Web Scraping

Module 2 | Chapter 1 | Notebook 3

Machine-learning models generally need data to be in a structured format. However, the data is often not available in this form and you have to prepare it first. In this exercise we'll continue reading out web pages. This time we won't just read out one page, but several. By the end of this lesson:

- You will know what to look for for when web scraping
- You will be able to easily iterate through the elements of several lists
- You will be able to search specifically for classes in HTML

Gathering information from multiple websites

Scenario: The Taiwanese investor from *Module 1, Chapter 1* gets in touch with you again. This time he's not interested in house prices. Instead, he wants to invest in DAX-listed companies. However, he doesn't yet have enough data on the companies to make an informed decision. So he asks you to collect publicly available data on the companies and to deliver it to him in a structured format.

In the last lesson you found out which companies are listed on the DAX stock index. We read out the names as well as some additional data from the website *https://en.wikipedia.org/wiki/DAX*. However, there is still some information that we have not considered: the links in the table. There are two types. Some lead to the relevant company page on Wikipedia. The others lead to the relevant company page on the Frankfurt Stock Exchange. These links were ignored by `my_element.text` because the link itself is in the attributes.

Retrieve the HTML code of the page *https://en.wikipedia.org/wiki/DAX* with `requests`. Print an error message if the request was not successful. Then use `beautifulsoup` to parse the HTML code. Save the resulting variable as `soup`. Use `soup` to get the table that has the `id` `'constituents'`. Store the table as a variable named `table`.

In [1]:
```
# connect to the website
import requests
from bs4 import BeautifulSoup  # import parser

website_url = 'https://en.wikipedia.org/wiki/DAX'
response = requests.get(website_url)  # get content of website
response.raise_for_status()  # give error if request failed

# get the table
soup = BeautifulSoup(response.text) # parse website
table = soup.find(id='constituents')  # select the table using the id
```

Now let's take a look at the links from the table. On the left are the `<a>` tags (a stands for *anchor*). Print the attributes of all the links in the table.

In [2]:
```
for element in table.find_all('a'):
    print(element.attrs)
```

{'href': '/wiki/Prime_Standard', 'title': 'Prime Standard'}
{'href': '#endnote_1'}
{'href': '/wiki/File:Adidas-group-logo-fr.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Adidas', 'title': 'Adidas'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=ADS.DE'}
{'href': '/wiki/File:Airbus_Logo_2017.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Airbus', 'title': 'Airbus'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=AIR.DE'}
{'href': '/wiki/File:Allianz.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Allianz', 'title': 'Allianz'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=ALV.DE'}
{'href': '/wiki/File:BASF-Logo_bw.svg', 'class': ['mw-file-description']}
{'href': '/wiki/BASF', 'title': 'BASF'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=BAS.DE'}
{'href': '/wiki/File:Logo_Bayer.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Bayer', 'title': 'Bayer'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=BAYN.DE'}
{'href': '/wiki/File:Beiersdorf_Logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Beiersdorf', 'title': 'Beiersdorf'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=BEI.DE'}
{'href': '/wiki/File:BMW_logo_(gray).svg', 'class': ['mw-file-description']}
{'href': '/wiki/BMW', 'title': 'BMW'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=BMW.DE'}
{'href': '/wiki/File:Brenntag_Logo_2022.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Brenntag', 'title': 'Brenntag'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=BNR.DE'}
{'href': '/wiki/File:Commerzbank_(2009).svg', 'class': ['mw-file-description']}
{'href': '/wiki/Commerzbank', 'title': 'Commerzbank'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=CBK.DE'}
{'href': '/wiki/File:Continental_wordmark.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Continental_AG', 'title': 'Continental AG'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=CON.DE'}
{'href': '/wiki/File:Covestro_Logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Covestro', 'title': 'Covestro'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=1COV.DE'}
{'href': '/wiki/File:Daimler_Truck_Logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Daimler_Truck', 'title': 'Daimler Truck'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=DTG.DE'}
{'href': '/wiki/File:Deutsche_Bank_logo_without_wordmark.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Deutsche_Bank', 'title': 'Deutsche Bank'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=DBK.DE'}
{'href': '/wiki/Deutsche_B%C3%B6rse', 'title': 'Deutsche Börse'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=DB1.DE'}
{'href': '/wiki/File:Deutsche_Post_DHL.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Deutsche_Post', 'title': 'Deutsche Post'}

{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=DHL.DE'}
{'href': '/wiki/File:Deutsche_Telekom_2022.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Deutsche_Telekom', 'title': 'Deutsche Telekom'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=DTE.DE'}
{'href': '/wiki/File:Logo_E.ON.svg', 'class': ['mw-file-description']}
{'href': '/wiki/E.ON', 'title': 'E.ON'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=EOAN.DE'}
{'href': '/wiki/File:Fresenius.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Fresenius_SE', 'class': ['mw-redirect'], 'title': 'Fresenius SE'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=FRE.DE'}
{'href': '/wiki/File:HannoverRe.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Hannover_Re', 'title': 'Hannover Re'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=HNR1.DE'}
{'href': '/wiki/File:HeidelbergCement_2022_logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/HeidelbergCement', 'class': ['mw-redirect'], 'title': 'HeidelbergCement'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=HEI.DE'}
{'href': '/wiki/File:Henkel-Logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Henkel', 'title': 'Henkel'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=HEN3.DE'}
{'href': '/wiki/File:Infineon-Logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Infineon_Technologies', 'title': 'Infineon Technologies'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=IFX.DE'}
{'href': '/wiki/File:Mercedes-Benz_Group_black.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Mercedes-Benz_Group', 'title': 'Mercedes-Benz Group'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=MBG.DE'}
{'href': '/wiki/File:Logo_Merck_KGaA_2015.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Merck_Group', 'title': 'Merck Group'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=MRK.DE'}
{'href': '/wiki/File:MTU_Aero_Engines_Logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/MTU_Aero_Engines', 'title': 'MTU Aero Engines'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=MTX.DE'}
{'href': '/wiki/File:M%C3%BCnchener_R%C3%BCck_logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Munich_Re', 'title': 'Munich Re'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=MUV2.DE'}
{'href': '/wiki/Porsche', 'title': 'Porsche'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=P911.DE'}
{'href': '/wiki/File:Porsche_SE_Logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Porsche_SE', 'title': 'Porsche SE'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frankfurt.de/en/equities/search/result?name_isin_wkn=PAH3.DE'}
{'href': '/wiki/File:Qiagen.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Qiagen', 'title': 'Qiagen'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank

```
furt.de/en/equities/search/result?name_isin_wkn=QIA.DE'}
{'href': '/wiki/File:Rheinmetall_Logo_2021.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Rheinmetall', 'title': 'Rheinmetall'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank
furt.de/en/equities/search/result?name_isin_wkn=RHM.DE'}
{'href': '/wiki/File:RWE_Logo_2020.svg', 'class': ['mw-file-description']}
{'href': '/wiki/RWE', 'title': 'RWE'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank
furt.de/en/equities/search/result?name_isin_wkn=RWE.DE'}
{'href': '/wiki/File:SAP_2011_logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/SAP', 'title': 'SAP'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank
furt.de/en/equities/search/result?name_isin_wkn=SAP.DE'}
{'href': '/wiki/File:Sartorius-Logo-2020.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Sartorius_AG', 'title': 'Sartorius AG'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank
furt.de/en/equities/search/result?name_isin_wkn=SRT3.DE'}
{'href': '/wiki/File:Siemens_AG_logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Siemens', 'title': 'Siemens'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank
furt.de/en/equities/search/result?name_isin_wkn=SIE.DE'}
{'href': '/wiki/File:Siemens_Energy_logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Siemens_Energy_AG', 'class': ['mw-redirect'], 'title': 'Siemens Energ
y AG'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank
furt.de/en/equities/search/result?name_isin_wkn=ENR.DE'}
{'href': '/wiki/File:Siemens_Healthineers_logo.svg', 'class': ['mw-file-descriptio
n']}
{'href': '/wiki/Siemens_Healthineers', 'title': 'Siemens Healthineers'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank
furt.de/en/equities/search/result?name_isin_wkn=SHL.DE'}
{'href': '/wiki/File:Symrise_Logo_2016.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Symrise', 'title': 'Symrise'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank
furt.de/en/equities/search/result?name_isin_wkn=SY1.DE'}
{'href': '/wiki/File:Volkswagen_Group.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Volkswagen_Group', 'title': 'Volkswagen Group'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank
furt.de/en/equities/search/result?name_isin_wkn=VOW3.DE'}
{'href': '/wiki/File:Vonovia_Logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Vonovia', 'title': 'Vonovia'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank
furt.de/en/equities/search/result?name_isin_wkn=VNA.DE'}
{'href': '/wiki/File:Zalando_201x_logo.svg', 'class': ['mw-file-description']}
{'href': '/wiki/Zalando', 'title': 'Zalando'}
{'rel': ['nofollow'], 'class': ['external', 'text'], 'href': 'http://www.boerse-frank
furt.de/en/equities/search/result?name_isin_wkn=ZAL.DE'}
```

You can see from the attributes that there are different types of links. Some links are to Wikipedia pages. We can identify these by the fact that their `href` attribute begins with `'/wiki'`. `href` stands for *hypertext reference* and indicates the address that it links to. The links are divided into two categories. One category includes image files (the companies' logos), which can be identified by the `class 'image'`. The other category leads to the company pages within Wikipedia. These links only have the attribute `title`. There are also links that link to external content, in our case websites of the Frankfurt Stock Exchange. These links are marked with the class `'external text'`. They also have the attribute `rel="nofollow"`. This simply means that these links should not have any influence on the ranking in search results.

Then there are links that start with a hash `'#'`. These redirect to other sections on the same page.

The links we are interested in are those to the company websites within Wikipedia. Create a `list` called `links_wiki`. Fill this with the values of the `href` attribute for the relevant links. Then print the first five addresses in `links_wiki`.

Tip: The links you are looking for do not contain the `class` attribute. You can check whether a certain *key* is contained in a *dictionary* with `if not 'key' in my_dict`.

In [3]:
```
links_wiki = [link.attrs['href'] for link in table.find_all('a') if not 'class' in lir
links_wiki[:5]
```

Out[3]:
```
['/wiki/Prime_Standard',
 '#endnote_1',
 '/wiki/Adidas',
 '/wiki/Airbus',
 '/wiki/Allianz']
```

Among other things, `links_wiki` contains the links `'/wiki/Prime_Standard'` and `'#endnote_1'`. These don't link to company sites, so we don't need them. If you also have links that don't lead to company sites, you can delete them using `my_list.remove()`. Apply this to the links `'/wiki/Prime_Standard'` and `'#endnote_1'` and print the first 5 rows of `links_wiki` again.

In [4]:
```
links_wiki.remove('/wiki/Prime_Standard')
links_wiki.remove('#endnote_1')
links_wiki[:5]
```

Out[4]:
```
['/wiki/Adidas', '/wiki/Airbus', '/wiki/Allianz', '/wiki/BASF', '/wiki/Bayer']
```

Now we have some links in a `list`. We'll follow these links and read the corresponding pages automatically. Searching entire web page structures is called *web crawling*. There are a few things to consider. On the one hand, some website operators don't want people to read out their websites. On the other hand, you can't make too many requests in a short time. This can cause the servers to become overloaded and the website to respond slowly. People generally don't want their websites to be slowed down in this way. Websites protect themselves from this by blocking the IP addresses of computers that request too many hits. Sometimes traps are also set for *web crawlers*. These traps, known as *honeypots*, are links that are not displayed on the browser interface, but *web crawlers* that simply follow all the links on the website still follow them and are then blocked.

The best thing to do is to observe the *robots.txt* of the website you want to read. This tells you whether you are allowed to read out the website automatically or whether they don't want you to do this. Its address path is always the base address of the web page with */robots.txt* at the end. So in our case, it's *https://en.wikipedia.org/robots.txt*.

If you take a look at Wikipedia's *robots.txt*, you will see the following bits of text:

```
User-agent: Zao
Disallow: /
```

A lot of reputable *web crawlers* pass their name as `User-agent` during an HTTP request. In this example, Wikipedia does not want a *web crawler* named `Zao` to search its page. That's why it says `Disallow: /`. This means that all the addresses starting from the base address are not allowed. If there is nothing next to `Disallow`, then all subpages are allowed.

What instructions do we need to follow? `requests` identifies itself by default with the module name and relevant version. In our case this is `User-agent: python-requests/2.21.0`. This name is not specifically indicated in the *robots.txt*. All *web crawlers* that are not explicitly listed should follow the instructions listed under `User-agent: *`. The star stands for all other names. You should see something like:

```
User-agent: *
Allow: /w/api.php?action=mobileview&
Allow: /w/load.php?
Allow: /api/rest_v1/?doc
Disallow: /w/
Disallow: /api/
Disallow: /trap/
Disallow: /wiki/Special:
Disallow: /wiki/Spezial:
Disallow: /wiki/Spesial:
Disallow: /wiki/Special%3A
Disallow: /wiki/Spezial%3A
Disallow: /wiki/Spesial%3A
```

It lists some paths that are explicitly allowed and not allowed. All others are allowed. Our link addresses are not listed here, so we can safely continue.

Wikipedia has a very consistent structure. So the relevant pages for the companies have a similar structure. Visit the page *https://en.wikipedia.org/wiki/Adidas*. You'll see a table on the right-hand side containing some key figures such as *Revenue*. How can you identify this table? Check it with the *Page Inspector* or the developer tools or similar tools of your browser. You'll probably see that the table has the attribute `class="infobox vcard"`. The other pages have a similar table with the same class. We can now use this to read the table.

It's a good idea to just use the first link in `links_wiki` start with, in order to find a procedure for extracting the information we want from this link. Create a variable `link` and assign it the first page stored in `links_wiki`.

Now connect to the page stored in `link`. Note that you have to add the base address `'https://en.wikipedia.org'` before it. Store this as `base_url`. Also store the response as `response` and check the status code.

In [5]:
```
base_url =  'https://en.wikipedia.org'
link = links_wiki[0]
```

```
print(link)
response = requests.get(base_url+link)
response.raise_for_status()
```

```
/wiki/Adidas
```

If the query was successful, you can now search the contents of the page with `pandas.read_html()` and extract the table with the *class* `'infobox vcard'`.

Use `pd.read_html()` together with `'infobox vcard'` to extract the table you are looking for in `response.text`. Remember that `pd.read_html()` returns a normal list. But since the web page only has one table with the class `'infobox vcard'`, this list has only contains one element. Select it by using its index and save it as `df_table`.

In [6]:
```
import pandas as pd
df_table = pd.read_html(response.text, attrs={'class': 'infobox vcard'})[0]
df_table
```

Out[6]:

	0	1
0	NaN	NaN
1	Factory outlet in Herzogenaurach, Germany	Factory outlet in Herzogenaurach, Germany
2	Formerly	Gebrüder Dassler Schuhfabrik (1924–1949)
3	Company type	Public (AG)
4	Traded as	FWB: ADSDAX component
5	Industry	Textile, footwear
6	Founded	July 1924; 99 years agoHerzogenaurach, Germany...
7	Founder	Adolf Dassler
8	Headquarters	Herzogenaurach, Bavaria, Germany
9	Area served	Worldwide
10	Key people	.mw-parser-output .plainlist ol,.mw-parser-out...
11	Products	Apparel, footwear, sportswear, sports equipmen...
12	Revenue	€21.915 billion (2018)[3]
13	Operating income	€2.368 billion (2018)[3]
14	Net income	€1.702 billion (2018)[3]
15	Total assets	€15.612 billion (2018)[3]
16	Total equity	€6.364 billion (2018)[3]
17	Number of employees	57,016 (2018)[3]
18	Subsidiaries	Adidas RuntasticMatix
19	Website	adidas.com

Now we can see the structure of the table very well. The first column contains profile categories and the second column contains the corresponding values for the company. However, tables in

HTML elements often contain empty rows or columns to create spaces between elements. We should remove those. In particular, rows without an entry in the first column are of no use to us.

Modify `df_table` so that rows without an entry in the first column are deleted and that only the first two columns are selected afterwards.

Tip: `df_table` only has two columns, so selecting them seems unnecessary here. However, we will reuse our code later on, so it's a good idea to select them here as well.

In [7]:
```
df_table = df_table.loc[~df_table[0].isna(),:]
df_table = df_table.iloc[:, :2]
df_table
```

Out[7]:

	0	1
1	Factory outlet in Herzogenaurach, Germany	Factory outlet in Herzogenaurach, Germany
2	Formerly	Gebrüder Dassler Schuhfabrik (1924–1949)
3	Company type	Public (AG)
4	Traded as	FWB: ADSDAX component
5	Industry	Textile, footwear
6	Founded	July 1924; 99 years agoHerzogenaurach, Germany...
7	Founder	Adolf Dassler
8	Headquarters	Herzogenaurach, Bavaria, Germany
9	Area served	Worldwide
10	Key people	.mw-parser-output .plainlist ol,.mw-parser-out...
11	Products	Apparel, footwear, sportswear, sports equipmen...
12	Revenue	€21.915 billion (2018)[3]
13	Operating income	€2.368 billion (2018)[3]
14	Net income	€1.702 billion (2018)[3]
15	Total assets	€15.612 billion (2018)[3]
16	Total equity	€6.364 billion (2018)[3]
17	Number of employees	57,016 (2018)[3]
18	Subsidiaries	Adidas RuntasticMatix
19	Website	adidas.com

Now that you've determined a fixed number of columns for `df_table`, you can fill the table with information. We've identified the first column as profile categories. So we can call it `'key'`. The second column contains specific values for the company whose Wikipedia page you've just landed on. So it makes sense to name this column with the company name.

You can extract the name of the company from `link`. Use `my_string.split()` to do this. Store the name in the variable `company_name` and then print it.

In [8]:
```
# extract company name from link
company_name = link.split('/')[-1]
company_name
```

Out[8]:
```
'Adidas'
```

Now you can rename the columns in `df_table` as `'key'` and `company_name`. Afterwards you should set the values in the `'key'` column as the index. The result should be a `DataFrame` with just one column about one single company.

In [9]:
```
df_table.columns=['key',company_name]
df_table = df_table.set_index('key')
df_table
```

Out[9]:

key	Adidas
Factory outlet in Herzogenaurach, Germany	Factory outlet in Herzogenaurach, Germany
Formerly	Gebrüder Dassler Schuhfabrik (1924–1949)
Company type	Public (AG)
Traded as	FWB: ADSDAX component
Industry	Textile, footwear
Founded	July 1924; 99 years agoHerzogenaurach, Germany...
Founder	Adolf Dassler
Headquarters	Herzogenaurach, Bavaria, Germany
Area served	Worldwide
Key people	.mw-parser-output .plainlist ol,.mw-parser-out...
Products	Apparel, footwear, sportswear, sports equipmen...
Revenue	€21.915 billion (2018)[3]
Operating income	€2.368 billion (2018)[3]
Net income	€1.702 billion (2018)[3]
Total assets	€15.612 billion (2018)[3]
Total equity	€6.364 billion (2018)[3]
Number of employees	57,016 (2018)[3]
Subsidiaries	Adidas RuntasticMatix
Website	adidas.com

Now we'll do the same steps for all the company pages from `links_wiki`. However we want to make sure that we don't get blocked by making too many requests in a short time period. So we have to wait a certain amount of time between each request. You can do this by using the `time` module. Import it and use the `time.sleep()` function. You pass it the number of full

seconds you want your program to wait. Try 5 seconds. Then print the sentence `5 seconds passed` so you can see when the cell has finished.

In [10]:
```
import time
time.sleep(5)
print('5 seconds passed')
```

```
5 seconds passed
```

But how long should you typically wait? *Web crawlers* from Yahoo and Microsoft follow a specification in the *robots.txt* file. This default is indicated by `crawl-delay`. `Crawl-delay: 20` means that you should wait for 20 seconds, for example. But sometimes this value is not specified, as in our case. Then we recommend waiting 5 times as long as it takes for the server response to arrive. You can find this out with `my_response.elapsed.total_seconds()`. How long did it take you to get the response?

In [11]:
```
response.elapsed.total_seconds()
```

Out[11]:
```
0.05351
```

For us it took about 0.2 seconds. This value may vary slightly for you. This depends on the current server load and your own connection.

Now we've extracted all the information from the first entry of `links_wiki` and know how to define an acceptable time interval between two queries. It's best to combine the previous steps into one function, which you can then call in a loop. Define the function `load_link()`, with the following parameters:

```
def load_link(link, base_url='https://en.wikipedia.org'):
    """Extracts information in table with class 'infobox vcard' from a
Wikipedia link. Returns a single column `DataFrame` containing basic
company information.

    Args:
        link (str): Subpage of wikipedia.
        base_url (str): Wikipedia main page defaults to
'https://en.wikipedia.org'.

    Returns:
        DataFrame: Shape (x,2) with column name [company_name] and keys as
index.
    """
```

However, we have to take care of one thing, which will lead to an error message otherwise: Siemens (at location [-4]) has a duplicated *key* - "Headquarters". We have a *key-value* pair with *Headquarters: Headquarters* and then, a little later, *Headquarters: Munich, Germany[1]*. You will notice that the latter is the image caption. As a result, `pandas` doesn't know which of the two values it should write in the DataFrame. If we don't take care of the duplicate, we will get an error message `ValueError: Shape of passed values is (56, 28), indices imply`

`(55, 28)` when creating the DataFrame, i.e., that we want to put 56 rows into the DataFrame, but there are only 55 rows are available.

This web page information can always change over time. What we know now is that duplicate keys can occur repeatedly. We can learn a lesson from this: **Web pages change their content and structure regularly and sometimes unexpectedly.** What worked once may not always work. Because of this, *parsers* need to be adjusted regularly!

Your function should perform the following steps:

1. Connect to the respective page (remember to use `base_url` as well)
2. Check the status code
3. Select the table with the class `'infobox vcard'` and convert it into a `DataFrame`
4. Remove empty rows and limit the `DataFrame` to two columns
5. Rename columns to `'key'` and `company_name`
6. Set the `'key'` column as the index
7. Wait for five times the server response time
8. We need to make sure when we create the DataFrames for each company, that there are no duplicates in the index. The best way to do this is `df_table.loc[~df_table.index.duplicated(keep='last')]`. There's a boolean mask in here where all duplicated indexes in `df_table` are filtered by, except for the last index (`keep='last'`). The tilde `~` inverts a boolean mask, i.e. we only want to get back the rows of `df_table` that are not duplicates.
9. Return the `DataFrame`

Tip: Copy the relevant code from the previous code cells.

In [12]:
```
def load_link(link, base_url='https://en.wikipedia.org'):
    """Extracts information in table with class 'infobox vcard' from a wikipedia link.
    Returns a single column DataFrame with basic company information.

    Args:
        link (str): Subpage of wikipedia.
        base_url (str): Wikipedia main page defaults to 'https://en.wikipedia.org'.

    Returns:
        DataFrame: Shape (x,1) with column name [company_name] and keys as index.
    """
    #connect to page
    response = requests.get(base_url+link)

    #raise error if no connection
    response.raise_for_status()

    #extact table
    dfTable = pd.read_html(response.text,attrs={'class':'infobox vcard'})[0]

    #clean table
    dfTable = dfTable.loc[~dfTable[0].isna()]
    dfTable = dfTable.iloc[:,:2]
    company_name = link.split('/')[-1]
```

```
    dfTable.columns=['key',company_name]
    dfTable = dfTable.set_index('key')

    #add delay
    time.sleep(response.elapsed.total_seconds()*5)

    #remove duplicated values
    dfTable = dfTable.loc[~dfTable.index.duplicated(keep='last')]

    return dfTable
```

Test your function by executing the following line of code. If you don't get any error messages, you've done everything correctly.

In [13]:
```
# check for differences between manually cleaned df_table and return of load_link
(load_link(links_wiki[0]) != df_table).sum()
```

Out[13]:
```
Adidas    0
dtype: int64
```

Since the company pages all have a similar structure in the table, we can now iterate through `links_wiki` with a loop and pass each value to `load_link()`. Add the outputs to a list `company_dfs`.

Tip: Since the function waits each time, it might take a while to complete the loop. To check the progress, you can add a `print()` command to display the link currently being processed.

In [14]:
```
company_dfs = []
for link in links_wiki:
    print(link)
    df = load_link(link)
    company_dfs.append(df)
```

```
/wiki/Adidas
/wiki/Airbus
/wiki/Allianz
/wiki/BASF
/wiki/Bayer
/wiki/Beiersdorf
/wiki/BMW
/wiki/Brenntag
/wiki/Commerzbank
/wiki/Continental_AG
/wiki/Covestro
/wiki/Daimler_Truck
/wiki/Deutsche_Bank
/wiki/Deutsche_B%C3%B6rse
/wiki/Deutsche_Post
/wiki/Deutsche_Telekom
/wiki/E.ON
/wiki/Hannover_Re
/wiki/Henkel
/wiki/Infineon_Technologies
/wiki/Mercedes-Benz_Group
/wiki/Merck_Group
/wiki/MTU_Aero_Engines
/wiki/Munich_Re
/wiki/Porsche
/wiki/Porsche_SE
/wiki/Qiagen
/wiki/Rheinmetall
/wiki/RWE
/wiki/SAP
/wiki/Sartorius_AG
/wiki/Siemens
/wiki/Siemens_Healthineers
/wiki/Symrise
/wiki/Volkswagen_Group
/wiki/Vonovia
/wiki/Zalando
```

Now we just have to combine the individual DataFrames from `dfs` into to one big `DataFrame`. You can use the `pd.concat()` function to do this. This "sticks" all the rows or columns in a list of DataFrames together.

```
pd.concat(objs=list # list with DataFrames or Series
          axis=int # 0: concatenates all rows together one below the other
1: concatenates all columns side by side
         )
```

Use `pd.concat()` to create a `DataFrame` from `company_dfs` where each column corresponds to a DAX company and each row to a property.

In [15]:
```
df_company = pd.concat(company_dfs, axis=1)
df_company.head()
```

Out[15]:

key	Adidas	Airbus	Allianz	BASF	Bayer	Beiersdorf	
Factory outlet in Herzogenaurach, Germany	Factory outlet in Herzogenaurach, Germany	NaN	NaN	NaN	NaN	NaN	
Formerly	Gebrüder Dassler Schuhfabrik (1924–1949)	.mw-parser-output .plainlist ol,.mw-parser-out...	NaN	NaN	NaN	NaN	Mote
Company type	Public (AG)	Public (Societas Europaea)	Public (SE)	Public (Societas Europaea)	Public	Public (AG)	(Aktie
Traded as	FWB: ADSDAX component	BMAD: AIR Euronext Paris: AIR FWB: AIR CAC 40 ...	.mw-parser-output .plainlist ol,.mw-parser-out...	.mw-parser-output .plainlist ol,.mw-parser-out...	.mw-parser-output .plainlist ol,.mw-parser-out...	FWB: BEIDAX Component	FV
Industry	Textile, footwear	Aerospace, Defence	Financial services	Chemicals	Pharmaceuticals Chemicals Biotechnology Health...	Consumer goods	

5 rows × 37 columns

`df_company` now contains all the data in one place. However, it's still in the wrong format, because the observations (companies) are currently in the columns rather than in the rows. This data structure is referred to as a *wide* format. But the usual format is the *long* format, with the observations in the rows and attributes in the columns.
To change from a *wide* format to a *long* format, you simply need to transpose your `DataFrame` with `my_df.T`

In [16]:
```
df_company = df_company.T
df_company.head()
```

Out[16]:

key	Factory outlet in Herzogenaurach, Germany	Formerly	Company type	Traded as	Industry	Founded
Adidas	Factory outlet in Herzogenaurach, Germany	Gebrüder Dassler Schuhfabrik (1924–1949)	Public (AG)	FWB: ADSDAX component	Textile, footwear	July 1924; 99 years agoHerzogenaurach, Germany...
Airbus	NaN	.mw-parser-output .plainlist ol,.mw-parser-out...	Public (Societas Europaea)	BMAD: AIR Euronext Paris: AIR FWB: AIR CAC 40 ...	Aerospace, Defence	18 December 1970; 53 years ago
Allianz	NaN	NaN	Public (SE)	.mw-parser-output .plainlist ol,.mw-parser-out...	Financial services	05 February 1890; 134 years ago
BASF	NaN	NaN	Public (Societas Europaea)	.mw-parser-output .plainlist ol,.mw-parser-out...	Chemicals	6 April 1865; 159 years agoMannheim, Baden
Bayer	NaN	NaN	Public	.mw-parser-output .plainlist ol,.mw-parser-out...	Pharmaceuticals Chemicals Biotechnology Health...	1 August 1863; 160 years ago[1]

5 rows × 76 columns

The investor is mainly interested in the key financial figures. In this table, these are the values in the `'Revenue'`, `'Operating income'`, `'Net income'`, `'Total assets'` and `'Total equity'` columns. The best thing to do is to read out all the values first and see if we were able to extract enough entries. Then we take the values that we actually want to have.

Get the sum of the `NaN` values for each column in `df_company`.

In [17]:
```
df_company.isna().sum()
```

Out[17]:
```
key
Factory outlet in Herzogenaurach, Germany    36
Formerly                                     31
Company type                                  0
Traded as                                     1
Industry                                      1
                                             ..
Headquarters in Wolfsburg, Germany           36
Headquarters of Vonovia                      36
Areas served                                 36
Footnotes / referencesAnnual Report 2020     36
Headquarters in Berlin                       36
Length: 76, dtype: int64
```

We have a lot of missing values. But this is quite normal, because although the tables on Wikipedia have a similar structure, the information they contain is different. When *web crawling* several web pages, it's very rare for the information to match up in such a way that you don't get any missing values.

Now let's just look at the values that the investor is most interested in. Simply run the next code cell.

In [18]:
```
# show only cols investor is most interested in
cols = ['Revenue', 'Operating income', 'Net income', 'Total assets', 'Total equity']
df_company.loc[:,cols]
```

Out[18]:

key	Revenue	Operating income	Net income	Total assets	Total equity
Adidas	€21.915 billion (2018)[3]	€2.368 billion (2018)[3]	€1.702 billion (2018)[3]	€15.612 billion (2018)[3]	€6.364 billion (2018)[3]
Airbus	€65.45 billion (2023)	€4.60 billion (2023)	€3.79 billion (2023)	€118.87 billion (2023)	€17.73 billion (2023)
Allianz	€152.7 billion (2022)	€14.16 billion (2022)	€7.18 billion (2022)	€1.02 trillion (2022)	€51.0 billion (2022)
BASF	€87.3 billion (2022)[1]	€6.55 billion (2022)[1]	€−627 million (2022)[1]	€84.5 billion (2022)[1]	€40.9 billion (2022)[1]
Bayer	€50.74 billion (2023)[2]	€7.01 billion (2022)[2]	€4.15 billion (2022)[2]	€124.9 billion (2022)[2]	€38.93 billion (2022)[2]
Beiersdorf	€7.653 billion (2019)[1]	€1.095 billion (2019)[1]	€736 million (2019)[1]	€9.63 billion (2019)[2]	NaN
BMW	€142.610 billion (2022)[1]	€23.509 billion (2022)[1]	€18.582 billion (2022)[1]	€246.926 billion (2022)[1]	€91.288 billion (2022)[1]
Brenntag	19.4 billion EUR (2022)[1]	NaN	NaN	NaN	NaN
Commerzbank	€8.57 billion (2018)	€1.245 billion (2018)	€865 million (2018)	€462 billion (2018)	€29 billion (2018)
Continental_AG	€41,030 million (2022)	€754.8 million (2022)	€66.6 million (2022)	€37,926.7 million (2022)	€13,735.0 million (2022)
Covestro	€18.0 billion (2022)[1]	€0.3 billion (2022)[1]	€-0.3 billion (2022)[1]	€14.6 billion (2022)[1]	€7.1 billion (2022)[1]
Daimler_Truck	€50.9 billion (2022)	NaN	NaN	NaN	NaN
Deutsche_Bank	€28.9 billion (2023)[2]	€5.7 billion (2023)[2]	€4.9 billion (2023)[2]	€1.31 trillion (2023)[2]	€64 billion (2023)[2]
Deutsche_B%C3%B6rse	€4.34 billion (2022)[2]	€2.17 billion (2022)[2]	€1.49 billion (2022)[2]	€269.1 billion (2022)[2]	€9.06 billion (2022)[2]
Deutsche_Post	NaN	NaN	NaN	NaN	NaN
Deutsche_Telekom	€114,4 billion (2022)[1]	€16.2 billion (2022)[1]	€4.5 billion (2022)[1]	€298.6 billion (2022)[1]	€87.3 billion (2022)[1]
E.ON	€115.66 billion (2022)[1]	€2.242 billion (2022)[1]	€2.728 billion (2022)[1]	€134.009 billion (2022)[1]	€21.867 billion (2022)[1]
Hannover_Re	€ 24.5 billion (Reinsurance revenue, 2023) [1]	NaN	NaN	NaN	NaN
Henkel	€21.514 billion (2023)[1]	€2.011 billion (2023) [1]	€1.340 billion (2023)[1]	€17.965 billion (2023) [1]	€6.624 billion (2023) [1]
Infineon_Technologies	€16.309 billion (2023)	€3.948 billion (2023)	€3.137 billion (2023)	€28.439 billion (2023)	€17.044 billion (2023)
Mercedes-Benz_Group	€133.9 billion	€16.0 billion	€23.4 billion	€258.8 billion	€73.2 billion

key	Revenue	Operating income	Net income	Total assets	Total equity
	(2021)[2]	(2021)[2]	(2021)[2]	(2021)[2]	(2021)[2]
Merck_Group	€20.99 billion (2023)	€3.609 billion (2023)	€2.834 billion (2023)	€48.49 billion (2023)	€26.68 billion (2023)
MTU_Aero_Engines	€4.628 billion (2019)[1]	€706 million (2019)[1]	€488 million (2019)[1]	€7.765 billion (31 December 2019)[1]	€2.421 billion (31 December 2019)[1]
Munich_Re	€67.1 billion (2022)[3]	€3.58 billion (2022)[4]	€3.41 billion (2022)[5]	€298.5 billion (2022)[6]	€21.2 billion (2022)[7]
Porsche	€37.630 billion (2022)[2]	€6.770 billion (2022)[2]	€4.957 billion (2022)[2]	€47.673 billion (2022)[2]	€17.027 billion (2022)[2]
Porsche_SE	NaN	NaN	€824 million (2021)[3]	€42.533 billion (2021)[3]	€42.196 billion (2021)[3]
Qiagen	US$1.97 billion (2023)	US$410 million (2023)	US$341 million (2023)	US$6.12 billion (2023)	US$3.81 billion (2023)
Rheinmetall	€6.410 billion (2022)[1]	€754 million (2022)[1]	€535 million (2022)[1]	€8.089 billion (end 2022)[1]	€3.083 billion (end 2022)[1]
RWE	€13.125 billion (2019)[1]	€1.267 billion (2019)[1]	€1.210 billion (2019)[1]	€39.846 billion (2019)[1]	€5.738 billion (2019)[1]
SAP	€31.207 billion (2023)	€5.785 billion (2023)	€5.928 billion (2023)	€68.291 billion (2023)	€43.365 billion (2023)
Sartorius_AG	€4.17 billion (2022)[1]	NaN	NaN	NaN	NaN
Siemens	€77.769 billion (2023)[2]	€11.201 billion (2023)[2]	€8.529 billion (2023)[2]	€145.067 billion (2023)[2]	€53.060 billion (2023)[2]
Siemens_Healthineers	€21.7 billion (2022)[1]	€2.927 billion (2022)[1]	€2.054 billion (2022)[1]	€33.614 billion (2022)[1]	€19.852 billion (2022)[1]
Symrise	€4.62 billion (2022)[1]	€618 million (2022)[1]	€280 million (2022)[1]	€7.78 billion (2022)[1]	€3.61 billion (2022)[1]
Volkswagen_Group	€322.2840 billion (2023)	€21.586 billion (2023)	€16.013 billion (2023)	€630.826 billion (2023)	€174.757 billion (2023)
Vonovia	€4.400 billion (2020)	NaN	€1,909.8 million (2020)	€58,910.7 million (2020)	€21,069.7 million (2019)
Zalando	€10.3 billion (2022)[1]	€81.0 million (2022)[2]	€16.8 million (2022)[3]	€7.626 billion (2022)[4]	€2.199 billion (2022)[5]

Just like in the last exercise, we now have a table with the individual observations (companies) in the rows. But we also have cells that contain several values here. In the `'Total assets'` column, for example, we have the value `€897,567 billion (2018)[1]`. This cell content is very impractical if we intend on doing more with it. On the one hand, it contains text, although it represents a number. On the other hand, it contains two different pieces of information: an amount of money and a year number that describes the amount in more detail. To fix this, it

would be helpful to know how to extract information from text elegantly. We'll look at that in the next lessons.

Now save `df_company` as a *pickle* named `company_data.p`. In the following exercise we'll merge the data we received.

In [20]:
```
df_company.to_pickle('company_data.p')
```

Congratulations: You have followed the links from one page to various other web pages. You then converted them and automatically read out the most important information and gave it a structured format. The Taiwanese investor is very satisfied that you found the data so quickly and put it into a good format for further analysis. He is looking forward to you extracting the most important information and preparing it even better.

Remember:

- Combine several DataFrames (or Series) with `pd.concat()` to make a bigger `DataFrame`
- Don't send too many queries in a short period of time. You can use `sleep()` from the `time` module here.
- Pay attention to similarities and differences in the web pages when reading them out
- Web pages change their content and structure regularly and sometimes unexpectedly. Because of this, parsers need to be adjusted regularly!

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.
