Web Scraping Recap

Module 2 | Chapter 2 | Notebook 1

In this notebook we will recap the most important things from the previous chapter. This is all repetition. If you discover a lot of new things here, we recommend taking a look back at Chapter 1. The relevant lessons for each section are clearly marked.

Accessing and reading a website

The internet offers a variety of data. However, it's not always available in a structured format. Instead, you might need to extract it directly from the HMTL code of the website. But first we have to retrieve the contents of the website. You can use the `requests` module here, which you got to know in *Communicating with Websites (Module 2 Chapter 1)*. As an example in this exercise, we'll use Heise's news ticker that is stored in a reproducible version in *https://web.archive.org/web/20210401004837/www.heise.de/newsticker/*. It contains news reports from the last week and links to the corresponding articles.

We can use the function `requests.get()` to request the content of the website. We will then receive a response. This contains a status code that indicates whether the request was successful.

In [1]:
```
import requests
website_url = "https://web.archive.org/web/20210401004837/www.heise.de/newsticker/"  #
response = requests.get(website_url)  # ask for the content of the website
response.status_code  # print status code
```

Out[1]:
```
200
```

So what exactly does that mean? We've received a reply to our request to send us all the information we would need to reproduce the website. This generall has the status code 200. The most important status codes can be divided into 5 categories, indicated by the first digit:

number	meaning
1XX	The request was received (Information)
2XX	The request was received and accepted (successful request)
3XX	Further action has to be taken to fulfill the request (redirection)
4XX	The request cannot be carried out, which is probably due to the client (client error)
4XX	The request cannot be carried out, which is probably due to the server(server error)

The next two digits then specify the answer. You can find an extensive list of response codes here.

The status code 200 indicates that our request worked. We can also generate an error message straight away if the request was not successful.

```
In [2]: response.raise_for_status()  # raise error if get was not successful
```

If it doesn't generate an error message, then you can move on and look at the website's HTML code.

```
In [3]: response.text[:1000]  # show first 1000 characters of html code of website
```

```
Out[3]: '<!DOCTYPE html>\n<html lang="de" data-responsive data-consentmanagement>\n  <head><s
cript type="text/javascript" src="https://web-static.archive.org/_static/js/bundle-pl
ayback.js?v=t1Bf4PY_" charset="utf-8"></script>\n<script type="text/javascript" src
="https://web-static.archive.org/_static/js/wombat.js?v=txqj7nKC" charset="utf-8"></s
cript>\n<script>window.RufflePlayer=window.RufflePlayer||{};window.RufflePlayer.confi
g={"autoplay":"on","unmuteOverlay":"hidden"};</script>\n<script type="text/javascrip
t" src="https://web-static.archive.org/_static/js/ruffle/ruffle.js"></script>\n<scrip
t type="text/javascript">\n  __wm.init("https://web.archive.org/web");\n  __wm.wombat
("https://www.heise.de/newsticker/","20210401004837","https://web.archive.org/","we
b","https://web-static.archive.org/_static/",\n\t      "1617238117");\n</script>\n<li
nk rel="stylesheet" type="text/css" href="https://web-static.archive.org/_static/css/
banner-styles.css?v=S1zqJCYt" />\n<link rel="stylesheet" type="text/css" href="https'
```

At first glance, this looks pretty messy. However, data in HTML form is not completely unstructured. We call this semi-structured data. Semi-structured data carries part of the structure information with it, instead of conforming to a general structure already like a table.

In addition to semi-structured, there's also structured data and unstructured data. Most of the data companies have access to is unstructured. This is mainly texts in natural language. But audio and video recordings as well as images also come under unstructured data. These are characterized by the fact that it is particularly difficult to extract information from them and make it usable for analyses and models.

It's best to work with structured data where possible. This is data that's available in tabular form, such as SQL databases. You can put this directly into a machine learning model without much effort. However, since most data is not available in this form, data scientists often have to structure the data first. This takes up a lot of your work time. When structuring data, you should pay attention to three principles to ensure that the data can be used as easily as possible for automated evaluations and models. These principles are often called *tidy data principles*. They are as follows:

- Each observation has its own row
- Each variable has its own column
- Each value has its own cell

To convert the data from the HTML code into a structured format according to the *tidy data principles*, we need to know how the HTML code is structured. The structural information is contained in what are called tags. The text is enclosed by an opening tag consisting of a word in angle brackets and a closing tag consisting of a word in angle brackets with a slash `'/'` (e.g. '' `). The opening tag opens an element, for example, the title. The text that follows corresponds to the value of this element (e.g. `'DAX - Wikipedia'`). The closing tag with a slash (e.g. `'</title>'`) ends the element. You can also start new elements within an element, so they can be nested. This nesting is indicated by indentations in the example body.

Every HTML document has a fixed structure, which looks like this:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    The header contains bits of information about the document. These are
displayed as separate elements.
    <title>Title Goes Here</title>
    ...
  </head>
  <body>
    The website body goes here. This is what's displayed in the browser.
    ...
  </body>
</html>
```

HTML tags can contain attributes as well as text. These provide further information on the element. You can see an example of this in the `<html>` tag. For example, This has the attribute `lang='en'`, which specifies the language of the document as English.

HTML offers a vast number of tags and even more attributes, you generally don't have to keep them all in your head. This is why browsers such as Firefox and Chrome offer special tools to view this information on a website. This is called the *Page Inspector* in Firefox and Developer Tools in Chrome.

To extract information from HTML efficiently, you learned about the `beautifulsoup` module in the lesson *Reading Websites* (Module 1 Chapter 1)*. `beautifulsoup` is a very performant parser for HTML documents. You can find the parser in the `BeautifulSoup` class, which you can import as follows:

```
from bs4 import BeautifulSoup
```

If we pass an HTML *string* to BeautifulSoup, we get an object back, which makes it easier to search for elements in the HTML code.

In [4]:
```
from bs4 import BeautifulSoup
soup = BeautifulSoup(response.text)
```

In this example, we'll look at a news ticker. It contains single-line news reports from the last few days and links to the corresponding articles. We'll use this to create a structured `DataFrame` which contains the title, date and time of the message in the columns. To do this, we have to extract these elements from the document. We got to know `my_soup.findall()` as a way to find different elements. With `my_soup.findall()` you can iterate through specified tags. For example, if you iterate through the headings specified by the tag `<h2>`, it looks like this

In [5]:
```
for element in soup.find_all('h2'):
    print(element.attrs)
```

```
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading']}
{'class': ['a-container-header__heading']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
{'class': ['a-container-header__heading', 'a-container-header__heading--small']}
```

We've output the attributes for the headings here. They are contained in `my_element.attrs`. We can see that the elements have the `'class'` attribute. This is often used in HTML to define similar elements. We can take advantage of this to access elements. If you pass a *dict* with the structure `{'my_attribute':'my_value'}` to the `attrs` parameter of `my_soup.findall()`, it will only output the elements which have the value `'my_value'` in `'my_attribute'`. For example, if we pass `{'class':'my_class'}` we will get a list back that only contains items with the class we are looking for.

In [6]:
```
soup.find_all('h2', attrs={'class':'a-container-header__heading'})
#root.find_class('a--header__heading')
```

```
Out[6]: [<h2 class="a-container-header__heading a-container-header__heading--small">
               Donnerstag, 01.04.2021
             </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
               Mittwoch, 31.03.2021
             </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
               Dienstag, 30.03.2021
             </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
               Montag, 29.03.2021
             </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
               Sonntag, 28.03.2021
             </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
               Samstag, 27.03.2021
             </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
               Freitag, 26.03.2021
             </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
         <span class="a-container-header__logo a-container-header__logo--foto">
                   c't Fotografie
                 </span>
         </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
               Hören Sie von uns
             </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
               Video-Tipp
             </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
         <span class="a-container-header__logo a-container-header__logo--security">
                   heise Security
                 </span>
         </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
               heise tipps+tricks
             </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
         <span class="a-container-header__logo a-container-header__logo--developer">
                   heise Developer
                 </span>
         </h2>,
         <h2 class="a-container-header__heading a-container-header__heading--small">
               Spielen bei heise online
             </h2>,
         <h2 class="a-container-header__heading">
         <span class="a-container-header__logo a-container-header__logo--hardware_hacks a-con
        tainer-header__logo--margin">
                   Make
                 </span>
         <span class="a-container-header__slug">
                   Das aktuelle Sonderheft
                 </span>
         </h2>,
         <h2 class="a-container-header__heading">
         <span class="a-container-header__logo a-container-header__logo--events a-container-h
        eader__logo--margin">
```

```
            heise Events
          </span>
 <span class="a-container-header__slug">
            secIT Special
          </span>
 </h2>,
 <h2 class="a-container-header__heading a-container-header__heading--small">
        DSL-Vergleich
      </h2>,
 <h2 class="a-container-header__heading a-container-header__heading--small">
 <span class="a-container-header__logo a-container-header__logo--ho a-container-heade
r__logo--margin">
            heise online
          </span>
 <span class="a-container-header__slug">
            Top-News der Redaktion von heise online
          </span>
 </h2>]
```

Another attribute that can help with your search is `'id'`. Each `'id'` only occurs once on a website. So if you see that the information you are looking for is marked with an `'id'`, you can search for it directly. `my_soup.find(attrs={'id':'my_id'})` from *Reading Websites (Modul 2 Chapter 1)*.

For example, the following cell searches the page for an advertisement with job offers:

In [7]:
```
soup.find(attrs={'id':'jobware-teaser'})
#root.get_element_by_id('jobware-teaser')  # get the element with the id 'jobware-teas
```

```
Out[7]: <section id="jobware-teaser">
<header><a href="https://web.archive.org/web/20210401004837/http://jobs.heise.de/" na
me="spalte.ho.jobs.logo">
<figure>
<img alt="Logo heise Jobs" height="62" loading="lazy" src="//web.archive.org/web/2021
0401004837im_/http://1.f.ix.de/icons/svg/logos/svg/heise_jobs.svg" width="201"/>
<figcaption>Der Stellenmarkt auf heise online.</figcaption>
</figure>
</a></header>
<ul>
<li>
<a href="https://web.archive.org/web/20210401004837/https://jobs.heise.de/Job/IT-Syst
em-Engineer-w-m-d-Microsoft-SCCM-und-Intune.530285925.html" name="spalte.ho.jobs.job
1" target="heisejobs">
<p>
<b class="teaser-title">IT System Engineer (w/m/d) Microsoft SCCM und Intune</b>
<span class="jobware-teaser__company">Bechtle AG</span>
</p>
</a>
</li>
<li>
<a href="https://web.archive.org/web/20210401004837/https://jobs.heise.de/Job/IT-Anwe
ndungssupport-w-m-d.530278665.html" name="spalte.ho.jobs.job2" target="heisejobs">
<p>
<b class="teaser-title">IT Anwendungssupport (w/m/d)</b>
<span class="jobware-teaser__company">Bechtle AG</span>
</p>
</a>
</li>
<li>
<a href="https://web.archive.org/web/20210401004837/https://jobs.heise.de/Job/IT-Cons
ultant-w-m-d-Microsoft-Security-und-Intune.530255755.html" name="spalte.ho.jobs.job3"
target="heisejobs">
<p>
<b class="teaser-title">IT Consultant (w/m/d) Microsoft Security und Intune</b>
<span class="jobware-teaser__company">Bechtle AG</span>
</p>
</a>
</li>
</ul>
</section>
```

The elements can contain other elements. So they are just as nested as the HTML code itself. If you want to get the text content of an element, you also get the content of any sub-elements as well. Sometimes this is very useful, for example when reading table rows. For this purpose, the elements contain the attribute `my_element.text`.

```
In [8]: print(soup.find(attrs={'id':'jobware-teaser'}).text) # get the text content of the cho
```

```
Der Stellenmarkt auf heise online.




IT System Engineer (w/m/d) Microsoft SCCM und Intune
Bechtle AG




IT Anwendungssupport (w/m/d)
Bechtle AG




IT Consultant (w/m/d) Microsoft Security und Intune
Bechtle AG
```

You can use the *Page Inspector* or the developer tools to find out how the news reports are organized.

All the reports are contained in one element with class `class='archive'`. The messages, each belonging to a date, are organized in elements of the class `a-article-teaser__preceded-kicker-container`. The date is specified in it by the tag `<time>` and the attribute `datetime`. The elements with the tag `<h1>` then contain the title about the report.

We'll also store the time and number of comments. We'll extract these from the text later. In the following cell we'll iterate through the elements mentioned. We'll use the three methods you've just recapped to do this. We'll store the selected contents in the lists `news_datetimess`, `news_headlines` and `news_comments`.

In [9]:
```
news_datetimes = []  # list for the timestamps
news_headlines = []  # list for the titles
news_comments = [] #list for the number of comments

archive = soup.find(attrs={'class':'archive'})  # get the website element, which conta

for element in archive.find_all(attrs={'class':'a-article-teaser__preceded-kicker-cont
    # extract information
    headline = element.find('h1').text.strip() # the headline is in the <h1> tag. it c
    datetime = element.find('time').attrs['datetime'] # the datetime is the datetime c

    # some feeds have comments some don't
    comments_raw = element.find(attrs={"class":"a-article-meta__comments"})
    if comments_raw:
        comments = comments_raw.text.replace('\n','')
        comments = comments.replace('     ','')
#           print(comments)
    else:
        comments = '0 Kommentare'

    # store information
    news_datetimes.append(datetime)  # we want to have the date for every headline, th
    news_headlines.append(headline)  # the headline is nice and clean in the title att
    news_comments.append(comments)  # save the complete content for later processing
```

Now we have three lists with the same number of entries:

In [10]:

```
print(len(news_datetimes), len(news_headlines), len(news_comments))
```

```
256 256 256
```

The entries in the same positions belong to the same news reports. You can summarize lists like these with `zip()`. You pass `zip()` two or more lists. Then `zip()` creates a list of tuples containing the elements of the list, so `zip()` follows this pattern:

zip pattern

`zip()` does not directly create a list, but an object you can use for a `for` loop, just like the `range()` function. Run the following code cell to try it out:

In [11]:

```
my_list_1 = ['A', 'B', 'C']
my_list_2 = [1, 2, 3]

for tup in zip(my_list_1, my_list_2):  # zip the lists and print the resulting tuples
    print(tup)
```

```
('A', 1)
('B', 2)
('C', 3)
```

If we get lists or tuples in the run variable in a `for` loop, we can use the correct number instead of just one run variable. Python then understands this and splits up the list or tuple. Run the following code cell, to look at it:

In [12]:

```
for letter, number in zip(my_list_1, my_list_2):  # we can split the tuples inside the
    print('The letter is {} and the number is {}'.format(letter, number))
```

```
The letter is A and the number is 1
The letter is B and the number is 2
The letter is C and the number is 3
```

Now we can print the related information side by side (just the first 15 elements here):

In [13]:

```
for date, headline, comment in zip(news_datetimes[:15],news_headlines[:15],news_commer
    print('#'*10)  # print a dividing line between the individual articles
    print(date, headline, comment)
```

```
##########
2021-04-01T01:04:00+02:00 heise online scherzt nicht mehr zum April, April   9  Komme
ntare
##########
2021-03-31T20:12:00 Von 2,4 bis 4 Zoll: Die kleinsten Smartphones mit Android 0 Komme
ntare
##########
2021-03-31T19:40:00+02:00 Neues Jugendschutzgesetz: USK erwartet teils höhere Alterse
instufungen   18  Kommentare
##########
2021-03-31T19:02:00+02:00 Xiaomi will Elektroauto bauen   11  Kommentare
##########
2021-03-31T19:00:00 Messenger-Backup: So können Sie WhatsApp-Chats ohne Account archi
vieren   2  Kommentare
##########
2021-03-31T18:36:00+02:00 Erneuerbare Energien: Apple holt mehr Zulieferer ins Boot
7  Kommentare
##########
2021-03-31T17:36:00 Oneplus 9 Pro: Spitzen-Smartphone mit Hasselblad-Kamera im Test 0
Kommentare
##########
2021-03-31T17:33:00+02:00 Tesla in Grünheide: Behörde erteilt weitere Genehmigung für
Gigafactory   31  Kommentare
##########
2021-03-31T17:23:00+02:00 Apple steigt bei Herausforderer großer Musiklabels ein   2
Kommentare
##########
2021-03-31T17:02:00+02:00 Small Cells: Berliner Litfaßsäulen bekommen LTE von der Tel
ekom   17  Kommentare
##########
2021-03-31T17:01:00+02:00 Cochstedt: Nationales Testzentrum soll die Drohnenökonomie
beflügeln 0 Kommentare
##########
2021-03-31T17:00:00+02:00 Kurz informiert: Stromversorgung, Chrome, Autoindustrie, In
genuity 0 Kommentare
##########
2021-03-31T16:26:00+02:00 Hippe Platinen von Aisler   18  Kommentare
##########
2021-03-31T16:22:00+02:00 Deutsche und Schweizerische Corona-Warn-Apps sind jetzt int
eroperabel   9  Kommentare
##########
2021-03-31T16:19:00+02:00 Microsoft Teams für Administratoren: Das XXL-Webinar von He
ise   2  Kommentare
```

Congratulations: You have recapped how to access websites and read them. You also recapped how HTML is structured. You also used the useful `zip()` function.

Regular expressions

Data is often contained in texts that are otherwise unstructured. This is where *regular expressions* - or *regexes* come in handy. We learned about these in the lesson *Extracting Data from Texts (Chapter 1)*. The `re` module helps us to look through texts for specific search patterns.

In [14]:
```python
import re
```

These search patterns are the *regexes*. The following tables contain the most important codes:

Generalizations

Expression	Meaning
.	matches any character (except line break)
\d	digit
\w	letter, digit or underscore
\s	whitespace: space, tab, line break
\D, \W, \S	negation of \d, \w und \s

Repetitions

Expression	Meaning
{min, max}	Repetition of the preceding expression at least min-times and no more than max-times
{3}	Repeat the preceding expression exactly 3 times
+	Repeat the preceding expression at least once
+	Repeat the preceding expression at least zero times

Combinations

Expression	Meaning
...|...	"or" operator: Matches the expression to the left or right of |
[...]	"or" operator: Matches an expression within the brackets
[a-z]	"from-to" operator: Matches a lower case letter between a and z
[A-Z0-9]	from-to-or operator: A capital letter between A and Z or a digit between 0 and 9 is searched for
[^...]	except: Only matches expressions that are not listed in the brackets
(...)	*capture group*: All expressions within the parentheses are searched for together and form a separate result
?	Optional: The preceding expression may apply exactly zero or one times
\	*Escape*: The following character is searched for as it is (e.g. \. searches for a period)

Boundaries

Expression	Meaning
^	Matches the beginning of the string
\$	Matches the end of the string

Expression	Meaning
\b	Matches the beginning or the end of a word

As you can see, a lot of characters are interpreted differently than how Python normally interprets them. As a result, it's good style to define *raw strings* for the search patterns. The following cell defines a search pattern for a point in time. A time is made up of four numbers, divided by a colon in the middle. The `r` before the actual *string* tells Python that the characters in the *string* should not interpreted, but should be used as they appear in the code.

In [15]:
```
expression = r'\d\d:\d\d'
```

With `re.findall()` we can use the search pattern to extract the times of the news reports from the previously stored time stamps.

In [16]:
```
re.findall(expression,news_datetimes[0])[0]
```

Out[16]:
```
'01:04'
```

For example, the number of comments on an article follows this search pattern:

In [17]:
```
expression = r'(\d+)\s+Kommentare'
re.findall(expression, news_comments[0])
```

Out[17]:
```
['9']
```

If the texts are in a `DataFrame`, `pandas` allows us to work directly with *regexes*. For this purpose we'll convert the lists into a `DataFrame` called `df_news`.

In [18]:
```
import pandas as pd

df_news = pd.DataFrame({'datetime': news_datetimes, 'headline': news_headlines, 'comme
df_news.head()
```

Out[18]:

	datetime	headline	comments
0	2021-04-01T01:04:00+02:00	heise online scherzt nicht mehr zum April, April	9 Kommentare
1	2021-03-31T20:12:00	Von 2,4 bis 4 Zoll: Die kleinsten Smartphones ...	0 Kommentare
2	2021-03-31T19:40:00+02:00	Neues Jugendschutzgesetz: USK erwartet teils h...	18 Kommentare
3	2021-03-31T19:02:00+02:00	Xiaomi will Elektroauto bauen	11 Kommentare
4	2021-03-31T19:00:00	Messenger-Backup: So können Sie WhatsApp-Chats...	2 Kommentare

`pandas` offers the `my_Series.str.extract()` to use a *regex* on all the elements in a column. You learned about this in *Regular Expressions in DataFrames (Module 2 Chapter 1)*. The parameter `expand=false` specifies that the output should also be a `Series` and not a `DataFrame`. So it's easier to store the result as a new column. First let's separate the `'date'` column into the date and time. The search pattern has to be passed as a *capture group*, i.e. enclosed by round parentheses.

In [19]:
```
# find all weekdays
expression_date = r'(\d{4}-\d{2}-\d{2})'  # date is like YYYY-MM-DD
df_news.loc[:, 'weekday'] = df_news.loc[:, 'datetime'].str.extract(expression_date, ex

#find all times
expression_time = r'(\d{2}\:\d{2})'  # the time follows the format hh:mm, {} tells reg
df_news.loc[:, 'time'] = df_news.loc[:, 'datetime'].str.extract(expression_time, expar

df_news.head()
```

Out[19]:

	datetime	headline	comments	weekday	time
0	2021-04-01T01:04:00+02:00	heise online scherzt nicht mehr zum April, April	9 Kommentare	2021-04-01	01:04
1	2021-03-31T20:12:00	Von 2,4 bis 4 Zoll: Die kleinsten Smartphones ...	0 Kommentare	2021-03-31	20:12
2	2021-03-31T19:40:00+02:00	Neues Jugendschutzgesetz: USK erwartet teils h...	18 Kommentare	2021-03-31	19:40
3	2021-03-31T19:02:00+02:00	Xiaomi will Elektroauto bauen	11 Kommentare	2021-03-31	19:02
4	2021-03-31T19:00:00	Messenger-Backup: So können Sie WhatsApp-Chats...	2 Kommentare	2021-03-31	19:00

Next, we'll use the *datetime* functionality of Pandas to extract the day of the week from the date in `'weekday'`.

In [20]:
```
df_news['weekday'] = pd.to_datetime(df_news['weekday']).dt.day_name()
df_news.head()
```

Out[20]:

	datetime	headline	comments	weekday	time
0	2021-04-01T01:04:00+02:00	heise online scherzt nicht mehr zum April, April	9 Kommentare	Thursday	01:04
1	2021-03-31T20:12:00	Von 2,4 bis 4 Zoll: Die kleinsten Smartphones ...	0 Kommentare	Wednesday	20:12
2	2021-03-31T19:40:00+02:00	Neues Jugendschutzgesetz: USK erwartet teils h...	18 Kommentare	Wednesday	19:40
3	2021-03-31T19:02:00+02:00	Xiaomi will Elektroauto bauen	11 Kommentare	Wednesday	19:02
4	2021-03-31T19:00:00	Messenger-Backup: So können Sie WhatsApp-Chats...	2 Kommentare	Wednesday	19:00

Last but not least, we'll find the number of comments. We'll use the capture group just for the numbers here. The word `'comment'` is taken into account in the search, but is not included in the result. Then the column is converted into a numeric column.

In [21]:
```
#find the number of comments
expression_comments = r'(\d+)\s+Kommentare'  # select the number of comments. Whitespa
df_news.loc[:, 'comments'] = df_news.loc[:, 'comments'].str.extract(expression_comment
df_news.loc[:, 'comments'] = pd.to_numeric(df_news.loc[:, 'comments'])  # tranform to
df_news.head()
```

Out[21]:

	datetime	headline	comments	weekday	time
0	2021-04-01T01:04:00+02:00	heise online scherzt nicht mehr zum April, April	9	Thursday	01:04
1	2021-03-31T20:12:00	Von 2,4 bis 4 Zoll: Die kleinsten Smartphones ...	0	Wednesday	20:12
2	2021-03-31T19:40:00+02:00	Neues Jugendschutzgesetz: USK erwartet teils h...	18	Wednesday	19:40
3	2021-03-31T19:02:00+02:00	Xiaomi will Elektroauto bauen	11	Wednesday	19:02
4	2021-03-31T19:00:00	Messenger-Backup: So können Sie WhatsApp-Chats...	2	Wednesday	19:00

Now we have taken the information from the Heise news ticker and given it a structured format. This means that this data can now be used for further analysis.

Congratulations: You have recapped regular expressions. You have recapped how you can use them in DataFrames to extract interesting data from columns of text.

Remember:

- Read a web page with `requests.get(website_url)` and `BeautifulSoup(my_response.text)`
- Find elements in HTML code with `my_soup.find_all()` or `my_soup.find()`
- Output all text content within an element and its sub-elements with `my_element.text`
- Combine lists into a list of tuples with `for my_tuple in zip(my_list_1, my_List_2)`
- Extract text in `DataFrame` columns with `my_series.str.extract()` using *regexes*

Do you have any questions about this exercise? Look in the forum to see if they have already been discussed.

Found a mistake? Contact Support at support@stackfuel.com.