

# Pandas: scrape, wrangle, and visualize data

For this module, we will use the Pandas package along with several other popular Python packages to scrape html data, wrangle it using a dataframe, and then conduct some visual analysis.

To get started, you will need a Python installation (3.6.3 or later is recommended).

```
$ python --version
3.6.3
```

Clone or download the repository [https://github.com/LSU-Analytics/sql\\_saturday\\_2018.git](https://github.com/LSU-Analytics/sql_saturday_2018.git) ([https://github.com/LSU-Analytics/sql\\_saturday\\_2018.git](https://github.com/LSU-Analytics/sql_saturday_2018.git)).

```
$ git clone https://github.com/LSU-Analytics/sql_saturday_2018.git
```

Run this command to install the packages:

```
$ pip install -r requirements.txt
```

Or you can install the packages individually.

```
$ pip install pandas
$ pip install beautifulsoup4
$ pip install jupyter
$ pip install matplotlib
$ pip install numpy
$ pip install requests
$ pip install seaborn
```

## Jupyter Notebook

Using Jupyter Notebook, you should start by importing the necessary modules (pandas, numpy, matplotlib.pyplot, seaborn). If you don't have Jupyter Notebook installed, you can find it here: <http://jupyter.org/install> (<http://jupyter.org/install>).

```
In [81]: # import the required packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from urllib.request import urlopen
from bs4 import BeautifulSoup

%matplotlib inline
```

## Get some data

Where do we get data? That's easy...data is everywhere. Pandas makes it easy to import data from multiple file types (csv, xlsx, txt, sql), pull from APIs (usually as JSON), or obtain raw HTML. For this example, we will use HTML since it usually requires a bit more cleaning.

```
In [82]: url = "http://www.hubertiming.com/results/2017GPTR10K"
html = urlopen(url)
```

Next step is to create a Beautiful Soup object from the html. This is done by passing the html to the BeautifulSoup() function. The BeautifulSoup package is used to parse the html. The second argument 'lxml' is the html parser whose details you do not need to worry about at this point.

```
In [83]: gumbo = BeautifulSoup(html, 'lxml')
type(gumbo)
```

```
Out[83]: bs4.BeautifulSoup
```

The soup object allows you to extract interesting information about the website you're scraping such as getting the title of the page as shown below.

```
In [84]: # Get the page title
title = gumbo.title
print(title)
```

```
<title>2017 Intel Great Place to Run 10K \ Urban Clash Games Race Results</title>
```

You can also get the text of the webpage and quickly print it out to check if it is what you expect.

```
In [85]: # Print out the text
text = gumbo.get_text()
# print(text)
```

You can use the find\_all() method of on our gumbo object to extract useful html tags within the html. Examples of useful tags include < a > for hyperlinks, < table > for tables, < tr > for table rows, < th > for table headers, and < td > for table cells. The code below shows how to extract all the hyperlinks within the webpage.

```
In [86]: gumbo.find_all('a')
```

```
Out[86]: [<a class="btn btn-primary btn-lg" href="/results/2017GPTR" role="button" style
="margin: 0px 0px 5px 5px">5K</a>,
<a href="http://hubertiming.com/">Huber Timing Home</a>,
<a href="#individual">Individual Results</a>,
<a href="#team">Team Results</a>,
<a href="mailto:timing@hubertiming.com">timing@hubertiming.com</a>,
<a href="#tabs-1" style="font-size: 18px">Results</a>,
<a name="individual"></a>,
<a name="team"></a>,
<a href="http://www.hubertiming.com/">Huber Timing</a>,
<a href="http://facebook.com/hubertiming/"></a>]
```

As you can see from the output above, html tags sometimes come with attributes such as class, src, etc. These attributes provide additional information about html elements. You can use a little python coding to extract only the hyperlinks.

```
In [87]: all_links = gumbo.find_all("a")
for link in all_links:
    print(link.get("href"))
```

```
/results/2017GPTR
http://hubertiming.com/ (http://hubertiming.com/)
#individual
#team
mailto:timing@hubertiming.com
#tabs-1
None
None
http://www.hubertiming.com/ (http://www.hubertiming.com/)
http://facebook.com/hubertiming/ (http://facebook.com/hubertiming/)
```

To print out table rows only, pass the 'tr' argument in `gumbo.find_all()`.

```
In [88]: # Print the first 10 rows for sanity check
rows = gumbo.find_all('tr')
print(rows[:10])
<td>2 of 154</td>
<td>00:00:03</td>
<td>00:39:24</td>
<td>INTEL TEAM F</td>
</tr>, <tr>
<td>6</td>
<td>642</td>
<td>JONATHON TRAN</td>
<td>M</td>
<td>PORTLAND</td>
<td>OR</td>
<td>00:39:49</td>
<td>06:25</td>
<td>6 of 414</td>
<td>M 18-25</td>
<td>1 of 34</td>
<td>00:00:06</td>
<td>00:39:55</td>
<td></td>
</tr>]
```

The goal is to take data from an html table and convert it into a dataframe for easier manipulation using Python. To get there, you should get all table rows in list form first and then convert that list into a dataframe. Below is a for loop that iterates through table rows and prints out the cells of the rows.

```
In [89]: for row in rows:
row_td = row.find_all('td')
print(row_td)
type(row_td)
```

```
[<td>14TH</td>, <td>INTEL TEAM M</td>, <td>04:43:23</td>, <td>00:58:59 - DANIEL
LE CASILLAS</td>, <td>01:02:06 - RAMYA MERUVA</td>, <td>01:17:06 - PALLAVI J SH
INDE</td>, <td>01:25:11 - NALINI MURARI</td>]
```

```
Out[89]: bs4.element.ResultSet
```

The output shows that each row is printed with html tags embedded in each row. This is not what you want.

The easiest way to remove html tags is to use BeautifulSoup, and it takes just one line of code to do this. Pass the string of interest into BeautifulSoup() and use the get\_text() method to extract the text without html tags.

```
In [90]: str_cells = str(row_td)
cleantext = BeautifulSoup(str_cells, "lxml").get_text()
print(cleantext)
```

```
[14TH, INTEL TEAM M, 04:43:23, 00:58:59 - DANIELLE CASILLAS, 01:02:06 - RAMYA M
ERUVA, 01:17:06 - PALLAVI J SHINDE, 01:25:11 - NALINI MURARI]
```

Using a loop, we will create a list of cleaned text from the rows.

```
In [91]: list_rows = []

for row in rows:
    row_td = row.find_all('td')
    str_cells = str(row_td)
    cleantext = BeautifulSoup(str_cells, "lxml").get_text()
    list_rows.append(cleantext)

type(list_rows)
```

Out[91]: list

The next step is to convert the list into a dataframe and get a quick view of the first 10 rows using Pandas.

```
In [92]: df = pd.DataFrame(list_rows)
df.head(10)
```

```
Out[92]:
```

|   | 0   |
|---|---|
| 0 | [Finishers:, 577]                                 |
| 1 | [Male:, 414]                                      |
| 2 | [Female:, 163]                                    |
| 3 | []  |
| 4 | [1, 814, JARED WILSON, M, TIGARD, OR, 00:36:21... |
| 5 | [2, 573, NATHAN A SUSTERSIC, M, PORTLAND, OR, ... |
| 6 | [3, 687, FRANCISCO MAYA, M, PORTLAND, OR, 00:3... |
| 7 | [4, 623, PAUL MORROW, M, BEAVERTON, OR, 00:38:... |
| 8 | [5, 569, DEREK G OSBORNE, M, HILLSBORO, OR, 00... |
| 9 | [6, 642, JONATHON TRAN, M, PORTLAND, OR, 00:39... |

## Regulators, mount up! It's time to wrangle.

The dataframe is not in the format we want. To clean it up, you should split the "0" column into multiple columns at the comma position. This is accomplished by using the `str.split()` method.

```
In [93]: df1 = df[0].str.split(',', expand=True)
df1.head(10)
```

```
Out[93]:
```

|   | 0           | 1    | 2                     | 3    | 4         | 5    | 6        | 7     | 8           | 9              | 10          |    |
|---|-------------|------|-----------------------|------|-----------|------|----------|-------|-------------|----------------|-------------|----|
| 0 | [Finishers: | 577] | None                  | None | None      | None | None     | None  | None        | None           | None        |    |
| 1 | [Male:      | 414] | None                  | None | None      | None | None     | None  | None        | None           | None        |    |
| 2 | [Female:    | 163] | None                  | None | None      | None | None     | None  | None        | None           | None        |    |
| 3 | []          | None | None                  | None | None      | None | None     | None  | None        | None           | None        |    |
| 4 | [1          | 814  | JARED<br>WILSON       | M    | TIGARD    | OR   | 00:36:21 | 05:51 | 1 of<br>414 | M<br>36-<br>45 | 1 of<br>152 | 00 |
| 5 | [2          | 573  | NATHAN A<br>SUSTERSIC | M    | PORTLAND  | OR   | 00:36:42 | 05:55 | 2 of<br>414 | M<br>26-<br>35 | 1 of<br>154 | 00 |
| 6 | [3          | 687  | FRANCISCO<br>MAYA     | M    | PORTLAND  | OR   | 00:37:44 | 06:05 | 3 of<br>414 | M<br>46-<br>55 | 1 of<br>64  | 00 |
| 7 | [4          | 623  | PAUL<br>MORROW        | M    | BEAVERTON | OR   | 00:38:34 | 06:13 | 4 of<br>414 | M<br>36-<br>45 | 2 of<br>152 | 00 |
| 8 | [5          | 569  | DEREK G<br>OSBORNE    | M    | HILLSBORO | OR   | 00:39:21 | 06:20 | 5 of<br>414 | M<br>26-<br>35 | 2 of<br>154 | 00 |
| 9 | [6          | 642  | JONATHON<br>TRAN      | M    | PORTLAND  | OR   | 00:39:49 | 06:25 | 6 of<br>414 | M<br>18-<br>25 | 1 of<br>34  | 00 |

This looks much better, but there is still work to do. The dataframe has unwanted square brackets surrounding each row. You can use the `strip()` method to remove the opening square bracket on column "0."

```
In [94]: df1[0] = df1[0].str.strip('[')
df1.head(10)
```

```
Out[94]:
```

|   | 0          | 1    | 2                     | 3    | 4         | 5    | 6        | 7     | 8           | 9              | 10          |    |
|---|------------|------|-----------------------|------|-----------|------|----------|-------|-------------|----------------|-------------|----|
| 0 | Finishers: | 577] | None                  | None | None      | None | None     | None  | None        | None           | None        |    |
| 1 | Male:      | 414] | None                  | None | None      | None | None     | None  | None        | None           | None        |    |
| 2 | Female:    | 163] | None                  | None | None      | None | None     | None  | None        | None           | None        |    |
| 3 | ]          | None | None                  | None | None      | None | None     | None  | None        | None           | None        |    |
| 4 | 1          | 814  | JARED<br>WILSON       | M    | TIGARD    | OR   | 00:36:21 | 05:51 | 1 of<br>414 | M<br>36-<br>45 | 1 of<br>152 | 00 |
| 5 | 2          | 573  | NATHAN A<br>SUSTERSIC | M    | PORTLAND  | OR   | 00:36:42 | 05:55 | 2 of<br>414 | M<br>26-<br>35 | 1 of<br>154 | 00 |
| 6 | 3          | 687  | FRANCISCO<br>MAYA     | M    | PORTLAND  | OR   | 00:37:44 | 06:05 | 3 of<br>414 | M<br>46-<br>55 | 1 of<br>64  | 00 |
| 7 | 4          | 623  | PAUL<br>MORROW        | M    | BEAVERTON | OR   | 00:38:34 | 06:13 | 4 of<br>414 | M<br>36-<br>45 | 2 of<br>152 | 00 |
| 8 | 5          | 569  | DEREK G<br>OSBORNE    | M    | HILLSBORO | OR   | 00:39:21 | 06:20 | 5 of<br>414 | M<br>26-<br>35 | 2 of<br>154 | 00 |
| 9 | 6          | 642  | JONATHON<br>TRAN      | M    | PORTLAND  | OR   | 00:39:49 | 06:25 | 6 of<br>414 | M<br>18-<br>25 | 1 of<br>34  | 00 |

The table is missing table headers. You can use the `find_all()` method to get the table headers.

```
In [95]: col_labels = gumbo.find_all('th')
```

Once again, you can use Beautiful Soup to extract text in between html tags for table headers.

```
In [96]: all_header = []
col_str = str(col_labels)
cleantext2 = BeautifulSoup(col_str, "lxml").get_text()
all_header.append(cleantext2)
print(all_header)
```

```
['[Place, Bib, Name, Gender, City, State, Chip Time, Chip Pace, Gender Place, A  
ge Group, Age Group Place, Time to Start, Gun Time, Team]']
```

```
In [97]: df2 = pd.DataFrame(all_header)
df2.head()
```

```
Out[97]:
```

| 0   |
|---|
| [Place, Bib, Name, Gender, City, State, Chip T... |

As before, split column "0" into multiple columns at the comma position for all rows.

```
In [98]: df3 = df2[0].str.split(',', expand=True)
df3.head()
```

```
Out[98]:
```

|   | 0      | 1   | 2    | 3      | 4    | 5     | 6         | 7         | 8            | 9         | 10              | 11            | 12       | 13    |
|---|--------|-----|------|--------|------|-------|-----------|-----------|--------------|-----------|-----------------|---------------|----------|-------|
| 0 | [Place | Bib | Name | Gender | City | State | Chip Time | Chip Pace | Gender Place | Age Group | Age Group Place | Time to Start | Gun Time | Team] |

The two dataframes can be concatenated into one using the `concat()` method as illustrated below.

```
In [99]: frames = [df3, df1]

df4 = pd.concat(frames)
df4.head(10)
```

```
Out[99]:
```

|   | 0          | 1    | 2                  | 3      | 4         | 5     | 6         | 7         | 8            | 9         | 10              | 11            | 12       | 13    |
|---|------------|------|--------------------|--------|-----------|-------|-----------|-----------|--------------|-----------|-----------------|---------------|----------|-------|
| 0 | [Place     | Bib  | Name               | Gender | City      | State | Chip Time | Chip Pace | Gender Place | Age Group | Age Group Place | Time to Start | Gun Time | Team] |
| 0 | Finishers: | 577] | None               | None   | None      | None  | None      | None      | None         | None      | None            | None          | None     | None  |
| 1 | Male:      | 414] | None               | None   | None      | None  | None      | None      | None         | None      | None            | None          | None     | None  |
| 2 | Female:    | 163] | None               | None   | None      | None  | None      | None      | None         | None      | None            | None          | None     | None  |
| 3 | ]          | None | None               | None   | None      | None  | None      | None      | None         | None      | None            | None          | None     | None  |
| 4 | 1          | 814  | JARED WILSON       | M      | TIGARD    | OR    | 00:36:21  | 05:51     | 1 of 414     | M 36-45   | 1 of 15         |               |          |       |
| 5 | 2          | 573  | NATHAN A SUSTERSIC | M      | PORTLAND  | OR    | 00:36:42  | 05:55     | 2 of 414     | M 26-35   | 1 of 15         |               |          |       |
| 6 | 3          | 687  | FRANCISCO MAYA     | M      | PORTLAND  | OR    | 00:37:44  | 06:05     | 3 of 414     | M 46-55   | 1 of 6          |               |          |       |
| 7 | 4          | 623  | PAUL MORROW        | M      | BEAVERTON | OR    | 00:38:34  | 06:13     | 4 of 414     | M 36-45   | 2 of 15         |               |          |       |
| 8 | 5          | 569  | DEREK G OSBORNE    | M      | HILLSBORO | OR    | 00:39:21  | 06:20     | 5 of 414     | M 26-35   | 2 of 15         |               |          |       |

Below shows how to assign the first row to be the table header.



```
In [100]: df5 = df4.rename(columns=df4.iloc[0])
df5.head()
```

Out[100]:

|   | [Place     | Bib  | Name | Gender | City | State | Chip Time | Chip Pace | Gender Place | Age Group | Age Group Place | Time to Start | Gun Time |
|---|------------|------|------|--------|------|-------|-----------|-----------|--------------|-----------|-----------------|---------------|----------|
| 0 | [Place     | Bib  | Name | Gender | City | State | Chip Time | Chip Pace | Gender Place | Age Group | Age Group Place | Time to Start | Gun Time |
| 0 | Finishers: | 577] | None | None   | None | None  | None      | None      | None         | None      | None            | None          | None     |
| 1 | Male:      | 414] | None | None   | None | None  | None      | None      | None         | None      | None            | None          | None     |
| 2 | Female:    | 163] | None | None   | None | None  | None      | None      | None         | None      | None            | None          | None     |
| 3 | ]          | None | None | None   | None | None  | None      | None      | None         | None      | None            | None          | None     |

At this point, the table is (mostly) properly formatted. For analysis, you can start by getting an overview of the data as shown below.

```
In [101]: df5.info()
df5.shape
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 597 entries, 0 to 595
Data columns (total 14 columns):
[Place          597 non-null object
Bib              596 non-null object
Name            593 non-null object
Gender          593 non-null object
City            593 non-null object
State           593 non-null object
Chip Time       593 non-null object
Chip Pace       578 non-null object
Gender Place    578 non-null object
Age Group       578 non-null object
Age Group Place 578 non-null object
Time to Start   578 non-null object
Gun Time        578 non-null object
Team]           578 non-null object
dtypes: object(14)
memory usage: 70.0+ KB
```

Out[101]: (597, 14)

The table has 597 rows and 14 columns. You can drop all rows with any missing values.

```
In [102]: df6 = df5.dropna(axis=0, how='any')
df6.head(5)
```

Out[102]:

|   | [Place | Bib | Name               | Gender | City      | State | Chip Time | Chip Pace | Gender Place | Age Group | Age Group Place |
|---|--------|-----|--------------------|--------|-----------|-------|-----------|-----------|--------------|-----------|-----------------|
| 0 | [Place | Bib | Name               | Gender | City      | State | Chip Time | Chip Pace | Gender Place | Age Group | Age Group Place |
| 4 | 1      | 814 | JARED WILSON       | M      | TIGARD    | OR    | 00:36:21  | 05:51     | 1 of 414     | M 36-45   | 1 of 152        |
| 5 | 2      | 573 | NATHAN A SUSTERSIC | M      | PORTLAND  | OR    | 00:36:42  | 05:55     | 2 of 414     | M 26-35   | 1 of 154        |
| 6 | 3      | 687 | FRANCISCO MAYA     | M      | PORTLAND  | OR    | 00:37:44  | 06:05     | 3 of 414     | M 46-55   | 1 of 64         |
| 7 | 4      | 623 | PAUL MORROW        | M      | BEAVERTON | OR    | 00:38:34  | 06:13     | 4 of 414     | M 36-45   | 2 of 152        |



Also, notice how the table header is replicated as the first row in df5. It can be dropped using the following line of code.

```
In [103]: df7 = df6.drop(df6.index[0])
df7.head()
```

Out[103]:

|   | [Place | Bib | Name               | Gender | City      | State | Chip Time | Chip Pace | Gender Place | Age Group | Age Group Place |
|---|--------|-----|--------------------|--------|-----------|-------|-----------|-----------|--------------|-----------|-----------------|
| 4 | 1      | 814 | JARED WILSON       | M      | TIGARD    | OR    | 00:36:21  | 05:51     | 1 of 414     | M 36-45   | 1 of 152        |
| 5 | 2      | 573 | NATHAN A SUSTERSIC | M      | PORTLAND  | OR    | 00:36:42  | 05:55     | 2 of 414     | M 26-35   | 1 of 154        |
| 6 | 3      | 687 | FRANCISCO MAYA     | M      | PORTLAND  | OR    | 00:37:44  | 06:05     | 3 of 414     | M 46-55   | 1 of 64         |
| 7 | 4      | 623 | PAUL MORROW        | M      | BEAVERTON | OR    | 00:38:34  | 06:13     | 4 of 414     | M 36-45   | 2 of 152        |
| 8 | 5      | 569 | DEREK G OSBORNE    | M      | HILLSBORO | OR    | 00:39:21  | 06:20     | 5 of 414     | M 26-35   | 2 of 154        |



You can perform more data cleaning by renaming the '[Place' and ' Team]' columns.

```
In [104]: df7.rename(columns={'Place': 'Place'},inplace=True)
df7.rename(columns={' Team}': 'Team'},inplace=True)
df7['Team'] = df7['Team'].str.strip(' ')
df7.head()
```

Out[104]:

|   | Place | Bib | Name               | Gender | City      | State | Chip Time | Chip Pace | Gender Place | Age Group | Age Group Place |
|---|-------|-----|--------------------|--------|-----------|-------|-----------|-----------|--------------|-----------|-----------------|
| 4 | 1     | 814 | JARED WILSON       | M      | TIGARD    | OR    | 00:36:21  | 05:51     | 1 of 414     | M 36-45   | 1 of 152        |
| 5 | 2     | 573 | NATHAN A SUSTERSIC | M      | PORTLAND  | OR    | 00:36:42  | 05:55     | 2 of 414     | M 26-35   | 1 of 154        |
| 6 | 3     | 687 | FRANCISCO MAYA     | M      | PORTLAND  | OR    | 00:37:44  | 06:05     | 3 of 414     | M 46-55   | 1 of 64         |
| 7 | 4     | 623 | PAUL MORROW        | M      | BEAVERTON | OR    | 00:38:34  | 06:13     | 4 of 414     | M 36-45   | 2 of 152        |
| 8 | 5     | 569 | DEREK G OSBORNE    | M      | HILLSBORO | OR    | 00:39:21  | 06:20     | 5 of 414     | M 26-35   | 2 of 154        |

It took a while to get here, but at this point, the dataframe is in the desired format. Now you can move on to the exciting part and start plotting the data and computing interesting statistics.

## Visualization & Analysis

The first question to answer is, what was the average finish time (in minutes) for the runners? You need to convert the column "Chip Time" into just minutes. One way to do this is to convert the column to a list first for manipulation.

```
In [105]: time_list = df7[' Chip Time'].tolist()

# You can use a for Loop to convert 'Chip Time' to minutes

time_mins = []
for i in time_list:
    h, m, s = i.split(':')
    math = (int(h) * 3600 + int(m) * 60 + int(s))/60
    time_mins.append(math)
# print(time_mins)
```

The next step is to convert the list back into a dataframe and make a new column ("Runner\_mins") for runner chip times expressed in just minutes.

```
In [106]: df7['Runner_mins'] = time_mins
df7.head()
```

Out[106]:

|   | Place | Bib | Name               | Gender | City      | State | Chip Time | Chip Pace | Gender Place | Age Group | Age Group Place |
|---|-------|-----|--------------------|--------|-----------|-------|-----------|-----------|--------------|-----------|-----------------|
| 4 | 1     | 814 | JARED WILSON       | M      | TIGARD    | OR    | 00:36:21  | 05:51     | 1 of 414     | M 36-45   | 1 of 152        |
| 5 | 2     | 573 | NATHAN A SUSTERSIC | M      | PORTLAND  | OR    | 00:36:42  | 05:55     | 2 of 414     | M 26-35   | 1 of 154        |
| 6 | 3     | 687 | FRANCISCO MAYA     | M      | PORTLAND  | OR    | 00:37:44  | 06:05     | 3 of 414     | M 46-55   | 1 of 64         |
| 7 | 4     | 623 | PAUL MORROW        | M      | BEAVERTON | OR    | 00:38:34  | 06:13     | 4 of 414     | M 36-45   | 2 of 152        |
| 8 | 5     | 569 | DEREK G OSBORNE    | M      | HILLSBORO | OR    | 00:39:21  | 06:20     | 5 of 414     | M 26-35   | 2 of 154        |



The code below shows how to calculate statistics for numeric columns only in the dataframe.

```
In [107]: # df7.describe(include=[np.number])
df7.describe()
```

Out[107]:

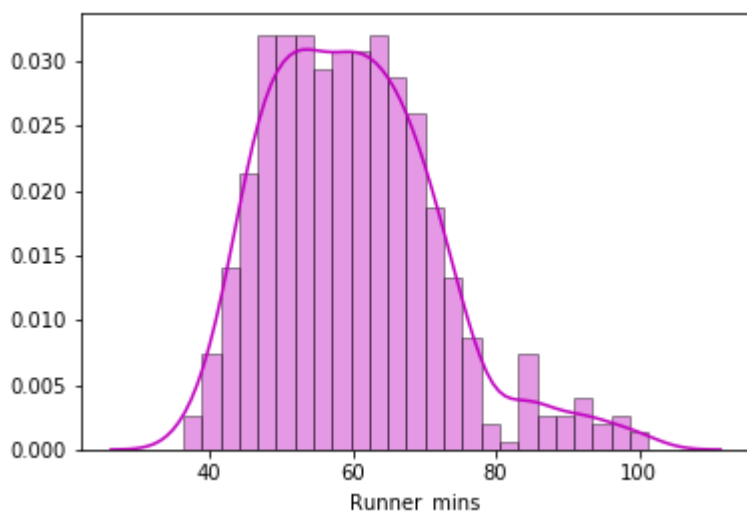
|       | Runner_mins |
|-------|-------------|
| count | 577.000000  |
| mean  | 60.035933   |
| std   | 11.970623   |
| min   | 36.350000   |
| 25%   | 51.000000   |
| 50%   | 59.016667   |
| 75%   | 67.266667   |
| max   | 101.300000  |

Interestingly, the average chip time for all runners was ~60 mins. The fastest 10K runner finished in 36.35 mins, and the slowest runner finished in 101.30 minutes.

Another question to answer is: Did the runners' finish times follow a normal distribution?

Below is a distribution plot of runners' chip times plotted using the seaborn library. The distribution looks almost normal.

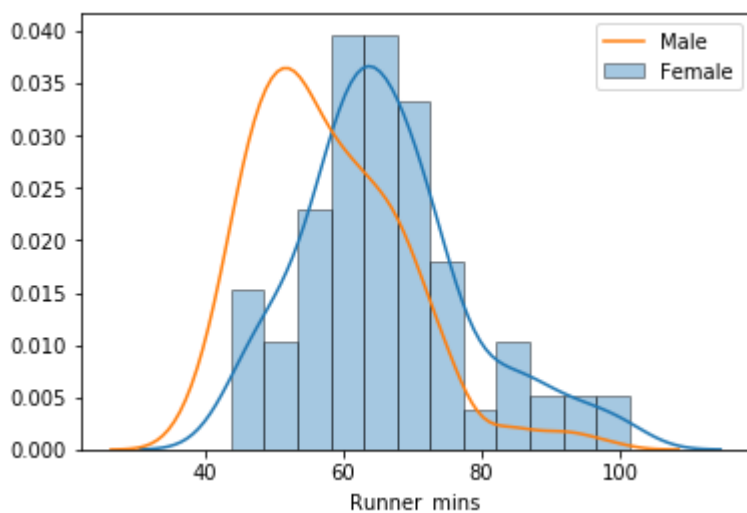
```
In [108]: x = df7['Runner_mins']
ax = sns.distplot(x, hist=True, kde=True, rug=False, color='m', bins=25, hist_kws:
plt.show()
```



Are there were any performance differences between the males and females runners? Below is a distribution plot of chip times for males and females.

```
In [109]: f_fuko = df7.loc[df7[' Gender']==' F']['Runner_mins']
m_fuko = df7.loc[df7[' Gender']==' M']['Runner_mins']
sns.distplot(f_fuko, hist=True, kde=True, rug=False, hist_kws={'edgecolor':'black
sns.distplot(m_fuko, hist=False, kde=True, rug=False, hist_kws={'edgecolor':'blac
plt.legend()
```

Out[109]: <matplotlib.legend.Legend at 0x10a983e48>



The distribution indicates that females were slower than males on average. You can use the `groupby()` method to compute summary statistics for males and females separately as shown below.

```
In [110]: g_stats = df7.groupby(" Gender", as_index=True).describe()
print(g_stats)
```

```
      Runner_mins
      count      mean      std      min      25%      50%
Gender
F         163.0  66.119223  12.184440  43.766667  58.758333  64.616667
M         414.0  57.640821  11.011857  36.350000  49.395833  55.791667

      75%      max
Gender
F         72.058333  101.300000
M         64.804167   98.516667
```

## Conclusion

In this tutorial, you performed web scraping using Python. You used the Beautiful Soup library to parse html data and convert it into a form that can be used for analysis. Using Pandas, You cleaned the data and created useful plots to reveal interesting trends using Python's matplotlib and seaborn libraries. After this tutorial, you should be able to use Python to easily scrape data from the web, apply cleaning techniques and extract useful insights from the data.

If you would like to learn more about Python, take DataCamp's free Intro to Python for Data Science course. <https://www.datacamp.com/courses/intro-to-python-for-data-science>  
(<https://www.datacamp.com/courses/intro-to-python-for-data-science>)

```
In [ ]:
```