[rvest][https://www.analyticsvidhya.com/blog/2017/03/beginners-guide-on-web-scraping-in-r-using-rvest-with-hands-on-knowledge/]

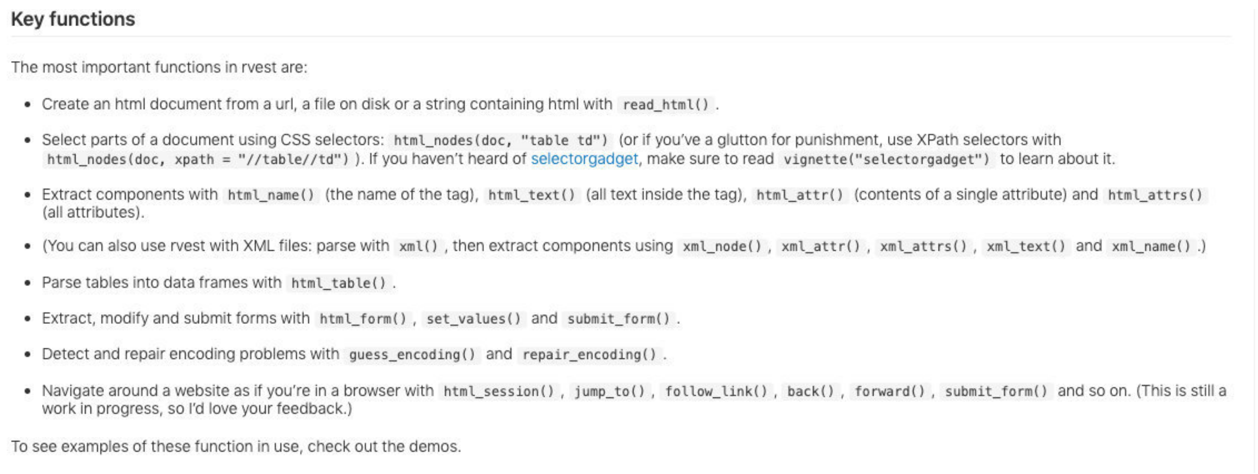
Easily Harvest(Scrape) Web Pages

With the amount of data available over the web, it opens new horizons of possibility for a Data Scientist. I strongly believe web scraping is a must have skill for any data scientist. In today's world, all the data that you need is already available on the internet - the only thing limiting you from using it is the ability to access it.

Most of the data available over the web is not readily available. It is present in an unstructured format(HTML format) and is not dowloadable.

Web scraping is a technique for conveting the data present in unstructured format over the web to the structured format which can easily be accessed and used.

Selector gadget使用: 选中匹配单元(显绿色). SelectorGadget将会针对所选单元形成最小匹配结果, 匹配单元颜色变黄. 再在高亮单元中进一步选择, 单击拒绝单元(显红), 或选择未高亮区域增加选择单元(显绿)



Key functions

The most important functions in rvest are:

- Create an html document from a url, a file on disk or a string containing html with `read_html()` .
- Select parts of a document using CSS selectors: `html_nodes(doc, "table td")` (or if you've a glutton for punishment, use XPath selectors with `html_nodes(doc, xpath = "//table//td")`). If you haven't heard of selectorgadget, make sure to read `vignette("selectorgadget")` to learn about it.
- Extract components with `html_name()` (the name of the tag), `html_text()` (all text inside the tag), `html_attr()` (contents of a single attribute) and `html_attrs()` (all attributes).
- (You can also use rvest with XML files: parse with `xml()` , then extract components using `xml_node()` , `xml_attr()` , `xml_attrs()` , `xml_text()` and `xml_name()` .)
- Parse tables into data frames with `html_table()` .
- Extract, modify and submit forms with `html_form()` , `set_values()` and `submit_form()` .
- Detect and repair encoding problems with `guess_encoding()` and `repair_encoding()` .
- Navigate around a website as if you're in a browser with `html_session()` , `jump_to()` , `follow_link()` , `back()` , `forward()` , `submit_form()` and so on. (This is still a work in progress, so I'd love your feedback.)

To see examples of these function in use, check out the demos.

1. 从网路读取HTML code

`read_html`/`read_xml`: 读取html/xml; 获得, 一个XML文档, HTML也将会标准化为XML

`read_xml(x, encoding="",as_html=FALSE...)`: x, 字符串, 一个链接; encoding, 指定文件默认的encoding

`library('rvest')`

`url <- https://www.imdb.com/search/title/?release_date=2018-01-01,2019-01-01`

`webpage <- read_html(url)`

2. 根据对应的CSS selector 提示, 抓取对应部分; 转换格式为文本

`html_nodes/html_node`: 使用XPath和CSS selectors从HTML文件中提取内容块. CSS selectors和[http://selectorgadget.com]连用很有用.

`html_modes(x, css, xpath)`: x, 文件, 一组节点或单个节点; css, 选择的节点; xpath, XPath 1.0 selector

`html_text/html_name/html_attrs`: 从html中提取属性, 文本和标签名称

`html_text(x, trim=FALSE)`: x, 文本/节点/节点集; trim, 是否去除首位的空格

`rank_data_html <- html_nodes(webpage, '.text-primary')`

`rank_data <- html_text(rank_data_html)`

`rank_data <- as.numeric(rank_data)`

3. 重复上两部获得对应信息(前两步获得电影序号, 该步骤重复获得电影名称)

`title_data_html <- html_nodes(webpage,".lister-item-header a") title_data <- html_text(title_data_html)`

`description_data_html <- html_nodes(webpage,".ratings-bar+ .text-muted") description_data <- html_text(description_data_html,trim=TRUE)`

`runtime_data_html <- html_nodes(webpage,".runtime") runtime_data <- html_text(runtime_data_html,trim=TRUE)`

`star_data_html <- html_nodes(webpage,".ratings-imdb-rating") star_data <- html_text(star_data_html,trim=TRUE)`

`genre_data_html <- html_nodes(webpage,".genre") genre_data <- html_text(genre_data_html) genre_data <- gsub("\n","",genre_data)`

`vote_data_html <- html_nodes(webpage,".sort-num_votes-visible span:nth-child(2)") vote_data <- html_text(vote_data_html)`

`movies_df <- data.frame(Rank=rank_data,Title=title_data,Description=description_data, Genre= genre_data,Votes=vote_data)`
