

Group Members: Caroline-Marie Jacobsen, Kamilla Dalgaard, Viktor Sybrandt, Victor Hjuler

Week 8 assignment is, inevitably, about Regular Expressions and OpenRefine

Upload a text file or a PDF with your answers/solutions to the problems below. Beware of making the submission legible and understandable to another reader; for example, consider using the "Save regex" functionality in regex101.com, which allows you to create a link out of your solution and share the link for easy use by your colleagues. Remember that you can elaborate solutions in groups, but need to submit **individually**.

1. What regular expressions do you use to extract all the dates in this blurb:

<http://bit.ly/regexexercise2> and to put them into the following format YYYY-MM-DD ?

Answer: <https://regex101.com/r/FewRxZ/1>

We start by writing the regular expression `(\d+).(\d+).\s?(\d+)`. Each parenthesis refers to one part of the date, which we want to change.

`\d` matches any digit between 0-9 and the `+` ensures that the digit can be repeated as many times as possible.

The `.` matches any character and the `\s` matches any blank space. Combined with the question mark after `\s` ensures that this element can be repeated as many times as needed.

By writing `$` it refers to any part of the regular expression. `$1` refers to the first parentheses, `$2` to the next and so on. We would like to change the order of the dates to YYYY-MM-DD instead of the current MM-DD-YYYY. In order to do so, we will write the following: `$3-$1-$2`.

The screenshot shows the regex101.com interface. The regular expression `/(\d+).(\d+).\s?(\d+)/gm` is entered in the 'REGULAR EXPRESSION' field. The 'TEST STRING' contains several historical dates in MM-DD-YYYY format, such as 'Juan Ponce de León sights Florida for the first time, on 3.27.1513'. The 'SUBSTITUTION' field shows the result `$3-$1-$2`, which reorders the dates to YYYY-MM-DD. The 'EXPLANATION' panel on the right details the components of the regex: `\d` for digits, `.` for any character, and `\s?` for an optional space. The 'MATCH INFORMATION' table lists the matches found in the test string.

Match	Start	End	Text
Match 1	57	67	3.27.1513
Group 1	57	58	3
Group 2	59	61	27
Group 3	63	67	1513

2. Write a regular expression to convert the stopwordslist (list of most frequent Danish words) from Voyant in <http://bit.ly/regexexercise3> into a neat stopwords list for R (which comprises "words" separated by commas, such as <http://bit.ly/regexexercise4>). Then take the stopwordslist from R <http://bit.ly/regexexercise4> and convert it into a Voyant list (words on separate line without interpunction)

From Voyant list to stop list for R: <https://regex101.com/r/traDyL/1>

When making the list, we used the regular expression `([a-æøå0-9]+)`. This expression captures one group: it will match a single character present in the list. `+` matches the previous token between one and unlimited times, as many times as possible, giving back as needed. `a-æ` matches a single character in the range between a and æ. `øå` matches a single character in the list øå. `0-9` matches a single character in the range between 0 and 9. In the substitution we have written `"$1"` which matches the end of the string. It "refers" to the parenthesis in the top text. we wanted `"` on both sides of the words, which is why it around the `$1`.

The screenshot shows the regex101.com interface. The regular expression `([a-æøå0-9]+)` is entered in the top field. The test string contains several lines of text, including 'aaen', 'ad', 'ænder', 'af', and 'aarschoen'. The substitution result shows the words separated by commas, e.g., 'aaen', 'ad', 'ænder', 'af', 'aarschoen'. The right sidebar provides an explanation of the regex, match information, and a quick reference for various tokens.

Working with the same expression, we then have to separate the lines without interpunction. We used `([a-æøå0-9]+\s+)`. A `\s+` has been added to the expression, and matches any whitespace character. `+` matches the previous token between one and

Group Members: Caroline-Marie Jacobsen, Kamilla Dalgaard, Viktor Sybrandt, Victor Hjuler

unlimited times, as many times as possible, giving back as needed. The substitution will also change and now makes it one text.

The screenshot shows a regex testing interface. The 'REGULAR EXPRESSION' field contains `"/ ([a-æøå0-9]+) \s+ / gm`. The 'TEST STRING' field contains a list of names: `"2""3""4""aaen""ad""ænr""af""agerschou""akdogan""aldrig""alene""alexandrines""alfred""alle""allerede""alligevel""alt""altid""ammitzbøll""amsterdamtraktaten""amtoft""anden""andet""andre""annette""anni""antonsen""arbo""at""augustforlig""augustforliget""augustforligets""augustforligspartierne""augustforligspartiernes""baagø""baastrup""bastrup""bæhr""bag""bare""barfod""begge""beskæftigelsesminister""b`. The 'SUBSTITUTION' field shows the result: `"$1"`. The 'EXPLANATION' panel on the right details the 1st Capturing Group `([a-æøå0-9]+)`, stating it matches a single character present in the list below `[a-æøå0-9]` unlimited times. The 'MATCH INFORMATION' panel shows Match 1 at positions 0-3 with Group 1 at 0-1, and Match 2 at positions 3-6 with Group 1 at 3-4. The 'QUICK REFERENCE' panel lists various token types like All Tokens, Common Tokens, General Tokens, and Anchors.

From stop list for R to Yoyant list: <https://regex101.com/r/3C4QXD/1>

When making the expression, we used `(")([a-æøå0-9éü.']+)(,)`, the `(")` captures group one and `"` matches every `"` character. The next group is `([a-æøå0-9éü.']+)` and `+` matches the previous token between one and unlimited times, as many times as possible, giving back as needed. `a-z` matches a single character between a and z. `æøå` matches a single character in the list æøå, and `0-9` matches a single character in the range 0-9. `éü.´` matches a single character in the list éü.´. the last group `(,)` matches the characters `,`. In substitution we used `$2` to refer to group 2 and `\n` for the new line.

Group Members: Caroline-Marie Jacobsen, Kamilla Dalgaard, Viktor Sybrandt, Victor Hjuler

The screenshot shows a regular expression tool interface. The 'REGULAR EXPRESSION' field contains the pattern `/(("[a-zæøåö-9éü.']+)(,))/gm`. The 'TEST STRING' field contains a list of names in Danish, each enclosed in double quotes and separated by commas. The 'SUBSTITUTION' field shows the result of the regex, where each name is replaced by `$2\n`. The 'EXPLANATION' panel on the right provides details about the matches, including the 1st and 2nd capturing groups. The 'MATCH INFORMATION' table lists the matches and their corresponding groups.

Match	Start	End	Text
Match 1	20	32	"højtærede",
Group 1	20	21	"
Group 2	21	30	højtærede
Group 3	30	32	,

3. Does OpenRefine alter the raw data during sorting and filtering? Fix the [interviews](#)

[dataset](#) in OpenRefine enough to answer this question: "Which two months are reported as the most water-deprived/dryest by the interviewed farmer households?"

Openrefine doesn't alter the raw data by changing, but makes it more visually tidy, and can make it easier to read. First we upload the raw data to openrefine. To answer the question, we sorted/declustered/filtered the data. When making it more tidy, we took the Facets and removed the brackets, "", ', and the bigger spaces. To specifically find the months, we used the expression `Value.split(",")` to find the month with the most matches. You can also use a facet filter to check, but this way is more visually tidy. We found that the two most water-deprived/dyrest months are October with 80 matches and September with 70 matches.

Group Members: Caroline-Marie Jacobsen, Kamilla Dalgaard, Viktor Sybrandt, Victor Hjuler

Custom facet on column months_no_water

Expression Language **General Refine Expression Language (GREL)**

value No syntax error

[Preview](#) [History](#) [Starred](#) [Help](#)

From **Expression**

[Reuse](#) This project `grel: value.split(",")`

Facet / Filter Undo / Redo 50 / 50

[Refresh](#) [Reset all](#) [Remove all](#)

months_no_water [invert reset](#)

☐ case sensitive ☐ regular expression

months_no_water [change](#)

11 choices Sort by: name count

- Oct 80
- Sept 70
- Nov 51
- NULL 45
- Aug 19
- Dec 6
- Jan 2
- July 2
- Apr 1
- June 1
- May 1

131 rows

Show as: **rows** records Show: 5 10 25 50 100 500 1000 rows

othe	no_f	plots	wate	no_g	yes	no_e	months_r	peric	expe	othe	res	mem	resp
2	2	no	2	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	N
3	3	yes	NULL	3	yes	Aug;Sept;Oct	2	yes	no	NULL	yes	no	n
1	1	no	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	N
3	3	no	3	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	N
2	2	no	2	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	N
1	1	no	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	N
4	4	yes	NULL	4	yes	Aug;Sept;Oct	10	yes	no	NULL	no	NULL	n
2	2	yes	NULL	2	yes	Sept;Oct	10	yes	no	NULL	yes	yes	n
3	3	yes	NULL	3	yes	Oct;Nov	6	yes	no	NULL	no	NULL	n
2	2	yes	NULL	2	yes	Sept;Oct;Nov	22	yes	no	NULL	no	NULL	n

The expressions to remove brackets etc.

[Preview](#) [History](#) [Starred](#) [Help](#)

Reuse	This project	grel: value.split(",")
Reuse	This project	grel: value.replace("","").replace(" ","")
Reuse	This project	grel: value.replace("[","").replace("]", "")
Reuse	This project	grel: value.replace("]", "")
Reuse	This project	grel: value.replace("[", "")
Reuse	This project	grel: value.replace("[","").replace("]", "").replace("","").replace(" ","")
Reuse	This project	grel: value

- 4. Real-Data Challenge: What are the 10 most frequent occupations "erhverv" among unmarried men or women of 20-30 years in 1801 Aarhus census dataset? (hint: first select either men or women to shrink the dataset to a manageable size, then filter by age, and then use merging to cut the erhvervvariation ruthlessly.)**

Women unmarried - 20-30 years 10 most frequent occupations:

1. Tjenestepige
2. Væverske
3. Huusholderske
4. Huusjomfru
5. Inderste
6. Kokkepige
7. Lever af sine midler
8. Tjener faderen
9. Væverpige
10. Bryggerpige

Men unmarried 20-30 years 10 most frequent occupations:

1. Soldat
2. landsoldat
3. tjenestekarl
4. væver
5. gårdkarl
6. land cerut
7. Bonde og Gaardbeboer
8. Land-rytter
9. Lærerdreng
10. Nationalrytter

To get these answers we had to go through a variety of steps, which is shown below.

Note: this explanation only shows the results for women, though it is representative for both cases.

Group Members: Caroline-Marie Jacobsen, Kamilla Dalgaard, Viktor Sybrandt, Victor Hjuler

step 1, upload data, text facet: After we uploaded the data we chose to make the “occupation” column into a text facet, to get a raw overview of the data.



as we can see here, the data is messy and disorganised.

step 2, cluster and filtering: To clean up the data and, we had to get rid of all the possible data corruption, such as a word thats spelled the same, but one has a lowercase letter and the other has an uppercase letter. to do this we clustered the data to make it even more clean.



as we only needed to get the most popular job/occupation for unmarried women, we filtered the data so as to only show the data we needed

✕

koen

invert reset

kvinde

☐ case sensitive

☐ regular expression

✕

civilstand

invert reset

ugift

☐ case sensitive

☐ regular expression

Step 3, age filtering: Then we had to only get data from unmarried women who were between the ages of 20-30, which proved a little tricky.

At first we decided to make a numeric facet of the “age” column, and sort it into “count”(picture 1).

Since the ages ranges from 1-86 years, we had to only include the ages from 20-30 and exclude all other data from that column (picture 2).

1.

alder		change invert reset
86 choices Sort by: name count		Cluster
1	634	
2	611	
3	596	edit include
15	568	
4	567	
5	562	
6	541	
7	524	
8	499	
10	481	
9	479	
12	474	

2.

alder		change invert reset
86 choices Sort by: name count		Cluster
20	378	exclude
18	337	
21	312	edit exclude
22	297	exclude
19	290	
24	272	exclude
17	245	
23	231	exclude
26	206	exclude
25	187	exclude
28	139	exclude
27	123	exclude