Web scraping

Web scraping, also known as web harvesting or web data extraction, is a type of data scraping used to gather information from websites.

In this session, we will cover the following concepts with the help of a business use case:

- Data acquisition through Web scraping

- Warnings are given to developers to alert them about circumstances that are not necessarily exceptions. Warnings are not the same as errors. It shows some message but the program will run. The `filterwarnings()` function, defined in the `warning` module, is used to handle warnings (presented, disregarded, or raised to exceptions).

In [1]:

```python
import warnings
warnings.filterwarnings("ignore")
```

Health Care Rankings for Different European Countries

Beautiful Soup is a Python package that is used for web scraping. The urllib package is used to simplify the tasks of building, loading and parsing URLs. The Python datetime module supplies classes to work with date and time.

In [3]:

```python
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup
import requests
import csv
import re
import urllib.request as urllib2
from datetime import datetime
import os
import sys
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

- We are going to scrape data from Wikipedia. The data indicate rankings on different health indices such as patient rights and information, accessibility (waiting time for treatment), outcomes, range, the reach of services provided, prevention, and pharmaceuticals. The data are from the Euro Health Consumer index. In the following code, we read the data and use Beautiful Soup to convert the data into **bs4.BeautifulSoup** data.

In [4]:

```python
url = 'https://en.wikipedia.org/wiki/Healthcare_in_Europe'
r = requests.get(url)
HCE = BeautifulSoup(r.text)
type(HCE)
```

Out[4]:

```
bs4.BeautifulSoup
```

- First, we must choose the table that we want to scrape. As many webpages have tables, we'll retrieve the exact table names from the HTML and store them in a list called `lst.`

In [31]:
```
r
```
Out[31]:
```
<Response [200]>
```
In [32]:
```
HCE
```
Out[32]:
```
<!DOCTYPE html>
<html class="client-nojs" dir="ltr" lang="en">
<head>
<meta charset="utf-8"/>
<title>Healthcare in Europe - Wikipedia</title>
<script>document.documentElement.className="client-js";RLCONF={"wgBreakFrames":fals
e,"wgSeparatorTransformTable":["",""],"wgDigitTransformTable":["",""],"wgDefaultDateF
ormat":"dmy","wgMonthNames":["","January","February","March","April","May","June","Ju
ly","August","September","October","November","December"],"wgRequestId":"5442c776-fda
f-4a32-a6ce-cc7c976ac70a","wgCSPNonce":false,"wgCanonicalNamespace":"","wgCanonicalSp
ecialPageName":false,"wgNamespaceNumber":0,"wgPageName":"Healthcare_in_Europe","wgTit
le":"Healthcare in Europe","wgCurRevisionId":1108727875,"wgRevisionId":1108727875,"wg
ArticleId":15973120,"wgIsArticle":true,"wgIsRedirect":false,"wgAction":"view","wgUser
Name":null,"wgUserGroups":["*"],"wgCategories":["CS1 maint: url-status","Use dmy date
s from December 2021","All articles with unsourced statements","Articles with unsourc
ed statements from January 2021","Articles with unsourced statements from September 2
022","Articles with LCCN identifiers","Health in Europe"],
"wgPageContentLanguage":"en","wgPageContentModel":"wikitext","wgRelevantPageName":"He
althcare_in_Europe","wgRelevantArticleId":15973120,"wgIsProbablyEditable":true,"wgRel
```
In [33]:
```
webpage = urllib2.urlopen(url)
htmlpage= webpage.readlines()
lst = []
for line in htmlpage:
    line = str(line).rstrip()
    if re.search('table class', line) :
        lst.append(line)
```
In [37]:
```
len(lst)
```
Out[37]:
```
5
```
In [35]:
```
webpage
```
Out[35]:
```
<http.client.HTTPResponse at 0x20beeb2b4f0>
```
In [36]:
```
htmlpage
```
Out[36]:
```
[b'<!DOCTYPE html>\n',
 b'<html class="client-nojs" lang="en" dir="ltr">\n',
 b'<head>\n',
 b'<meta charset="UTF-8"/>\n',
 b'<title>Healthcare in Europe - Wikipedia</title>\n',
 b'<script>document.documentElement.className="client-js";RLCONF={"wgBreakFrames":fal
se,"wgSeparatorTransformTable":["",""],"wgDigitTransformTable":["",""],"wgDefaultDate
Format":"dmy","wgMonthNames":["","January","February","March","April","May","June","J
uly","August","September","October","November","December"],"wgRequestId":"5442c776-fd
af-4a32-a6ce-cc7c976ac70a","wgCSPNonce":false,"wgCanonicalNamespace":"","wgCanonicalS
pecialPageName":false,"wgNamespaceNumber":0,"wgPageName":"Healthcare_in_Europe","wgTi
tle":"Healthcare in Europe","wgCurRevisionId":1108727875,"wgRevisionId":1108727875,"w
gArticleId":15973120,"wgIsArticle":true,"wgIsRedirect":false,"wgAction":"view","wgUse
rName":null,"wgUserGroups":["*"],"wgCategories":["CS1 maint: url-status","Use dmy dat
es from December 2021","All articles with unsourced statements","Articles with unsour
ced statements from January 2021","Articles with unsourced statements from September
2022","Articles with LCCN identifiers","Health in Europe"],\n',
 b'"wgPageContentLanguage":"en","wgPageContentModel":"wikitext","wgRelevantPageNam
e":"Healthcare_in_Europe","wgRelevantArticleId":15973120,"wgIsProbablyEditable":tru
```

- This list `lst` has a length of 5.

- Now let us display `lst` .

In [5]:
```
lst
```

Out[5]:
```
['b\'<table class="wikitable floatright sortable" style="font-size: 90%">\\n\'',
 'b\'<div class="navbox-styles nomobile"><style data-mw-deduplicate="TemplateStyles:r
1061467846">.mw-parser-output .navbox{box-sizing:border-box;border:1px solid #a2a9b1;
width:100%;clear:both;font-size:88%;text-align:center;padding:1px;margin:1em auto 0}.
mw-parser-output .navbox .navbox{margin-top:0}.mw-parser-output .navbox+.navbox,.mw-p
arser-output .navbox+.navbox-styles+.navbox{margin-top:-1px}.mw-parser-output .navbox
-inner,.mw-parser-output .navbox-subgroup{width:100%}.mw-parser-output .navbox-grou
p,.mw-parser-output .navbox-title,.mw-parser-output .navbox-abovebelow{padding:0.25em
1em;line-height:1.5em;text-align:center}.mw-parser-output .navbox-group{white-space:n
owrap;text-align:right}.mw-parser-output .navbox,.mw-parser-output .navbox-subgroup{b
ackground-color:#fdfdfd}.mw-parser-output .navbox-list{line-height:1.5em;border-colo
r:#fdfdfd}.mw-parser-output .navbox-list-with-group{text-align:left;border-left-widt
h:2px;border-left-style:solid}.mw-parser-output tr+tr>.navbox-abovebelow,.mw-parser-o
utput tr+tr>.navbox-group,.mw-parser-output tr+tr>.navbox-image,.mw-parser-output tr+
tr>.navbox-list{border-top:2px solid #fdfdfd}.mw-parser-output .navbox-title{backgrou
nd-color:#ccf}.mw-parser-output .navbox-abovebelow,.mw-parser-output .navbox-group,.m
w-parser-output .navbox-subgroup .navbox-title{background-color:#ddf}.mw-parser-outpu
t .navbox-subgroup .navbox-group,.mw-parser-output .navbox-subgroup .navbox-abovebelo
w{background-color:#e6e6ff}.mw-parser-output .navbox-even{background-color:#f7f7f7}.m
```

- We will scrape the first table, and use index 0 in `lst` to capture the first table name. Now, read the table using Beautiful Soup's `find` function. A simple option is to type the table name. You can simply select the name in `lst` , which in this case is "wikitable floatright sortable".

In [38]:
```
table=HCE.find('table', {'class', 'wikitable floatright sortable'})
```

In [40]:
```
type(table)
```

Out[40]:
```
bs4.element.Tag
```

- Alternatively, there is a way to automate this step by capturing the first data from the list and then stripping off the unnecessary characters like `^ " *` .

In [41]:
```
x=lst[0]
extr=re.findall('"([^"]*)"', x)
table=HCE.find('table', {'class', extr[0]})
```

In [42]:
```
type(table)
```

Out[42]:
```
bs4.element.Tag
```

- Now, it would be good to read the header and row names separately, so later we can easily make a DataFrame.

In [43]:
```
headers= [header.text for header in table.find_all('th')]
```

In [44]:
```
headers
```

Out[44]:
```
['WorldRank\n', 'EURank\n', 'Country\n', 'Life expectancyat birth (years)\n']
```

In [45]:
```
rows = []
for row in table.find_all('tr'):
    rows.append([val.text.encode('utf8').decode() for val in row.find_all('td')])
```

- Now, all elements, rows, and headers are available to build the DataFrame, which we will call `df1` .

In [46]:

```
df1 = pd.DataFrame(rows, columns=headers)
```

- Let's display first seven rows of the `df1`

In [47]:

```
df1.head(7)
```

Out[47]:

	WorldRank\n	EURank\n	Country\n	Life expectancyat birth (years)\n
0	None	None	None	None
1	5.\n	1.\n	Spain\n	83.4\n
2	6.\n	2.\n	Italy\n	83.4\n
3	11.\n	3.\n	Sweden\n	82.7\n
4	12.\n	4.\n	France\n	82.5\n
5	13.\n	5.\n	Malta\n	82.4\n
6	16.\n	6.\n	Ireland\n	82.1\n

Health Expenditure

Let's scrape health expenditure as well. These are data per capita, which means that expenditure was corrected for the number of habitants in a country.

- Just like we did for above web page (**Health Care Rankings for Different European Countries**), we have to repeat the same steps in this web page as well (**Health Expenditure**).
- Finally, we will be directed to the first table "wikitable sortable static" in this web page as well.

In [48]:

```
url = 'https://en.wikipedia.org/wiki/List_of_countries_by_total_health_expenditure_p
r = requests.get(url)
HEE = BeautifulSoup(r.text)
webpage = urllib2.urlopen(url)
htmlpage= webpage.readlines()
lst = []
for line in htmlpage:
    line = str(line).rstrip()
    if re.search('table class', line) :
        lst.append(line)
x=lst[1]
print(x)
extr=re.findall('"([^"]*)"', x)
table=HEE.find('table', {'class', extr[0]})
headers= [header.text for header in table.find_all('th')]
rows = []
for row in table.find_all('tr'):
    rows.append([val.text.encode('utf8').decode() for val in row.find_all('td')])
headers = [i.replace("\n", "") for i in headers]
df2 = pd.DataFrame(rows, columns=headers)
```

```
b'<table class="wikitable sortable static-row-numbers plainrowheaders srn-white-backgro
und" border="1" style="text-align:right;">\n'
```

- Let's display the first five rows of the table "wikitable sortable static"

In [49]:

```
df2.head()
```

Out[49]:

	Location	2017	2018	2019
0	None	None	None	None
1	Australia *\n	4,711\n	4,965\n	5,187\n
2	Austria *\n	5,360\n	5,538\n	5,851\n
3	Belgium *\n	5,014\n	5,103\n	5,428\n
4	Canada *\n	5,155\n	5,287\n	5,418\n

Additional Preprocessing Steps

If we look at the DataFrame, we can see that there are still some issues that prohibit numeric computations.

- There are undesired characters ('\n')
- The undesired decimal format (,) should be removed
- There are cells with non-numeric characters ('x') that should be NAN

In [50]:

```
def preproc(dat):
    dat.dropna(axis=0, how='all', inplace=True)
    dat.columns = dat.columns.str.replace("\n", "")
    dat.replace(["\n"], [""], regex=True, inplace=True)
    dat.replace([r"\s\*$"], [""], regex=True, inplace=True)
    dat.replace([","], [""], regex=True, inplace=True)
    dat.replace(r"\b[a-zA-Z]\b", np.nan, regex=True, inplace=True)
    dat.replace([r"^\s"], [""], regex=True, inplace=True)
    dat = dat.apply(pd.to_numeric, errors='ignore')
    return(dat)
```

In [51]:

```
df1 = preproc(df1)
df2 = preproc(df2)
```

In [52]:

```
df1
```

Out[52]:

	WorldRank	EURank	Country	Life expectancyat birth (years)
1	5.0	1.0	Spain	83.4
2	6.0	2.0	Italy	83.4
3	11.0	3.0	Sweden	82.7
4	12.0	4.0	France	82.5
5	13.0	5.0	Malta	82.4
6	16.0	6.0	Ireland	82.1
7	17.0	7.0	Netherlands	82.1
8	19.0	8.0	Luxembourg	82.1
9	20.0	9.0	Greece	82.1

- Apparently, after this preprocessing, there are some NANs.

In [53]:

```
df2
```

Out[53]:

	Location	2017	2018	2019
1	Australia	4711	4965	5187
2	Austria	5360	5538	5851
3	Belgium	5014	5103	5428
4	Canada	5155	5287	5418
5	Chile	2030	2126	2159
6	Colombia	1156	1201	1213
7	Czech Republic	2891	3171	3428
8	Denmark	5107	5295	5568
9	Estonia	2217	2368	2579
10	Finland	4222	4331	4578
11	France	5057	5154	5376
12	Germanv	6011	6224	6646

In [19]:

```
print(df1.isnull().sum().sum())
print(df2.isnull().sum().sum())
```

```
0
0
```

- Now we display where the NANs occur. In fact, when we check the original table, we can see that Cyprus has values "x", which were in our preproc function changed to NANs (https://en.wikipedia.org/wiki/Healthcare_in_Europe (https://en.wikipedia.org/wiki/Healthcare_in_Europe)).

In [20]:

```
df1[df1.isnull().any(axis=1)]
```

Out[20]:

WorldRank	EURank	Country	Life expectancyat birth (years)

Now we remove the NANs.

In [21]:

```
df1.dropna(axis=0, how='any', inplace=True)
```

At this point we inspect the data types:

In [22]:

```
df1.dtypes
```

Out[22]:

```
WorldRank                           float64
EURank                              float64
Country                              object
Life expectancyat birth (years)     float64
dtype: object
```

In [23]:

```
df2.dtypes
```

Out[23]:

```
Location    object
2017         int64
2018         int64
2019         int64
dtype: object
```

The column names are a bit long, so it would be good to use shorter names.

In [24]:

```
df1.columns = ['WorldRank', 'EURank', 'Country', 'Life expectancy in (years)']
df2.columns = ['Country', '2017', '2018', '2019']
```

Analyzing Final Tables

In [25]:

```
df1.head()
```

Out[25]:

	WorldRank	EURank	Country	Life expectancy in (years)
1	5.0	1.0	Spain	83.4
2	6.0	2.0	Italy	83.4
3	11.0	3.0	Sweden	82.7
4	12.0	4.0	France	82.5
5	13.0	5.0	Malta	82.4

In [26]:

```
df2.head()
```

Out[26]:

	Country	2017	2018	2019
1	Australia	4711	4965	5187
2	Austria	5360	5538	5851
3	Belgium	5014	5103	5428
4	Canada	5155	5287	5418
5	Chile	2030	2126	2159

Merging Different Data

###It should be clear from this example that web scraping can be important to quickly grasp data. Web scraping may be particularly useful when you need to automate data processing:

- Webdata change regularly and need to be stored repeatedly.
- A large number of data sources, for example, tables, need to be loaded and merged.

Let us elaborate on the last point a bit more. If the two tables that we just scraped need to be merged, it can be done in Python. For example, if we want to merge on the column "Country", we would use the following code (we use the `.head()` function to limit the output).

In [27]:

```
pd.merge(df1, df2, how='left', on='Country').head()
```

Out[27]:

	WorldRank	EURank	Country	Life expectancy in (years)	2017	2018	2019
0	5.0	1.0	Spain	83.4	3322.0	3430.0	3616.0
1	6.0	2.0	Italy	83.4	3399.0	3485.0	3649.0
2	11.0	3.0	Sweden	82.7	5318.0	5434.0	5782.0
3	12.0	4.0	France	82.5	5057.0	5154.0	5376.0
4	13.0	5.0	Malta	82.4	NaN	NaN	NaN

Note: In this lesson, we saw the use of the data wrangling and web scraping methods, but in the next lesson we are going to use one of these methods as a sub component of "Feature Engineering".