

Big Data and Automated Content Analysis

Week 2 – Thursday

Data harvesting and storage

Damian Trilling

d.c.trilling@uva.nl

@damian0604

www.damiantrilling.net

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

9 April 2020

Today

This week's exercise

Discussing the code

Reading a JSON file into a dict, looping over the dict

Task 1: Print all titles of all videos

```
1 import json
2
3 with open("/home/damian/pornexercise/xhamster.json") as fi:
4     data=json.load(fi)
5
6 for k,v in data.items():
7     print (v["title"])
```

Reading a JSON file into a dict, looping over the dict

Task 1: Print all titles of all videos

```
1 import json
2
3 with open("/home/damian/pornexercise/xhamster.json") as fi:
4     data=json.load(fi)
5
6 for k,v in data.items():
7     print (v["title"])
```

NB: You have to know (e.g., by reading the documentation of the dataset) that the key is called title

Reading a JSON file into a dict, looping over the dict

Task 1: Print all titles of all videos

```
1 import json
2
3 with open("/home/damian/pornexercise/xhamster.json") as fi:
4     data=json.load(fi)
5
6 for k,v in data.items():
7     print (v["title"])
```

NB: You have to know (e.g., by reading the documentation of the dataset) that the key is called `title`

NB: `data` is in fact a dict of dicts, such that each value `v` is another dict.

For each of these dicts, we retrieve the value that corresponds to the key `title`

What to do if you do not know the structure of the dataset?

Inspecting your data: use the functions `type()` and `len()` and/or the dictionary method `.keys()`

```
1 len(data)
2 type(data)
3 data.keys()
```

What to do if you do not know the structure of the dataset?

Inspecting your data: use the functions `type()` and `len()` and/or the dictionary method `.keys()`

```
1 len(data)
2 type(data)
3 data.keys()
```

`len()` returns the number of items of an object; `type()` returns the type object; `.keys()` returns a list of all available keys in the dictionary

What to do if you do not know the structure of the dataset?

Inspecting your data: use the module `pprint`

```
1 from pprint import pprint
2 pprint(data)
```

(but better do this on a smaller subset of the data!)

```
{'1002968': {'channels': ['Tits'],
  'description': 'NA',
  'id': 1002968,
  'nb_comments': 6,
  'nb_views': 26683,
  'nb_votes': 68,
  'runtime': 1083,
  'title': 'big tit chick gets fucked',
  'upload_date': '2012-01-22',
  'uploader': '894621d2f89ab1b859c17b9b81f182e82a375c55'},
'1003007': {'channels': ['Hardcore', 'Spanking', 'Squirting'],
  'description': 'Antonia plays with her 16 inch toy and her '
    'crop,till she sprays cum into her own mouth!',
  'id': 1003007,
  'nb_comments': 5,
  'nb_views': 5008,
  'nb_votes': 35,
  'runtime': 213,
  'title': 'antonia just playin around',
  'upload_date': '2012-01-22',
```


For the sake of completeness. . .

`.items()` returns a key-value *pair*, that's why we need to assign *two* variables in the for statement.

These alternatives would also work:

```
1 for v in data.values():  
2     print(v["title"])
```

```
1 for k in data:    #or: for k in data.keys():  
2     print(data[k]["title"])
```

Do you see (dis-)advantages?

Working with a subset of the data

What to do if you want to work with a smaller subset of the data?

Taking a random sample of 10 items in a dict:

```
1 import random
2 mydict_short = dict(random.sample(mydict.items(),10))
```

Taking the first 10 elements in a list:

```
1 mylist_short = mylist[:10]
```

Initializing variables, merging two lists, using a counter

Task 2: Average tags per video and most frequently used tags

```
1 from collections import Counter
2
3 alltags=[]
4 i=0
5 for k,v in data.items():
6     i+=1
7     alltags.extend(v["channels"])
8
9 print(len(alltags),"tags are describing",i,"different videos")
10 print("Thus, we have an average of",len(alltags)/i,"tags per video")
11
12 c=Counter(alltags)
13 print (c.most_common(100))
```

(there are other, more efficient ways of doing this)

Nesting blocks, using a defaultdict to count, error handling

Task 3: What porn category is most frequently commented on?

```
1 from collections import defaultdict
2
3 commentspercat=defaultdict(int)
4 for k,v in data.items():
5     for tag in v["channels"]:
6         try:
7             commentspercat[tag]+=int(v["nb_comments"])
8         except:
9             pass
10 print(commentspercat)
11 # if you want to print in a fancy way, you can do it like this:
12 for tag in sorted(commentspercat, key=commentspercat.get, reverse=True):
13     print( tag,"\t", commentspercat[tag])
```

A defaultdict is a normal dict, with the difference that the type of each value is pre-defined and it doesn't give an error if you look up a non-existing key

Nesting blocks, using a defaultdict to count, error handling

Task 3: What porn category is most frequently commented on?

```
1 from collections import defaultdict
2
3 commentspercat=defaultdict(int)
4 for k,v in data.items():
5     for tag in v["channels"]:
6         try:
7             commentspercat[tag]+=int(v["nb_comments"])
8         except:
9             pass
10 print(commentspercat)
11 # if you want to print in a fancy way, you can do it like this:
12 for tag in sorted(commentspercat, key=commentspercat.get, reverse=True):
13     print( tag,"\t", commentspercat[tag])
```

A defaultdict is a normal dict, with the difference that the type of each value is pre-defined and it doesn't give an error if you look up a non-existing key

NB: In line 7, we assume the value to be an int, but the datasets sometimes contains the string "NA" instead of a string representing an int. That's why we need the try/except construction

Adding elements to a list, sum() and len()

Task 4: Average length of descriptions

```
1 length=[]  
2 for k,v in data.items():  
3     length.append(len(v["description"]))  
4  
5 print ("Average length",sum(length)/len(length))
```

Extending (merging) vs appending

Merging:

```
1 l1 = [1,2,3]
2 l2 = [4,5,6]
3 # either:
4 l1 = l1 + l2
5 # or:
6 l1.extend(l2)
7 print(l1)
```

gives [1,2,3,4,5,6]

Appending:

```
1 l1 = [1,2,3]
2 l2 = [4,5,6]
3 l1.append(l2)
4 print(l1)
```

gives [1,2,3,[4,5,6]]

l2 is seen as *one* element to append to l1

Tokenizing with .split()

Task 5: Most frequently used words

```
1 allwords=[]
2 for k,v in data.items():
3     allwords+=v["description"].split()
4 c2=Counter(allwords)
5 print(c2.most_common(100))
```

.split() changes a string to a list of words.

"This is cool".split()

results in

["This", "is", "cool"]

Concluding remarks

Make sure you fully understand the code!

Re-read the corresponding chapters

PLAY AROUND!!!