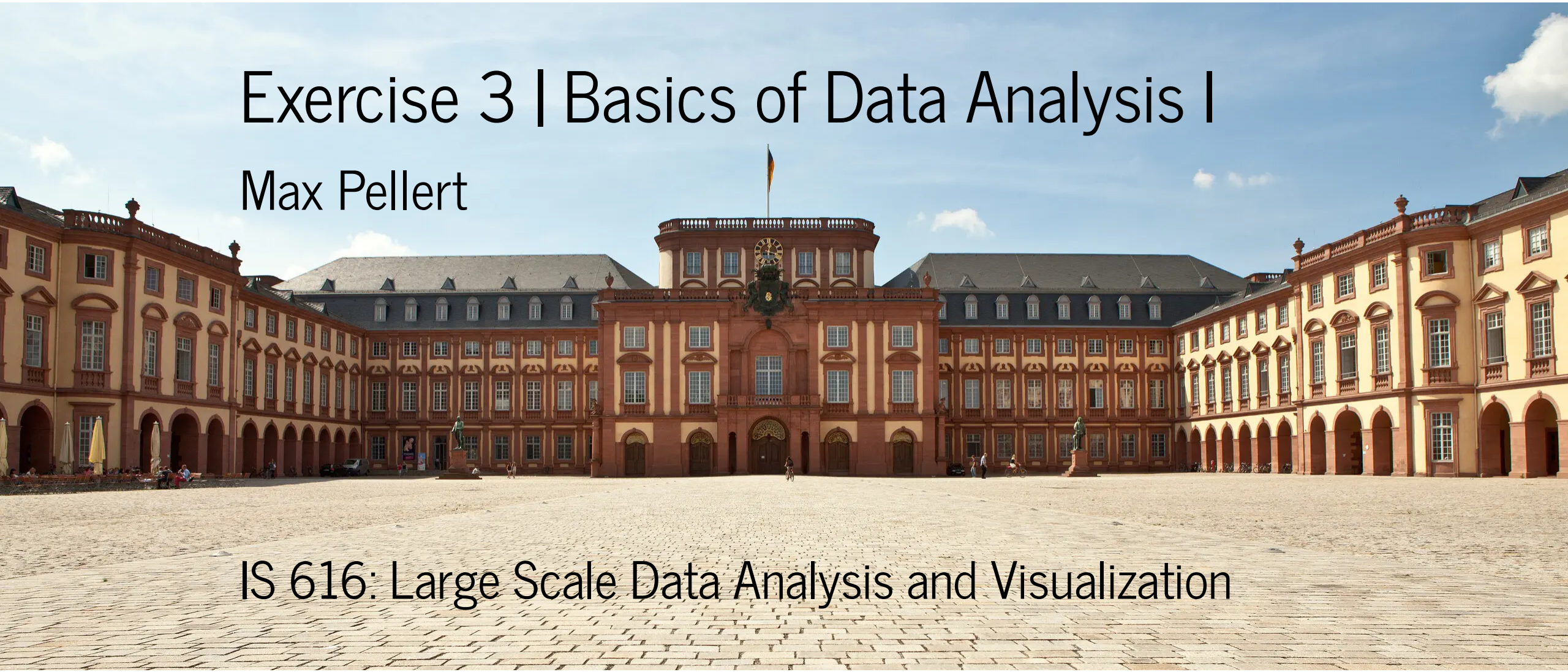
Exercise 3 | Basics of Data Analysis I

Max Pellert



IS 616: Large Scale Data Analysis and Visualization

Data

the *Reader*, that he hath found, that the *Apertures*, which *Optick-Glaſſes* can bear with diſtinctneſs, are in about a *ſubduplicate proportion* to their *Lengths*; whereof he tells us he intends to give the reaſon and demonſtration in his *Diopticks*, which he is now writing, and intends to finiſh, as ſoon as his Health will permit. In the mean time, he preſents the *Reader* with a *Table* of ſuch *Apertures*; which is here exhibited to the Conſideration of the Ingenious, there being of this *French* Book but one Copy, that is known, in *England*.

A *TABLE* of the *Apertures* of *Object-Glaſſes*.

The Points put to ſome of theſe Numbers denote Fractions.

Lengths of Glaſſes.		For excellent ones.		For good ones.		For ordinary ones.		Lengths of Glaſſes.		For excellent ones.		For good ones.		For ordinary ones.	
Feet.	Inches.	Inch.	Lines.	Inch.	Lines.	Inch.	Lines.	Feet.	Inches.	Inch.	Lines.	Inch.	Lines.	Inch.	Lines.
	4	4		4		3		25		3	4	2	10	2	4
	6	5		5		4		30		3	8	3		2	7
	9	7		6		5		35		4	0	3	4	2	10
1	0	8		7		6		40		4	3	3	7	3	
1	6	9		8		7		45		4	6	3	10	3	2
2	0	11		10		8		50		4	9	4	0	3	4
2	6	1	0	1	11	9		55		5	0	4	3	3	6
3	0	1	1	1	0	10		60		5	2	4	6	3	8
3	6	1	2	1	1	11		65		5	4	4	8	3	10
4	0	1	4	1	2	1	0	70		5	7	4	10	4	
4	6	1	5	1	3	1		75		5	9	5	0	4	2
5	0	1	6	1	4	1		80		5	11	5	2	4	5
6		1	7	1	5	1	2	90		6	4	5	6	4	7
7		1	9	1	6	1	3	100		6	8	5	9	4	10
8		1	10	1	8	1	4	120		7	5	6	5	5	3
9		1	11	1	9	1	5	150		8	0	7	0	5	11
10		2	1	1	10	1	6	200		9	6	8	0	6	
12		2	4	2	0	1	8	250		10	6	9	2	7	8
14		2	6	2	2	1	9	300		11	6	10	0	8	5
16		2	8	2	4	1	11	350		12	6	10	9	9	0
18		2	10	2	6	2	1	400		13	4	11	6	9	8
20		3	0	2	7	2	2								

How to store data?

We will talk about two broad kinds of data formats

Human readable formats

Classical: Comma-seperated values (CSV) or Tabulator-seperated (TSV)

Also: JSON, YAML and many others

“Binary” formats

Those would show up as garbage in your text editor

Maybe not so accessible, but other advantages

```
company,surname,forename
Foo Tech,Jones,Alice
Top Bar Hardware,Smith,Bob
Quxcorp,Garcia,Carlos
```

Human readable formats

Easy to inspect

Pretty straightforward to use

What about data types? Long float/numeric or character string? Character string or date object?

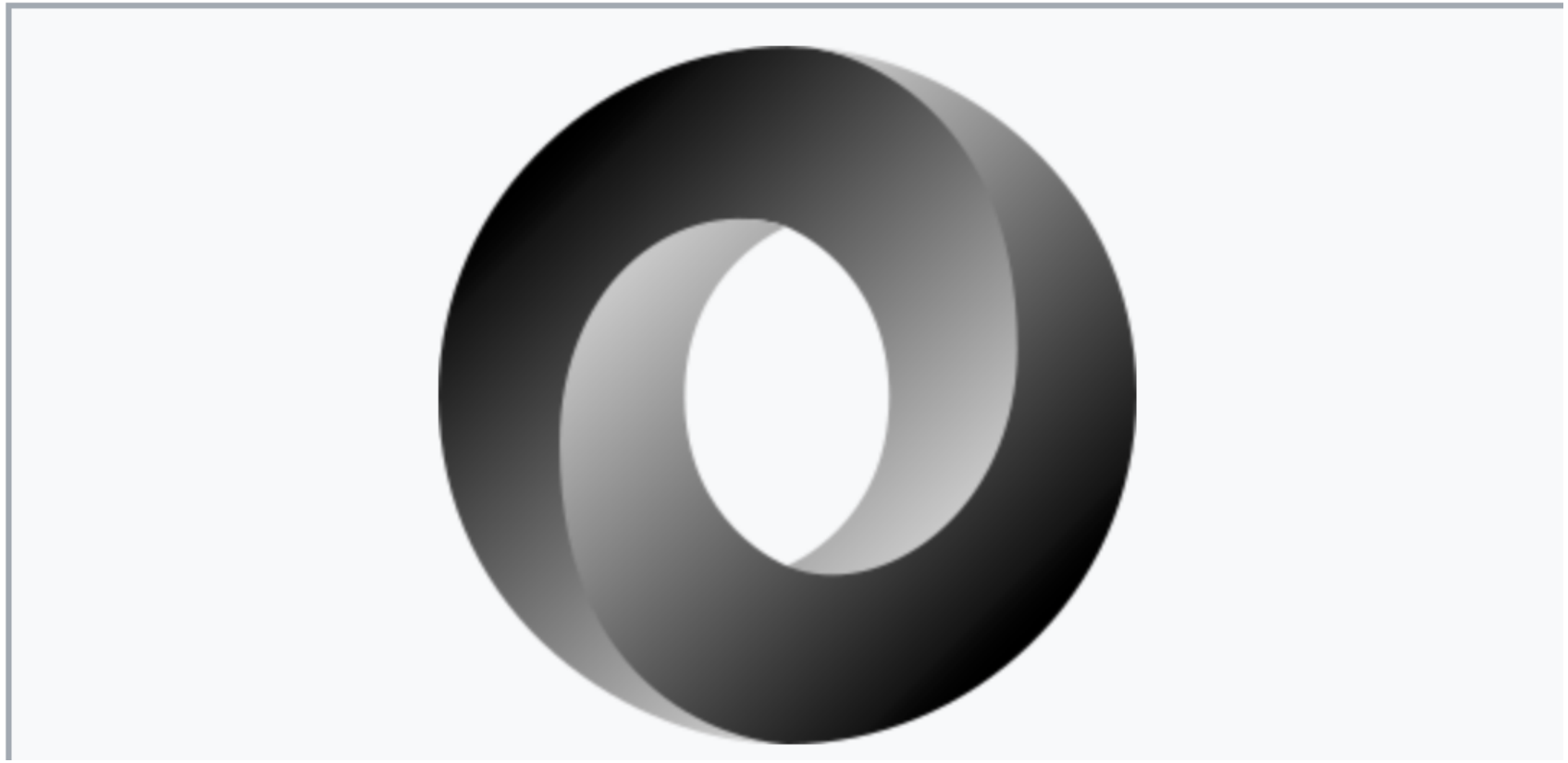
Performance? I/O

Corruption? What if the field separator is contained within the field? Problems like this lead to many quoting and escaping rules and other differences between users and software packages

Convenience?

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Catherine",
    "Thomas",
    "Trevor"
  ],
  "spouse": null
}
```

JavaScript Object Notation



```yaml
---
receipt:     Oz-Ware Purchase Invoice
date:        2012-08-06
customer:
    first_name:   Dorothy
    family_name:  Gale

items:
    - part_no:   A4786
      descrip:   Water Bucket (Filled)
      price:     1.47
      quantity:  4

    - part_no:   E1628
      descrip:   High Heeled "Ruby" Slippers
      size:      8
      price:     133.7
      quantity:  1

bill-to:  &id001
    street: |
            123 Tornado Alley
            Suite 16
    city:   East Centerville
    state:  KS

ship-to:  *id001

specialDelivery:  >
    Follow the Yellow Brick
    Road to the Emerald City.
    Pay no attention to the
    man behind the curtain.
...
```

YAML (/ˈjæməl/) *(see § History and name)* is a human-readable data-serialization language. It is commonly used for configuration files and in applications where data is being stored or transmitted. YAML targets many of the same communications applications as Extensible Markup Language (XML) but has a minimal syntax which intentionally differs from Standard Generalized Markup Language (SGML).[2] It uses both Python-style indentation to indicate nesting, and a more compact format that uses `[...]` for lists and `{...}` for maps[2] but forbids tab characters to use as indentation[3] thus only some JSON files are valid YAML 1.2.[4]

YAML

Filename extensions	`.yaml`, `.yml`
Internet	*Not registered*

Binary formats

Can store the data type of the column: no more userids read in as numeric and converted to scientific notation

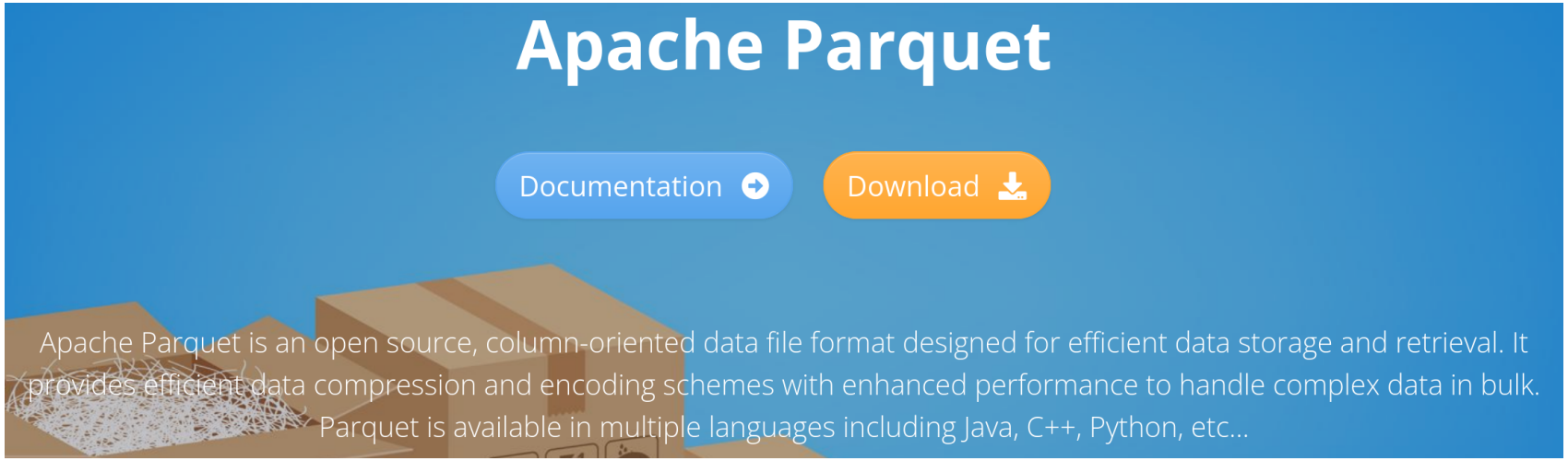
Can optimize for read speed

Can optimize for disk space (compression)

Depending on the format, can offer to do column subsets for reading in

Advanced: sometimes you can use database query language on some formats with special packages


APACHE
ARROW


Apache Parquet
Documentation
Download
Apache Parquet is an open source, column-oriented data file format designed for efficient data storage and retrieval. It provides efficient data compression and encoding schemes with enhanced performance to handle complex data in bulk. Parquet is available in multiple languages including Java, C++, Python, etc...

`pickle` — Python object serialization

Source code: Lib/pickle.py

The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are "pickling" and "unpickling".

readRDS {base}

Serialization Interface for Single Objects

Description

Functions to write a single `R` object to a file, and to restore it.

Usage

```
saveRDS(object, file = "", ascii = FALSE, version = NULL,
        compress = TRUE, refhook = NULL)

readRDS(file, refhook = NULL)
infoRDS(file)
```

How to load data?

Easy answer: Depends on the storage format

For humanly readable data a lot of different functions (for example from pandas, data.table, dplyr, …): read.csv, read_csv, fread, *.from_csv, …

The binary formats usually offer a library that can be loaded to provide import and export functions

How to decide between formats?

If there are no big constraints on disk space, I/O performance and similar things: (almost) everybody can work with TSVs

If you need to read in repeatedly large files, you can make your life much easier when you choose a format that optimizes I/O (like feather from Appache Arrow for example)

If disk space is an issue, use a format that supports good compression like parquet

A note on CSVs

What if the field seperator (a comma for example) appears within a field? (in a text for example)

One solution: We quote the field a,“<TEXTWITHCOMMA>”,b

What about an actual quotation mark appearing in <TEXTWITHCOMMAANDQUOTATIONMARK>?

We escape (\“), double (””) or change to single quotation marks (’)

–> Quoting and escape rules with no real standard

TSV are a bit “safer” (because tabulators are more rare within fields), but still better to pay attention to the possibility

How to manipulate the data?

Usually the bottleneck is RAM

For R as well as Python all objects are handled in memory

Some tricks can help like the one we will discuss shortly

Most important: avoid unnecessary copies!

Many functions that are not well-implemented copy a lot

One tip from practice

Make use of column subsetting, i.e. specify the columns you need in the loading function (many formats support this)

Especially handy when you work with text: huge data sets tend to overwhelm the memory you have on typical machines quite easily

One workaround can be to work with a (row) index instead of the full text and do all kind of preprocessing with the (lightweight) metadata

Then write out the index of those rows that you actually keep and use a lightweight UNIX tool like AWK to select line by line only those from the original text file

A not(e) on loops in R

R works best on vectors (“vectorized” functions are usually way faster)

Avoid loops wherever possible

Functions like the apply family (lapply, sapply, …) sometimes also loop, but avoid some performance bottlenecks

If you have to loop, prespecify at least the size of the output object, for example a list, first

Use the remaining time to

Load the data set that you selected in the last exercise

Save it in different binary formats:

Apache Arrow, Parquet

Depending on your choice of programming language: pickle or saveRDS

Check and load each of the saved files again and take note of any differences (in terms of speed, functionality, disk space)