ADVANCED PYTHON PROGRAMMING

Regular Expression and Yahoo Finance

Peter Lo

Regular Expression

Pattern Matching

Regular Expression Module

- A regular expression in a programming language is a special text string used for describing a search pattern.
- Extremely useful for extracting information from text such as code, files, log, spreadsheets or even documents.
- Widely used in natural language processing, web applications that require validating string and pretty much most data science projects that involve text mining.
- Implemented in the standard module ***re***, detail information can be found from:
  - https://docs.python.org/3/library/re.html

What is a regex pattern?

- A regex pattern is a special language used to represent generic text, numbers or symbols so it can be used to extract texts that conform to that pattern.
- Consider an example expression "**\s+**".
  - Here the "**\s**" matches any whitespace character.
  - By adding a '**+**' notation at the end will make the pattern match at least 1 or more spaces.
  - So this pattern will match even tab characters as well.

Split String Separated by regex

- If you intend to use a particular pattern multiple times, then you are better off compiling a regular expression rather than using ***re.split*** over and over again.

```
COM             101     Computers

MAT      205              Mathematics
ENG 189    English
```

This file contain three column, but the separator are different.

```
import re

with open("Course.txt", "r") as inFile:
    # Read the data to End of File
    InStrem = inFile.read()

    # Split the data by space, tab, new line
    ResultList = re.split('\s+', InStrem)

# Print the result out
print(ResultList)
```

The '\s' matches any whitespace character. By adding a '+' notation at the end will make the pattern match at least 1 or more spaces. This pattern will match even tab '\t' characters as well

```
['COM', '101', 'Computers', 'MAT', '205', 'Mathematics', 'ENG', '189', 'English']
```

Greedy vs Non-greedy Matching

- Greedy matching gets the longest results possible
- Nongreedy matching gets the shortest possible
- Consider a string = "***123 ABC 456 xyz***"
  - For greedy expression: **\d+**
    - Result: ['123', '456']
    - Maximizes the length of **\d**
  - For non-greedy expression: **\d+?**
    - Result: ['1', '2', '3', '4', '5', '6']
    - Minimizes the length of **\d**

Wildcards and Anchors

- . (a dot) matches any character except \n
  - ".oo.y" matches "Doocy", "goofy", "LooPy", ...
  - use \. to literally match a dot . character

```
import re

String = "Dog. Doocy Doooth"

regex = re.compile(".oo.y")
Result = regex.findall(String)
print(Result)

regex = re.compile("og\.")
Result = regex.findall(String)
print(Result)
```

```
['Doocy']
['og.']
```

Wildcards and Anchors

- ^ matches the beginning of a line; $ the end
  - "^fi$" matches lines that consist entirely of fi

```
import re

String = "This is a demo"

regex = re.compile("^This")
Result = regex.findall(String)
print(Result)

regex = re.compile("demo$")
Result = regex.findall(String)
print(Result)

if String.startswith("This") and String.endswith("demo"):
    print(String)
```

```
['This']
['demo']
This is a demo
```

Boolean

- | means OR
  - "abc|def|g" matches lines with "abc", "def", or "g"

```
import re

String = "abc is different from def or g"

regex = re.compile("abc|def|g")
Result = regex.findall(String)
print(Result)
```

```
['abc', 'def', 'g']
```

Grouping

- () are for grouping
  - "(Homer|Marge)" matches lines containing "Homer" or "Marge"

```
import re

String = "Red Apple and Green Apple are fruit"

regex = re.compile("(Red|Green)")
Result = regex.findall(String)
print(Result)
```

```
['Red', 'Green']
```

Finding Matched Pattern

- There are three method for searching matched pattern:
  - ***findall()*** returns the matched portions of the text as a list
  - ***search()*** returns a particular match object that contains the starting and ending positions of the first occurrence of the pattern.
  - ***match()*** also returns a match object, but the difference is, it requires the pattern to be present at the beginning of the text itself.

Finding Pattern using findall

- The ***findall()*** method extracts all occurrences of the 1 or more digits from the text and returns them in a list.

```
import re

with open("Course.txt", "r") as inFile:
    # Read the data to End of File
    InStream = inFile.read()

    # Define the Regular expression
    regex = re.compile("\d+")

    # Find all result match the criteria
    Result = regex.findall(InStream)

# Print the result out
print(Result)
```

```
['101', '205', '189']
```

the special character '\d' is a regular expression which matches any digit. Adding a '+' symbol to it mandates the presence of at least 1 digit to be present in order to be found.

Finding Pattern using search

- The ***search()*** method return the found value together with its position.

```
import re

with open("Course.txt", "r") as inFile:
    # Read the data to End of File
    InStream = inFile.read()

    # Define the Regular expression
    regex = re.compile("\d+")

    # Find all result match the criteria
    Result = regex.search(InStream)

# Print the result
print(Result.group())
print("Start Position:", Result.start())
print("End Position:", Result.end())
```

```
101
Start Position: 5
End Position: 8
```

Finding Pattern using match

- If the pattern is not present at the beginning of the text itself, the ***match()*** method unable to find the result.

```
import re

with open("Course.txt", "r") as inFile:
    # Read the data to End of File
    InStream = inFile.read()

    # Define the Regular expression
    regex = re.compile("\d+")

    # Find all result match the criteria
    Result = regex.match(InStream)

# Print the result out
print(Result)
```

```
None
```

Substitute Text with Another

- To replace texts, use the ***sub()*** method.

```
import re

with open("Course.txt", "r") as inFile:
    # Read the data to End of File
    InStream = inFile.read()

    # Replace one or more spaces with single space
    regex = re.compile("\s+")

    # Find all result match the criteria
    Result = regex.sub(" ", InStream)

# Print the result out
print(Result)
```

```
COM 101 Computers MAT 205 Mathematics ENG 189 English
```

Regular Expression Quick Guide

Symbol	Meaning
^	Matches the beginning of a line
$	Matches the end of the line
.	Matches any character except line terminators like \n.
*	Repeats a character zero or more times
*?	Repeats a character zero or more times (non-greedy)
+	Repeats a character one or more times
+?	Repeats a character one or more times (non-greedy)
(	Indicates where string extraction is to start
)	Indicates where string extraction is to end

Regular Expression Quick Guide

Symbol	Meaning
\s	Matches whitespace
\S	Matches any non-whitespace character
\d	Matches a digit
\D	Matches a non-digit
\w	Matches alphanumeric characters which means a-z, A-Z, 0-9 and underscore, _.
\W	Matches a non-alphanumeric characters
\n	Matches a new line
[*abcde*]	Matches a single character in the listed set
[^*xyz*]	Matches a single character not in the listed set
[*a-z0-9*]	The set of characters can include a range

Yahoo_fin

Scraping from Yahoo Finance

What is yahoo_fin?

- Yahoo_fin is a Python 3 package for scraping historical stock price data, as well as to provide current information on market caps, dividend yields, and which stocks comprise the major exchanges.
- Additional functionality includes scraping income statements, balance sheets, cash flows, holder information, and analyst data.

Using Module

- Before we can use yahoo_fin we will have to install it.

```
# Install Yahoo Finance
!pip install yahoo_fin
```

```
# Install Request HTML
!pip install requests_html
```

- It has to be imported like any other module:

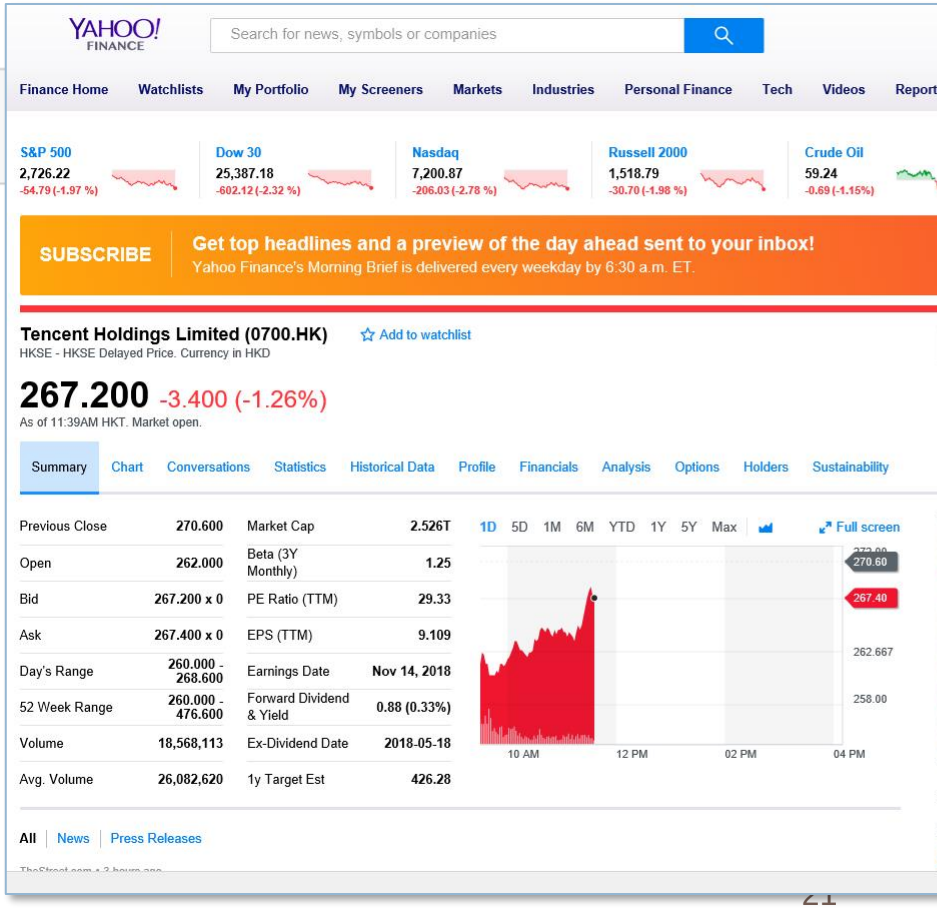
```
# import stock_info module from yahoo_fin
from yahoo_fin import stock_info as si
```

Obtain Live Quote Price

- The function ***get_live_price()*** scrapes the live quote price.

```
# Get live Stock Price
si.get_live_price("0700.HK")
```

```
267.20001220703125
```



Download Historical Price

- The function ***get_data()*** downloads historical price data of a stock into a pandas data frame.

```
si.get_data("0700.HK", start_date = "11/01/2018", end_date = "11/30/2018")
```

date	open	high	low	close	adjclose	volume	ticker
2018-11-01	275.000000	283.200012	271.399994	277.799988	277.799988	41075613	0700.HK
2018-11-02	295.000000	303.799988	289.399994	303.600006	303.600006	66912440	0700.HK
2018-11-05	294.600006	296.399994	286.799988	292.399994	292.399994	27220707	0700.HK
2018-11-06	294.799988	294.799988	286.000000	292.000000	292.000000	23335427	0700.HK
2018-11-07	293.399994	301.799988	286.000000	293.399994	293.399994	25726022	0700.HK
2018-11-08	301.000000	301.600006	292.000000	293.600006	293.600006	30646347	0700.HK
2018-11-09	287.600006	287.600006	278.200012	279.200012	279.200012	28764444	0700.HK
2018-11-12	275.399994	276.799988	270.000000	270.600006	270.600006	27299303	0700.HK
2018-11-13	262.000000	270.600006	260.000000	269.200012	269.200012	19513406	0700.HK

Scrape Quote Table

- The function ***get_quote_table()*** scrapes the primary table found on the quote page of an input ticker from Yahoo Finance

```
si.get_quote_table("0700.HK", dict_result = False)
```

	attribute	value
0	1y Target Est	426.28
1	52 Week Range	251.400 - 476.600
2	Ask	271.800 x 0
3	Avg. Volume	2.75384e+07
4	Beta (3Y Monthly)	1.25
5	Bid	271.600 x 0
6	Day's Range	260.000 - 272.600
7	EPS (TTM)	9.109
8	Earnings Date	Nov 14, 2018
9	Ex-Dividend Date	2018-05-18
10	Forward Dividend & Yield	0.88 (0.33%)

Scrape Statistics Information

- The function ***get_stats()*** scrapes data off the statistics page for the input ticker, which includes information on Price / Sales, P/E, and moving averages

```
si.get_stats("0700.HK")
```

	Attribute	Value
0	Market Cap (intraday) 5	2.57T
1	Enterprise Value 3	2.56T
2	Trailing P/E	29.82
3	Forward P/E 1	27.13
4	PEG Ratio (5 yr expected) 1	1.01
5	Price/Sales (ttm)	60.60
6	Price/Book (mrq)	57.22
7	Enterprise Value/Revenue 3	60.50
8	Enterprise Value/EBITDA 6	156.45
9	Fiscal Year Ends	Dec 31, 2017
10	Most Recent Quarter (mrq)	Jun 30, 2018

Scrape Analyst Info

- The function ***get_analysts_info()*** scrapes data from the Analysts page in Yahoo Finance.
  - This includes information on earnings estimates, EPS trends / revisions etc. Returns a dictionary containing the tables visible on the 'Analysts' page.

```
si.get_analysts_info("0700.hk")
```

```
{'Earnings Estimate':   Earnings Estimate  Current Qtr. (Sep 2018)  Next Qtr. (Dec 2018)  \
 0   No. of Analysts                    12.00                 10.00
 1     Avg. Estimate                     1.99                  2.24
 2      Low Estimate                     1.71                  1.80
 3     High Estimate                     2.35                  2.74
 4      Year Ago EPS                     2.13                  2.46

    Current Year (2018)  Next Year (2019)
 0                32.00             32.00
 1                 8.23             10.01
 2                 7.47              8.18
 3                10.11             13.47
 4                 8.47              8.23  ,
 'Revenue Estimate':          Revenue Estimate Current Qtr. (Sep 2018) Next Qtr. (Dec 2018)
 0          No. of Analysts                        18                    15
 1            Avg. Estimate                  91.45B                96.27B
```

Scrape Balance Sheet

- The function ***get_balance_sheet()*** scrapes the balance sheet from Yahoo Finance

```
si.get_balance_sheet("0700.HK")
```

	Period Ending	12/31/2017	12/31/2016	12/31/2015	12/31/2014
0	Current Assets	NaN	NaN	NaN	NaN
1	Cash And Cash Equivalents	105697000	71902000	43438000	42713000
2	Short Term Investments	36724000	50320000	37331000	10798000
3	Net Receivables	20352000	14384000	11214000	6918000
4	Inventory	295000	263000	222000	244000
5	Other Current Assets	11283000	9779000	57646000	11024000
6	Total Current Assets	178446000	149154000	155378000	75321000
7	Long Term Investments	278505000	170104000	114958000	72243000
8	Property Plant and Equipment	26760000	18574000	14221000	11748000
9	Goodwill	23608000	22927000	7155000	6356000
10	Intangible Assets	21769000	18714000	8577000	3699000
11	Accumulated Amortization	-	-	-	-

Scrape Cash Flow Statement

- The function ***get_cash_flow()*** scrapes the cash flow statement from Yahoo Finance

```
si.get_cash_flow("0700.HK")
```

	Period Ending	12/31/2017	12/31/2016	12/31/2015	12/31/2014
0	Net Income	71510000	41095000	28806000	23810000
1	Operating Activities, Cash Flows Provided By o...	NaN	NaN	NaN	NaN
2	Depreciation	22722000	11989000	6354000	4617000
3	Adjustments To Net Income	-12644000	715000	3562000	-1473000
4	Changes In Accounts Receivables	-6400000	-2930000	-2469000	-1418000
5	Changes In Liabilities	25251000	15488000	10408000	4475000
6	Changes In Inventories	-39000	-38000	-17000	1300000
7	Changes In Other Operating Activities	4851000	-1553000	-1533000	1199000
8	Total Cash Flow From Operating Activities	106140000	65518000	45431000	32711000
9	Investing Activities, Cash Flows Provided By o...	NaN	NaN	NaN	NaN
10	Capital Expenditures	-12108000	-8399000	-5440000	-4296000
11	Investments	-69244000	-57593000	-51365000	-22061000

Scrape Income Statement

- The function ***get_income_statement()*** scrapes the income statement, which includes information on Price / Sales, P/E, and moving averages

```
si.get_income_statement("0700.HK")
```

	Revenue	12/31/2017	12/31/2016	12/31/2015	12/31/2014
0	Total Revenue	237760000	151938000	102863000	78932000
1	Cost of Revenue	120835000	67439000	41631000	30873000
2	Gross Profit	116925000	84499000	61232000	48059000
3	Operating Expenses	NaN	NaN	NaN	NaN
4	Research Development	-	-	-	-
5	Selling General and Administrative	50703000	34595000	24818000	21952000
6	Non Recurring	-	-	-	-
7	Others	-7703000	-874000	-173000	-14000
8	Total Operating Expenses	163835000	101160000	66276000	52811000
9	Operating Income or Loss	73925000	50778000	36587000	26121000
10	Income from Continuing Operations	NaN	NaN	NaN	NaN

Get Holder

- The function ***get_holders()*** is used to scrapes data from the Holders tab from Yahoo Finance

```
si.get_holders("0700.HK")
```

```
{'Major Holders':    N/A        % of Shares Held by All Insider
 0  NaN       % of Shares Held by Institutions
 1  NaN        % of Float Held by Institutions
 2  NaN  Number of Institutions Holding Shares,
 'Direct Holders (Forms 3 and 4)':                          Holder   Shares Date Reported  % C
 0  Gilder, Gagnon, Howe & Co.  1566115  Jun 30, 2018  0.02%  591051791,
 'Top Institutional Holders':                                                Holder    Shares
 0  Vanguard International Stock Index-Emerging Ma...  91724377  Oct 31, 2017
 1  Vanguard International Stock Index-Total Intl ...  70449100  Jan 31, 2018
 2                           Europacific Growth Fund  47200711  Jun 30, 2018
 3            iShares Core MSCI Emerging Markets ETF  44680900  Jun 30, 2018
 4                 iShares MSCI Emerging Markets ETF  33568500  Jun 30, 2018
 5               Vanguard International Growth Fund  32265800  Feb 28, 2018
 6         Price (T.Rowe) Blue Chip Growth Fund Inc.  22001800  Jun 30, 2018
 7                  Harbor Capital Appreciation Fund  20921515  Jun 30, 2018
 8             Price (T.Rowe) Growth Stock Fund Inc.  18572500  Jun 30, 2018
 9  Fidelity Series Emerging Markets Opportunities...  16638100  Jun 30, 2018

    % Out        Value
 0  0.96%  32066842759
```