The Joy of debugging

import logging

Who reads the logs?

You're not going to read the logs, unless...

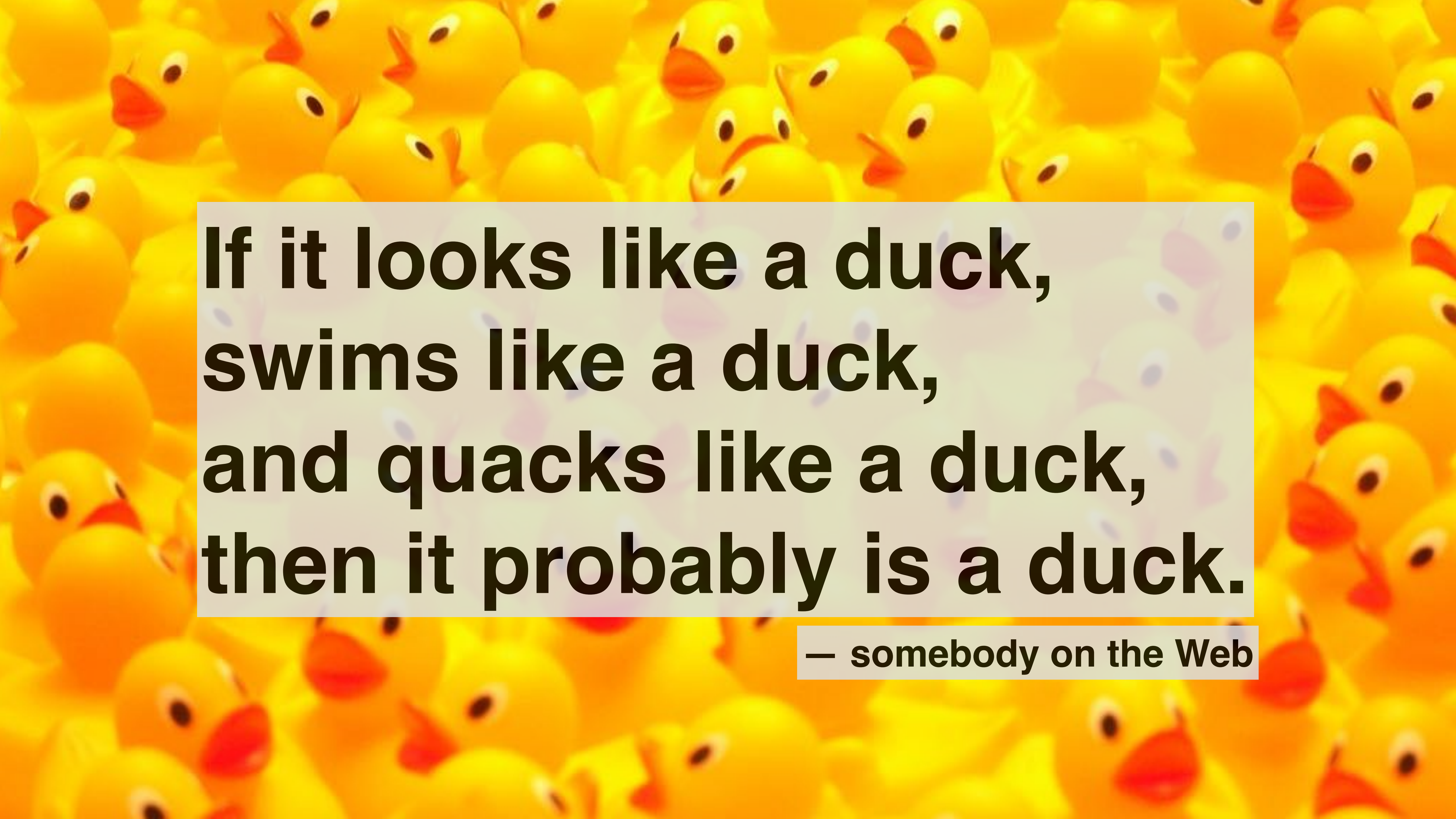
- E-mail notifications (built-in in Luigi)
- Slack notifications

```
$ pip install luigi_slack # WIP
```

Towards Good Data Pipelines (h)

Static Analysis

The Joy of Duck Typing



**If it looks like a duck,
swims like a duck,
and quacks like a duck,
then it probably is a duck.**

— somebody on the Web


```
>>> 1.0 == 1 == True
```

```
True
```

```
>>> 1 + True
```

```
2
```

```
>>> '1' * 2
```

```
'11'
```

```
>>> '1' + 2
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: Can't convert 'int' object  
to str implicitly
```

PEP 3107 — Function Annotations (since Python 3.0)

```
def do_stuff(a: int,  
             b: int) -> str:  
    ...  
    return something
```

(annotations are ignored by the interpreter)

PEP 484 — Type Hints (since Python 3.5)

`typing` **module: semantically coherent**

(still ignored by the interpreter)

```
pip install mypy
```

- Add optional types
- Run:
`mypy --follow-imports silent mylib`
- Refine gradual typing (e.g. `Any`)

Summary

Basic engineering principles help
(packaging, testing, orchestration, logging, static analysis, ...)

Summary

R&D is not Engineering:
can we meet halfway?

Vanity Slide

- speakerdeck.com/marcobonzanini
- github.com/bonzanini
- marcobonzanini.com
- @MarcoBonzanini