A Practical Introduction to Machine Learning in Python
Day 2 - Tuesday Morning
»File formats and data structures, again«

Damian Trilling
Anne Kroon

d.c.trilling@uva.nl, @damian0604
a.c.kroon@uva.nl, @annekroon

September 20, 2022

Gesis

Today

Data structures and files

Encodings and dialects

Dataframes

Beyond standard data files

- API's
- Messy data

Everything clear from yesterday?

Data structures and files

(a bit more in-depth than yesterday)

Some ways of storing data

use case	data type	structure	typical file format
(long) texts	string	unstructured	multiple .txt files
list of messages/words/...	list of strings	newline-delimited	.txt
hierarchical data	dict	(semi-)structured	.json
table	nested lists, dict, dataframe	tabular	.csv (.json)

(Of course, there are many, many, many other formats we can use)

Some ways of storing data

use case	data type	structure	typical file format
(long) texts	string	unstructured	multiple .txt files
list of messages/words/...	list of strings	newline-delimited	.txt
hierarchical data	dict	(semi-)structured	.json
table	nested lists, dict, dataframe	tabular	.csv (.json)

(Of course, there are many, many, many other formats we can use)

Some ways of storing data

use case	data type	structure	typical file format
(long) texts	string	unstructured	multiple .txt files
list of messages/words/...	list of strings	newline-delimited	.txt
hierarchical data	dict	(semi-)structured	.json
table	nested lists, dict, dataframe	tabular	.csv (.json)

(Of course, there are many, many, many other formats we can use)

Some ways of storing data

use case	data type	structure	typical file format
(long) texts	string	unstructured	multiple .txt files
list of messages/words/...	list of strings	newline-delimited	.txt
hierarchical data	dict	(semi-)structured	.json
table	nested lists, dict, dataframe	tabular	.csv (.json)

(Of course, there are many, many, many other formats we can use)

Some ways of storing data

use case	data type	structure	typical file format
(long) texts	string	unstructured	multiple .txt files
list of messages/words/...	list of strings	newline-delimited	.txt
hierarchical data	dict	(semi-)structured	.json
table	nested lists, dict, dataframe	tabular	.csv (.json)

(Of course, there are many, many, many other formats we can use)

Some ways of storing data

use case	data type	structure	typical file format
(long) texts	string	unstructured	multiple .txt files
list of messages/words/...	list of strings	newline-delimited	.txt
hierarchical data	dict	(semi-)structured	.json
table	nested lists, dict, dataframe	tabular	.csv (.json)

(Of course, there are many, many, many other formats we can use)

string $\leftrightarrow$ file

```
1  data = "Hi I'm a string."
2  with open("mytext.txt", mode="w") as fo:
3      fo.write(data)
```

$\Rightarrow$ create (or overwrite(!)) file, assign handler name `fo`, write string to it.

```
1  with open("mytext.txt", mode="r") as fi:
2      data = fi.read()
```

$\Rightarrow$ make connection with file for reading, assign handler name `fi`, read string from it

string $\leftrightarrow$ file

```
1  data = "Hi I'm a string."
2  with open("mytext.txt", mode="w") as fo:
3      fo.write(data)
```

$\Rightarrow$ create (or overwrite(!)) file, assign handler name `fo`, write string to it.

```
1  with open("mytext.txt", mode="r") as fi:
2      data = fi.read()
```

$\Rightarrow$ make connection with file for reading, assign handler name `fi`, read string from it

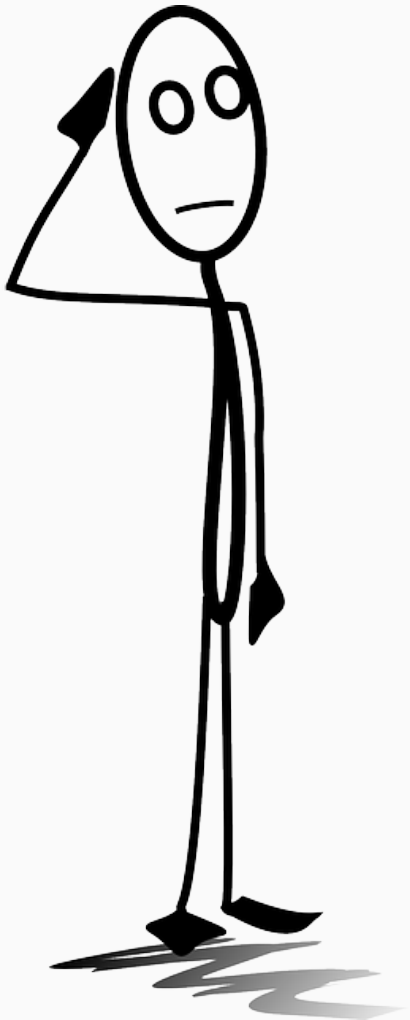
string $\leftrightarrow$ file

```
1  data = "Hi I'm a string."
2  with open("mytext.txt", mode="w") as fo:
3      fo.write(data)
```

$\Rightarrow$ create (or overwrite(!)) file, assign handler name `fo`, write string to it.

```
1  with open("mytext.txt", mode="r") as fi:
2      data = fi.read()
```

$\Rightarrow$ make connection with file for reading, assign handler name `fi`, read string from it



For what can you use this?

list → file

```
1  data = ["ik", "jij", "je", "hij", "zij", "ze", "wij", "we", "jullie"]
2
3  with open("pronouns.txt", mode="w") as fo:
4      for pronoun in data:
5          fo.write(pronoun)
6          fo.write("\n")
```

⇒ create file, assign handler name `fo`, write each element from list to `fo` followed by a line break

Result: a file `pronouns.txt` with this content:

```
1  ik
2  jij
3  je
4  hij
5  zij
6  ze
7  wij
8  we
```

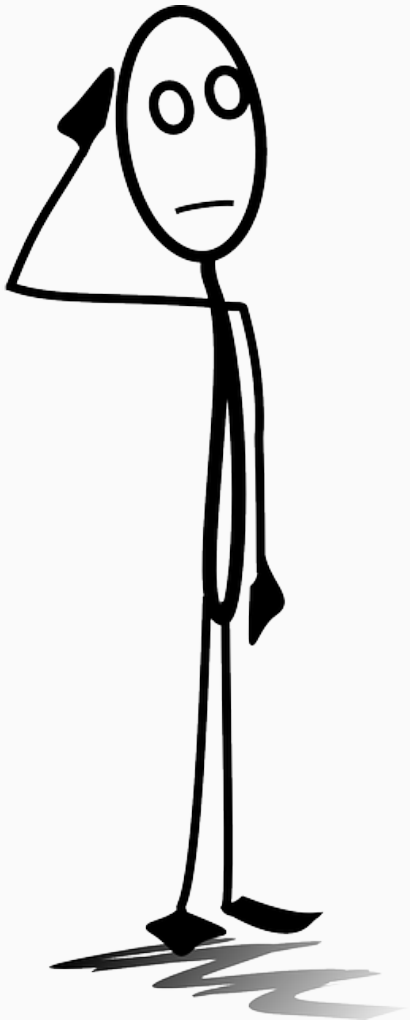
list ← file

```
1 with open("pronouns.txt", mode="r") as fi:
2     data = [line.strip() for line in fi]
3 print(data)
```

⇒ Open file for reading, assign handler name `fi`, loop over all lines in `fi`, strip whitespace from end (such as line endings), put in new list.

Output:

```
1 ['ik', 'jij', 'je', 'hij', 'zij', 'ze', 'wij', 'we', 'jullie']
```



For what can you use this?

dict → file

```
1  import json
2
3  data = {'Alice': {'office': '020222', 'mobile': '0666666'},
4      'Bob': {'office': '030111'},
5      'Carol': {'office': '040444', 'mobile': '0644444'},
6      'Daan': "020222222",
7      'Els': ["010111", "06222"]}
8
9  with open("phonebook.json", mode="w") as f:
10     json.dump(data, f)
```

⇒ Open file for writing, convert dict to JSON, dump in file.

Creates a file `phonebook.json` that looks like this:

```
1  {"Alice": {"office": "020222", "mobile": "0666666"}, "Bob": {"office":
       "030111"}, "Carol": {"office": "040444", "mobile": "0644444"}, "
       Daan": "020222222", "Els": ["010111", "06222"]}
```

dict ← file

```
1  import json
2
3  with open("phonebook.json", mode="r") as f:
4          data = json.load(f)
```

⇒ Reads data from f, converts to dict (or list of dicts. . .), store in `data`

JSON lines

There is also a dialect in which you write *one JSON object per line instead of per file*. For this, you can use a for loop as in the one-string-per-file example, but convert each string with `json.loads` or `json.dumps` to a dict.

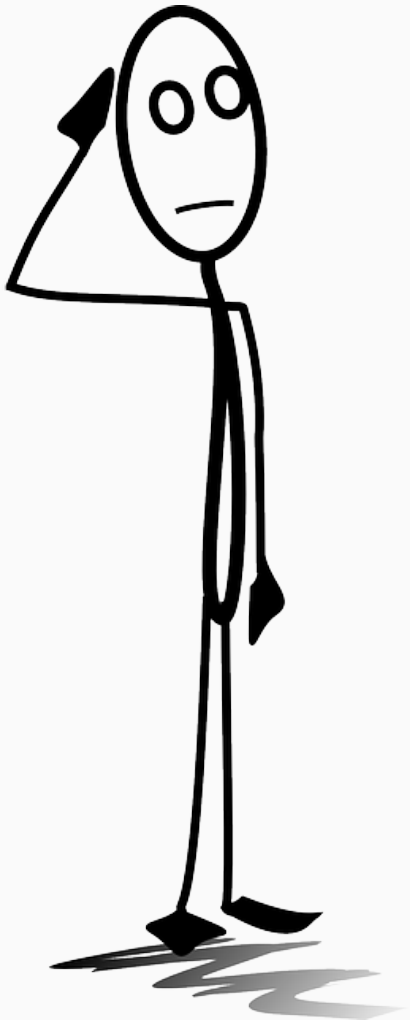
dict ← file

```
1  import json
2
3  with open("phonebook.json", mode="r") as f:
4          data = json.load(f)
```

⇒ Reads data from f, converts to dict (or list of dicts. . .), store in `data`

JSON lines

There is also a dialect in which you write *one JSON object per line instead of per file*. For this, you can use a for loop as in the one-string-per-file example, but convert each string with `json.loads` or `json.dumps` to a dict.



For what can you use this?

Tabular data

How can we store this data?

```
1  names = ['Alice', 'Bob', 'Carol', 'Daan', 'Els']
2  phonenumbers = ['020111111', '020222222', '020333333', '020444444',
       '020555555']
```

1. We could convert to some dict and store as json (not too bad...)
2. We can store in a table (next slide)

Tabular data

How can we store this data?

```
1  names = ['Alice', 'Bob', 'Carol', 'Daan', 'Els']
2  phonenumbers = ['020111111', '020222222', '020333333', '020444444',
       '020555555']
```

1. We could convert to some dict and store as json (not too bad...)
2. We can store in a table (next slide)

Tabular data

How can we store this data?

```
names = ['Alice', 'Bob', 'Carol', 'Daan', 'Els']
phonenumbers = ['020111111', '020222222', '020333333', '020444444',
    '020555555']
```

1. We could convert to some dict and store as json (not too bad...)
2. We can store in a table (next slide)

Tabular data

```
1  import csv
2  with open("mytable.csv", mode="w") as f:
3      mywriter = csv.writer(f)
4      for row in zip(names,phonenumbers):
5          mywriter.writerow(row)
```

Results in a file `mytable.csv` that looks like this:

```
1  Alice,020111111
2  Bob,020222222
3  Carol,020333333
4  Daan,020444444
5  Els,020555555
```

But you don't have to do it like this! Often, pandas is more user-friendly. Yet, also pandas doesn't save you from thinking about encodings and dialects. . .

Tabular data

```
1  import csv
2  with open("mytable.csv", mode="w") as f:
3      mywriter = csv.writer(f)
4      for row in zip(names,phonenumbers):
5          mywriter.writerow(row)
```

Results in a file `mytable.csv` that looks like this:

```
1  Alice,020111111
2  Bob,020222222
3  Carol,020333333
4  Daan,020444444
5  Els,020555555
```

But you don't have to do it like this! Often, pandas is more user-friendly. Yet, also pandas doesn't save you from thinking about encodings and dialects...

Encodings and dialects

Choices to make

For all text-based (as opposed to binary) file formats:

How to separate data?

- new line = new record?
- Unix-style (`\n`, also known as LF), or Windows-style (`\r\n`, also known as CRLF) line endings?
- some delimiter = new field?
- or new file = new record?

How to convert bytes to characters?

- choose right encoding (e.g., UTF-8)
- (seldom) does the file start with a so-called byte-order-marker (BOM) – then the encoding is often referred to as sth like UTF-8-BOM

Choices to make

For all text-based (as opposed to binary) file formats:

How to separate data?

- new line = new record?
- Unix-style (`\n`, also known as LF), or Windows-style (`\r\n`, also known as CRLF) line endings?
- some delimiter = new field?
- or new file = new record?

How to convert bytes to characters?

- choose right encoding (e.g., UTF-8)
- (seldom) does the file start with a so-called byte-order-marker (BOM) – then the encoding is often referred to as sth like UTF-8-BOM

Let's look at csv files

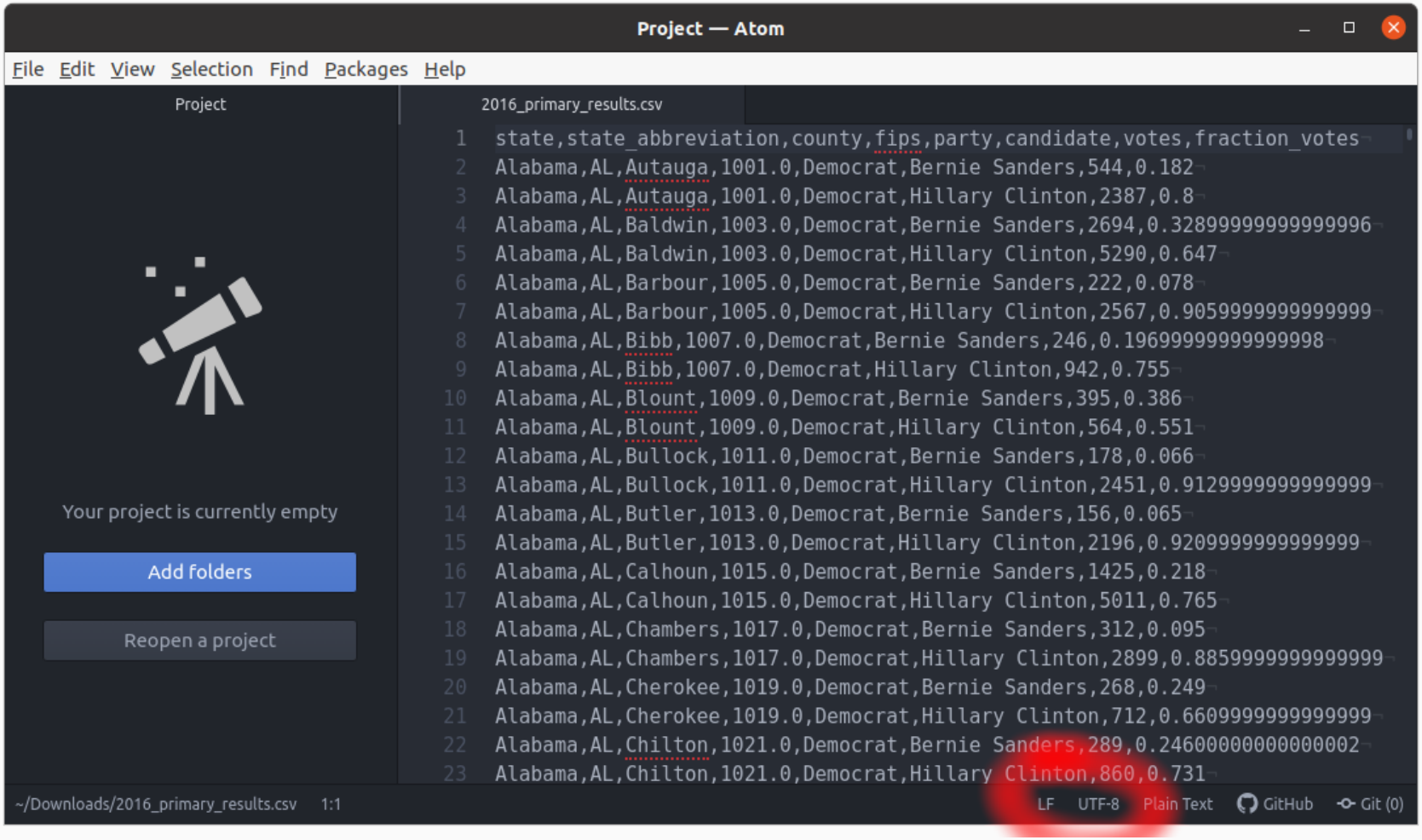
comma-separated values: always a good choice

- *All* programs can read it
- Even human-readable in a simple text editor
- Plain text, with a comma (or a semicolon) denoting column breaks
- No limits regarging the size
- But: several dialects (e.g., `,` vs. `;` as delimiter)
- Also: tab-separated files (.csv, .tab, .txt – no consensus) (delimiter is `\t`)

A CSV-file with tweets

- delimiter is ,
- with header row

```
1 text,to_user_id,from_user,id,from_user_id,iso_language_code,source,
      profile_image_url,geo_type,geo_coordinates_0,geo_coordinates_1,
      created_at,time
2 :-) #Lectrr #wereldleiders #uitspraken #Wikileaks #klimaattop http://t.
      co/Udjpk48EIB,,henklbr,4070859170110791 69,118374840,nl,web,http://
      pbs.twimg.com/profile_images/378800000673845195/
      b47785b1595e6a1c63b93e463f3d0ccc_normal.jpeg,,0,0,Sun Dec 01
      09:57:00 +0000 2013,1385891820
3 Wat zijn de resulaten vd #klimaattop in #Warschau waard? @EP_Environment
       ontmoet voorzitter klimaattop @MarcinKorolec http://t.co/4
      Lmiaopf60,,Europarl_NL,406058792573730816,37623918,en,<a href="http
      ://www.hootsuite.com" rel="nofollow">HootSuite</a>,http://pbs.twimg
      .com/profile_images/2943831271/
      b6631b23a86502fae808ca3efde23d0d_normal.png,,0,0,Thu Nov 28
      13:55:35 +0000 2013,1385646935
```



Opening a file in a (good) text editor (here: Atom) to check its encoding and line-ending style.

Why encodings are really, really, REALLY important – and why you shouldn't let Excel mess with them

- Unicode is around for decades, but sometimes legacy encodings (ASCII, ANSI, Windows-1252, ...) are still used
- Problem 1: They don't support all Unicode symbols (emoticons, different scripts, ...)
- Problem 2: What is an ä in the one encoding may be an ø in another $\Rightarrow$ big confusion if you use the wrong one
- Some programs (looking at you, Excel!) may use legacy encodings when saving CSV files without telling you!

Make sure to use UTF-8 from beginning to end, unless you know what you are doing!

Dataframes

What are dataframes?

`pd.DataFrames` (from the pandas package) are

- objects that store tabular data in rows and columns.
- columns and rows can have names
- they have methods built-in for data wrangling and analysis

Creating dataframes

Option 1: transform existing data into a dataframe

`df = pd.DataFrame(list-of-lists, dict, or similar)`

(use `pd.DataFrame?` for help if necessary)

Option 2: read from file

Using tab-completion to see methods to read dataframes from a file)

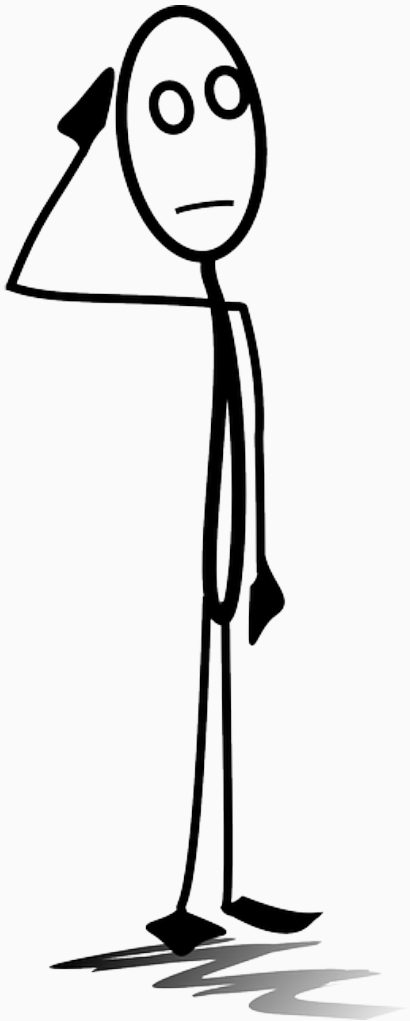
In [6]:
```
# just use the default....
df = pd.read_csv("2016_primary_results.csv")
df
```

Out[6]:

	state	state_abbreviation	county	fips	party	candidate	votes	fraction_votes
0	Alabama	AL	Autauga	1001.0	Democrat	Bernie Sanders	544	0.182
1	Alabama	AL	Autauga	1001.0	Democrat	Hillary Clinton	2387	0.800
2	Alabama	AL	Baldwin	1003.0	Democrat	Bernie Sanders	2694	0.329
3	Alabama	AL	Baldwin	1003.0	Democrat	Hillary Clinton	5290	0.647
4	Alabama	AL	Barbour	1005.0	Democrat	Bernie Sanders	222	0.078
...	...	...	...	...	...	...	...	...
24606	Wyoming	WY	Teton-Sublette	95600028.0	Republican	Ted Cruz	0	0.000
24607	Wyoming	WY	Uinta-Lincoln	95600027.0	Republican	Donald Trump	0	0.000
24608	Wyoming	WY	Uinta-Lincoln	95600027.0	Republican	John Kasich	0	0.000
24609	Wyoming	WY	Uinta-Lincoln	95600027.0	Republican	Marco Rubio	0	0.000
24610	Wyoming	WY	Uinta-Lincoln	95600027.0	Republican	Ted Cruz	53	1.000

24611 rows × 8 columns

In [10]:
```
# ... or specify encoding, delimiter, or possibly other things (header etc)
df = pd.read_csv("2016_primary_results.csv", encoding="utf-8", delimiter=',')
```



Can you think of a situation when you would use the for-loop approach to reading or writing tabular data instead of the pandas approach? What are pros and cons?

When (not) to use dataframes

use it!

- tabular data
- visual inspection
- data wrangling or statistical analysis

don't use it

- non-tabular data
- when it does not make sense to consider rows as "cases" and columns as "variables"
- if you only care about one (or maybe two) column anyway
- size of dataset $>$ available RAM
- long or expensive operations, play safe and write to / read from file line by line*

* imagine scraping 10,000 pages for a week and your program crashes at nr. 9,999 just before saving the dataframe. . .

Beyond standard data files

Beyond standard data files

API's

Beyond files

It probably has occured to you by now that

it's all about the structure:

- we can write *anything* to files
- it doesn't really matter whether sth is called "csv" or whatever – as long as we know how records are delimited and what the encoding is
- Maybe check out Chapter 9 in the old book for an example of how we can write files in a strange format called GDF (for network data) *even though it is not natively supported* (**Trilling2016**) (https://github.com/damian0604/bdaca/tree/master/book)
- and ... do we then even need files?

Beyond files

It probably has occured to you by now that

it's all about the structure:

- we can write *anything* to files
- it doesn't really matter whether sth is called "csv" or whatever – as long as we know how records are delimited and what the encoding is
- Maybe check out Chapter 9 in the old book for an example of how we can write files in a strange format called GDF (for network data) *even though it is not natively supported* (**Trilling2016**) (https://github.com/damian0604/bdaca/tree/master/book)
- and ... do we then even need files?

Beyond files

It probably has occured to you by now that

it's all about the structure:

- we can write *anything* to files
- it doesn't really matter whether sth is called "csv" or whatever – as long as we know how records are delimited and what the encoding is
- Maybe check out Chapter 9 in the old book for an example of how we can write files in a strange format called GDF (for network data) *even though it is not natively supported* (**Trilling2016**) (https://github.com/damian0604/bdaca/tree/master/book)
- and ... do we then even need files?

Beyond files

It probably has occured to you by now that

it's all about the structure:

- we can write *anything* to files
- it doesn't really matter whether sth is called "csv" or whatever – as long as we know how records are delimited and what the encoding is
- Maybe check out Chapter 9 in the old book for an example of how we can write files in a strange format called GDF (for network data) *even though it is not natively supported* (**Trilling2016**) (https://github.com/damian0604/bdaca/tree/master/book)
- and ... do we then even need files?

Beyond files

It probably has occured to you by now that

it's all about the structure:

- we can write *anything* to files
- it doesn't really matter whether sth is called "csv" or whatever – as long as we know how records are delimited and what the encoding is
- Maybe check out Chapter 9 in the old book for an example of how we can write files in a strange format called GDF (for network data) *even though it is not natively supported* (**Trilling2016**) (https://github.com/damian0604/bdaca/tree/master/book)
- and ... do we then even need files?

APIs

Why not just send a JSON object (with a question/request) directly through the internet and get another one (with an answer/response) back?

That's what (most) Application Programming Interfaces (APIs) do.

Great if we have a nested data structure

- Items within news feeds
- Personal data within authors within books
- Bio statements within authors within tweets

APIs

Why not just send a JSON object (with a question/request) directly through the internet and get another one (with an answer/response) back?

That's what (most) Application Programming Interfaces (APIs) do.

Great if we have a nested data structure

- Items within news feeds
- Personal data within authors within books
- Bio statements within authors within tweets

APIs

Why not just send a JSON object (with a question/request) directly through the internet and get another one (with an answer/response) back?

That's what (most) Application Programming Interfaces (APIs) do.

Great if we have a nested data structure

- Items within news feeds
- Personal data within authors within books
- Bio statements within authors within tweets

APIs

Why not just send a JSON object (with a question/request) directly through the internet and get another one (with an answer/response) back?

That's what (most) Application Programming Interfaces (APIs) do.

Great if we have a nested data structure

- Items within news feeds
- Personal data within authors within books
- Bio statements within authors within tweets

A JSON object containing GoogleBooks data

```
1 {'totalItems': 574, 'items': [{'kind': 'books#volume', 'volumeInfo': {'
      publisher': '"O\'Reilly Media, Inc."', 'description': u'Get a
      comprehensive, in-depth introduction to the core Python language
      with this hands-on book. Based on author Mark Lutz\u2019s popular
      training course, this updated fifth edition will help you quickly
      write efficient, high-quality code with Python. It\u2019s an ideal
      way to begin, whether you\u2019re new to programming or a
      professional developer versed in other languages. Complete with
      quizzes, exercises, and helpful illustrations, this easy-to-follow,
       self-paced tutorial gets you started with both Python 2.7 and 3.3\
      u2014 the
2 ...
3 ...
4 'kind': 'books#volumes'}
```

Who offers APIs?

The usual suspects: Twitter, Facebook, Google – but also Reddit, Youtube, ...

Who offers APIs?

The usual suspects: Twitter, Facebook, Google – but also Reddit, Youtube, ...

If you ever leave your bag on a bus on Chicago

... but do have Python on your laptop, watch this: https://www.youtube.com/watch?v=RrPZza_vZ3w.

That guy queries the Chicago bus company's API to calculate when *exactly the vehicle* with his bag arrives the next time at the bus stop in front of his office.

(Yes, he tried calling the help desk before, but they didn't know. He got his bag back.)

Who offers APIs?

The usual suspects: Twitter, Facebook, Google – but also Reddit, Youtube, ...

If you ever leave your bag on a bus on Chicago

...but do have Python on your laptop, watch this:
https://www.youtube.com/watch?v=RrPZza_vZ3w.
That guy queries the Chicago bus company's API to calculate when *exactly the vehicle* with his bag arrives the next time at the bus stop in front of his office.

(Yes, he tried calling the help desk before, but they didn't know. He got his bag back.)

APIs

Pro

- Structured data (JSON!)
- Easy to process automatically
- Can be directy embedded in your script

Con

- Often limitations (requests per minute, sampling, . . .)
- You have to trust the provider that he delivers the right content (Morstatter2013)
- Some APIs won't allow you to go back in time!

⇒ We work with a simple API in the exercise.

⇒ More about APIs versus webscraping this afternoon

APIs

Pro

- Structured data (JSON!)
- Easy to process automatically
- Can be directy embedded in your script

Con

- Often limitations (requests per minute, sampling, . . .)
- You have to trust the provider that he delivers the right content (**Morstatter2013**)
- **Some APIs won't allow you to go back in time!**

⇒ We work with a simple API in the exercise.

⇒ More about APIs versus webscraping this afternoon

Beyond standard data files

Messy data

Collecting data:

Parsing text files

For messy input data or for semi-structured data

Guiding question: Can we identify some kind of pattern?

Examples

- LexisNexis gives you a chunk of text (rather than a structured JSON object)
- But as long as you can find any pattern or structure to it, you can try to write a Python script to parse the data (and maybe end up with a dict, lists, or a dataframe)

For messy input data or for semi-structured data

Guiding question: Can we identify some kind of pattern?

Examples

- LexisNexis gives you a chunk of text (rather than, e.g., a structured JSON object)
- But as long as you can find any pattern or structure in it, you can try to write a Python script to *parse* the data (and put it in a dict, lists, or a dataframe)

For messy input data or for semi-structured data

Guiding question: Can we identify some kind of pattern?

Examples

- LexisNexis gives you a chunk of text (rather than, e.g., a structured JSON object)
- But as long as you can find any pattern or structure in it, you can try to write a Python script to *parse* the data (and put it in a dict, lists, or a dataframe)

1 of 200 DOCUMENTS

De Telegraaf

21 maart 2014 vrijdag

Safe

SECTION: Binnenland; Blz. 3

LENGTH: 257 woorden

Na ruim twee jaar onderhandelen zijn de EU-landen, de Europese Commissie en het Europees Parlement het eens geworden over een nieuwe Europese bankenunie. Een verstrekkende afspraak die noodzakelijk bleek na de diepe financiële crisis,

De_Telegraaf2014-03-22_22-08.TXT

File Path : ~/Dropbox/uva/tcst/De_Telegraaf2014-03-22_22-08.TXT

De_Telegraaf2014-03-22_22-08.TXT

zorgen dat we niet in zo'n situatie terecht kunnen komen. Europese banken mogen niet meer omvallen. De nu gemaakte afspraken kunnen daarvoor zorgen.

LOAD-DATE: 21 March 2014

LANGUAGE: DUTCH; NEDERLANDS

PUBLICATION-TYPE: Krant

JOURNAL-CODE: Telegraaf

2 of 200 DOCUMENTS

Line 1 Col 1 | (none) | Unicode (UTF-8) | Classic Mac (CR) | Last saved: 24-03-14 21:53:18 | 471.467 / 64.074...

```
tekst={}
section={}
length={}
...
...
with open(bestandsnaam) as f:
    for line in f:
        line=line.replace("\r","")
        if line=="\n":
            continue
        matchObj=re.match(r"\s+(\d+) of (\d+) DOCUMENTS",line)
        if matchObj:
            artikelnr = int(matchObj.group(1))
            tekst[ artikelnr ]=""
            continue
        if line.startswith("SECTION"):
            section[ artikelnr ]=line.replace("SECTION: ","").rstrip("\n")
        elif line.startswith("LENGTH"):
            length[ artikelnr ]=line.replace("LENGTH: ","").rstrip("\n")
...
...
...

        else:
            tekst[ artikelnr ]=tekst[ artikelnr ]+line
```