Chapter 1)

1)
Importing flat files from the web: your turn!

You are about to import your first file from the web! The flat file you will import will be `'winequality-red.csv'` from the University of California, Irvine's **Machine Learning repository**. The flat file contains tabular data of physiochemical properties of red wine, such as pH, alcohol content and citric acid content, along with wine quality rating.
The URL of the file is
`'https://s3.amazonaws.com/assets.datacamp.com/production/course_1606/datasets/winequality-red.csv'`
After you import it, you'll check your working directory to confirm that it is there and then you'll load it into a `pandas` DataFrame.

- Import the function `urlretrieve` from the subpackage `urllib.request`.
- Assign the URL of the file to the variable `url`.
- Use the function `urlretrieve()` to save the file locally as `'winequality-red.csv'`.
- Execute the remaining code to load `'winequality-red.csv'` in a pandas DataFrame and to print its head to the shell.

```
# Import package
from urllib.request import urlretrieve

# Import pandas
import pandas as pd

# Assign url of file: url
url =
'https://s3.amazonaws.com/assets.datacamp.com/production/course_1606/datasets/winequality-red.csv'


# Save file locally
urlretrieve(url,'winequality-red.csv')

# Read file into a DataFrame and print its head
df = pd.read_csv('winequality-red.csv', sep=';')
print(df.head())
```

2)
Opening and reading flat files from the web

You have just imported a file from the web, saved it locally and loaded it into a DataFrame. If you just wanted to load a file from the web into a DataFrame without first saving it locally, you can do that easily using `pandas`. In particular, you can use the function `pd.read_csv()` with the URL as the first argument and the separator `sep` as the second argument.
The URL of the file, once again, is

- Assign the URL of the file to the variable `url`.
- Read file into a DataFrame `df` using `pd.read_csv()`, recalling that the separator in the file is `';'`.
- Print the head of the DataFrame `df`.
- Execute the rest of the code to plot histogram of the first feature in the DataFrame `df`.

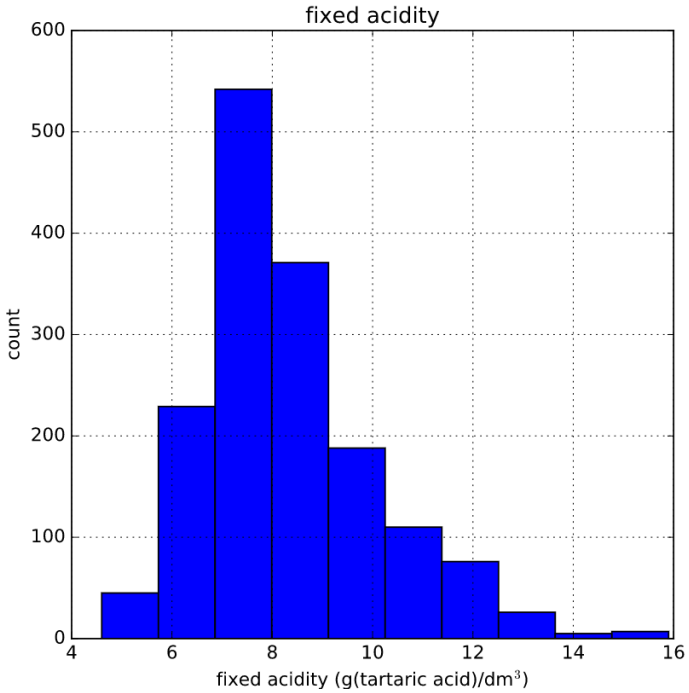
```
# Import packages
import matplotlib.pyplot as plt
import pandas as pd

# Assign url of file: url
url =
'https://s3.amazonaws.com/assets.datacamp.com/production/course_1606/datasets/winequality-red.csv'


# Read file into a DataFrame: df
df = pd.read_csv(url,sep=';')

# Print the head of the DataFrame
print(df.head())

# Plot first column of df
pd.DataFrame.hist(df.ix[:, 0:1])
plt.xlabel('fixed acidity (g(tartaric acid)/dm$^3$)')
plt.ylabel('count')
plt.show()
```



3)

Importing non-flat files from the web

Congrats! You've just loaded a flat file from the web into a DataFrame without first saving it locally using the `pandas` function `pd.read_csv()`. This function is super cool because it has close relatives that allow you to load all types of files, not only flat ones. In this interactive exercise, you'll use `pd.read_excel()` to import an Excel spreadsheet.

The URL of the spreadsheet is

```
'http://s3.amazonaws.com/assets.datacamp.com/course/importing_data_into_r/latitude.xls'
```

Your job is to use `pd.read_excel()` to read in all of its sheets, print the sheet names and then print the head of the first sheet *using its name, not its index*.

Note that the output of `pd.read_excel()` is a Python dictionary with sheet names as keys and corresponding DataFrames as corresponding values.

- Assign the URL of the file to the variable `url`.
- Read the file in `url` into a dictionary `xl` using `pd.read_excel()` recalling that, in order to import all sheets you need to pass `None` to the argument `sheetname`.
- Print the names of the sheets in the Excel spreadsheet; these will be the keys of the dictionary `xl`.
- Print the head of the first sheet *using the sheet name, not the index of the sheet*! The sheet name is `'1700'`

# Import package
import pandas as pd

Assign url of file: url

```
url =
'http://s3.amazonaws.com/assets.datacamp.com/course/importing_data_into_r/latitude.xls'


# Read in all sheets of Excel file: xl
xl = pd.read_excel(url,sheetname=None)

# Print the sheetnames to the shell
print (xl.keys())

# Print the head of the first sheet (using its name, NOT its index)
print (xl['1700'].head())
```

```
dict_keys(['1700', '1900'])
                country       1700
0           Afghanistan  34.565000
1  Akrotiri and Dhekelia  34.616667
2               Albania  41.312000
3               Algeria  36.720000
4        American Samoa -14.307000
```

Note)

GET requests using requests

- One of the most downloaded Python packages

```
In [1]: import requests

In [2]: url = "https://www.wikipedia.org/"

In [3]: r = requests.get(url)

In [4]: text = r.text
```

4)

Performing HTTP requests in Python using urllib

Now that you know the basics behind HTTP GET requests, it's time to perform some of your own. In this interactive exercise, you will ping our very own DataCamp servers to perform a GET request to extract information from our teach page, `"http://www.datacamp.com/teach/documentation"`.
In the next exercise, you'll extract the HTML itself. Right now, however, you are going to package and send the request and then catch the response.

- Import the functions `urlopen` and `Request`from the subpackage `urllib.request`.
- Package the request to the url `"http://www.datacamp.com/teach/documentation"`using the function `Request()` and assign it to `request`.
- Send the request and catch the response in the variable `response` with the function `urlopen()`.
- Run the rest of the code to see the datatype of `response` and to close the connection!

```
# Import packages
from urllib.request import urlopen,Request

# Specify the url
url = "http://www.datacamp.com/teach/documentation"

# This packages the request: request
request = Request(url)

# Sends the request and catches the response: response
response = urlopen(request)

# Print the datatype of response
```

```
print(type(response))
```
```
<class 'http.client.HTTPResponse'>
```
```
# Be polite and close the response!
response.close()
```

5)

Printing HTTP request results in Python using urllib

You have just packaged and sent a GET request to `"http://www.datacamp.com/teach/documentation"` and then caught the response. You saw that such a response is a `http.client.HTTPResponse` object. The question remains: what can you do with this response?
Well, as it came from an HTML page, you could *read* it to extract the HTML and, in fact, such a `http.client.HTTPResponse` object has an associated `read()` method. In this exercise, you'll build on your previous great work to extract the response and print the HTML.

- Send the request and catch the response in the variable `response` with the function `urlopen()`, as in the previous exercise.
- Extract the response using the `read()` method and store the result in the variable `html`.

```
# Import packages
from urllib.request import urlopen, Request

# Specify the url
url = "http://www.datacamp.com/teach/documentation"

# This packages the request
request = Request(url)

# Sends the request and catches the response: response
response = urlopen(request)

# Extract the response: html
html = response.read()

# Print the html
print (html)

# Be polite and close the response!
response.close()
```

b'\n<!DOCTYPE HTML>\n<html lang="" >\n <head>\n   <meta charset="UTF-8">\n <meta content="text/html; charset=utf-8" http-equiv="Content-Type">\n   <title>Welcome \xc2\xb7 Authoring Content for DataCamp</title>\n   <meta http-equiv="X-UA-Compatible" content="IE=edge" />\n   <meta name="description" content="">\n   <meta name="generator" content="GitBook 3.2.3">\n

6)
Performing HTTP requests in Python using requests

Now that you've got your head and hands around making HTTP requests using the urllib package, you're going to figure out how to do the same using the higher-level requests library. You'll once again be pinging DataCamp servers for their `"http://www.datacamp.com/teach/documentation"` page.
Note that unlike in the previous exercises using urllib, you don't have to close the connection when using requests!

- Import the package `requests`.
- Assign the URL of interest to the variable `url`.
- Package the request to the URL, send the request and catch the response with a single function `requests.get()`, assigning the response to the variable `r`.
- Use the `text` attribute of the object `r` to return the HTML of the webpage as a string; store the result in a variable `text`.

```
# Import package
import requests

# Specify the url: url
url = 'http://www.datacamp.com/teach/documentation'

# Packages the request, send the request and catch the response: r
r = requests.get(url)

# Extract the response: text
text = r.text

# Print the html
print(text)
```


<meta name="HandheldFriendly" content="true"/>
<meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=no">

Note)

BeautifulSoup

```
In [1]: from bs4 import BeautifulSoup

In [2]: import requests

In [3]: url = 'https://www.crummy.com/software/BeautifulSoup/'

In [4]: r = requests.get(url)

In [5]: html_doc = r.text

In [6]: soup = BeautifulSoup(html_doc)
```

Prettified Soup

```
In [7]: print(soup.prettify())
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitio
<html>
 <head>
  <meta content="text/html; charset=utf-8" http-equiv
  <title>
   Beautiful Soup: We called him Tortoise because he
  </title>
  <link href="mailto:leonardr@segfault.org" rev="made
  <link href="/nb/themes/Default/nb.css" rel="stylesh
  <meta content="Beautiful Soup: a library designed f
  <meta content="Markov Approximation 1.4 (module: le
  <meta content="Leonard Richardson" name="author"/>
 </head>
```

Exploring BeautifulSoup

- Many methods such as:

```
In [9]: print(soup.title)
<title>Beautiful Soup: We called him Tortoise because he taught
us.</title>
```

```
In [8]: print(soup.get_text())

Beautiful Soup: We called him Tortoise because he taught us.

You didn't write that awful page. You're just trying to get some
data out of it. Beautiful Soup is here to help. Since 2004, it's
been saving programmers hours or days of work on quick-turnaroun
screen scraping projects.
```

7)
Parsing HTML with BeautifulSoup

In this interactive exercise, you'll learn how to use the BeautifulSoup package to *parse*, *prettify* and *extract* information from HTML. You'll scrape the data from the webpage of Guido van Rossum, Python's very own **Benevolent Dictator for Life**. In the following exercises, you'll prettify the HTML and then extract the text and the hyperlinks.
The URL of interest is `url = 'https://www.python.org/~guido/'`.

- Import the function `BeautifulSoup` from the package `bs4`.
- Assign the URL of interest to the variable `url`.
- Package the request to the URL, send the request and catch the response with a single function `requests.get()`, assigning the response to the variable `r`.
- Use the `text` attribute of the object `r` to return the HTML of the webpage as a string; store the result in a variable `html_doc`.
- Create a BeautifulSoup object `soup` from the resulting HTML using the function `BeautifulSoup()`.
- Use the method `prettify()` on `soup` and assign the result to `pretty_soup`.

```
# Import packages
import requests
from bs4 import BeautifulSoup

# Specify url: url
url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r
r = requests.get(url)

# Extracts the response as html: html_doc
html_doc = r.text

# Create a BeautifulSoup object from the HTML: soup
soup = BeautifulSoup(html_doc)

# Prettify the BeautifulSoup object: pretty_soup
pretty_soup = soup.prettify()

# Print the response
print(pretty_soup)
```

<script.py> output:

```
   <html>
    <head>
     <title>
      Guido's Personal Home Page
     </title>
    </head>
    <body bgcolor="#FFFFFF" text="#000000">
     <h1>
      <a href="pics.html">
       <img border="0" src="images/IMG_2192.jpg"/>
      </a>
```

8)

Turning a webpage into data using BeautifulSoup: getting the text

As promised, in the following exercises, you'll learn the basics of extracting information from HTML soup. In this exercise, you'll figure out how to extract the text from the BDFL's webpage, along with printing the webpage's title.

- In the sample code, the HTML response object `html_doc` has already been created: your first task is to Soupify it using the function `BeatifulSoup()` and to assign the resulting soup to the variable `soup`.
- Extract the title from the HTML soup `soup` using the attribute `title` and assign the result to `guido_title`.
- Print the title of Guido's webpage to the shell using the `print()` function.
- Extract the text from the HTML soup `soup` using the method `get_text()`and assign to `guido_text`.

```
# Import packages
import requests
from bs4 import BeautifulSoup

# Specify url: url
url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r
r = requests.get(url)

# Extract the response as html: html_doc
html_doc = r.text

# Create a BeautifulSoup object from the HTML: soup
```

```
soup = BeautifulSoup(html_doc)
# Get the title of Guido's webpage: guido_title
guido_title = soup.title

# Print the title of Guido's webpage to the shell
print (guido_title)
```

```
<title>Guido's Personal Home Page</title>
```

```
# Get Guido's text: guido_text
guido_text = soup.get_text()

# Print Guido's text to the shell
print(guido_text)
```

```
Guido's Personal Home Page


    Guido van Rossum - Personal Home Page
    "Gawky and proud of it."
    Who
    I Am
```

9)

Turning a webpage into data using BeautifulSoup: getting the hyperlinks

In this exercise, you'll figure out how to extract the URLs of the hyperlinks from the BDFL's webpage. In the process, you'll become close friends with the soup method `find_all()`.

- Use the method `find_all()` to find all hyperlinks in `soup`, remembering that hyperlinks are defined by the HTML tag `<a>`; store the result in the variable `a_tags`.
- The variable `a_tags` is a results set: your job now is to enumerate over it, using a `for` loop and to print the actual URLs of the hyperlinks; to do this, for every element `link` in `a_tags`, you want to `print()` `link.get('href')`

```
# Import packages
import requests
from bs4 import BeautifulSoup

# Specify url
url = 'https://www.python.org/~guido/'

# Package the request, send the request and catch the response: r
```

```
r = requests.get(url)

# Extracts the response as html: html_doc
html_doc = r.text

# create a BeautifulSoup object from the HTML: soup
soup = BeautifulSoup(html_doc)

# Print the title of Guido's webpage
print(soup.title)

# Find all 'a' tags (which define hyperlinks): a_tags
a_tags = soup.find_all('a')

# Print the URLs to the shell
for link in a_tags:
    print (link.get('href'))
```

pics.html
http://www.washingtonpost.com/wp-srv/business/longterm/microsoft/stories/1998/raymond120398.htm
http://metalab.unc.edu/Dave/Dr-Fun/df200004/df20000406.jpg
http://neopythonic.blogspot.com/2016/04/kings-day-speech.html
http://www.python.org
Resume.html
Publications.html

Chapter 2)

Note)

Loading JSONs in Python

```
In [1]: import json

In [2]: with open('snakes.json', 'r') as json_file:
   ...:     json_data = json.load(json_file)

In [3]: type(json_data)
Out[3]: dict
```

1)
Loading and exploring a JSON

Now that you know what a JSON is, you'll load one into your Python environment and explore it yourself. Here, you'll load the JSON `'a_movie.json'` into the variable `json_data`, which will be a dictionary. You'll then explore the JSON contents by printing the key-value pairs of `json_data` to the shell.

- Load the JSON `'a_movie.json'` into the variable `json_data` *within the context* provided by the `with` statement. To do so, use the function `json.load()` *within the context manager*.
- Use a `for` loop to print all key-value pairs in the dictionary `json_data`. Recall that you can access a value in a dictionary using the syntax: *dictionary*`[`key`]`.

```
# Load JSON: json_data
with open("a_movie.json") as json_file:
    json_data = json.load(json_file)

# Print each key-value pair in json_data
for k in json_data.keys():
    print(k + ': ', json_data[k])
```

```
DVD:  11 Jan 2011
Actors:  Jesse Eisenberg, Rooney Mara, Bryan Barter, Dustin Fitzsimons
Website:  http://www.thesocialnetwork-movie.com/
Writer:  Aaron Sorkin (screenplay), Ben Mezrich (book)
Director:  David Fincher
imdbRating:  7.7
Year:  2010
```

2)
API requests

Now it's your turn to pull some movie data down from the Open Movie Database (OMDB) using their API. The movie you'll query the API about is *The Social Network*. Recall that, in the video, to query the API about the movie *Hackers*, Hugo's *query string* was `'http://www.omdbapi.com/?t=hackers'` and had a single argument `t=hackers`.
Note: recently, OMDB has changed their API: you now also have to specify an API key. This means you'll have to add another argument to the URL: `apikey=ff21610b`.

```
# Import requests package
import requests

# Assign URL to variable: url
```

```
url = 'http://www.omdbapi.com/?apikey=ff21610b&t=social+network'

# Package the request, send the request and catch the response: r
r = requests.get(url)

# Print the text of the response
print (r.text)
```

{"Title":"The Social Network","Year":"2010","Rated":"PG-13","Released":"01 Oct 2010","Runtime":"120 min","Genre":"Biography, Drama","Director":"David Fincher","Writer":"Aaron Sorkin (screenplay), Ben Mezrich (book)","Actors":"Jesse Eisenberg, Rooney Mara, Bryan Barter, Dustin Fitzsimons","Plot":"Harvard student Mark Zuckerberg creates the social networking site that would become known as Facebook, but is later sued by two brothers who claimed he stole their idea, and the co-founder who was later squeezed out of the business.","Language":"English, French","Country":"USA","Awards":"Won 3 Oscars. Another 165 wins & 168 nominations.","Poster":"https://images-na.ssl-images-amazon.com/images/M/MV5BMTM2ODk0NDAwMF5BMl5BanBnXkFtZTcwNTM1MDc2Mw@@._V1_SX300.jpg","Ratings":[{"Source":"Internet Movie Database","Value":"7.7/10"},{"Source":"Rotten Tomatoes","Value":"96%"},{"Source":"Metacritic","Value":"95/100"}],"Metascore":"95","imdbRating":"7.7","imdbVotes":"528,379","imdbID":"tt1285016","Type":"movie","DVD":"11 Jan 2011","BoxOffice":"$96,400,000","Production":"Columbia Pictures","Website":"http://www.thesocialnetwork-movie.com/","Response":"True"}

3)
JSON–from the web to Python

Wow, congrats! You've just queried your first API programmatically in Python and printed the text of the response to the shell. However, as you know, your response is actually a JSON, so you can do one step better and decode the JSON. You can then print the key-value pairs of the resulting dictionary. That's what you're going to do now!

- Pass the variable `url` to the `requests.get()` function in order to send the relevant request and catch the response, assigning the resultant response message to the variable `r`.
- Apply the `json()` method to the response object `r` and store the resulting dictionary in the variable `json_data`.

```
# Import package
import requests

# Assign URL to variable: url
url = 'http://www.omdbapi.com/?apikey=ff21610b&t=social+network'

# Package the request, send the request and catch the response: r
r = requests.get(url)
```

```
# Decode the JSON data into a dictionary: json_data
json_data = r.json()

# Print each key-value pair in json_data
for k in json_data.keys():
    print(k + ': ', json_data[k])
```

```
Response:  True
Title:  The Social Network
Rated:  PG-13
imdbID:  tt1285016
Poster:  https://images-na.ssl-images-amazon.com/images/M/MV5BMTM2ODk0NDAwMF5BMl5BanBnXkFtZTcwNTM1MDc2Mw@@._V1_SX300.jpg
Year:  2010
Metascore:  95
```

4)

Checking out the Wikipedia API

You're doing so well and having so much fun that we're going to throw one more API at you: the Wikipedia API (documented **here**). You'll figure out how to find and extract information from the Wikipedia page for *Pizza*. What gets a bit wild here is that your query will return *nested* JSONs, that is, JSONs with JSONs, but Python can handle that because it will translate them into dictionaries within dictionaries. The URL that requests the relevant query from the Wikipedia API is

```
https://en.wikipedia.org/w/api.php?action=query&prop=extracts&format=json&exintro=&titles=pizza
```

- Assign the relevant URL to the variable `url`.
- Apply the `json()` method to the response object `r` and store the resulting dictionary in the variable `json_data`.
- The variable `pizza_extract` holds the HTML of an extract from Wikipedia's *Pizza* page as a string;

```
# Import package
import requests

# Assign URL to variable: url
url =
'https://en.wikipedia.org/w/api.php?action=query&prop=extracts&format=json
&exintro=&titles=pizza'

# Package the request, send the request and catch the response: r
```

```
r = requests.get(url)

# Decode the JSON data into a dictionary: json_data
json_data = r.json()

# Print the Wikipedia page extract
pizza_extract = json_data['query']['pages']['24768']['extract']
print (pizza_extract)
```

Pizza is a traditional Italian dish consisting of a yeasted flatbread typically topped with tomato sauce and cheese and baked in an oven. It can also be topped with additional vegetables, meats, and condiments.

The term pizza was first recorded in the 10th century, in a Latin manuscript from the Southern Italy town of Gaeta in Lazio, on the border with Campania. Modern pizza was invented in Naples, and the dish and its variants have since become popular and common in many areas of the world. In 2009, upon Italy's request, Neapolitan pizza was registered with the European Union as a Traditional Speciality Guaranteed dish. Associazione Verace Pizza Napoletana (True Neapolitan Pizza Association), a non-profit organization founded in 1984 with headquarters in Naples, aims to "promote and protect... the true Neapolitan pizza".

Chapter 3)

Note)

Using Tweepy: Authentication handler

tw_auth.py

```
import tweepy, json

access_token = "..."
access_token_secret = "..."
consumer_key = "..."
consumer_secret = "..."

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
```

Tweepy: define stream listener class

st_class.py

```
class MyStreamListener(tweepy.StreamListener):
    def __init__(self, api=None):
        super(MyStreamListener, self).__init__()
        self.num_tweets = 0
        self.file = open("tweets.txt", "w")

    def on_status(self, status):
        tweet = status._json
        self.file.write(json.dumps(tweet) + '\n')
        tweet_list.append(status)
        self.num_tweets += 1
        if self.num_tweets < 100:
            return True
        else:
            return False
        self.file.close()
```

Using Tweepy: stream tweets!!

tweets.py

```
# Create Streaming object and authenticate
l = MyStreamListener()
stream = tweepy.Stream(auth, l)

# This line filters Twitter Streams to capture data by keywords:
stream.filter(track=['apples', 'oranges'])
```

1)
API Authentication

The package `tweepy` is great at handling all the Twitter API OAuth Authentication details for you. All you need to do is pass it your authentication credentials. In this interactive exercise, we have created some mock authentication credentials (if you wanted to replicate this at home, you would need to create a **Twitter App** as Hugo detailed in the video). Your task is to pass these credentials to tweepy's OAuth handler.

- Import the package `tweepy`.
- Pass the parameters `consumer_key` and `consumer_secret` to the function `tweepy.OAuthHandler()`.
- Complete the passing of OAuth credentials to the OAuth handler `auth` by applying to it the method `set_access_token()`, along with arguments `access_token` and `access_token_secret`

# Import package
import tweepy

Store OAuth authentication credentials in relevant variables

```
access_token = "1092294848-
aHN7DcRP9B4VMTQIhwqOYiB14YkW92fFO8k8EPy"
access_token_secret =
"X4dHmhPfaksHcQ7SCbmZa2oYBBVSD2g8uIHXsp5CTaksx"
consumer_key = "nZ6EA0FxZ293SxGNg8g8aP0HM"
consumer_secret =
"fJGEodwe3KiKUnsYJC3VRndj7jevVvXbK2D5EiJ2nehafRgA6i"

# Pass OAuth details to tweepy's OAuth handler
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token,access_token_secret)
```

2)

Streaming tweets

Now that you have set up your authentication credentials, it is time to stream some tweets! We have already defined the tweet stream listener class, `MyStreamListener`, just as Hugo did in the introductory video. You can find the code for the tweet stream listener class **here**.
Your task is to create the `Stream`object and to filter tweets according to particular keywords.

- Create your `Stream` object with authentication by passing `tweepy.Stream()` the authentication handler `auth` and the Stream listener `l`;
- To filter Twitter streams, pass to the `track` argument in `stream.filter()` a list containing the desired keywords `'clinton'`, `'trump'`, `'sanders'`, and `'cruz'`

```
# Initialize Stream listener
l = MyStreamListener()

# Create you Stream object with authentication
stream = tweepy.Stream(auth, l)


# Filter Twitter Streams to capture data by the keywords:
stream.filter(track=['clinton','trump','sanders','cruz'])
```

3)
Load and explore your Twitter data

Now that you've got your Twitter data sitting locally in a text file, it's time to explore it! This is what you'll do in the next few interactive exercises. In this exercise, you'll read the Twitter data into a list: `tweets_data`.

- Assign the filename `'tweets.txt'` to the variable `tweets_data_path`.
- Initialize `tweets_data` as an empty list to store the tweets in.
- Within the `for` loop initiated by `for line in tweets_file:`, load each tweet into a variable, `tweet`, using `json.loads()`, then append `tweet` to `tweets_data` using the `append()` method.

```
# Import package
import json

# String of path to file: tweets_data_path
tweets_data_path = 'tweets.txt'

# Initialize empty list to store tweets: tweets_data
tweets_data = []

# Open connection to file
tweets_file = open(tweets_data_path, "r")

# Read in tweets and store in list: tweets_data
for line in tweets_file:
    tweet = json.loads(line)
    tweets_data.append(tweet)

# Close connection to file
tweets_file.close()

# Print the keys of the first tweet dict
print(tweets_data[0].keys())
```

dict_keys(['coordinates', 'contributors', 'in_reply_to_status_id_str', 'truncated', 'retweet_count', 'lang', 'in_reply_to_status_id', 'place', 'extended_entities', 'in_reply_to_screen_name', 'favorited', 'retweeted_status', 'retweeted', 'source', 'favorite_count', 'timestamp_ms', 'is_quote_status', 'entities', 'created_at', 'user', 'in_reply_to_user_id', 'id_str', 'text', 'filter_level', 'id', 'geo', 'in_reply_to_user_id_str', 'possibly_sensitive'])

4)
Twitter data to DataFrame

Now you have the Twitter data in a list of dictionaries, `tweets_data`, where each dictionary corresponds to a single tweet. Next, you're going to extract the text and language of each tweet. The text in a tweet, `t1`, is stored as the value `t1['text']`; similarly, the language is stored in `t1['lang']`. Your task is to build a DataFrame in which each row is a tweet and the columns are `'text'` and `'lang'`.

- Use `pd.DataFrame()` to construct a DataFrame of tweet texts and languages; to do so, the first argument should be `tweets_data`, a list of dictionaries. The second argument to `pd.DataFrame()` is a *list* of the keys you wish to have as columns. Assign the result of the `pd.DataFrame()` call to `df`.

```
# Import package
import pandas as pd

# Build DataFrame of tweet texts and languages
df = pd.DataFrame(tweets_data, columns=['text','lang'])

# Print head of DataFrame
print (df.head())
```

```
                                                text lang
0  b"RT @bpolitics: .@krollbondrating's Christoph...   en
1  b'RT @HeidiAlpine: @dmartosko Cruz video found...   en
2  b'Njihuni me Zonj\\xebn Trump !!! | Ekskluzive...   et
3  b"Your an idiot she shouldn't have tried to gr...   en
4  b'RT @AlanLohner: The anti-American D.C. elite...   en
```

5)
A little bit of Twitter text analysis

Now that you have your DataFrame of tweets set up, you're going to do a bit of text analysis to count how many tweets contain the words `'clinton'`, `'trump'`, `'sanders'` and `'cruz'`. In the pre-exercise code, we have defined the following function `word_in_text()`, which will tell you whether the first argument (a word) occurs within the 2nd argument (a tweet).

```
import re
```

```
def word_in_text(word, tweet):
    word = word.lower()
    text = tweet.lower()
    match = re.search(word, tweet)

    if match:
        return True
    return False
```

You're going to iterate over the rows of the DataFrame and calculate how many tweets contain each of our keywords! The list of objects for each candidate has been initialized to 0.

- Within the `for` loop `for index, row in df.iterrows():`, the code currently increases the value of `clinton` by `1` each time a tweet mentioning 'Clinton' is encountered; complete the code so that the same happens for `trump`, `sanders` and `cruz`.

```
# Initialize list to store tweet counts
[clinton, trump, sanders, cruz] = [0, 0, 0, 0]

# Iterate through df, counting the number of tweets in which
# each candidate is mentioned
for index, row in df.iterrows():
    clinton += word_in_text('clinton', row['text'])
    trump += word_in_text('trump', row['text'])
    sanders += word_in_text('sanders', row['text'])
    cruz += word_in_text('cruz', row['text'])
```

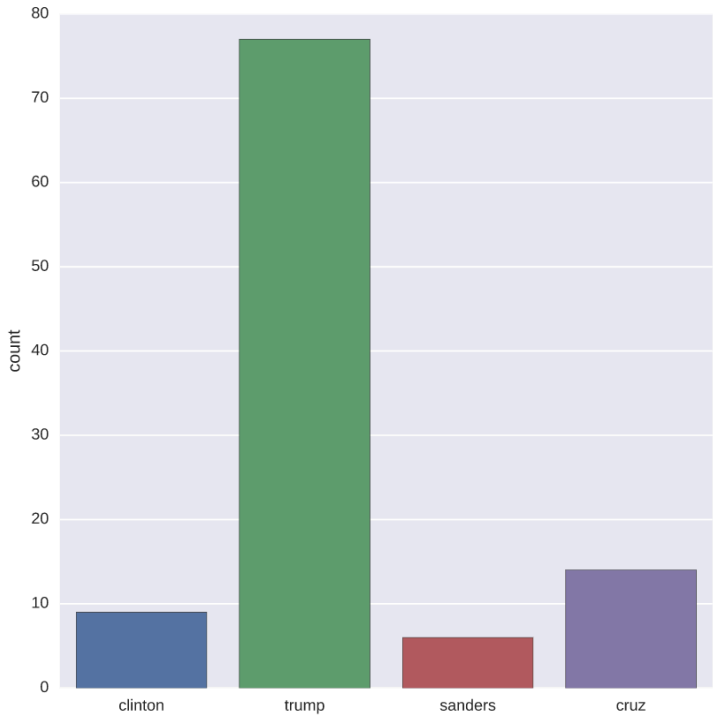
6)

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns

# Set seaborn style
sns.set(color_codes=True)

# Create a list of labels:cd
cd = ['clinton', 'trump', 'sanders', 'cruz']

# Plot histogram
ax = sns.barplot(cd, [clinton,trump,sanders,cruz])
ax.set(ylabel="count")
plt.show()
```


80
70
60
50
40
30
20
10
0
count
clinton
trump
sanders
cruz