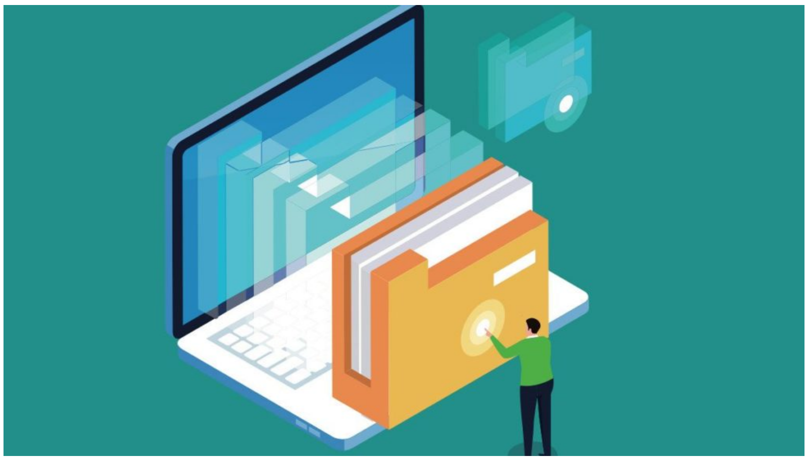
Reading HTML tables

In this lecture we'll learn how to read and parse HTML tables from websites into a list of `DataFrame` objects to work with.



Hands on!

In [ ]:

```
!pip install lxml
```

In [ ]:

```
import pandas as pd
```

Parsing raw HTML strings

Another useful pandas method is `read_html()`. This method will read HTML tables from a given URL, a file-like object, or a raw string containing HTML, and return a list of `DataFrame` objects.

Let's try to read the following `html_string` into a `DataFrame`.

(Open sample.html for the working example)

In [ ]:

```
html_string = """
<table>
    <thead>
      <tr>
        <th>Order date</th>
        <th>Region</th>
        <th>Item</th>
        <th>Units</th>
        <th>Unit cost</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>1/6/2018</td>
        <td>East</td>
        <td>Pencil</td>
        <td>95</td>
        <td>1.99</td>
      </tr>
      <tr>
        <td>1/23/2018</td>
        <td>Central</td>
        <td>Binder</td>
        <td>50</td>
        <td>19.99</td>
      </tr>
      <tr>
        <td>2/9/2018</td>
        <td>Central</td>
        <td>Pencil</td>
        <td>36</td>
        <td>4.99</td>
      </tr>
      <tr>
        <td>3/15/2018</td>
        <td>West</td>
        <td>Pen</td>
        <td>27</td>
        <td>19.99</td>
      </tr>
    </tbody>
</table>
"""
```

In [ ]:

```
from IPython.core.display import display, HTML
display(HTML(html_string))
```

In [ ]:

```
dfs = pd.read_html(html_string)
```

The `read_html` just returned one `DataFrame` object:

In [ ]:

```
len(dfs)
```

In [ ]:

```
df = dfs[0]

df
```

Previous `DataFrame` looks quite similar to the raw HTML table, but now we have a `DataFrame` object, so we can apply any pandas operation we want to it.

In [ ]:

```
df.shape
```

In [ ]:

```
df.loc[df['Region'] == 'Central']
```

In [ ]:

```
df.loc[df['Units'] > 35]
```

Defining header

Pandas will automatically find the header to use thanks to the tag.

But in many cases we'll find wrong or incomplete tables that make the `read_html` method parse the tables in a wrong way without the proper headers.

To fix them we can use the `header` parameter.

In [ ]:

```
html_string = """
<table>
  <tr>
    <td>Order date</td>
    <td>Region</td>
    <td>Item</td>
    <td>Units</td>
    <td>Unit cost</td>
  </tr>
  <tr>
    <td>1/6/2018</td>
    <td>East</td>
    <td>Pencil</td>
    <td>95</td>
    <td>1.99</td>
  </tr>
  <tr>
    <td>1/23/2018</td>
    <td>Central</td>
    <td>Binder</td>
    <td>50</td>
    <td>19.99</td>
  </tr>
  <tr>
    <td>2/9/2018</td>
    <td>Central</td>
    <td>Pencil</td>
    <td>36</td>
    <td>4.99</td>
  </tr>
  <tr>
    <td>3/15/2018</td>
    <td>West</td>
    <td>Pen</td>
    <td>27</td>
    <td>19.99</td>
  </tr>
</table>
"""
```

In [ ]:

```
pd.read_html(html_string)[0]
```

In this case, we'll need to pass the row number to use as header using the `header` parameter.

In [ ]:

```
pd.read_html(html_string, header=0)[0]
```

Parsing HTML tables from the web

Now that we know how `read_html` works, go one step beyond and try to parse HTML tables directly from an URL.

To do that we'll call the `read_html` method with an URL as paramter.

Simple example

In [ ]:

```
html_url = "https://www.basketball-reference.com/leagues/NBA_2019_per_game.html"
```

In [ ]:

```
nba_tables = pd.read_html(html_url)
```

In [ ]:

```
len(nba_tables)
```

We'll work with the only one table found:

In [ ]:

```
nba = nba_tables[0]
```

In [ ]:

```
nba.head()
```

In [ ]:

```
nba.head(25)
```

Complex example

We can also use the `requests` module to get HTML code from an URL to parse it into `DataFrame` objects.

If we look at the given URL we can see multiple tables about The Simpsons TV show.

We want to keep the table with information about each season.

In [ ]:

```
import requests

html_url = "https://en.wikipedia.org/wiki/The_Simpsons"
```

In [ ]:

```
r = requests.get(html_url)

wiki_tables = pd.read_html(r.text, header=0)
```

In [ ]:

```
len(wiki_tables)
```

In [ ]:

```
simpsons = wiki_tables[1]
```

In [ ]:

```
simpsons.head()
```

Quick clean on the table: remove extra header rows and set `Season` as index.

In [ ]:

```
simpsons.drop([0, 1], inplace=True)
```

In [ ]:

```
simpsons.set_index('Season', inplace=True)
```

Which season has the lowest number of episodes?

In [ ]:

```
simpsons['No. ofepisodes'].unique()
```

In [ ]:

```
simpsons = simpsons.loc[simpsons['No. ofepisodes'] != 'TBA']
```

In [ ]:

```
min_season = simpsons['No. ofepisodes'].min()

min_season
```

In [ ]:

```
simpsons.loc[simpsons['No. ofepisodes'] == min_season]
```

Save to CSV file

Finally save the `DataFrame` to a CSV file as we saw on previous lectures.

In [ ]:

```
simpsons.head()
```

In [ ]:

```
simpsons.to_csv('out.csv')
```

In [ ]:

```
pd.read_csv('out.csv', index_col='Season').head()
```