Python Automation - A Case Study

Maciej Skórski[1]

PyDays Vienna, Vienna

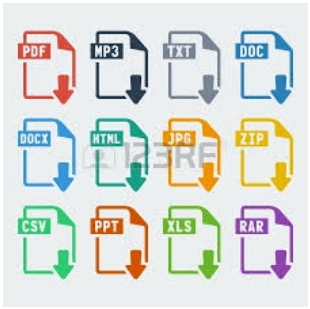
[1] IST Austria

Plan

Overview

Figure: A typical data-analytic project

Part I: Extract-Transform-Load



Part II: Data Analytics

A closer look at the ETL process

Why a self-made solution?

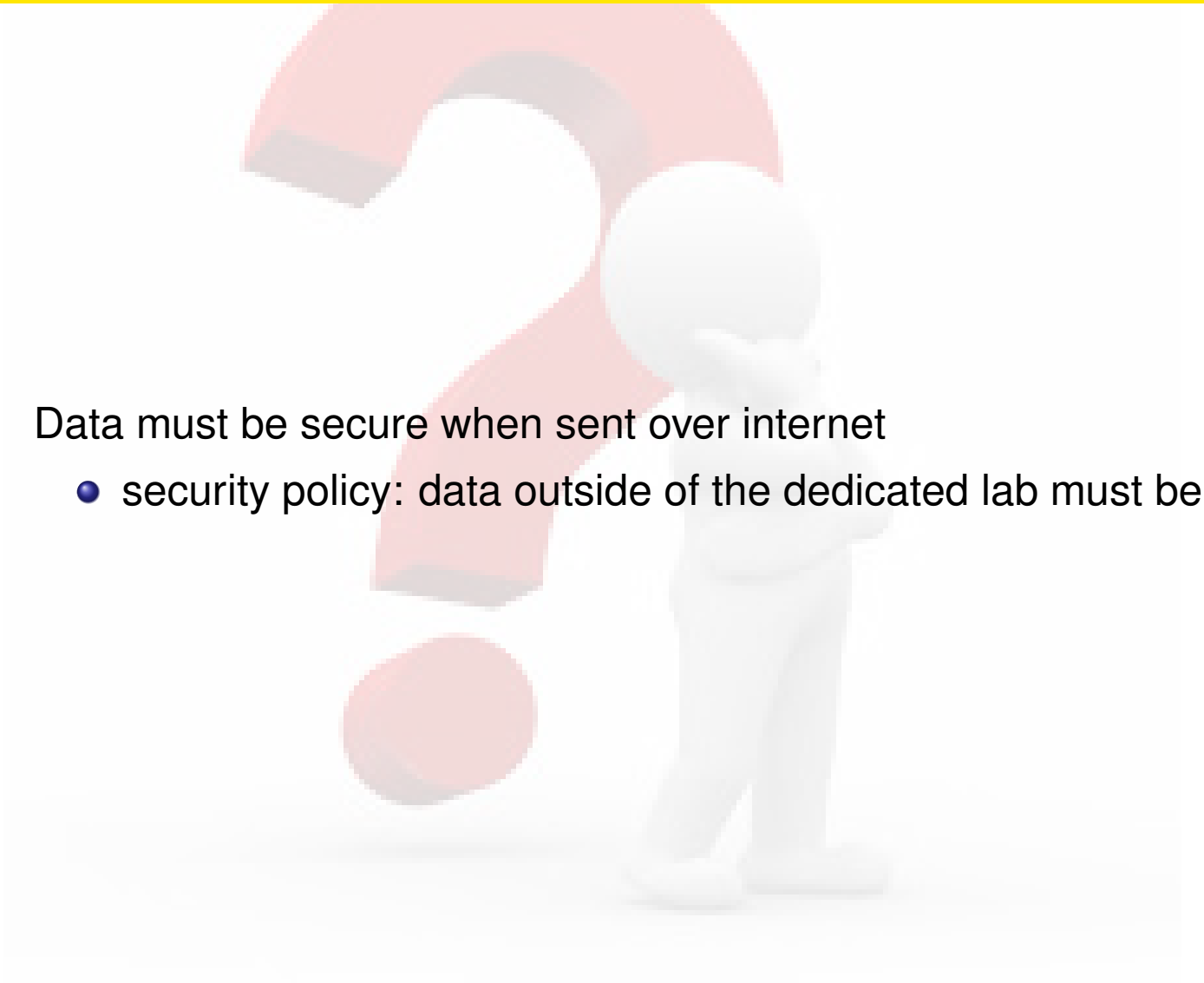
- tailored exactly to the project scope
- can be adapted to changes
- can add ad-hoc countermeasures against ”unpredictable” issues

Why python?

- Known to staff members $\Longrightarrow$ not dependent on one developer!
- Popular worldwide $\Longrightarrow$ can find more co-workers even with FDA background
- Interpreted language $\Longrightarrow$ quick fixes/changes on demand

Plan

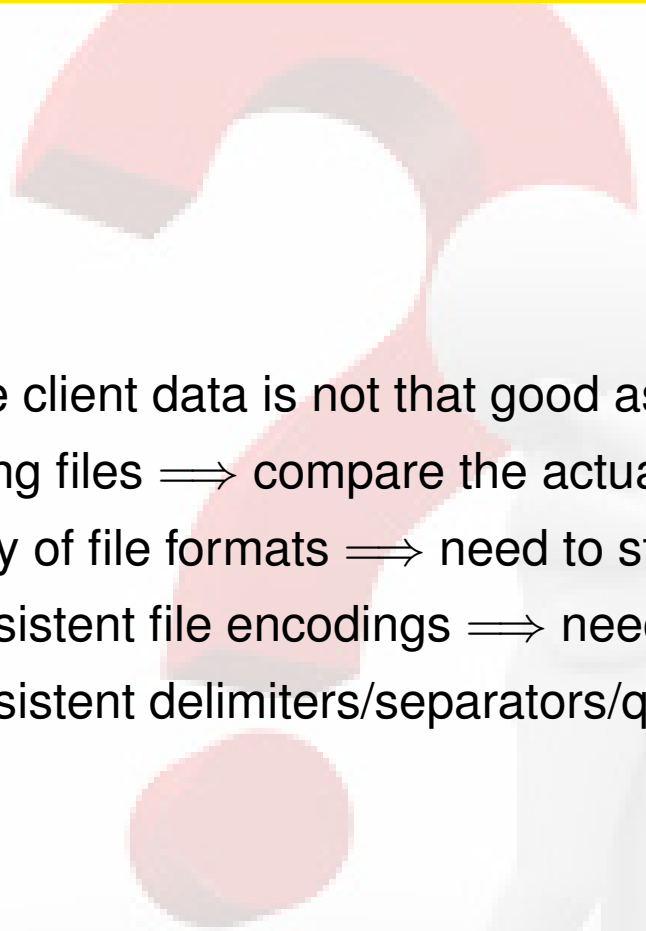
Guaranteeing privacy to client's data



Data must be secure when sent over internet

- security policy: data outside of the dedicated lab must be encrypted

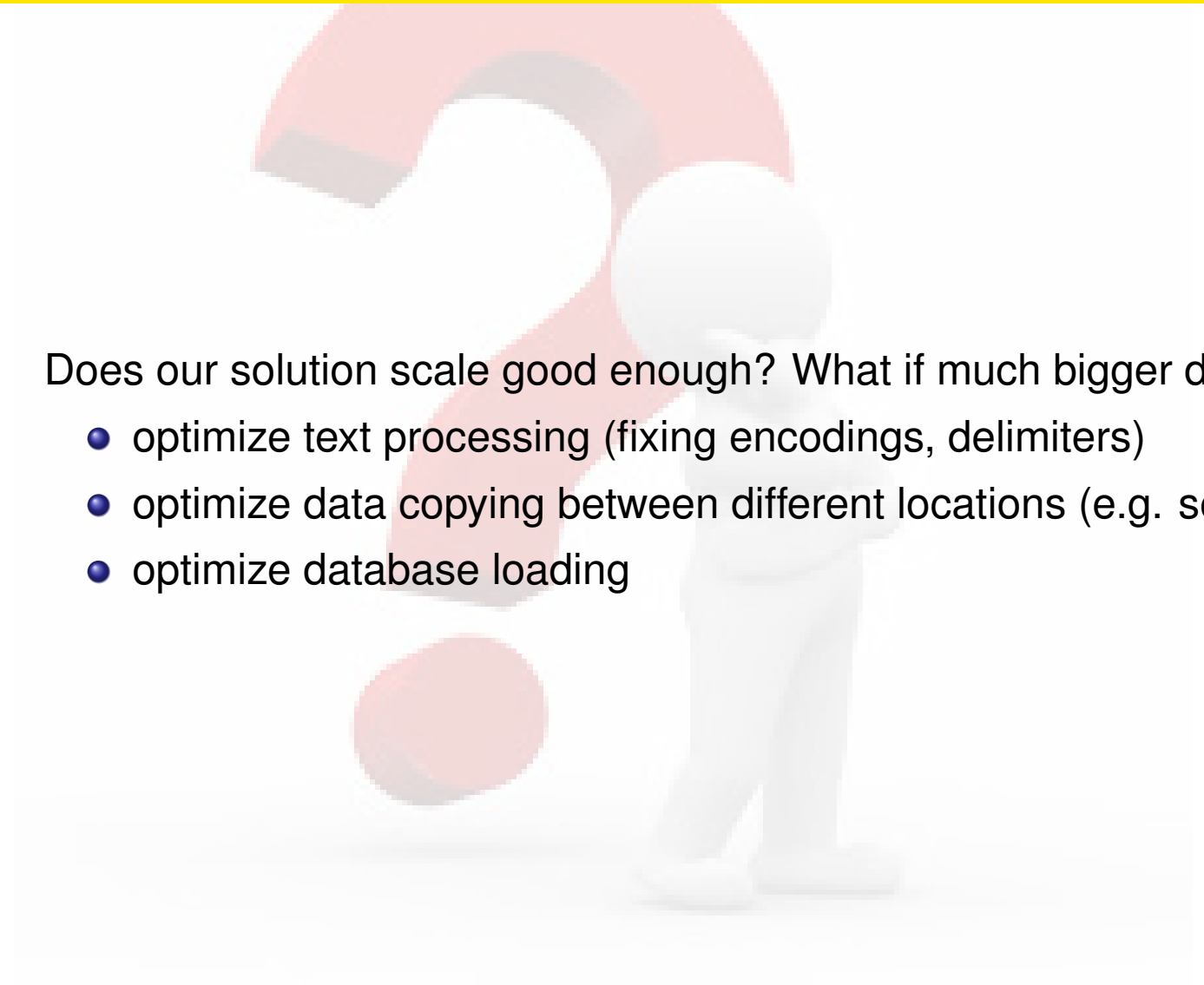
Processing unstructured data



What if the client data is not that good as promised?

- missing files $\Longrightarrow$ compare the actual data against the list
- variety of file formats $\Longrightarrow$ need to standardize before loading
- inconsistent file encodings $\Longrightarrow$ need to detect and fix or report to the client asap
- inconsistent delimiters/separators/quotations $\Longrightarrow$ detect and fix before loading?

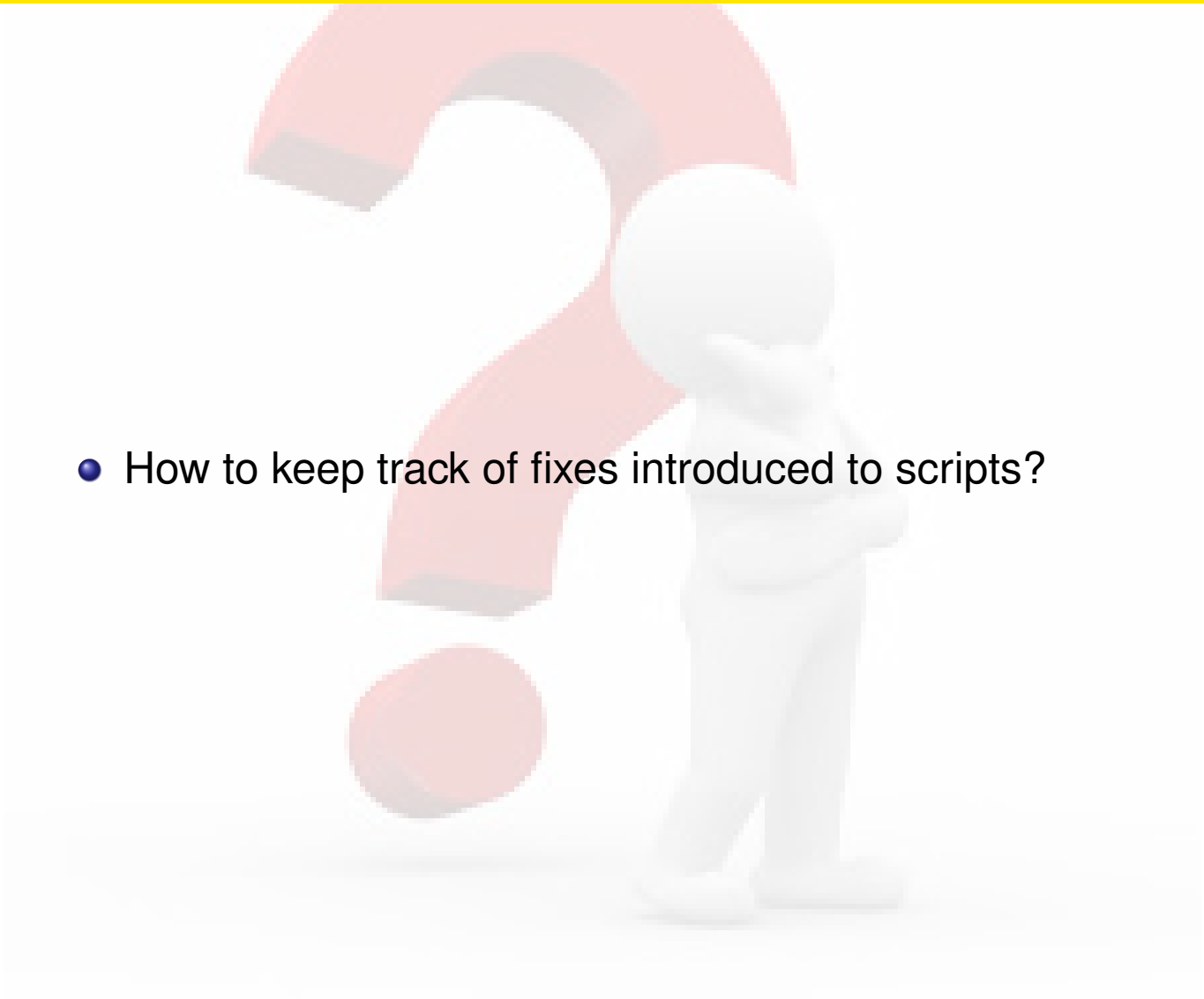
Optimizing pre-processing and loading



Does our solution scale good enough? What if much bigger data sets arrive?

- optimize text processing (fixing encodings, delimiters)
- optimize data copying between different locations (e.g. servers)
- optimize database loading

Code maintenance



- How to keep track of fixes introduced to scripts?

Plan

Learn structures by sampling

Tabular structures (fields, delimiters) may be learned from first few lines.

Example (How to automatically learn delimiters)

```
...
delims = ['|',",",";"]
with open('file') as f:
  # analyse the header
  line = f.readline()
  for c in delims:
    if line.split(c).count() > 1:
      delim = c
      fields = line.split(c).count()
  # compare with the second line
  for i in [1..10]:
    line = f.readline()
    if line.split(delim).count != fields:
      report_failure
```

Listing 1: learning delimiters from few first lines

Manipulating large strings

Implement efficient string manipulations.

Example (Replacing delimiters by a custom char)

- database loading may have insufficient support for custom (weird) csv files
- may need to standardised delimiters $\Longrightarrow$ parse and rewrite a lot of text

```
...
for line_in in open('file_in'):
  line_out = ''
  for word in line.split('|'):
    line_out = line_out + word + chr
        (30)
```

Listing 2: naive concatenating (slow)

```
...
for line_in in open('file_in'):
  line_out = chr(30).join( line.split('
      |') )
```

Listing 3: efficient concatenating (fast)

Regular expressions in file processing

Use regular expressions for clever searching/replacing.

Example (Replacing delimiters in a quoted csv file)

- replace separators, that may be also (quoted) text characters?
- need ”context” parsing $\Longrightarrow$ use regular expressions

```
import re
"""replace comma separators unless are quoted"""
...
r = '[^"]\\|[^"]' # look behind / look ahead
new_delim = '#'
for line_in in open('file_in'):
  line_out = re.sub(r, new_delim, line_in)
```

Listing 4: replacing not-quoted delimiters

Use regular expressions for clever searching/replacing.

Example (Replacing delimiters in a badly quoted csv file)

- understand the context of non-delimiter occurrences

firstname, lastname, payment
John, Smith, 1000,00
John, Snow, 2000.00

```
import re
"""replace comma delimiters unless appear as decimal separators  """
...
r = '(?<!\d),(?!\d{2})'
new_delim = '#'
for line_in in open('file_in'):
  line_out = re.sub(r, new_delim, line_in)
```

Listing 5: replacing badly quoted delimiters

Compile regular expressions

When matching a given pattern multiple times (e.g. in a loop over lines in a file) it's more efficient to compile the regular expression. See [Fou17].

Example (Checking if line breaks are correct)

- a csv file has in the last column either "true" or "false"
- want to do a quick check before loading into the database

```
import re
...
pattern  = "false|true$"
r = re.compile(pattern)
...
for line in open('myfile'):
  ...
        if r.match(line):
        report_corrupted_file
```

Countermeasures against wrong encodings

Files should not contain ”weird” characters when decoded properly .

Example (Fast reading with searching weird chars)

- client files often contain certain wrongly encoded characters
- can scan the file against the list of ”weird” chars

```
import codecs
..
weird_chars = [...]
with codecs.open('myfile', mode='rt', encoding='utf-8') as f:
  while True:
    # read a chunk of data.
    chunk = input.read(4096)
    if not chunk:
      break
    # check if characters seem to be correct
    for c in chunk:
      if (ord(c) > 255) || (c in weird_chars):
        report_weird_chars
```

Note: codecs library is faster [Nel15].

Parrallelization in Python is bit tricky

Workarounds to Python Global Interpreter Lock [Knu13]

- single instance of Python interpreter may be constrained by GIL..
- ...but can run multiple interpreters independently!

In practice we can adopt it by either

(a) use the `multiprocessing` package, which implements the above idea

(b) let different persons execute scripts over chunks of data

Boosting database loading

Some ideas, not discussed during this talk

- load big data in batches (bulk loading features)
- load from parallel processes
- use table locks
- ...

Plan

Good practices learned by experience

- don't trust the data quality $\Longrightarrow$ implement countermeasures against broken files!
- think of scaling $\Longrightarrow$ text-processing (splitting, searching, replacing) can be a performance bottleneck!

Plan

References I

P. S. Foundation. Python Documentation. accessed on 20.03.2017. 2017. URL: https://docs.python.org/3.6/library/re.html.

J. Knupp. accessed on 20.03.2017. 2013. URL: https://jeffknupp.com/blog/2013/06/30/pythons-hardest-problem-revisited/.

Nelson. accessed on 20.03.2017. 2015. URL: https://nelsonslog.wordpress.com/2015/02/26/python-file-reading-benchmarks/.

Thank you for your attention!

Questions?