DataEng S22: Data Gathering Activity

# *[this lab activity originally came from datacamp.com]*

This lab activity explains how to extract data from a website, manipulate and clean data using Python's Pandas library, and then visualize using Python's Matplotlib library.

This lab touches on:

* Data extraction from the web using Python's Beautiful Soup module
* Data transformation using Python's Pandas library
* Data visualization using Python's Matplotlib library

For this lab we use data from a 10K road race that took place in Hillsboro, OR on June 2017. Specifically, you will analyze the performance of the 10K runners and answer questions such as:

* What was the average finish time for the runners?
* Did the runners' finish times follow a normal distribution?
* Were there any performance differences between males and females of various age groups?

Be warned: some of the data mentioned in this tutorial has aged, meaning that it has changed over time in subtle ways. This means that some of the steps in the tutorial will not work perfectly, and you will need to adjust your code to get correct results.

Due Friday at the end of the week: April 8 (10pm). Use the submission form linked from the class Materials page to submit your completed assignment.

## Web Data Gathering using Beautiful Soup

Using your preferred python environment, you should start by importing the necessary modules (pandas, numpy, matplotlib.pyplot, seaborn). I suggest you use Jupyter Notebook or Google Colab for this lab, but any python environment probably will work.

(Jupyter notebook is available via the Anaconda Python package)

BTW, to easily display the plots in Jupyter Notebook, include the line %matplotlib inline as shown below.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

Also import the libraries shown below. You will use the urllib.request module to open URLs. And you’ll use BeautifulSoup to extract data from html files. The Beautiful Soup library's name is bs4 which stands for BeautifulSoup, version 4.

from urllib.request import urlopen

from bs4 import BeautifulSoup

After importing necessary modules, you should specify the URL containing the dataset and pass it to urlopen() to get the html of the page.

url = "http://www.hubertiming.com/results/2017GPTR10K"

html = urlopen(url)

[hubertiming.com](https://www.hubertiming.com/) is a husband/wife team in Bend, OR who provide timing and race result services to race organizers throughout the USA. They provide timing information for dozens of races each year.

After you get the HTML page create a Beautiful Soup object from the html. This is done by passing the html to the BeautifulSoup() function. The Beautiful Soup package is used to parse the html, that is, take the raw html text and break it into Python objects. The second argument 'lxml' is the html parser, one of several such parsers available for use.

soup = BeautifulSoup(html, 'lxml')

type(soup)

bs4.BeautifulSoup

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

# Get the title

title = soup.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.

# Print out the text

text = soup.get\_text()

#print(soup.text)

Next, use the find\_all() method of soup to extract useful html tags within a webpage. 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.

soup.find\_all('a')

[<a class="btn btn-primary btn-lg" href="/results/2017GPTR" role="button">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/"><img height="65" src="/sites/all/themes/hubertiming/images/clockWithFinishSign\_small.png" width="50"/>Huber Timing</a>,

<a href="http://facebook.com/hubertiming/"><img src="/results/FB-f-Logo\_\_blue\_50.png"/></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 for loop and the get('"href") method to extract and print out only hyperlinks.

all\_links = soup.find\_all("a")

for link in all\_links:

print(link.get("href"))

/results/2017GPTR

http://hubertiming.com/

#individual

#team

mailto:timing@hubertiming.com

#tabs-1

None

None

http://www.hubertiming.com/

http://facebook.com/hubertiming/

To print out table rows only, pass the 'tr' argument in soup.find\_all().

# Print the first 10 rows for sanity check

rows = soup.find\_all('tr')

print(rows[:10])

[<tr><td>Finishers:</td><td>577</td></tr>, <tr><td>Male:</td><td>414</td></tr>, <tr><td>Female:</td><td>163</td></tr>, <tr class="header">

<th>Place</th>

<th>Bib</th>

<th>Name</th>

<th>Gender</th>

<th>City</th>

<th>State</th>

<th>Chip Time</th>

<th>Chip Pace</th>

<th>Gender Place</th>

<th>Age Group</th>

<th>Age Group Place</th>

<th>Time to Start</th>

<th>Gun Time</th>

<th>Team</th>

</tr>, <tr>

<td>1</td>

<td>814</td>

<td>JARED WILSON</td>

<td>M</td>

<td>TIGARD</td>

<td>OR</td>

<td>00:36:21</td>

<td>05:51</td>

<td>1 of 414</td>

<td>M 36-45</td>

<td>1 of 152</td>

<td>00:00:03</td>

<td>00:36:24</td>

<td></td>

</tr>, <tr>

<td>2</td>

<td>573</td>

<td>NATHAN A SUSTERSIC</td>

<td>M</td>

<td>PORTLAND</td>

<td>OR</td>

<td>00:36:42</td>

<td>05:55</td>

<td>2 of 414</td>

<td>M 26-35</td>

<td>1 of 154</td>

<td>00:00:03</td>

<td>00:36:45</td>

<td>INTEL TEAM F</td>

</tr>, <tr>

<td>3</td>

<td>687</td>

<td>FRANCISCO MAYA</td>

<td>M</td>

<td>PORTLAND</td>

<td>OR</td>

<td>00:37:44</td>

<td>06:05</td>

<td>3 of 414</td>

<td>M 46-55</td>

<td>1 of 64</td>

<td>00:00:04</td>

<td>00:37:48</td>

<td></td>

</tr>, <tr>

<td>4</td>

<td>623</td>

<td>PAUL MORROW</td>

<td>M</td>

<td>BEAVERTON</td>

<td>OR</td>

<td>00:38:34</td>

<td>06:13</td>

<td>4 of 414</td>

<td>M 36-45</td>

<td>2 of 152</td>

<td>00:00:03</td>

<td>00:38:37</td>

<td></td>

</tr>, <tr>

<td>5</td>

<td>569</td>

<td>DEREK G OSBORNE</td>

<td>M</td>

<td>HILLSBORO</td>

<td>OR</td>

<td>00:39:21</td>

<td>06:20</td>

<td>5 of 414</td>

<td>M 26-35</td>

<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>]

One goal of this lab is to learn how to convert such table data to a Python Pandas dataframe for easier transformation and analysis. You should get all of the table’s rows in list form first and then convert that list into a pandas dataframe. Below is a for loop that iterates through table rows and prints out the cells of the rows. Again, BeautifulSoup helps you to process/access the data in ways that make sense for an HTML document with a call to find\_all(‘td’). “td” is the tag used to delimit table cells.

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 - DANIELLE CASILLAS</td>, <td>01:02:06 - RAMYA MERUVA</td>, <td>01:17:06 - PALLAVI J SHINDE</td>, <td>01:25:11 - NALINI MURARI</td>]

bs4.element.ResultSet

The output above shows that each row is printed with html tags embedded in each row. This is not what you want. You can remove the html tags using Beautiful Soup or regular expressions.

To remove html tags using Beautiful Soup, pass the string of interest into BeautifulSoup() and use the get\_text() method to extract the text without html tags.

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 MERUVA, 01:17:06 - PALLAVI J SHINDE, 01:25:11 - NALINI MURARI]

To gain a better understanding of the problem, try doing the same extraction with regular expressions. Be sure to import the re (regular expressions) module in your python environment. The code below shows how to build a regular expression that finds all the characters inside the < td > html tags and replace them with an empty string for each table row.

First, the code compiles a regular expression by passing the regex to re.compile(). Compilation is not required, but many python programming resources recommend compilation of regex to improve performance. This is especially the case if the same regex is to be used over many lines of input text.

The **'<.\*?>'** regex will match an opening angle bracket followed by anything and followed by a closing angle bracket. The ‘?’ character causes the regex to match text in a non-greedy fashion, that is, it matches the shortest possible string. If you omit the question mark, it will match all the text between the first opening angle bracket and the last closing angle bracket which would lead to confusing results.

After compiling the regex, use the re.sub() method to find all the substrings where the regular expression matches and replace each with an empty string. The full code below generates an empty list, extracts text between html tags for each row, and appends it to the assigned list.

import re

list\_rows = []

for row in rows:

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

str\_cells = str(cells)

clean = re.compile('<.\*?>')

clean2 = (re.sub(clean, '',str\_cells))

list\_rows.append(clean2)

print(clean2)

type(clean2)

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

str

Using regex works well for this task, but I also hope that you can see what an advantage BeautifulSoup gives you for the specific job of parsing html documents!

The next step is to convert the list into a pandas dataframe and get a quick view of the first 10 rows of data. Pandas is a wonderful tool for manipulating python data as tables. It’s almost like having a database server or google sheets within your python environment. We will use a couple of features in this lab (and throughout the DataEng course), but I encourage you to explore it much further, see [the pandas wikipedia page](https://en.wikipedia.org/wiki/Pandas_%28software%29) for a start.

df = pd.DataFrame(list\_rows)

df.head(10)

|  |  |
| --- | --- |
|  | **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... |

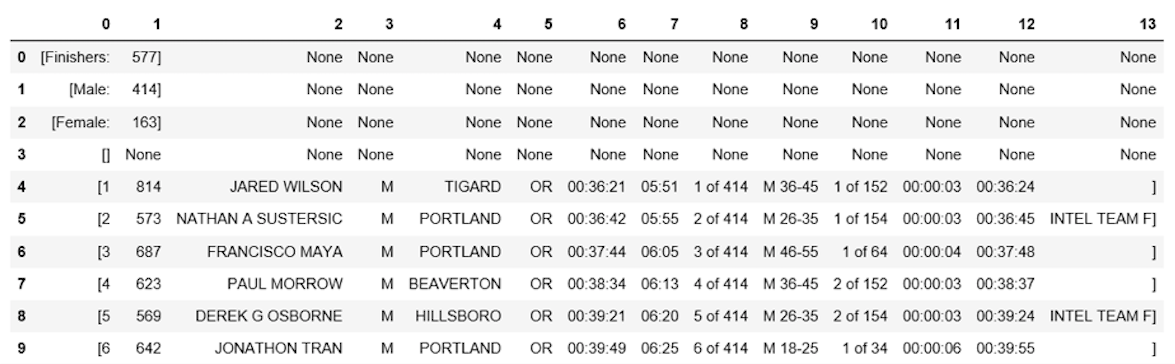
## Data Transformation

Our topic this week is “Data Gathering”, and in a sense, you are done with the gathering. But to make things interesting you should continue on and use the data that you gathered. Next we need to transform the data into something more meaningful.

The dataframe is not in the format we want. To clean it up, split the "0" column into multiple columns at the comma position. Use the str.split() method.

df1 = df[0].str.split(',', expand=True)

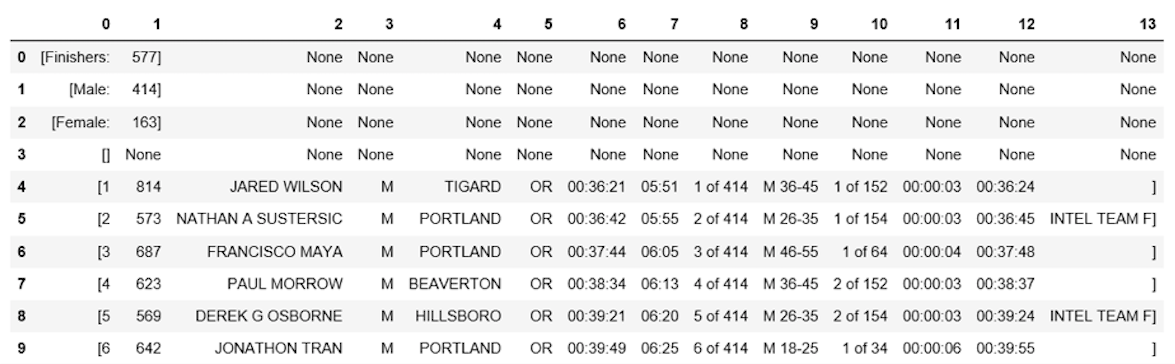
df1.head(10)



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

df1[0] = df1[0].str.strip('[')

df1.head(10)



Note that column 13 also has some ugly ‘]’ characters. We could strip these as well, but for demonstration purposes we will do that later when we show how pandas lets us refer to that column in a symbolic way instead of using the mysterious number 13.

The table is missing table headers. So go back to BeautifulSoup and use its find\_all() method to get the table headers.

col\_labels = soup.find\_all('th')

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

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, Age Group, Age Group Place, Time to Start, Gun Time, Team]']

Next convert the table headers to a new pandas dataframe.

df2 = pd.DataFrame(all\_header)

df2.head()

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

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

df3 = df2[0].str.split(',', expand=True)

df3.head()

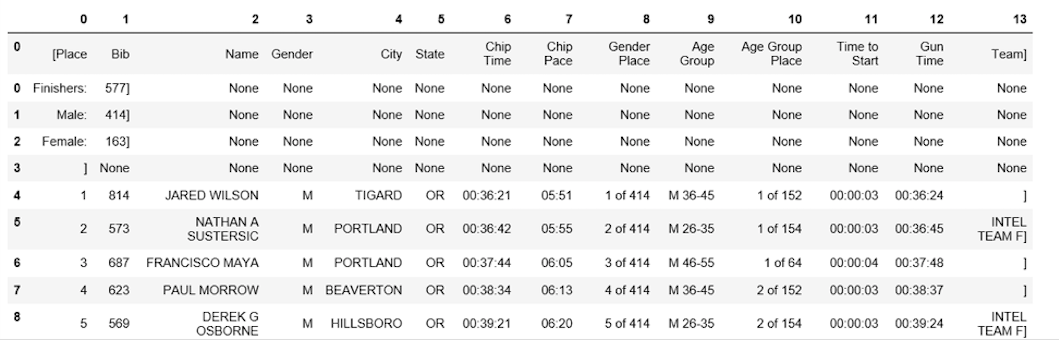


Next, concatenate the two dataframes into one using the concat() method.

frames = [df3, df1]

df4 = pd.concat(frames)

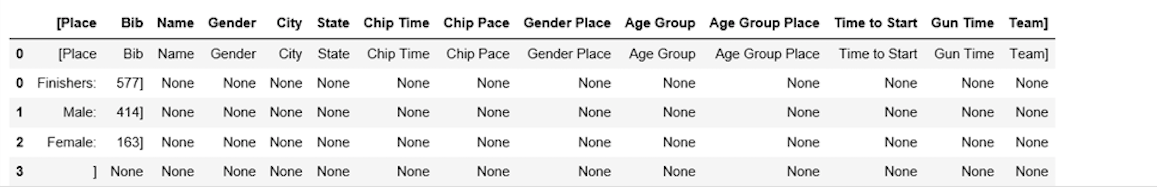
df4.head(10)



Next, re-configure the data frame so that the first row is the table header.

df5 = df4.rename(columns=df4.iloc[0])

df5.head()



Behold all of the progress that you have made! At this point, the table is almost properly formatted. For analysis, start by getting an overview of the data as shown below.

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

(597, 14)

The table has 597 rows and 14 columns. One statement that you will hear me say over and over throughout the course is, “if you did not validate the data then it’s probably wrong”, and this data is no exception. How do we know? Because it has 597 rows, but the various columns have varying numbers of “non-null object” values. So transform it again to drop all rows with any missing values.

df6 = df5.dropna(axis=0, how='any')

df6.info()

df6.shape

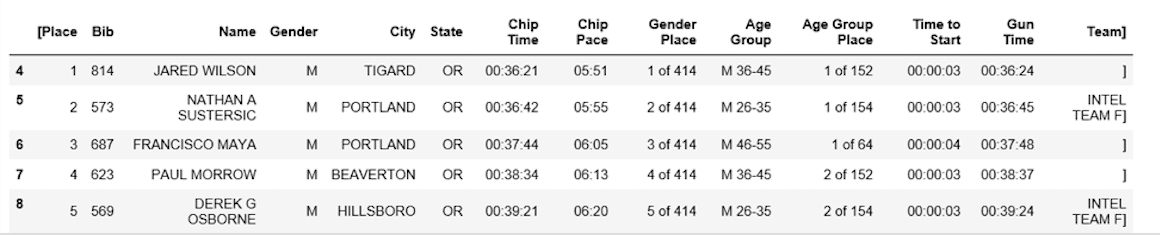
Wasn’t that easy? When I say “pandas is useful”, this is one example of what I mean. You can do the same type of thing within a relational database, but by keeping all of the data in memory within our python program we can accomplish this type of transformation task much more quickly and cleanly.

In a real data engineering project probably we would validate the data thoroughly by devising predicates for every aspect of the data. But to keep the focus on data gathering we will instead move forward with the data we have.

Notice how the table header is replicated as the first row in df5 and df6. Drop this redundant row like this:

df7 = df6.drop(df6.index[0])

df7.head()



Clean up the headers a bit more by renaming the '[Place' and ' Team]' columns. Python is picky about whitespace. Make sure you include a space after the quotation mark in ' Team]'.

df7.rename(columns={'[Place': 'Place'},inplace=True)

df7.rename(columns={' Team]': 'Team'},inplace=True)

df7.head()



The final data cleaning step involves removing the closing bracket for cells in the "Team" column.

df7['Team'] = df7['Team'].str.strip(']')

df7.head()



The dataframe is now in the desired format. One thing you could do at this point is to further transform the data by eliminating redundancies and unneeded information. For example, some of the columns appear to be redundant and potentially could be dropped.

Next move on to the Data Science part, including computation of summary statistics and plotting of results.

## Data Analysis and Visualization

The first question to answer is, ***what was the average finish time (in minutes) for the runners?*** Unfortunately, the “Chip Time” field gives the finish time in hh:mm:ss , so you need to transform that column into just minutes. One way to do this is to convert the column to a list first for manipulation.

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)

Next, convert the list back into a dataframe and create a new column ("Runner\_mins") for runner chip times expressed in just minutes.

df7['Runner\_mins'] = time\_mins

df7.head()



Finally, time for analysis! Pandas provides a handy “describe” method that computes a generous list of explanatory statistics for the dataframe.

df7.describe(include=[np.number])

|  |  |
| --- | --- |
|  | **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.

Boxplots can help to visualize summary statistics (maximum, minimum, medium, first quartile, third quartile, including outliers) for a given column. Below are data summary statistics for the runners shown in a boxplot. For data visualization, it is convenient to first import parameters from the pylab module that comes with matplotlib and set the same size for all figures to avoid doing it for each figure.

from pylab import rcParams

rcParams['figure.figsize'] = 15, 5

df7.boxplot(column='Runner\_mins')

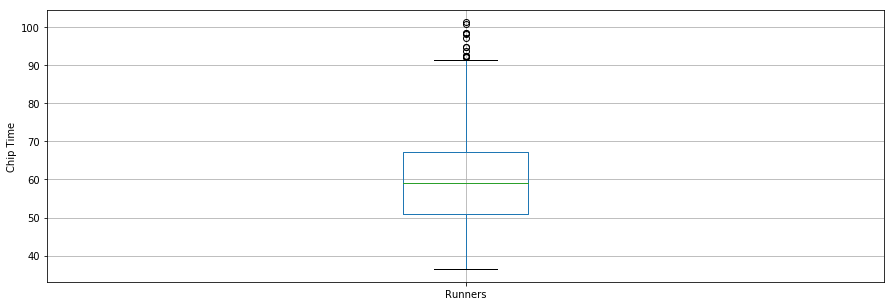
plt.grid(True, axis='y')

plt.ylabel('Chip Time')

plt.xticks([1], ['Runners'])

([<matplotlib.axis.XTick at 0x570dd106d8>],

<a list of 1 Text xticklabel objects>)



The boxplot visualization clearly shows that most runners finished between low 50s and upper 60s (minutes) with a few outliers far up above 90 minutes.

Our second question: ***Did the runners' finish times follow a normal distribution?***

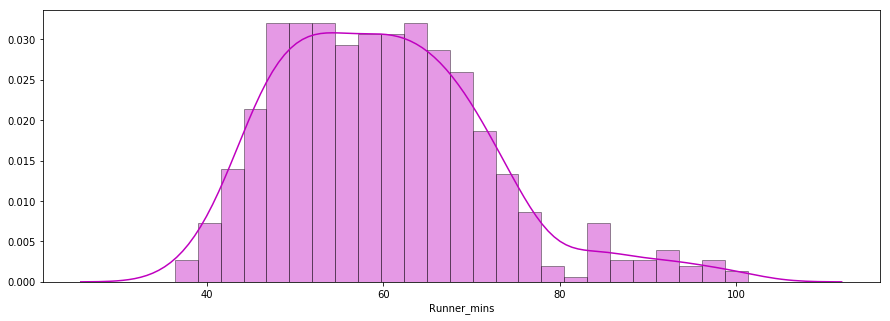
To study the shape of the distribution, plot the runners’ chip times using the seaborn python plotting library. Seaborn provides a wealth of useful, compelling, visualization tools. The distribution looks almost normal. “Looks normal” of course is not sufficient, and at this point a good statistician would use various tests and methods to properly classify this distribution.

One thing to note here is the large number of outliers on the high end of the race times. If we ignored those, then the normal distribution might be an even better model of the data. But if we include those outliers, then we might favor a more sophisticated distribution to model the data properly.

x = df7['Runner\_mins']

ax = sns.distplot(x, hist=True, kde=True, rug=False, color='m', bins=25, hist\_kws={'edgecolor':'black'})

plt.show()



Question 3: Were there any performance differences between males and females of various age groups? To explore this question plot separate distributions of chip times for males and females.

f\_fuko = df7.loc[df7[' Gender']==' F']['Runner\_mins']

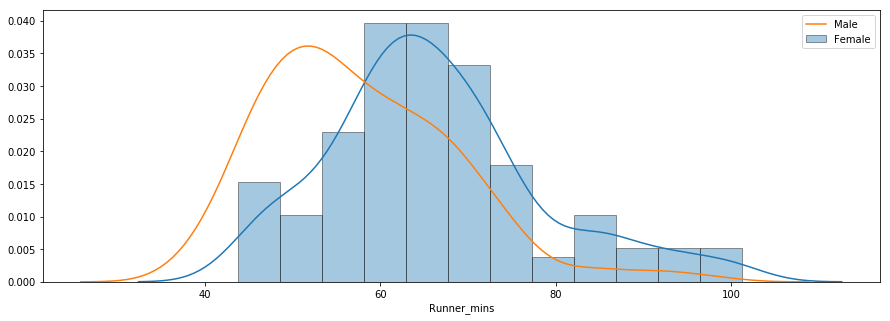
m\_fuko = df7.loc[df7[' Gender']==' M']['Runner\_mins']

sns.distplot(f\_fuko, hist=True, kde=True, rug=False, hist\_kws={'edgecolor':'black'}, label='Female')

sns.distplot(m\_fuko, hist=False, kde=True, rug=False, hist\_kws={'edgecolor':'black'}, label='Male')

plt.legend()

<matplotlib.legend.Legend at 0x570e301fd0>



The distribution indicates that females were slower than males on average. If you prefer summary, descriptive statistics to the visual plot, then use the pandas groupby() method to compute summary statistics for males and females separately.

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

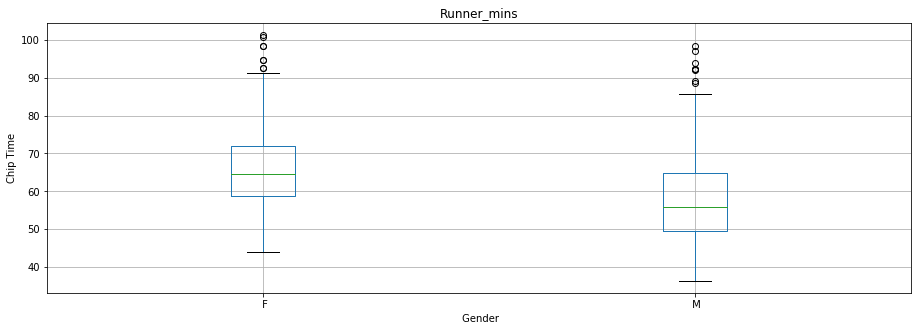
The average chip time for all females and males was ~66 mins and ~58 mins, respectively. Next, display a side-by-side boxplot comparison of male and female finish times.

df7.boxplot(column='Runner\_mins', by=' Gender')

plt.ylabel('Chip Time')

plt.suptitle("")

Text(0.5,0.98,'')



# Looks like both males and females had significant numbers of outliers in the > 90m range.

## Conclusion

In this lab activity, you explored data gathering, transformation, validation and analysis using Python, BeautifulSoup, pandas, matplotlib, Seaborn, etc. You should now have a bit more understanding and appreciation of the various types of operations common to data engineering. Plus a little bit of Data Science at the end to show the value of all of your DateEng efforts. Well done!

BeautifulSoup and Pandas are two examples of software libraries that enable clean, efficient, understandable code that saves time and effort when solving real world data engineering problems.

Submit: fill the in-class activity submission form (find this on the Materials page on the website) at the end of the week (by 10pm on Friday January 15). It is not necessary to reach the end of the assignment, just get as far as you can and indicate your status when submitting. Keep your code in a repository (github) visible to the instructors; the form will ask you for a reference to your repository.