Exercise 2: Programming & Algorithms

Exercise 2 for the lecture 'Foundations of Data Science'

Prof. Dr. Karsten Donnay, Assistant: Philipp Kling

This session covers

– Data processing & cleaning

– Manipulation of text data

– Regular expressions

Data processing & cleaning

Data processing & cleaning

- To make data machine readable, we often need to convert it to another format or extract only features we are interested in.

- Many ways to save and store data

- Overview:
  - HTML and XML
  - JSON

Data processing & cleaning

HTML (Hyper Text Markup Language)

```
<!DOCTYPE html>
<html>
    <head>
            <title>Page Title</title>
    </head>
    <body>
            <h1>This is a Heading</h1>
            <p>This is a paragraph.</p>
    </body>
</html>
```

Data processing & cleaning

HTML (Hyper Text Markup Language)

– Start tags `<title>` and end tags `</title>` indicate elements

– Elements have attributes: e.g. `<title id= "a">` ... `</title>` has the attribute `id="a"`

– Every website is built in HTML. HTML files can be displayed by your browser. Can be inspected by right clicking in your browser.

Workflow in R:

– Download page and parse to an XML-file format in R with the **rvest** and **xml2** package (they are very similar). Both have a function named **read_html().** Then use functions of both packages to inspect data.

Data processing & cleaning

XML (Extensible Markup Language)

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

Data processing & cleaning

XML (Extensible Markup Language)

–   Used to store data (not for visualization as compared to HTML)

–   Used to exchange information between web services (e.g. apps)

–   Very flexible and highly customizable

Workflow in R:

–   Download data and parse it into R with the **xml2** package (**read_xml()**)

–   Useful functions to inspect and extract information:

–   **xml_nodes:** to get the nodes of a file.   -   **xml_names**: to retrieve the names of the nodes

–   **xml_children**: to get the content of a node.   -   **xml_text**: to transform the content into text.

Data processing & cleaning

JSON (JavaScript Object Notation)

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

Data processing & cleaning

JSON (JavaScript Object Notation)

– Also used to store data

– More popular (especially for APIs)

– Less flexible than XML, more standardized

– Workflow in R:

 – Download data and parse it into R with the **jsonlite** package (**fromJSON()**)

 – If there aren't any encoding problems the package should already convert the data to a data.frame.

Text manipulation in R

String manipulation

– Concatenate text with **paste**:

```
> (text <- paste("Demonstration of", "text manipulation."))
  "Demonstration of text manipulation."
> (text <- paste("Demonstration of", "text manipulation.", sep = "-"))
  "Demonstration of-text manipulation."
```

– Concatenate text and vectors with **paste**:

```
> abc <- c("a", "b", "c")
> paste("Option", abc, collapse=", ")
        "Option a, Option b, Option c"
```

String manipulation

– Split strings with **strsplit**:

> ```
> > strsplit("2020-02-11", split="-")
> ```
>
> "2020"   "02"   "11"

– Extract parts of a string with **substr**:

> ```
> > substr("Rabarberrabarberrabarber", start=1, stop=8)
> ```
>
> "Rabarber"

String manipulation

– Detect a pattern in text with **grep** or the **stringr** - package

```
> grepl(pattern = "prize", "Somewhere I've hidden a prize in this sentence.")
```
    TRUE   (returns whether a hit or not)

```
> grep(pattern = "prize", c("Somewhere I've hidden a prize and another prize",
                            "But not in this one",
                            "But there is a prize in this one."))
```
    1 3   (returns positions of hits)

```
> stringr::str_extract(string="Somewhere I've hidden a prize and another prize.",
                       pattern = "prize")
```
    "prize"   (returns first match)

```
> stringr::str_extract_all(string="Somewhere I've hidden a prize and another prize",
                           pattern = "prize")
```
    "prize"   "prize"   (returns all matches)

Regular expressions

Regular expressions

– What if we want to extract or detect features more generally?

E.g. what would you do if you want to extract all the first names from such a text?

```
> text <- "Abou-Chadi Tarik AFL-H-359 +41 44 634 52 03Caramani Daniele AFL-H-344
+41 44 634 40 10Donnay Karsten AFL-H-350 +41 44 634 58 57"

> unlist(stringr::str_extract_all(string = text,
                                  pattern = "[A-Z]{1}[a-z]+ [A-Z]{1}[a-z]+"))

         "Chadi Tarik"  "Caramani Daniele"   "Donnay Karsten"
```

Regular expressions were the solution here!

– Formal language used in programming

– General pattern that matches text

– Cross-platform

– Can be used to clean text data or extract text features of interest

Regular expressions

- `[:punct:]`: punctuation.
- `[:alpha:]`: letters.
- `[:lower:]`: lowercase letters.
- `[:upper:]`: upperclass letters.
- `[:digit:]`: digits.
- `[:xdigit:]`: hex digits.
- `[:alnum:]`: letters and numbers.
- `[:cntrl:]`: control characters.
- `[:graph:]`: letters, numbers, and punctuation.
- `[:print:]`: letters, numbers, punctuation, and whitespace.
- `[:space:]`: space characters (basically equivalent to `\s`).
- `[:blank:]`: space and tab.

- `[abc]`: matches a, b, or c.
- `[a-z]`: matches every character between a and z (in Unicode code point order).
- `[^abc]`: matches anything except a, b, or c.
- `[\^\-]`: matches ^ or -

Regular expressions

– Grouping

– Anchors

– Repetition

Signs after an expression indicate
how often it should or may appear.

- `?`:   0 or 1.
- `+`:   1 or more.
- `*`:   0 or more.
- `{n}`:   exactly n
- `{n,}`:   n or more
- `{n,m}`:   between n and m

Regular expressions

– Grouping

– Anchors

– Repetition

Anchors indicate the start or end of a text.

- ^ matches the start of string.
- $ matches the end of the string.

Regular expressions

    –   Grouping

    –   Anchors

    –   Repetition

Parentheses define a group

```
> str_extract(c("grey", "gray"), "gre|ay")
  "gre"  "ay"
> str_extract(c("grey", "gray"), "gr(e|a)y")
  "grey"  "gray"
```

Regular expressions

- There are many useful website to test or look up regular expressions.

    - E.g. https://regexr.com/

Text data

Text data

– Text can be used in many ways. Some examples:

  – Classification of social media accounts via their published comments

  – Sentiment analyses of speeches

  – Usage of words development in news throughout the last decade

  – …

– To do so, we need to be able to clean text or e.g. count words in a text.

– Common cleaning steps are e.g. the removal of stop words, the stemming of words, or the summarization to word embeddings.

– Next, we switch over to R to do some basic word extraction with regular expressions.