**Web Scraping Lab**

For this lab, we are going to be using Python and several Python libraries. Some of these libraries might be installed in your lab environment or in SN Labs. Others may need to be installed by you. The cells below will install these libraries when executed.

!mamba install bs4==4.10.0 -y

!pip install lxml==4.6.4

!mamba install html5lib==1.1 -y

# !pip install requests==2.26.0



Import the required modules and functions

from bs4 import BeautifulSoup # this module helps in web scrapping.

import requests # this module helps us to download a web page

## Beautiful Soup Objects

Beautiful Soup is a Python library for pulling data out of HTML and XML files, we will focus on HTML files. This is accomplished by representing the HTML as a set of objects with methods used to parse the HTML. We can navigate the HTML as a tree and/or filter out what we are looking for.

Consider the following HTML:

%%html

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

</head>

<body>

<h3><b id='boldest'>Lebron James</b></h3>

<p> Salary: $ 92,000,000 </p>

<h3> Stephen Curry</h3>

<p> Salary: $85,000, 000 </p>

<h3> Kevin Durant </h3>

<p> Salary: $73,200, 000</p>

</body>

</html>

### Lebron James

Salary: $ 92,000,000

### Stephen Curry

Salary: $85,000, 000

### Kevin Durant

Salary: $73,200, 000

We can store it as a string in the variable HTML:

html="<!DOCTYPE html><html><head><title>Page Title</title></head><body><h3><b id='boldest'>Lebron James</b></h3><p> Salary: $ 92,000,000 </p><h3> Stephen Curry</h3><p> Salary: $85,000, 000 </p><h3> Kevin Durant </h3><p> Salary: $73,200, 000</p></body></html>"

To parse a document, pass it into the BeautifulSoup constructor, the BeautifulSoup object, which represents the document as a nested data structure:

soup = BeautifulSoup(html, "html.parser")

First, the document is converted to Unicode, (similar to ASCII), and HTML entities are converted to Unicode characters. Beautiful Soup transforms a complex HTML document into a complex tree of Python objects. The BeautifulSoup object can create other types of objects. In this lab, we will cover BeautifulSoup and Tag objects that for the purposes of this lab are identical, and NavigableString objects.

We can use the method prettify() to display the HTML in the nested structure:

print(soup.prettify())

<!DOCTYPE html>

<html>

<head>

<title>

Page Title

</title>

</head>

<body>

<h3>

<b id="boldest">

Lebron James

</b>

</h3>

<p>

Salary: $ 92,000,000

</p>

<h3>

Stephen Curry

</h3>

<p>

Salary: $85,000, 000

</p>

<h3>

Kevin Durant

</h3>

<p>

Salary: $73,200, 000

</p>

</body>

</html>

## Tags

Let's say we want the title of the page and the name of the top paid player we can use the Tag. The Tag object corresponds to an HTML tag in the original document, for example, the tag title.

tag\_object=soup.title

print("tag object:",tag\_object)

tag object: <title>Page Title</title>

we can see the tag type bs4.element.Tag

print("tag object type:",type(tag\_object))

tag object type: <class 'bs4.element.Tag'>

If there is more than one Tag with the same name, the first element with that Tag name is called, this corresponds to the most paid player:

tag\_object=soup.h3

tag\_object

<h3><b id="boldest">Lebron James</b></h3>

Enclosed in the bold attribute b, it helps to use the tree representation. We can navigate down the tree using the child attribute to get the name.

### Children, Parents, and Siblings

As stated above the Tag object is a tree of objects we can access the child of the tag or navigate down the branch as follows:

tag\_child =tag\_object.b

tag\_child

<b id="boldest">Lebron James</b>

You can access the parent with the  parent

parent\_tag=tag\_child.parent

parent\_tag

<h3><b id="boldest">Lebron James</b></h3>

this is identical to

tag\_object

<h3><b id="boldest">Lebron James</b></h3>

tag\_object parent is the body element.

tag\_object.parent

<body><h3><b id="boldest">Lebron James</b></h3><p> Salary: $ 92,000,000 </p><h3> Stephen Curry</h3><p> Salary: $85,000, 000 </p><h3> Kevin Durant </h3><p> Salary: $73,200, 000</p></body>

tag\_object sibling is the paragraph element

sibling\_1=tag\_object.next\_sibling

sibling\_1

<p> Salary: $ 92,000,000 </p>

sibling\_2 is the header element which is also a sibling of both sibling\_1 and tag\_object

sibling\_2=sibling\_1.next\_sibling

sibling\_2

<h3> Stephen Curry</h3>

### HTML Attributes

If the tag has attributes, the tag id="boldest" has an attribute id whose value is boldest. You can access a tag’s attributes by treating the tag like a dictionary:

tag\_child['id']

‘boldest’

You can access that dictionary directly as attrs:

tag\_child.attrs

{'id': 'boldest'}

You can also work with Multi-valued attribute check out [[1]](https://www.crummy.com/software/BeautifulSoup/bs4/doc/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0220ENSkillsNetwork23455606-2021-01-01) for more.

We can also obtain the content if the attribute of the tag using the Python get() method.

tag\_child.get('id')

'boldest'

### Navigable String

A string corresponds to a bit of text or content within a tag. Beautiful Soup uses the NavigableString class to contain this text. In our HTML we can obtain the name of the first player by extracting the sting of the Tag object tag\_child as follows:

tag\_string=tag\_child.string

tag\_string

'Lebron James'

we can verify the type is Navigable String

type(tag\_string)

bs4.element.NavigableString

A NavigableString is just like a Python string or Unicode string, to be more precise. The main difference is that it also supports some BeautifulSoup features. We can covert it to sting object in Python:

unicode\_string = str(tag\_string)

unicode\_string

'Lebron James'

## Filter

Filters allow you to find complex patterns, the simplest filter is a string. In this section we will pass a string to a different filter method and Beautiful Soup will perform a match against that exact string. Consider the following HTML of rocket launchs:

%%html

<table>

<tr>

<td id='flight' >Flight No</td>

<td>Launch site</td>

<td>Payload mass</td>

</tr>

<tr>

<td>1</td>

<td><a href='https://en.wikipedia.org/wiki/Florida'>Florida</a></td>

<td>300 kg</td>

</tr>

<tr>

<td>2</td>

<td><a href='https://en.wikipedia.org/wiki/Texas'>Texas</a></td>

<td>94 kg</td>

</tr>

<tr>

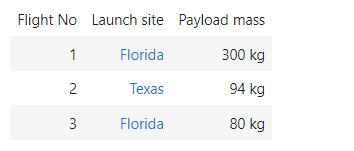
<td>3</td>

<td><a href='https://en.wikipedia.org/wiki/Florida'>Florida<a> </td>

<td>80 kg</td>

</tr>

</table>



We can store it as a string in the variable table:

table="<table><tr><td id='flight'>Flight No</td><td>Launch site</td> <td>Payload mass</td></tr><tr> <td>1</td><td><a href='https://en.wikipedia.org/wiki/Florida'>Florida<a></td><td>300 kg</td></tr><tr><td>2</td><td><a href='https://en.wikipedia.org/wiki/Texas'>Texas</a></td><td>94 kg</td></tr><tr><td>3</td><td><a href='https://en.wikipedia.org/wiki/Florida'>Florida<a> </td><td>80 kg</td></tr></table>"

table\_bs = BeautifulSoup(table, "html.parser")

## find All

The find\_all() method looks through a tag’s descendants and retrieves all descendants that match your filters.

The Method signature for find\_all(name, attrs, recursive, string, limit, \*\*kwargs)

### Name

When we set the name parameter to a tag name, the method will extract all the tags with that name and its children.

table\_rows=table\_bs.find\_all('tr')

table\_rows

[<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>,

<tr> <td>1</td><td><a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a></a></a></td><td>300 kg</td></tr>,

<tr><td>2</td><td><a href="<https://en.wikipedia.org/wiki/Texas>">Texas</a></td><td>94 kg</td></tr>,

<tr><td>3</td><td><a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a> </a></a></td><td>80 kg</td></tr>]

The result is a Python Iterable just like a list, each element is a tag object:

first\_row =table\_rows[0]

first\_row

<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>

The type is tag

print(type(first\_row))

<class 'bs4.element.Tag'>

we can obtain the child

first\_row.td

<td id="flight">Flight No</td>

If we iterate through the list, each element corresponds to a row in the table:

for i,row in enumerate(table\_rows):

print("row",i,"is",row)

row 0 is <tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>

row 1 is <tr> <td>1</td><td><a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a></a></a></td><td>300 kg</td></tr>

row 2 is <tr><td>2</td><td><a href="<https://en.wikipedia.org/wiki/Texas>">Texas</a></td><td>94 kg</td></tr>

row 3 is <tr><td>3</td><td><a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a> </a></a></td><td>80 kg</td></tr>

As row is a cell object, we can apply the method find\_all to it and extract table cells in the object cells using the tag td, this is all the children with the name td. The result is a list, each element corresponds to a cell and is a Tag object, we can iterate through this list as well. We can extract the content using the string attribute.

for i,row in enumerate(table\_rows):

print("row",i)

cells=row.find\_all('td')

for j,cell in enumerate(cells):

print('colunm',j,"cell",cell)

row 0

colunm 0 cell <td id="flight">Flight No</td>

colunm 1 cell <td>Launch site</td>

colunm 2 cell <td>Payload mass</td>

row 1

colunm 0 cell <td>1</td>

colunm 1 cell <td><a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a></a></a></td>

colunm 2 cell <td>300 kg</td>

row 2

colunm 0 cell <td>2</td>

colunm 1 cell <td><a href="<https://en.wikipedia.org/wiki/Texas>">Texas</a></td>

colunm 2 cell <td>94 kg</td>

row 3

colunm 0 cell <td>3</td>

colunm 1 cell <td><a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a> </a></a></td>

colunm 2 cell <td>80 kg</td>

If we use a list we can match against any item in that list.

list\_input=table\_bs .find\_all(name=["tr", "td"])

list\_input

[<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>,

<td id="flight">Flight No</td>,

<td>Launch site</td>,

<td>Payload mass</td>,

<tr> <td>1</td><td><a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a></a></a></td><td>300 kg</td></tr>,

<td>1</td>,

<td><a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a></a></a></td>,

<td>300 kg</td>,

<tr><td>2</td><td><a href="<https://en.wikipedia.org/wiki/Texas>">Texas</a></td><td>94 kg</td></tr>,

<td>2</td>,

<td><a href="<https://en.wikipedia.org/wiki/Texas>">Texas</a></td>,

<td>94 kg</td>,

<tr><td>3</td><td><a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a> </a></a></td><td>80 kg</td></tr>,

<td>3</td>,

<td><a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a> </a></a></td>,

<td>80 kg</td>]

## Attributes[¶](https://jupyterlab-5-labs-prod-jupyterlab-us-east-0.labs.cognitiveclass.ai/user/marinaust86/lab/tree/labs/PY0101EN/WebScraping_Review_Lab.ipynb#Attributes)

If the argument is not recognized it will be turned into a filter on the tag’s attributes. For example the id argument, Beautiful Soup will filter against each tag’s id attribute. For example, the first td elements have a value of id of flight, therefore we can filter based on that id value.

table\_bs.find\_all(id="flight")

[<td id="flight">Flight No</td>]

We can find all the elements that have links to the Florida Wikipedia page:

list\_input=table\_bs.find\_all(href="https://en.wikipedia.org/wiki/Florida")

list\_input

[<a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a></a></a>,

<a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a> </a></a>]

If we set the href attribute to True, regardless of what the value is, the code finds all tags with href value:

table\_bs.find\_all(href=True)

[<a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a></a></a>,

<a href="<https://en.wikipedia.org/wiki/Texas>">Texas</a>,

<a href="<https://en.wikipedia.org/wiki/Florida>">Florida<a> </a></a>]

There are other methods for dealing with attributes and other related methods; Check out the following [link](https://www.crummy.com/software/BeautifulSoup/bs4/doc/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0220ENSkillsNetwork23455606-2021-01-01#css-selectors)

### string

With string you can search for strings instead of tags, where we find all the elments with Florida:

table\_bs.find\_all(string="Florida")

['Florida', 'Florida']

## find

The find\_all() method scans the entire document looking for results, it’s if you are looking for one element you can use the find() method to find the first element in the document. Consider the following two table:

%%html

<h3>Rocket Launch </h3>

<p>

<table class='rocket'>

<tr>

<td>Flight No</td>

<td>Launch site</td>

<td>Payload mass</td>

</tr>

<tr>

<td>1</td>

<td>Florida</td>

<td>300 kg</td>

</tr>

<tr>

<td>2</td>

<td>Texas</td>

<td>94 kg</td>

</tr>

<tr>

<td>3</td>

<td>Florida </td>

<td>80 kg</td>

</tr>

</table>

</p>

<p>

<h3>Pizza Party </h3>

<table class='pizza'>

<tr>

<td>Pizza Place</td>

<td>Orders</td>

<td>Slices </td>

</tr>

<tr>

<td>Domino's Pizza</td>

<td>10</td>

<td>100</td>

</tr>

<tr>

<td>Little Caesars</td>

<td>12</td>

<td >144 </td>

</tr>

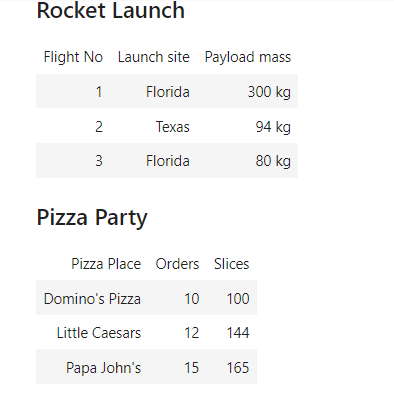
<tr>

<td>Papa John's </td>

<td>15 </td>

<td>165</td>

</tr>



We store the HTML as a Python string and assign two\_tables:

two\_tables="<h3>Rocket Launch </h3><p><table class='rocket'><tr><td>Flight No</td><td>Launch site</td> <td>Payload mass</td></tr><tr><td>1</td><td>Florida</td><td>300 kg</td></tr><tr><td>2</td><td>Texas</td><td>94 kg</td></tr><tr><td>3</td><td>Florida </td><td>80 kg</td></tr></table></p><p><h3>Pizza Party </h3><table class='pizza'><tr><td>Pizza Place</td><td>Orders</td> <td>Slices </td></tr><tr><td>Domino's Pizza</td><td>10</td><td>100</td></tr><tr><td>Little Caesars</td><td>12</td><td >144 </td></tr><tr><td>Papa John's </td><td>15 </td><td>165</td></tr>"

We create a BeautifulSoup object two\_tables\_bs

two\_tables\_bs= BeautifulSoup(two\_tables, 'html.parser')

We can find the first table using the tag name table

wo\_tables\_bs.find("table")

<table class="rocket"><tr><td>Flight No</td><td>Launch site</td> <td>Payload mass</td></tr><tr><td>1</td><td>Florida</td><td>300 kg</td></tr><tr><td>2</td><td>Texas</td><td>94 kg</td></tr><tr><td>3</td><td>Florida </td><td>80 kg</td></tr></table>

We can filter on the class attribute to find the second table, but because class is a keyword in Python, we add an underscore.

two\_tables\_bs.find("table",class\_='pizza')

<table class="pizza"><tr><td>Pizza Place</td><td>Orders</td> <td>Slices </td></tr><tr><td>Domino's Pizza</td><td>10</td><td>100</td></tr><tr><td>Little Caesars</td><td>12</td><td>144 </td></tr><tr><td>Papa John's </td><td>15 </td><td>165</td></tr></table>

## Downloading And Scraping The Contents Of A Web Page

We Download the contents of the web page:

url = <http://www.ibm.com>

We use get to download the contents of the webpage in text format and store in a variable called data:

data = requests.get(url).text

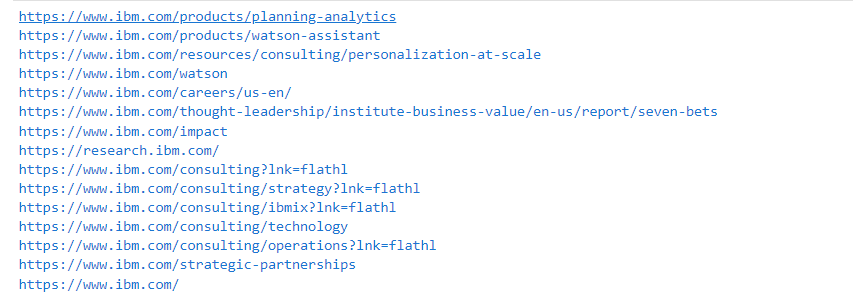
We create a BeautifulSoup object using the BeautifulSoup constructor

soup = BeautifulSoup(data,"html.parser") # create a soup object using the variable 'data'

Scrape all links

for link in soup.find\_all('a',href=True): # in html anchor/link is represented by the tag <a>

print(link.get('href'))



## Scrape all images Tags

for link in soup.find\_all('img'):# in html image is represented by the tag <img>

print(link)

print(link.get('src'))

## Scrape data from HTML tables

#The below url contains an html table with data about colors and color codes.

url = <https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/datasets/HTMLColorCodes.html>

Before proceeding to scrape a web site, you need to examine the contents, and the way data is organized on the website. Open the above url in your browser and check how many rows and columns are there in the color table.

# get the contents of the webpage in text format and store in a variable called data

data = requests.get(url).text

soup = BeautifulSoup(data,"html.parser")

#find a html table in the web page

table = soup.find('table') # in html table is represented by the tag <table>

#Get all rows from the table

for row in table.find\_all('tr'): # in html table row is represented by the tag <tr>

# Get all columns in each row.

cols = row.find\_all('td') # in html a column is represented by the tag <td>

color\_name = cols[2].string # store the value in column 3 as color\_name

color\_code = cols[3].string # store the value in column 4 as color\_code

print("{}--->{}".format(color\_name,color\_code))

Color Name--->None

lightsalmon--->#FFA07A

salmon--->#FA8072

darksalmon--->#E9967A

lightcoral--->#F08080

coral--->#FF7F50

tomato--->#FF6347

orangered--->#FF4500

gold--->#FFD700

orange--->#FFA500

darkorange--->#FF8C00

lightyellow--->#FFFFE0

lemonchiffon--->#FFFACD

papayawhip--->#FFEFD5

moccasin--->#FFE4B5

peachpuff--->#FFDAB9

palegoldenrod--->#EEE8AA

khaki--->#F0E68C

darkkhaki--->#BDB76B

yellow--->#FFFF00

lawngreen--->#7CFC00

chartreuse--->#7FFF00

limegreen--->#32CD32

lime--->#00FF00

forestgreen--->#228B22

green--->#008000

powderblue--->#B0E0E6

lightblue--->#ADD8E6

lightskyblue--->#87CEFA

skyblue--->#87CEEB

deepskyblue--->#00BFFF

lightsteelblue--->#B0C4DE

dodgerblue--->#1E90FF

## Scrape data from HTML tables into a DataFrame using BeautifulSoup and Pandas

import pandas as pd

#The below url contains html tables with data about world population.

url = "https://en.wikipedia.org/wiki/World\_population"

Before proceeding to scrape a web site, you need to examine the contents, and the way data is organized on the website. Open the above url in your browser and check the tables on the webpage.

# get the contents of the webpage in text format and store in a variable called data

data = requests.get(url).text

soup = BeautifulSoup(data,"html.parser")

#find all html tables in the web page

tables = soup.find\_all('table') # in html table is represented by the tag <table>

# we can see how many tables were found by checking the length of the tables list

len(tables)

29

Assume that we are looking for the 10 most densly populated countries table, we can look through the tables list and find the right one we are look for based on the data in each table or we can search for the table name if it is in the table but this option might not always work.

for index,table in enumerate(tables):

if ("10 most densely populated countries" in str(table)):

table\_index = index

print(table\_index)

7

See if you can locate the table name of the table, 10 most densly populated countries, below.

print(tables[table\_index].prettify())

population\_data = pd.DataFrame(columns=["Rank", "Country", "Population", "Area", "Density"])

for row in tables[table\_index].tbody.find\_all("tr"):

col = row.find\_all("td")

if (col != []):

rank = col[0].text

country = col[1].text

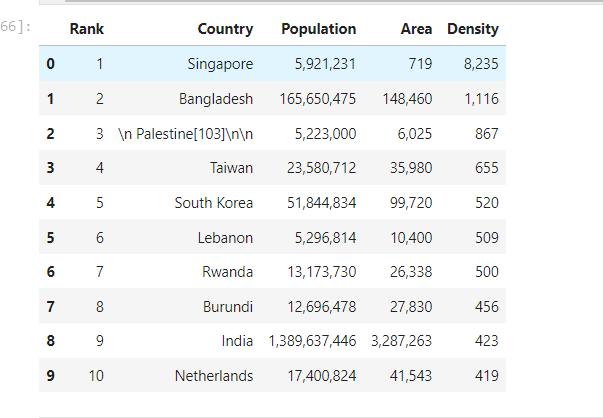
population = col[2].text.strip()

area = col[3].text.strip()

density = col[4].text.strip()

population\_data = population\_data.append({"Rank":rank, "Country":country, "Population":population, "Area":area, "Density":density}, ignore\_index=True)

population\_data



## Scrape data from HTML tables into a DataFrame using BeautifulSoup and read\_html

Using the same url, data, soup, and tables object as in the last section we can use the read\_html function to create a DataFrame.

Remember the table we need is located in tables[table\_index]

We can now use the pandas function read\_html and give it the string version of the table as well as the flavor which is the parsing engine bs4.

pd.read\_html(str(tables[5]), flavor='bs4')

[ Rank Country Population Area(km2) Density(pop/km2)

0 1 Singapore 5704000 710 8033

1 2 Bangladesh 171670000 143998 1192

2 3 Palestine 5266785 6020 847

3 4 Lebanon 6856000 10452 656

4 5 Taiwan 23604000 36193 652

5 6 South Korea 51781000 99538 520

6 7 Rwanda 12374000 26338 470

7 8 Haiti 11578000 27065 428

8 9 Netherlands 17660000 41526 425

9 10 Israel 9430000 22072 427]

The function read\_html always returns a list of DataFrames so we must pick the one we want out of the list.

population\_data\_read\_html = pd.read\_html(str(tables[5]), flavor='bs4')[0]

population\_data\_read\_html

## Scrape data from HTML tables into a DataFrame using read\_html

We can also use the read\_html function to directly get DataFrames from a url.

dataframe\_list = pd.read\_html(url, flavor='bs4')

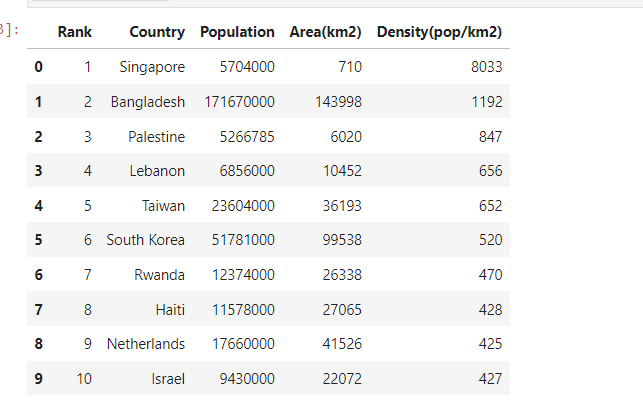
We can see there are 25 DataFrames just like when we used find\_all on the soup object.

len(dataframe\_list)

26

Finally we can pick the DataFrame we need out of the list.

dataframe\_list[5]



We can also use the match parameter to select the specific table we want. If the table contains a string matching the text it will be read.

pd.read\_html(url, match="10 most densely populated countries", flavor='bs4')[0]