Regular Expressions

Data Skills for Empirical Research

Winter, 2020

NORTHWESTERN UNIVERSITY

Kellogg School of Management

NORTHWESTERN UNIVERSITY

Kellogg School of Management

Dealing with Unstructured Data

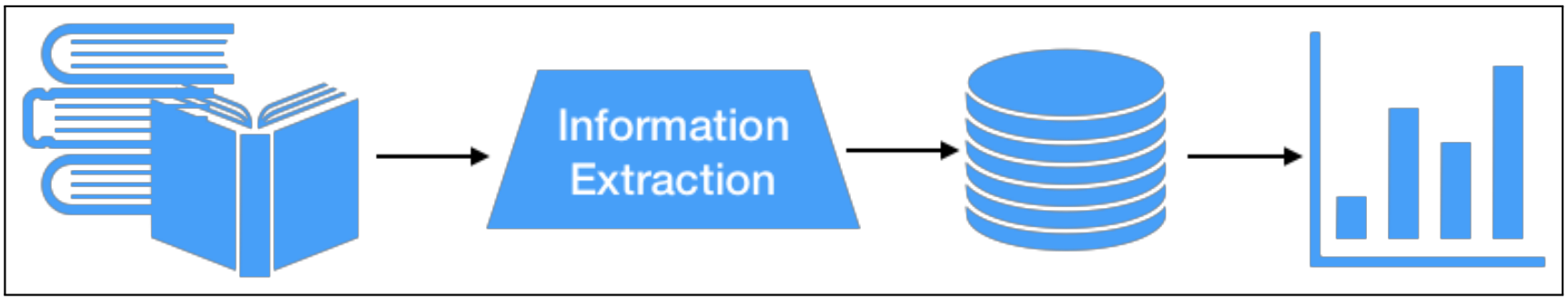
Vast quantities of information are encoded as unstructured data, in the form of natural language text.

But it can be hard to make this information available for computational analysis at scale.

Information Extraction from Text

Regular expressions can be used in many ways, including to perform information extraction, which converts information stored in text into structured data:

1. Search for relevant chunks of information in a collection of texts
2. Map these chunks to structured representations
3. Store the results as tabular data for downstream analysis



Example: Federal Register

Example: Federal Register

GOVERNMENT PRINTING OFFICE

NAME	OFFICIAL TITLE	Legal residence		Compensation
		State	Cong. Dist.	
George H. Carter	Public Printer	Iowa	9th	$10,000
John Greene	Deputy Public Printer	Mass	5th	7,500
Mary A. Tate	Assistant to the Public Printer	Tenn	2d	4,000
Henry H. Wright	Chief clerk	N. Y	28th	4,200
Edward J. Wilver	Disbursing clerk	Pa	16th	4,000
William A. Smith	Congressional Record clerk	D. C		4,000
Daniel P. Bush	Medical and sanitary officer	Nebr	1st	4,200
James K. Wallace	Superintendent of accounts and budget officer	Ohio	1st	5,000
Ernest E. Emerson	Purchasing agent	Md	5th	4,600
Edward O. Reed	Technical director	D. C		5,200
Burr G. Williams	Chief instructor of apprentices	Iowa	10th	3,600
Ellwood S. Moorhead	Production manager	Pa	6th	5,200
Edward A. Huse	Night assistant production manager	Mass	6th	4,800
Hermann B. Barnhart	Superintendent of printing	Ind	9th	4,400
Bert E. Bair	Superintendent of presswork	Mich	6th	4,400
Martin R. Speelman	Superintendent of binding	Mo	4th	4,400
Edward G. Whall	Superintendent of platemaking	Mass	12th	4,400
William A. Mitchell	Superintendent of planning	N. C	5th	4,400
Alfred E. Hanson	Superintendent of construction and maintenance.	Mass	14th	5,200
Alton P. Tisdel	Superintendent of documents	Ohio	22d	5,000
William H. Kervin	Storekeeper and traffic manager	N. Y	39th	4,200

LIBRARY OF CONGRESS

Example: Form 4 (Insider Trading)

UNITED STATES
SECURITIES AND EXCHANGE COMMISSION
Washington, DC 20549

FORM 4

STATEMENT OF CHANGES OF BENEFICIAL OWNERSHIP OF SECURITIES

The Commission is authorized to solicit the information required by this Form pursuant to Sections 16(a) and 23(a) of the Securities Exchange Act of 1934, Sections 17(a) and 20(a) of the Public Utility Holding Company Act of 1935, and Sections 30(h) and 38 of the Investment Company Act of 1940, and the rules and regulations thereunder.

Disclosure of information specified on this Form is mandatory, except for disclosure of the I.R.S. identification number of the reporting person if such person is an entity, which is voluntary. If such numbers are furnished, they will assist the Commission in distinguishing reporting persons with similar names and will facilitate the prompt processing of the Form. The information will be used for the primary purpose of disclosing the transactions and holdings of directors, officers, and beneficial owners of registered companies.

Example: Form 4 (Insider Trading)

FORM 4

UNITED STATES SECURITIES AND EXCHANGE COMMISSION
Washington, D.C. 20549

STATEMENT OF CHANGES IN BENEFICIAL OWNERSHIP

Filed pursuant to Section 16(a) of the Securities Exchange Act of 1934 or Section 30(h) of the Investment Company Act of 1940

OMB APPROVAL	
OMB Number:	3235-0287
Estimated average burden hours per response:	0.5

☐ Check this box if no longer subject to Section 16. Form 4 or Form 5 obligations may continue. *See* Instruction 1(b).

1. Name and Address of Reporting Person*	2. Issuer Name **and** Ticker or Trading Symbol	5. Relationship of Reporting Person(s) to Issuer (Check all applicable)
STIRITZ WILLIAM P	Post Holdings, Inc. [POST]	X Director; 10% Owner; Officer (give title below); Other (specify below)
(Last) (First) (Middle) C/O POST HOLDINGS, INC. 2503 S. HANLEY ROAD	3. Date of Earliest Transaction (Month/Day/Year) 02/06/2019	
(Street) ST. LOUIS MO 63144	4. If Amendment, Date of Original Filed (Month/Day/Year)	6. Individual or Joint/Group Filing (Check Applicable Line) X Form filed by One Reporting Person Form filed by More than One Reporting Person
(City) (State) (Zip)		

Table I - Non-Derivative Securities Acquired, Disposed of, or Beneficially Owned

1. Title of Security (Instr. 3)	2. Transaction Date (Month/Day/Year)	2A. Deemed Execution Date, if any (Month/Day/Year)	3. Transaction Code (Instr. 8)		4. Securities Acquired (A) or Disposed Of (D) (Instr. 3, 4 and 5)			5. Amount of Securities Beneficially Owned Following Reported Transaction(s) (Instr. 3 and 4)	6. Ownership Form: Direct (D) or Indirect (I) (Instr. 4)	7. Nature of Indirect Beneficial Ownership (Instr. 4)
			Code	V	Amount	(A) or (D)	Price			
Common Stock	02/06/2019		P		869	A	$95.1565(1)	250,942	I	By Spouse
Common Stock	02/06/2019		P		133,190	A	$96.7193(2)	384,132	I	By Spouse
Common Stock								2,659,862	D	
Common Stock								169,369	I	By Trust

Table II - Derivative Securities Acquired, Disposed of, or Beneficially Owned (e.g., puts, calls, warrants, options, convertible securities)

1. Title of Derivative Security (Instr. 3)	2. Conversion or Exercise Price of Derivative Security	3. Transaction Date (Month/Day/Year)	3A. Deemed Execution Date, if any (Month/Day/Year)	4. Transaction Code (Instr. 8)		5. Number of Derivative Securities Acquired (A) or Disposed of (D) (Instr. 3, 4 and 5)		6. Date Exercisable and Expiration Date (Month/Day/Year)		7. Title and Amount of Securities Underlying Derivative Security (Instr. 3 and 4)		8. Price of Derivative Security (Instr. 5)	9. Number of derivative Securities Beneficially Owned Following Reported Transaction(s) (Instr. 4)	10. Ownership Form: Direct (D) or Indirect (I) (Instr. 4)	11. Nature of Indirect Beneficial Ownership (Instr. 4)
				Code	V	(A)	(D)	Date Exercisable	Expiration Date	Title	Amount or Number of Shares				

Explanation of Responses:

1. The reported price in Column 4 is a weighted average price. These shares were purchased in multiple transactions at prices ranging from $95.09 to $95.29 per share. The reporting person undertakes to provide to the issuer, any security holder of the issuer, or the staff of the Securities and Exchange Commission, upon request, full information regarding the number of shares purchased at each separate price within the range set forth in this footnote.

2. The reported price in Column 4 is a weighted average price. These shares were purchased in multiple transactions at prices ranging from $96.23 to $97.10 per share. The reporting person undertakes to provide to the issuer, any security holder of the issuer, or the staff of the Securities and Exchange Commission, upon request, full information regarding the number of shares purchased at each separate price within the range set forth in this footnote.

Regular Expressions (Regex)

From the official Python Regex "howto" guide:

> Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the `re` module. Using this little language, you specify the rules for the set of possible strings that you want to match; this set might contain English sentences, or e-mail addresses, or TeX commands, or anything you like. You can then ask questions such as "Does this string match the pattern?", or "Is there a match for the pattern any where in this string?". You can also use REs to modify a string or to split it apart in vari ous ways.
>
> Regular expression patterns are compiled into a series of bytecodes which are then executed by a matching engine written in C. For advanced use, it may be necessary to pay careful attention to how the engine will execute a given RE, and write the RE in a certain way in order to produce bytecode that runs faster. Optimization isn't covered in this document, because it requires that you have a good understanding of the matching en gine's internals.

Regular Expressions (Regex)

Most letters and characters will simply match themselves.:

regex: **expression**

Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the re module. Using this little language, you specify the rules for the set of possible strings that you want to match; this set might contain English sentences, or e-mail addresses, or TeX commands, or anything you like. You can then ask questions such as "Does this string match the pattern?", or "Is there a match for the pattern anywhere in this string?". You can also use REs to modify a string or to split it apart in various ways.

Regular expression patterns are compiled into a series of bytecodes which are then executed by a matching engine written in C. For advanced use, it may be necessary to pay careful attention to how the engine will execute a given RE, and write the RE in a certain way in order to produce bytecode that runs faster. Optimization isn't covered in this document, because it requires that you have a good understanding of the matching engine's internals.

Regular Expressions (Regex)

Some characters are special metacharacters, and don't match themselves. For example, square brackets create character "classes":

regex: **[Rr]eg**

Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available throu gh the re module. Using this little language, you specify the rules for the set of poss ible strings that you want to match; this set might contain English sentences, or e-mai l addresses, or TeX commands, or anything you like. You can then ask questions such as "Does this string match the pattern?", or "Is there a match for the pattern anywhere in this string?". You can also use REs to modify a string or to split it apart in various ways.

Regular expression patterns are compiled into a series of bytecodes which are then exec uted by a matching engine written in C. For advanced use, it may be necessary to pay ca reful attention to how the engine will execute a given RE, and write the RE in a certai n way in order to produce bytecode that runs faster. Optimization isn't covered in this document, because it requires that you have a good understanding of the matching engin e's internals.

Regular Expressions (Regex)

Another metachacter is the wildcard '.':

regex: **RE.**

Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the re module. Using this little language, you specify the rules for the set of possible strings that you want to match; this set might contain English sentences, or e-mail addresses, or TeX commands, or anything you like. You can then ask questions such as "Does this string match the pattern?", or "Is there a match for the pattern anywhere in this string?". You can also use REs to modify a string or to split it apart in various ways.

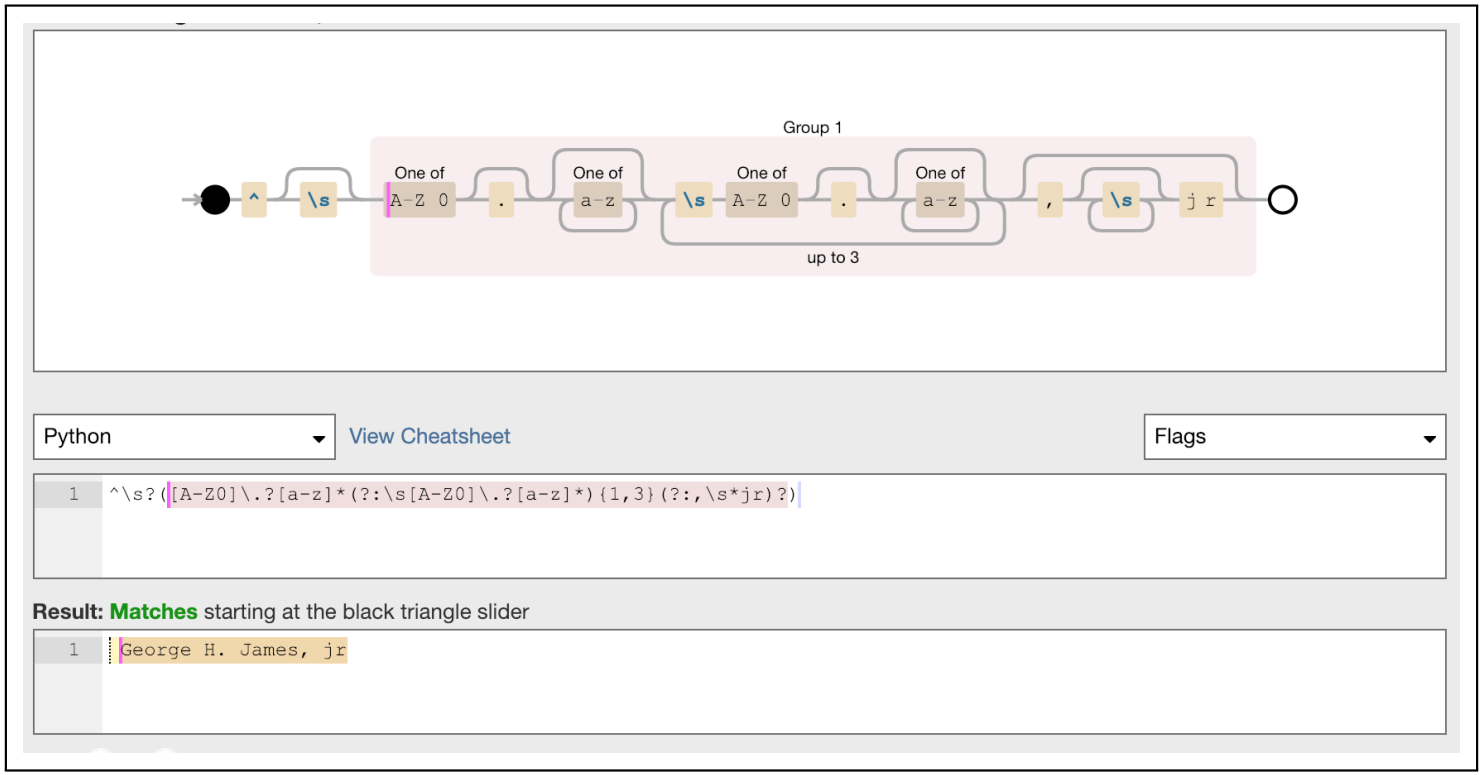
Regular expression patterns are compiled into a series of bytecodes which are then executed by a matching engine written in C. For advanced use, it may be necessary to pay careful attention to how the engine will execute a given RE, and write the RE in a certain way in order to produce bytecode that runs faster. Optimization isn't covered in this document, because it requires that you have a good understanding of the matching engine's internals.

Regular Expressions (Regex)

We'll explore these and more in the regex lab. Python regex methods include:

Method/Attribute	Purpose
`match()`	Determine if the RE matches at the beginning of the string.
`search()`	Scan through a string, looking for any location where this RE matches.
`findall()`	Find all substrings where the RE matches, and returns them as a list.
`finditer()`	Find all substrings where the RE matches, and returns them as an iterator.

Regex Tools



https://www.debuggex.com

https://jex.im/regulex

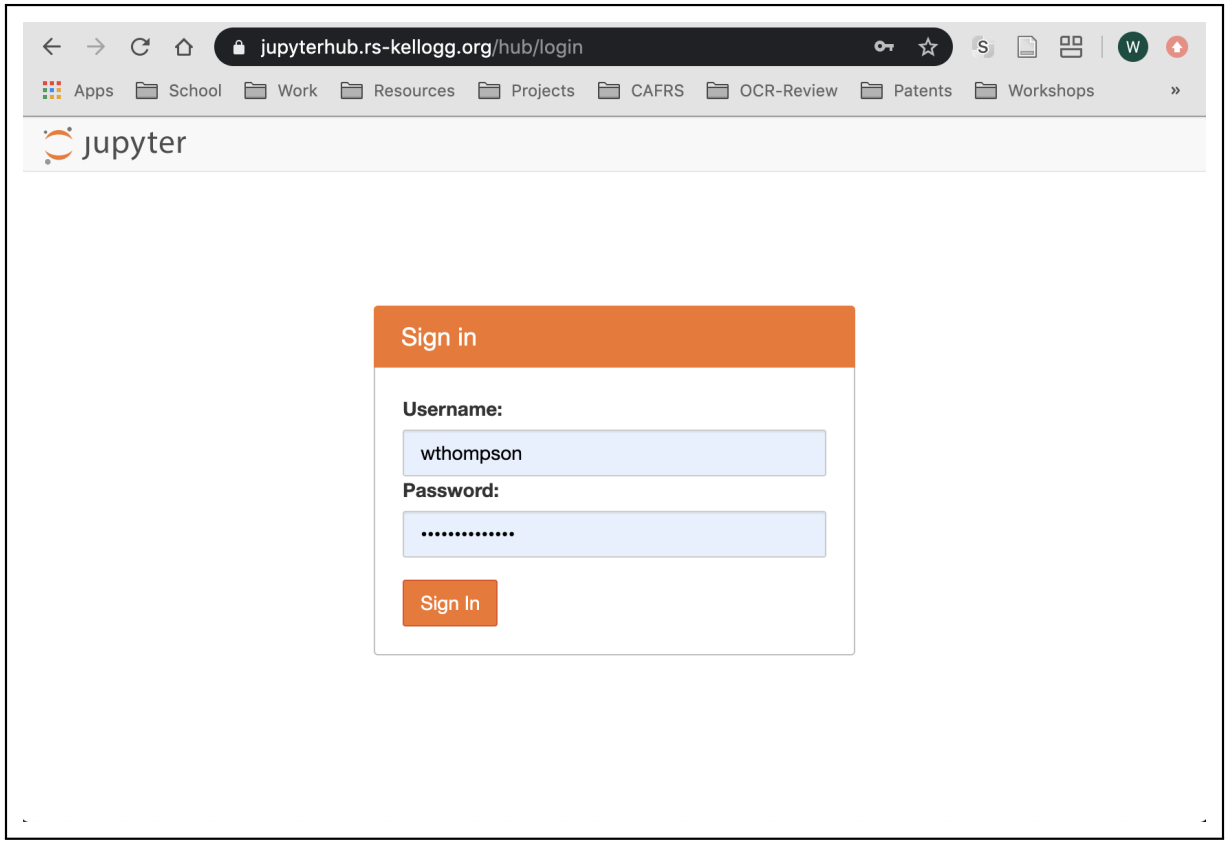
https://regex101.com

Friendly Regex Advice

DO use a tool for testing expressions against target strings.
DO break your parsing problems into smaller pieces when you can.
DO write lots of `print` statements and comments.
DO consider a variety of patterns for your job.
DO come ask us for help if you get stuck.

DON'T try to write one gigantic regex when a few simple ones will do.
DON'T fall in love with `.*`.
DON'T write programs that break if a regex fails, if you can avoid it.
DON'T get discouraged.

Regex Lab



https://jupyterhub.rs-kellogg.org/hub/user-redirect/git-pull?repo=https%3A%2F%2Fgithub.com%2Frs-kellogg%2Fempirical-workshop-2020&subPath=3-regex&app=lab

Resources

- Online tool: https://regex101.com
- Python cheat sheet: https://www.debuggex.com/cheatsheet/regex/python
- Python official guide: https://docs.python.org/3/howto/regex.html
- The surprisingly long history of Regex (6 min. video): http://bit.ly/38yprAl
- Book chapter (NLP focus): http://web.stanford.edu/~jurafsky/slp3
- Regex crosswords: https://regexcrossword.com