Business Analytics Programming

Lab 6 - Web Scraping

Dr. Wajahat Gilani

Rutgers Business School

June 11, 2019

Cherry Blossom 10 Mile Run

Aside from API's there is a lot of free and ubiquitous data on the web. One fun example is the Cherry Blossom 10 Mile run, that has decades of data. The complexity lies in the fact that web pages tend to be inconsistent and requires a lot of iterative development.

In the python language there are several different packages that allow you to pull the html markup from the pages. Two of the most popular ones scrapy and BeautifulSoup, the latter we will use for this project.

Import Libraries

```
1 import pandas as pd
2 import numpy as np
3 import requests
4 from bs4 import BeautifulSoup
5
6 res = requests.get('http://www.cherryblossom.org/results
      /2012/2012cucb10m-m.htm')
7 soup = BeautifulSoup(res.content,'lxml')
```

link: http://www.cherryblossom.org/results/2012/2012cucb10m-m.htm

Credit Union Cherry Blossom Ten Mile Run
Washington, DC Sunday, April 1, 2012

Official Male Results (Sorted By Net Time)

Place	Div /Tot	Num	Name	Ag	Hometown	5 Mile	Time	Pace
1	1/347	9	Allan Kiprono	22	Kenya	22:32	45:15	4:32
2	2/347	11	Lani Kiplagat	23	Kenya	22:38	46:28	4:39
3	1/1093	31	John Korir	36	Kenya	23:20	47:33	4:46
4	1/1457	15	Ian Burrell	27	Tucson AZ	23:50	47:34	4:46
5	3/347	19	Jesse Cherry	24	Blowing Rock NC	23:50	47:40	4:46
6	1/1490	37	Ketema Nugusse	31	Ethiopia	23:42	47:50	4:47
7	2/1457	13	Josh Moen	29	Minneapolis MN	24:06	48:38	4:52
8	3/1457	17	Patrick Rizzo	28	Boulder CO	24:24	49:14	4:56
9	4/1457	41	Stephen Hallinan	26	Washington DC	25:01	50:18	5:02

HTML Markup

```
1 print(soup)
```

link: http://www.cherryblossom.org/results/2012/2012cucb10m-m.htm

```
<!-- saved from url=(0022)http://internet.e-mail -->
<html>
<pre>
                 Credit Union Cherry Blossom Ten Mile Run
                 Washington, DC     Sunday, April 1, 2012

                Official Male Results (Sorted By Net Time)

Place Div   /Tot    Num    Name                  Ag Hometown            5 Mile  Time    Pace
===== =========== ====== ===================== == ==================== ======= ======= =====
    1     1/347        9 Allan Kiprono         22 Kenya                 22:32   45:15  4:32
    2     2/347       11 Lani Kiplagat         23 Kenya                 22:38   46:28  4:39
    3     1/1093      31 John Korir            36 Kenya                 23:20   47:33  4:46
    4     1/1457      15 Ian Burrell           27 Tucson AZ             23:50   47:34  4:46
    5     3/347       19 Jesse Cherry          24 Blowing Rock NC       23:50   47:40  4:46
    6     1/1490      37 Ketema Nugusse        31 Ethiopia              23:42   47:50  4:47
    7     2/1457      13 Josh Moen             29 Minneapolis MN        24:06   48:38  4:52
```

Console 1/A

```
7182   929/931    18958 William Lee          44 Washington DC      1:11:37 2:23:41 14:23
7183   647/648     9984 Dennis Smith         46 Brambleton VA      1:20:22 2:24:02 14:25
7184   196/197    18732 Santa Schreurs       63 Olney MD           1:09:37 2:24:05 14:25
7185   930/931    18974 Chau Nguyen          41 Rockville MD       1:11:17 2:24:48 14:29
7186   197/197    18948 Kerry Kilman         64 Rockville MD       1:11:40 2:25:00 14:30
7187    79/79     16022 Steve Scharf         66 Gaithersburg MD    1:07:39 2:25:52 14:36
7188   931/931    10769 Dana Brown           41 Randallstown MD    1:16:05 2:26:47 14:41
7189  1092/1093   19845 Jurek Grabowski      39 Fairfax VA         1:12:45 2:27:11 14:44
7190   375/375    18780 Larry Hume           56 Arlington VA       1:07:17 2:27:20 14:44
7191  1093/1093   19104 Sean-Patrick Alexander 35 Alexandria VA    1:08:44 2:27:30 14:45
```

Convert HTML Markup To List of Strings

```
table = soup.find("pre").contents[0]
type(table)
#bs4.element.NavigableString
table[0:5]
```

output: '\r\n '

```
table[0:100]
```

output: '\r\n Credit Union Cherry Blossom Ten Mile Run\r\n Washington, DC Sun'

```
tablerows=table.split('\r\n')
tablerows[0:5]
```

['', ' Credit Union Cherry Blossom Ten Mile Run', ' Washington, DC Sunday, April 1, 2012', '', ' Official Male Results (Sorted By Net Time)']

The variable tablerows now behaves like a list of strings.

```
x="the cat in the hat is fat"
y=x.split('at')
print(y)
```

Create Initial DataFrame

```
1 webdf=pd.DataFrame({'raw':tablerows})
```

Index	raw
0	
1	Credit Union Cherry Blossom Ten Mile Run
2	Washington, DC Sunday, April 1, 2012
3	
4	Official Male Results (Sorted By Net Time)
5	
6	Place Div /Tot Num Name Ag Hometown 5 Mile…
7	===== ========== ====== ==== == ====== ======…
8	1 1/347 9 Allan Kiprono 22 Kenya 22:3…
9	2 2/347 11 Lani Kiplagat 23 Kenya 22:3…
10	3 1/1093 31 John Korir 36 Kenya 23:2…
11	4 1/1457 15 Ian Burrell 27 Tucson AZ 23:5…
12	5 3/347 19 Jesse Cherry 24 Blowing Rock NC 23:5…
13	6 1/1490 37 Ketema Nugusse 31 Ethiopia 23:4…
14	7 2/1457 13 Josh Moen 29 Minneapolis MN 24:0…
15	8 3/1457 17 Patrick Rizzo 28 Boulder CO 24:2…
16	9 4/1457 41 Stephen Hallinan 26 Washington DC 25:0…

Notice how the table is just 1 column, and how the data is spaced out into looking like columns within 'raw' column. Also Notice how line 7 is all equals that look like they represent the spaces of each of the columns.

Step 1: Get Length of each column

We need to break up each of these rows by their spaces at the character level. A space is considered a character. The equals signs on line 7 tell us by what size we need to take a substring of each row. Because we aren't sure if the formatting will be exactly the same for each year, we want our program to do as much of the guess work as possible. In this case our line 7 of equal siqn lengths might be on a different line in another year.

```
1 rownumber=webdf.loc[webdf.raw.str.contains("=")]
```

rownumber is a Series, but what we really want is the index number or position in this case because the indexes right now are the position.

```
1 rownumber=webdf.loc[webdf.raw.str.contains("=")].index.values[0]
```

Index	raw
0	
1	Credit Union Cherry Blossom Ten Mile Run
2	Washington, DC Sunday, April 1, 2012
3	
4	Official Male Results (Sorted By Net Time)
5	
6	Place Div /Tot Num Name Ag Hometown 5 Mile…
7	===== ========== ====== ==================== == ================== ======…
8	1 1/347 9 Allan Kiprono 22 Kenya 22:3…
9	2 2/347 11 Lani Kiplagat 23 Kenya 22:3…

Step 1: Get Length of each column

We now get the actual data of line 7.

```
dash=webdf.loc[rownumber,'raw']
type(dash)
print(dash)
```

```
====== ============== =======
============================ ==
======================== ======== ======== ======
```

Each space in this sequence is where the column breaks, so the first column breaks at the 5th character (start counting as 0). To break up the strings of each row, we need to know at what number to break each string. To do this, we are going to use the fact that strings can be converted to lists of characters. For example:

```
vv='batman'
print(list(vv))
```

```
['b', 'a', 't', 'm', 'a', 'n']
```

Step 1: Get Length of each column

```
da=np.array(list(dash))
print(da)
```

```
['=' '=' '=' '=' '=' ' ' '=' '=' '=' '=' '=' '=' '=' '=' '=' '=' '=' ' ' '='
'=' '=' '=' '=' '=' ' ' '=' '=' '=' '=' '=' '=' '=' '=' '=' '=' '=' '=' '='
'=' '=' '=' '=' '=' '=' '=' '=' '=' ' ' '=' '=' ' ' '=' '=' '=' '=' '=' '='
'=' '=' '=' '=' '=' '=' '=' '=' '=' '=' '=' '=' '=' '=' ' ''=' '=' '=' '='
'=' '=' '=' ' ' '=' '=' '=' '=' '=' '=' '=' ' ' '=' '=' '=' '=' '=' ' ']
```

We converted our list into a numpy array so that we can make use of the numpy libraries. All we need to do is just find the position of every space, or ' ', in the da array.

```
np.argwhere(da==' ')
```

array([[5], [17], [24], [47], [50], [71], [79], [87], [93]])
I get the positions in a weird array within an array style, but I can flatten it.

```
dashpos=np.argwhere(da==' ').flatten().tolist()
print(dashpos)
```

[5, 17, 24, 47, 50, 71, 79, 87, 93]

Step 2: Break The Texts into Lists

For example, If I have the text "I love dark chocolate mousse from Mara's", I can break it up into 3 different sub array at the position 6 and 21.

```
ww="I love dark chocolate mousse from Mara's"
wwa=np.array(list(ww))
wwl=np.split(wwa,[6,21])
print(wwl)
```

[array(['I', ' ', 'l', 'o', 'v', 'e'], dtype='¡U1'),
array([' ', 'd', 'a', 'r', 'k', ' ', 'c', 'h', 'o', 'c', 'o', 'l', 'a', 't', 'e'],
dtype='¡U1'),
array([' ', 'm', 'o', 'u', 's', 's', 'e', ' ', 'f', 'r', 'o', 'm', ' ', 'M', 'a', 'r', 'a',
"'", 's'], dtype='¡U1')]

and then I can rejoin the individual array into their own string of words:

```
''.join(wwl[0])
```

output:'I love'

Step 2: Break The Texts into Lists

Now lets apply that same logic on a grand scale to the all the rows in the table.

```
1 webdf['rawa']=webdf.raw.apply(lambda x: np.array(list(x)))
2 webdf['rawal']=webdf.rawa.apply(lambda x: np.split(x, dashpos))
```

Index	raw	rawa	rawal
0		[]	[array([], dtype=float64), array([], dtype=float64), array([]…
1	…	[' ' ' ' ' …	[array([' ', ' ', ' ', ' ', ' '], dtype='<U1'), array([' ', '…
2	…	[' ' ' ' ' …	[array([' ', ' ', ' ', ' ', ' '], dtype='<U1'), array([' ', '…
3		[]	[array([], dtype=float64), array([], dtype=float64), array([]…
4	…	[' ' ' ' ' …	[array([' ', ' ', ' ', ' ', ' '], dtype='<U1'), array([' ', '…
5		[]	[array([], dtype=float64), array([], dtype=float64), array([]…
6	P…	['P' 'l' 'a…	[array(['P', 'l', 'a', 'c', 'e'], dtype='<U1'), array([' ', '…
7	=…	['=' '=' '=…	[array(['=', '=', '=', '=', '='], dtype='<U1'), array([' ', '…
8	…	[' ' ' ' ' …	[array([' ', ' ', ' ', ' ', '1'], dtype='<U1'), array([' ', '…
9	…	[' ' ' ' ' …	[array([' ', ' ', ' ', ' ', '2'], dtype='<U1'), array([' ', '…
10	…	[' ' ' ' ' …	[array([' ', ' ', ' ', ' ', '3'], dtype='<U1'), array([' ', '…
11	…	[' ' ' ' ' …	[array([' ', ' ', ' ', ' ', '4'], dtype='<U1'), array([' ', '…
12	…	[' ' ' ' ' …	[array([' ', ' ', ' ', ' ', '5'], dtype='<U1'), array([' ', '…
13	…	[' ' ' ' ' …	[array([' ', ' ', ' ', ' ', '6'], dtype='<U1'), array([' ', '…
14	…	[' ' ' ' ' …	[array([' ', ' ', ' ', ' ', '7'], dtype='<U1'), array([' ', '…
15	…	[' ' ' ' ' …	[array([' ', ' ', ' ', ' ', '8'], dtype='<U1'), array([' ', '…

Step 2: Break The Texts into Lists

When we combine the subarrays into words in the column 'rawal', we want those words to be in newly created separate columns. So first, we want to separate out the subarrays within column 'rawal' to their own columns.

```
expl=[['toyota','black'],['bmw','red'],['ford','grey'],['audi','
    white']]
dfexp = pd.DataFrame({'First':expl})
```

	First
0	[toyota, black]
1	[bmw, red]
2	[ford, grey]
3	[audi, white]

```
dfexp[['Second','Third']]=pd.DataFrame(dfexp.First.values.tolist
    ())
```

	First	Second	Third
0	[toyota, black]	toyota	black
1	[bmw, red]	bmw	red
2	[ford, grey]	ford	grey

Step 3: Arrays in Their Own Columns

Now applying the same logic from the previous example, we need to break column rawal so that each array in that cell, (each row is a list of arrays) is in its own column.

But we have 2 problems that we have to address:

1. We are not certain that every year will have the same amount of columns (same amount of arrays in the list)
2. What should we name our columns given the uncertainty

The solution is that we can name our columns as numbers, and we can use the dashpos list to tell us how many columns we need. For example:

```
1 ww="I love dark chocolate mousse from Mara's"
2 wwa=np.array(list(ww))
3 wwl=np.split(wwa,[6,21])
4 print(wwl)
```

```
['I', ' ', 'l', 'o', 'v', 'e']
[' ', 'd', 'a', 'r', 'k', ' ', 'c', 'h', 'o', 'c', 'o', 'l', 'a','t', 'e']
[' ', 'm', 'o', 'u', 's', 's', 'e', ' ', 'f', 'r', 'o', 'm', ' ','M', 'a', 'r', 'a', '"', 's']
```

Step 3: Arrays in Their Own Columns

```
ww="I love dark chocolate mousse from Mara's"
wwa=np.array(list(ww))
wwl=np.split(wwa,[6,21])
print(wwl)
```

```
['I', ' ', 'l', 'o', 'v', 'e']
[' ', 'd', 'a', 'r', 'k', ' ', 'c', 'h', 'o', 'c', 'o', 'l', 'a','t', 'e']
[' ', 'm', 'o', 'u', 's', 's', 'e', ' ', 'f', 'r', 'o', 'm', ' ','M', 'a', 'r', 'a', '"', 's']
```

So if we split by 2 spaces, we get 3 vectors. Given this logic, we need the number of spaces + 1, amount of columns.

```
newcol=list(range(len(dashpos)+1))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The length len(dashpos), has the number of spaces between the columns, we add 1 to that, and generate a range of numbers. To show the range as a list, we need to use the list() function. Using that list as our new names for the columns, we can get to work.

Step 3: Arrays in Their Own Columns

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
1 webdf[newcol] = pd.DataFrame(webdf.rawal.values.tolist(), index=
      webdf.index)
```

Now those 10 columns are created. But our data isnt' still a proper table because of extra rows

Index	raw
0	
1	Credit Union Cherry Blossom Ten Mile Run
2	Washington, DC Sunday, April 1, 2012
3	
4	Official Male Results (Sorted By Net Time)
5	
6	Place Div /Tot Num Name Ag Hometown 5 Mile…
7	===== ========== ====== ========== == ========== ======…
8	1 1/347 9 Allan Kiprono 22 Kenya 22:3…
9	2 2/347 11 Lani Kiplagat 23 Kenya 22:3…
10	3 1/1093 31 John Korir 36 Kenya 23:2…
11	4 1/1457 15 Ian Burrell 27 Tucson AZ 23:5…
12	5 3/347 19 Jesse Cherry 24 Blowing Rock NC 23:5…
13	6 1/1490 37 Ketema Nugusse 31 Ethiopia 23:4…
14	7 2/1457 13 Josh Moen 29 Minneapolis MN 24:0…
15	8 3/1457 17 Patrick Rizzo 28 Boulder CO 24:2…
16	9 4/1457 41 Stephen Hallinan 26 Washington DC 25:0…

Step 4: Select the Right Rows

Index	raw
0	
1	Credit Union Cherry Blossom Ten Mile Run
2	Washington, DC Sunday, April 1, 2012
3	
4	Official Male Results (Sorted By Net Time)
5	
6	Place Div /Tot Num Name Ag Hometown 5 Mile…
7	===== ========== ====== ==========…== ==========…======
8	1 1/347 9 Allan Kiprono 22 Kenya 22:3…
9	2 2/347 11 Lani Kiplagat 23 Kenya 22:3…
10	3 1/1093 31 John Korir 36 Kenya 23:2…
11	4 1/1457 15 Ian Burrell 27 Tucson AZ 23:5…
12	5 3/347 19 Jesse Cherry 24 Blowing Rock NC 23:5…
13	6 1/1490 37 Ketema Nugusse 31 Ethiopia 23:4…
14	7 2/1457 13 Josh Moen 29 Minneapolis MN 24:0…
15	8 3/1457 17 Patrick Rizzo 28 Boulder CO 24:2…
16	9 4/1457 41 Stephen Hallinan 26 Washington DC 25:0…

We need to create a new table where we select the 6^{th} row and the 8^{th} row till the end. The easiest way is to to create a vector that will have the values 6,8,9,10,.....,7201

```
1 newrows=[rownumber-1]+list(range(rownumber+1,webdf.shape[0]))
```

The [rownumber-1], will be a list with just the value 6, that will added to the list from 8 onwards.

Step 5: Create New Table

```
newrows=[rownumber-1]+list(range(rownumber+1,webdf.shape[0]))
webdf2=webdf.loc[newrows,newcol]
```

Index	0	1	2	3	4	5	6	7	8	9
6	['P' '…	[' ' …	[' '…	[' ' 'N…	[' ' 'A…	[' ' 'H…	[' ' '5…	[' ' …	[' ' …	[' ']
8	[' ' '…	[' ' …	[' '…	[' ' 'A…	[' ' '2…	[' ' 'K…	[' ' ' …	[' ' …	[' ' …	[' ']
9	[' ' '…	[' ' …	[' '…	[' ' 'L…	[' ' '2…	[' ' 'K…	[' ' ' …	[' ' …	[' ' …	[' ']
10	[' ' '…	[' ' …	[' '…	[' ' 'J…	[' ' '3…	[' ' 'K…	[' ' ' …	[' ' …	[' ' …	[' ']
11	[' ' '…	[' ' …	[' '…	[' ' 'I…	[' ' '2…	[' ' 'T…	[' ' ' …	[' ' …	[' ' …	[' ']
12	[' ' '…	[' ' …	[' '…	[' ' 'J…	[' ' '2…	[' ' 'B…	[' ' ' …	[' ' …	[' ' …	[' ']

We only selected the columns where we split the arrays from their list. The next step is now to combine all those arrays, back into words. Usually we would use the apply(lambda r: ''.join(r)) function, but this can become tedious, instead we will utilize a function that allows us to run apply on ALL the columns. The function applymap(). It works exactly like apply(), except we can specify multiple columns or the whole table.

```
webdf3=webdf2.applymap(lambda r: ''.join(r))
```

Step 5: Create New Table

```
1 webdf3=webdf2.applymap(lambda r: ''.join(r))
```

Index	0	1	2	3	4	5	6	7	8	9
6	Place	Div /Tot	Num	Name	Ag	Hometo…	5 Mile	Time	Pace	
8	1	1/347	9	All…	22	Kenya	22:32	45:15	4:32	
9	2	2/347	11	Lan…	23	Kenya	22:38	46:28	4:39	
10	3	1/10…	31	Joh…	36	Kenya	23:20	47:33	4:46	
11	4	1/14…	15	Ian…	27	Tucson…	23:50	47:34	4:46	
12	5	3/347	19	Jes…	24	Blowin…	23:50	47:40	4:46	
13	6	1/14…	37	Ket…	31	Ethiop…	23:42	47:50	4:47	
14	7	2/14…	13	Jos…	29	Minnea…	24:06	48:38	4:52	
15	8	3/14…	17	Pat…	28	Boulde…	24:24	49:14	4:56	
16	9	4/14…	41	Ste…	26	Washin…	25:01	50:18	5:02	
17	10	2/14…	345	Pao…	31	Washin…	25:20	50:44	5:05	
18	11	3/14…	346	Dav…	32	Bridge…	25:33	50:56	5:06	
19	12	4/347	299	Fra…	23	Washin…	25:28	50:57	5:06	
20	13	4/14…	112	Ber…	32	Arling…	25:31	50:57	5:06	
21	14	1/931	290	Chr…	41	Alexan…	25:28	51:10	5:07	
22	15	5/14…	108	Dar…	29	Exton …	25:28	51:16	5:08	
23	16	6/14…	119	Jay…	28	Denver…	25:22	51:17	5:08	

Step 5: Create Final Table

Now we just have 4 steps left:

1. Get rid of excessive white spaces before and after the words
2. Creating a final table (datadf) with all the rows except for the first row
3. Renaming the column names of datadf using the first row of webdf3
4. Reseting the indexes

```
webdf3=webdf3.applymap(lambda r: r.strip())
datadf = webdf3.iloc[1:]
datadf.columns = webdf3.iloc[0].tolist()
datadf=datadf.reset_index(drop=True)
```

Index	Place	Div /Tot	Num	Name	Ag	Hometown	5 Mile	Time	Pace
0	1	1/347	9	Allan K…	22	Kenya	22:32	45:15	4:32
1	2	2/347	11	Lani Ki…	23	Kenya	22:38	46:28	4:39
2	3	1/1093	31	John Ko…	36	Kenya	23:20	47:33	4:46
3	4	1/1457	15	Ian Bur…	27	Tucson AZ	23:50	47:34	4:46
4	5	3/347	19	Jesse C…	24	Blowing R…	23:50	47:40	4:46
5	6	1/1490	37	Ketema …	31	Ethiopia	23:42	47:50	4:47
6	7	2/1457	13	Josh Mo…	29	Minneapol…	24:06	48:38	4:52
7	8	3/1457	17	Patrick…	28	Boulder CO	24:24	49:14	4:56
8	9	4/1457	41	Stephen…	26	Washingto…	25:01	50:18	5:02