LECTURE 3
Introduction to Python for Data Science
LAB

1) Scraping Federal Reserve Statement

Your task is to scrape the Fed Statement (html format) located at:
https://www.federalreserve.gov/newsevents/pressreleases/monetary20200315a.htm

Steps:

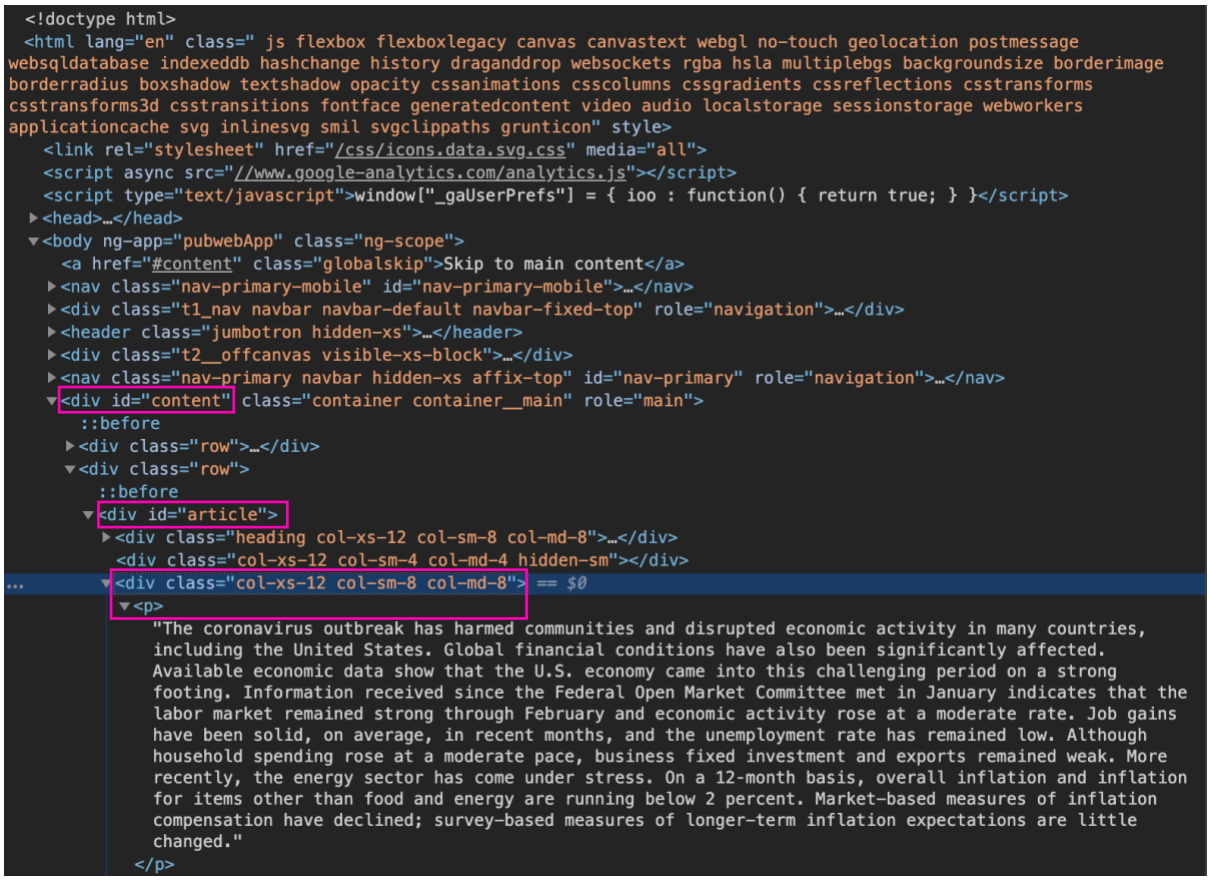
a) Import the necessary libraries in order to open a url with BeautifulSoup
b) Open the above url and read it into an html variable, called `html`
c) Convert `html` variable into BeautifulSoup object

Main task: extract the first 4 paragraphs you see on the Fed Statement webpage. Let's break the task down into steps:

I. Read in the Statement text, i.e. 5 paragraphs which you see on the webpage, and store them into a variable `data`.

Hint1: open web browser and go to the Fed page via the link. Right click anywhere on the page and press 'inspect' / 'inspect element'. You will see the HTML code appear. On the web page, hover over one of the paragraphs and right click selecting inspect again. This will open the paragraph tag you pointed on, similar to what is shown in the screenshot below.

I have highlighted some of the tags which could try to extract. I suggest that you try each tag and see print the result to see what you obtain. This will help you decide on what actually provides you the desired paragraphs text (since there is a lot of other content on the page, other paragraphs, links etc).



Hint2: observe code which pulled out data in `scrapeTest.py` file under section 'Extract text from bs object'.

Hint3: If you would like to double check how the statement html code tends to look upon each release (in order to ensure your code will work across any Fed Statement document), check another link from the meeting previous to the one given above: https://www.federalreserve.gov/newsevents/pressreleases/monetary20200303a.htm

II. Once you obtain `data` variable from step I. print out the following to examine content

```
# data              # should contain all paragraphs (you should see text)
# data[0]           # first paragraph with text
# data[0].text      # first paragraph with text only in string format (note the .text method works only on tags, i.e. a single element of data)
```

Check the data types of each of the 3 objects.

III. Obtain just the 4 first paragraphs as a **single string**. These are the 4 paragraphs that actually matter to the financial markets when examining the Fed Reserve intent with respect to interest rates. So, your last extracted paragraph should start with 'The Federal Reserve is prepared to use its full range of tools to support the flow of credit…'), i.e. everything following this para onwards should be ignored: "Voting for the monetary policy action were…".

Hint: You are free to decide on how to do this. Think of number of elements in your variable `data`. And use part II) to help you think of the solution. You may need a for loop.

2) Dictionaries

You have an example dictionary: `months = {'Jan':1, 'Feb':2, 'Mar':3}`

a) Use a `for` loop to iterate over the dictionary. Your task is to print out keys and their associated values.
Hint: remember you can print several items using a comma i.e. `print(a,b)`

A Machine Learning Class (i.e. object oriented code containing functions and values, which perform a specific task) may use a dictionary to store key:value pairs, where **key** is the algorithm name, e.g. string `KNN` and **value** is a tuple containing information relevant for that algorithm, e.g. for KNN relevant info is: `k` (number of neighbours), `N` (number of data points), `metric` (Euclidian distance).

b) Create an empty dictionary called `algo`

c) Create variables:

```
k = 3
N = 100
metric = 'Euclidian'
```

d) Populate the empty dictionary `algo` with a single `key:value` pair where **key** is `KNN` and **value** is a tuple containing variables `k`, `N`, `metric` as its elements.

e) Obtain number of algos you have in your dictionary. (You should only have a single algo at this stage). Such a dictionary may contain a number of algorithms with their associated values.

Remind yourself about **Tab Completion** in Python. Whenever we have methods which operate on an object, you can take a look at the available methods for that object by typing

out object name followed by a full stop `objName.` and then pressing **Tab**. All of the available methods will be listed.

f) Type out **`months.`** in Console Pane and press Tab. Observe all the methods available for object of type `dict`. So far we have covered 2 of them, which are used the most: `keys()` and `values()`.

Short dictionary-based sentiment analysis exercise. Sentiment analysis is used to compare words in a text against pre-established dictionaries of words containing positive, negative, or neutral sentiment words. In a simple case a simple count of positive vs negative words present in a text can be taken to indicate text polarity.

Analyse the single string obtained for the Federal Reserve Statement of the 15th of March 2020, which you have scraped in Q1. If you did not obtain the string, import string `s` into your script from the module which I saved under the name `L3_Fed_15Mar2020_str.py` by using the familiar import command (i.e. from module name import relevant variable):

```
from module import var
```

Your task is to build a dictionary which counts the occurrence of each word in the Statement and then perform a simple sentiment analysis.

g) The string currently contains unwanted characters inside of it, which we will remove:
   i) Remove all commas, full stops, dashes and slash characters.
   ii) Convert all upper-case letters to lower case.
   iii) Store the result into a list where each element is a word.

   iv) Create an empty dictionary `d`

   v) Use `for` loop to iterate over the list of words and create a dictionary which uses words as keys and number of times each word occurs as values.

   Hint: use string methods `replace()` `lower()` `split()` and keyword `in` to check whether a key is in dictionary.

h) Perform simple sentiment analysis using '*Loughran and McDonald Sentiment Word Lists*' which takes a form of a dictionary with 2 keys: 'Negative' and 'Positive'. Each key's value is a list of words stored as elements of the list, relating to either category.

   i) Import the variable `lmdict` from a module called `lmdict.py` in the same way you imported the string `s` above.

   ii) Use a `for` loop to loop over the keys of the dictionary to count the number of occurrences of positive and negative words, call the variable `score`.
   Hint: check if key from your dictionary is in the `lmdict positive` dictionary. If so add number of times you counted that key to `score`.
   If key from your dictionary is in the `lmdict negative` dictionary, take away the number of times you counted that key from `score`.

   iii) Find the average score and report it to 2 d.p.
   Hint: to answer this you need to keep a count of how many keys matched the `lmdict` and then simply find `score/counter`.

   You should find av sentiment score of latest Fed Statement to be 0.33, which is mildly positive.

3) Creating Numpy Arrays

a) Use the function `array()` (which accepts sequence-like objects including other arrays and produces a new numpy array) to create arrays from:
- List `[1, 2, 3]`
- List `[1.0, 2, 3.]` (remember the trailing zero is not necessary when specifying floats, just the dot is enough).
- List of lists `[[1,2,3], [4,5,6]]`

b) Use the commands `shape` and `dtype` to establish the size of each dimension, and data type of the array of the last 2 arrays you created.

c) Use function `empty()` to create a 1D, 2D and 3D array with the size of your choice. Re-run the code several times to see if the created arrays contain different values than the ones you observed the first time.

Conclusion: it is not safe to assume that `np.empty()` will return an array of all zeros!

d) Now create two 3x4 2D arrays: one filled with 0s and another filled with 1s.

[EXTRA]: (Complete after finishing other main questions first)

e) Introducing `asarray()` method.
- Create two arrays using the proposed methods:
  `a = np.array([1,2,3])` – official documentation here
  `b = np.array(a)`
  `c = np.asarray(a)` – official documentation here
- Check whether you have created an alias between `a-b` and `a-c` (i.e. whether two variables point to the same location in memory).

Explanation: the `np.asarray` method performs the same job as the array method, however does not copy input if it is already an `ndarray` of the same datatype.
- Change the value of array a in position 1 (indexed by 0) to `11`. Check contents of array `a`, `b`, `c`.

4) Arithmetic operations, Mathematical / Statistical calculations:

a) Create a 1-D array of prices for Intel, call it `intc` containing values `55, 42, 61`.
- Establish max value contained in `intc`
- Establish the index at which max value occurs in `intc`

b) Create an array called `a` using a list of lists: `[[1., 2., 3.], [4., 5., 6.]]`. Your task is to perform the following element-wise operations:
- Multiply `a` by `a`
- Take `a` away from `a`
- Find reciprocal of `a`
- Find square root of `a`

Create an array called `b` using a list of lists `[[0.1,2.1,32], [0.9,15,5.6]]`.
- Find a new array which would contain boolean values of element-wise comparison to check where elements of `a` are *greater* than elements of `b`.

c) You have the following 1-D array: `a = np.arange(9)`,

First, re-shape this array into a 2-D 3-by-3 array using `numpy.reshape()` method (if unsure read official documentation here; it is a good idea to start to get familiar with how official documentation looks).

Your task is to find the `sum`, `mean`, `std` (with and without `ddof=1` optional argument), and `cumsum` for:

- Whole data set
- Dimension `axis = 0`
- Dimension `axis = 1`

Note: name your variables in a descriptive way, for example for `sum` use: `a_sum`, `a0_sum`, `a1_sum` etc

Note2: remember that `np.std()` function calculates standard deviation either using 1/N or 1/(N-ddof) specified by the user, inside its formula.

[EXTRA]: (Complete after finishing other main questions first)

d) Create an array:
```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe']) # 7x1
```
- Use relational operator `==` to obtain a boolean array `boolBob` which gives `True` or `False` answer of where array names contains name `'Bob'` and another boolean array `boolWill` for `'Will'`.
- Reverse the logic of the array `boolBob`
- Create a new boolean array which is a result of a check for either `boolBob` **or** `boolWill`.

5) Applied Example

Q5 See L3_LAB_Questions.py Q5