stat547-hw06-thibodeau-mylinh

I installed these new packages of the ontologyX suite here (https://cran.r-project.org/web/packages/ontologyIndex/vignettes/intro-to-ontologyX.html), which offer tools to explore ontology data.

We can use the getNamespaceExports() to see what are the functions of these packages !

```
getNamespaceExports("ontologyIndex")
```

```
##  [1] "get_term_info_content"         "get_OBO"
##  [3] "minimal_set"                   "get_descendants"
##  [5] "get_term_property"             "exclude_descendants"
##  [7] "get_term_descendancy_matrix"   "check"
##  [9] "get_ancestors"                 "get_ontology"
## [11] "intersection_with_descendants" "prune_descendants"
## [13] "propagate_relations"           "ontology_index"
## [15] "get_term_frequencies"          "get_relation_names"
```

```
getNamespaceExports("ontologySimilarity")
```

```
##  [1] "get_similarity_rank_matrix"     "lin"
##  [3] "get_term_sim_mat"               "get_term_set_to_term_sims"
##  [5] "get_profile_sims"               "get_sim_grid"
##  [7] "create_sim_index"               "get_sim"
##  [9] "sample_group_sim_from_ontology" "sample_group_sim"
## [11] "get_sim_p"                      "descendants_IC"
## [13] "resnik"                         "get_asym_sim_grid"
## [15] "get_sim_p_from_ontology"
```

First step, get the Human Disease Ontology as explained here (https://github.com/DiseaseOntology/HumanDiseaseOntology/blob/master/src/ontology/README-editors.md), I had to clone the repository (which I called HumanDiseaseOntology_git, as suggested by the github README document).

```
get_relation_names("HumanDiseaseOntology_git/src/ontology/HumanDO.obo")
```

```
## [[1]]
## [1] "is_a"
```

```
HumanDO <- get_ontology("HumanDiseaseOntology_git/src/ontology/HumanDO.obo", propagate_relationships=c("is_a", "p
art_of"))
#View(HumanDO)
```

I have downloaded a basic version of the Gene Ontology (GO) at here (http://www.geneontology.org/page/download-ontology) and I will now look into what are the relationships between the GO terms.

```
get_relation_names("GO/go-basic.obo")
```

```
## [1] "is_a"                 "regulates"            "part_of"
## [4] "negatively_regulates" "positively_regulates"
```

```
GO <- get_ontology("GO/go-basic.obo")
#View(GO)
str(GO, max.level = 1) %>% head()
```

```
## List of 6
##  $ id       : Named chr [1:47068] "GO:0000001" "GO:0000002" "GO:0000003" "GO:0000005" ...
##   ..- attr(*, "names")= chr [1:47068] "GO:0000001" "GO:0000002" "GO:0000003" "GO:0000005" ...
##  $ name     : Named chr [1:47068] "mitochondrion inheritance" "mitochondrial genome maintenance" "reproductio
n" "obsolete ribosomal chaperone activity" ...
##   ..- attr(*, "names")= chr [1:47068] "GO:0000001" "GO:0000002" "GO:0000003" "GO:0000005" ...
##  $ parents  :List of 47068
##  $ children :List of 47068
##  $ ancestors:List of 47068
##  $ obsolete : Named logi [1:47068] FALSE FALSE FALSE TRUE FALSE FALSE ...
##   ..- attr(*, "names")= chr [1:47068] "GO:0000001" "GO:0000002" "GO:0000003" "GO:0000005" ...
##  - attr(*, "class")= chr "ontology_index"
##  - attr(*, "version")= chr [1:30] "format-version: 1.2" "data-version: releases/2017-11-03" "subsetdef: goanti
slim_grouping \"Grouping classes that can be excluded\"" "subsetdef: gocheck_do_not_annotate \"Term not to be use
d for direct annotation\"" ...
```

```
## NULL
```

Interestingly, the ontologyIndex package does come with an R version of HPO (Human Phenotype Ontology) and GO (Gene Ontology), which you can load as follow.

```
data(hpo)
#View(hpo)
data(go)
#View(go)
```

Note. I will try not use the R lists of the ontologyIndex, because I am trying to learn how to read and manipulate the OBO format files directly.

Homework instructions

Pick (at least) two of the six (numbered) topics below and do one of the exercise prompts listed, or something comparable using your dataset of choice.

The two tasks I picked are the following:

1. Character data
2. Work with a list

---

1. Character data

Let's take a peak at the data

For the first task, we will be exploring some character data, and we will use ontology terms for this. This examples is modelled on Daniel Greene's work here (https://cran.r-project.org/web/packages/ontologySimilarity/vignettes/ontologySimilarity-introduction.html).

However, I think it would be useful to try and understand the data of HumanDO a bit better first. Here are some key concepts about HumanDO:

- As opposed to the GO dataset above, HumanDO only has one type of relationship and it is "is_a"
- It is a large list of 6 elements
- These 6 elements are:

1. id: specific DOID (Disease Ontology ID) identifier
2. name: specific term attached to the identifier (e.g. angiosarcoma)
3. parents: an ontology goes from general terms (parents) to more specific terms (children)
4. children: on parent can have zero children (if it is a unique term), or many children (if it is a general term which can be further divided into more specific terms)
5. ancestors: this list keep an aggregate list of all the more general terms that preceded a term (all the parents, grand-parents, great grand-parents "terms", etc.)
6. obsolete: this is a boolean list, which includes all the terms/DOID identifiers that were once in the HumanOD ontology: most of these are currently valid ("TRUE"), but some are not in use anymore ("FALSE")

Let us look at an example. Let's look at the diseases that contain the word "encephalitis":

```
head(HumanDO$name[grep(x=HumanDO$name, pattern = "encephalitis")])
```

```
##                   DOID:0050015                   DOID:0050066
##      "Rocio virus encephalitis"  "Listeria meningoencephalitis"
##                   DOID:0050118                   DOID:0050123
##        "La Crosse encephalitis"      "tuberculous encephalitis"
##                   DOID:0050126                   DOID:0050170
##     "Tahyna virus encephalitis" "Jamestown Canyon encephalitis"
```

Note. Use of grep in ontology data also from Daniel Greene's work here (https://cran.r-project.org/web/packages/ontologyIndex/vignettes/intro-to-ontologyX.html)

Let's look at the diseases that contain the word "Japanese":

```
HumanDO$name[grep(x=HumanDO$name, pattern = "Japanese")]
```

```
##             DOID:0050050              DOID:10844
## "Japanese spotted fever"  "Japanese encephalitis"
```

We note that DOID:0050050 is associated with the disease "Japanese spotted fever". Let's look at the ancestor of this term.

```
get_term_property(ontology=HumanDO, property = "ancestors", term = "DOID:0050050",as_names=TRUE)
```

```
##                                DOID:4
##                            "disease"
##                          DOID:0050117
##          "disease by infectious agent"
##                              DOID:104
##         "bacterial infectious disease"
##                          DOID:0050338
## "primary bacterial infectious disease"
##                            DOID:11104
##                         "spotted fever"
##                            DOID:0050050
##               "Japanese spotted fever"
```

Note. So this is pretty intuitive when we think about it: some terms like "disease" are very general, and when the ontology tree gains more granularity, then the addition of characteristics such as "bacterial", "infectious" and "spotted fever" lead to the creation of a specific ontology disease identifier:diagnosis (DOID:0050050 = Japanese spotted fever).

We will use the HumanDO data and set a seed.

```
set.seed(1)
```

Then, we will use the `descendants_IC()` function to calculate information content of terms based on frequency with which it is an ancestor of other terms.

```
information_content <- descendants_IC(HumanDO)
```

Then, we generate 5 random sets of 8 terms.

```
term_sets <- replicate(simplify=FALSE, n=5, expr=minimal_set(HumanDO, sample(HumanDO$id, size=8)))
term_sets
```

```
## [[1]]
## [1] "DOID:0110818" "DOID:11574"   "DOID:262"     "DOID:8224"
## [5] "DOID:0110104" "DOID:8020"    "DOID:891"     "DOID:4006"
##
## [[2]]
## [1] "DOID:3486"    "DOID:0050738" "DOID:0110151" "DOID:0080169"
## [5] "DOID:4397"    "DOID:11824"   "DOID:5601"    "DOID:14433"
##
## [[3]]
## [1] "DOID:4872"    "DOID:9869"    "DOID:11747"   "DOID:5709"
## [5] "DOID:8692"    "DOID:0110218" "DOID:3847"    "DOID:0060443"
##
## [[4]]
## [1] "DOID:0110837" "DOID:11861"   "DOID:0050194" "DOID:118"
## [5] "DOID:7455"    "DOID:10933"   "DOID:14049"   "DOID:3029"
##
## [[5]]
## [1] "DOID:1432"    "DOID:0090077" "DOID:6641"    "DOID:4112"
## [5] "DOID:5983"    "DOID:0060252" "DOID:4942"    "DOID:12380"
```

Note that the term_sets variable is a small nested list of characters items: there are 5 lists, each of which contains one list of 8 items, as exemplified here:

```
str(term_sets)
```

```
## List of 5
##  $ : chr [1:8] "DOID:0110818" "DOID:11574" "DOID:262" "DOID:8224" ...
##  $ : chr [1:8] "DOID:3486" "DOID:0050738" "DOID:0110151" "DOID:0080169" ...
##  $ : chr [1:8] "DOID:4872" "DOID:9869" "DOID:11747" "DOID:5709" ...
##  $ : chr [1:8] "DOID:0110837" "DOID:11861" "DOID:0050194" "DOID:118" ...
##  $ : chr [1:8] "DOID:1432" "DOID:0090077" "DOID:6641" "DOID:4112" ...
```

In genomics, it can be helpful to compare sets of terms and determine how much similarity is shared between datasets (here, we have 5 lists, or 5 “mini datasets”). We can use the `get_sim_grid()` function to produce a similarity matrix and verify if any dataset is highly similar to another one.

```
similarity_matrix <- get_sim_grid(ontology= HumanDO, term_sets = term_sets)
# similarity_matrix %>% kable(format = "markdown", align="c")
similarity_matrix
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.0000000 0.1692971 0.2282228 0.1094473 0.3079935
## [2,] 0.1692971 1.0000000 0.1102068 0.1301982 0.0806324
## [3,] 0.2282228 0.1102068 1.0000000 0.1452106 0.2733638
## [4,] 0.1094473 0.1301982 0.1452106 1.0000000 0.1525723
## [5,] 0.3079935 0.0806324 0.2733638 0.1525723 1.0000000
```

Note. From the top-left corner to the bottom-right corner, the similarity score is always 1 because we are comparing respectively list 1 with list 1, list 2 with list 2, etc.

Let’s manipulate some strings now

I will be completed the tasks of this R for Data Science tutorial here (http://r4ds.had.co.nz/strings.html), as suggested in the homework 6 instructions.

String basics

```
string1 <- "anemia"
string2 <- 'I am looking for the word "anemia" in a list' # Using double quotes inside simple quotes
```

Using one double quote or single quote

```
double_quote <- "\""
double_quote2 <- '"'
single_quote <- '\''
single_quote2 <- "'"
```

```
double_quote
```

```
## [1] "\""
```

```
double_quote2
```

```
## [1] "\""
```

```
single_quote
```

```
## [1] "'"
```

```
single_quote2
```

```
## [1] "'"
```

Note. In the tutorial, it seems as if the backslash bar is not printed, but in the example above, it does get printed. It seems that only the use of single quotes provides us with the expected result, so I will keep this format in the future.

Let's try to print a backslash. These all produced an error message:

- “"
- “\”
- “ ”

```
"\ '"
```

```
## [1] " '"
```

Note. This simply does not print a backslash.

As shown in block028 of the character data stat545 website here (http://stat545.com/block028_character-data.html), using the function cat instead of print allow us to print a backslash (or “escape”)

```
cat("Here is a backslash: \\ ")
```

```
## Here is a backslash: \
```

There can be some printed representation of a string that is not the same than the string itself, and using the `writeLines()` function allows us to see the raw content:

```
x <- c("one \"", "two \\")
x
```

```
## [1] "one \"" "two \\"
```

```
writeLines(x)
```

```
## one "
## two \
```

Other handy special characters are `"\n"` (newline) and `"\t"`

```
cat("We can use \n to print a new line \n\n")
```

```
## We can use
##  to print a new line
```

```
cat("While \t inserts a tab")
```

```
## While     inserts a tab
```

Some strings actually represent non-English characters in all coding platforms, for example:

```
x <- "\u00b5"
x
```

```
## [1] "μ"
```

Strings can be put into a charactor vector

```
c("anemia", "low iron", "pallor")
```

```
## [1] "anemia"   "low iron" "pallor"
```

String length

How many character in this string? Use str_length

```
str_length(c("anemia", "low iron", "pallor"))
```

```
## [1] 6 8 6
```

Note. White spaces count as characters.

Combining strings

Use `str_c()`

```
str_c("anemia", "low iron", "pallor")
```

```
## [1] "anemialow ironpallor"
```

Note. As mentioned, white spaces count as characters, so the white space in “low iron” is preserved here.

You can also specify the separator:

```
str_c("anemia", "low iron", "pallor", sep=" = ")
```

```
## [1] "anemia = low iron = pallor"
```

You can specify what the missing values “NA” can be replaced with:

```
x <- c("anemia",  NA)
str_c("--> ", x , " <--")
```

```
## [1] "--> anemia <--" NA
```

```
str_c("--> ", str_replace_na(x), " <--")
```

```
## [1] "--> anemia <--" "--> NA <--"
```

In the example, `str_c()` is vectorized, and it makes the shorter vectors the same length as the longest vector:

```
str_c("prefix-", c("a", "b", "c"), "", "-suffix")
```

```
## [1] "prefix-a-suffix" "prefix-b-suffix" "prefix-c-suffix"
```

Note. Objects of length zero (e.g. “” above) are dropped.

This can be used with an if statement.

```
cancer_type <- "breast cancer"
time_since_diagnosis <- "2 years"
day_of_diagnosis <- FALSE

str_c(
  "It has been ", time_since_diagnosis, " since your diagnosis of ", cancer_type,
  if (day_of_diagnosis) "and I am sorry to give you this bad news today"
)
```

```
## [1] "It has been 2 years since your diagnosis of breast cancer"
```

We can also collapse a vector of strings as followed:

```
str_c(c("anemia", "low iron", "pallor"), collapse=" = ")
```

```
## [1] "anemia = low iron = pallor"
```

Subsetting strings

We can extract part of a string with `str_sub()`, and specify the inclusive positions (start and end) of the substring to extract.

```
list_of_strings <- c("dominant", "recessive", "X-linked", "mitochondrial")
str_sub(list_of_strings, 1, 4)
```

```
## [1] "domi" "rece" "X-li" "mito"
```

```
str_sub(list_of_strings, -4, -1)
```

```
## [1] "nant" "sive" "nked" "rial"
```

Even if you put “too big of a range”, it will still works !

```
str_sub(list_of_strings, 1, 15)
```

```
## [1] "dominant"      "recessive"     "X-linked"      "mitochondrial"
```

```
str_sub(list_of_strings, -1, -1) <- str_to_upper(str_sub(list_of_strings, -1, -1))
list_of_strings
```

```
## [1] "dominanT"      "recessivE"     "X-linkeD"      "mitochondriaL"
```

Locales

Here are some functions to change the letter format:

```
str_to_upper("recessive") # Change to upper case
```

```
## [1] "RECESSIVE"
```

```
str_to_lower("PIK3CA") # Change to lower case
```

```
## [1] "pik3ca"
```

```
str_to_title("MTOR pathway") # Capitalizes the first letter of each word
```

```
## [1] "Mtor Pathway"
```

Locales are used to specify special characters according to the specific language. Different languages have different rules for changing case, hence the need to specify. ISO is a 639 language code, you can peak here (http://www.loc.gov/standards/iso639-2/php/English_list.php).

```
str_to_upper("i") # English
```

```
## [1] "I"
```

```
str_to_upper("i", "tr") # Turkish
```

```
## [1] "İ"
```

Base R has some functions, like order() and sort() functions, which can sort strings using the current locale. But stringr functions is more consistent and flexible as it can take a locale argument.

```
list_of_strings <- c("recessive", "X-linked", "dominant",  "mitochondrial", "anticipation")
str_sort(list_of_strings, locale = "en")
```

```
## [1] "anticipation"  "dominant"      "mitochondrial" "recessive"
## [5] "X-linked"
```

```
str_sort(list_of_strings, locale = "haw")
```

```
## [1] "anticipation"  "dominant"      "mitochondrial" "recessive"
## [5] "X-linked"
```

```
str_order(list_of_strings, locale = "en")
```

```
## [1] 5 3 4 1 2
```

```
str_order(list_of_strings, locale = "haw")
```

```
## [1] 5 3 4 1 2
```

Note. Apparently, the order can change according to the locale (language) selected, as showed here (http://r4ds.had.co.nz/strings.html), but I must have tried a dozen languages and it never changed the order of my list. Oh well, I'll still keep it in mind just in case.

Exploring some functions.

paste() and paste0()

```
paste("PIK3CA", "BRCA1", "TP53", "PTEN", "MLH1")
```

```
## [1] "PIK3CA BRCA1 TP53 PTEN MLH1"
```

```
paste0("PIK3CA", "BRCA1", "TP53", "PTEN", "MLH1")
```

```
## [1] "PIK3CABRCA1TP53PTENMLH1"
```

I found with this blog here (https://www.r-bloggers.com/difference-between-paste-and-paste0/) that the difference between paste and paste0 is that their separator is " " and "" respectively.

Get the character in the middle using str_length() and str_sub()

```
gene1 <- "BRCA1"
str_sub(gene1, ceiling(str_length(gene1)/2), ceiling(str_length(gene1)/2))
```

```
## [1] "C"
```

Difference between sep and collapse?

```
str_c("BRCA1", "PTEN", sep = " and ") # Takes the list of strings as argument
```

```
## [1] "BRCA1 and PTEN"
```

```
str_c(c("BRCA1", "PTEN"), collapse = " + ") # Takes a vector as argument
```

```
## [1] "BRCA1 + PTEN"
```

The function str_wrap() can be used to reformat strings, but I see very little application for this function in my type of research work, so I will not illustrate it here.

str_trim() trim whitespace from start and enf of string.

```
cancer_diagnoses <- str_trim(c(" breast cancer", "paraganglioma   ", "medullary thyroid cancer", "   glioblastom
a", "pheochromocytoma"))
#str(cancer_diagnoses)
```

Matching patterns with regular expressions

Find a substring to highlight with str_view()

```
str_view(cancer_diagnoses, "oma")
```

breast cancer

paraganglioma

medullary thyroid cancer

glioblastoma

pheochromocytoma

A dot (“.”) can be used to replace any unknown character.

```
str_view(cancer_diagnoses, "..oma")
```

breast cancer

paraganglioma

medullary thyroid cancer

glioblastoma

pheochromocytoma

Note. The two dots highlight the 2 characters preceeding “oma”.

You have to use the escape `\\` in order to be able to search for the actual “.” character in strings.

```
dot <- "\\."
writeLines(dot)
```

```
## \.
```

```
str_view(c("abc", "a.c", "bef"), "a\\.c")
```

abc

a.c

bef

In order to look for a backslash, you need no less than 4 of them:

```
x <- "a\\b"
writeLines(x)
```

```
## a\b
```

```
#> a\b

str_view(x, "\\\\")
```

a\b

Why each of these strings don't match a `\` : `"\"`, `"\\"`, `"\\\"` ?

- `"\"` : this doesn't work because the backslash is leading to an "escape" for the `"` and the expression is still open.
- `"\\"` : this doesn't work because it literally prints `"\\"`.
- `"\\\"` : the expression is interpreted as is still open.

Anchor

Here are some useful options as well:

- ^ to match the start of the string.
- $ to match the end of the string.

```
str_view(cancer_diagnoses, "oma$")
```

breast cancer

paraganglioma

medullary thyroid cancer

glioblastoma

pheochromocytoma

```
str_view(cancer_diagnoses, "^p")
```

breast cancer

paraganglioma

medullary thyroid cancer

glioblastoma

pheochromocytoma

```
lung_cancer <- c("Non-Small Cell Lung Carcinoma", "Small Cell Lung Cancer", "Lung Carcinoid Tumor")
str_view(lung_cancer, "^Lung$") # Exact match is not found (starts with and finishes with), so it's not highlight
ed by this function
```

Non-Small Cell Lung Carcinoma

Small Cell Lung Cancer

Lung Carcinoid Tumor

```
str_view(lung_cancer, "Lung") # Highlights all the words mathing "Lung" regardless of position of the word
```

Non-Small Cell Lung Carcinoma

Small Cell Lung Cancer

Lung Carcinoid Tumor

Replacing matched

Replace matches with new string (here, replace vowels by "*")

```
str_replace(lung_cancer, "[aeiou]", "*")
```

```
## [1] "N*n-Small Cell Lung Carcinoma" "Sm*ll Cell Lung Cancer"
## [3] "L*ng Carcinoid Tumor"
```

```
str_replace_all(lung_cancer, "[aeiou]", "*")
```

```
## [1] "N*n-Sm*ll C*ll L*ng C*rc*n*m*" "Sm*ll C*ll L*ng C*nc*r"
## [3] "L*ng C*rc*n**d T*m*r"
```

Can be used for multiple replacements:

```
copy_number <- c("2 copies of PTEN", "4 copies of GSK3B", "1 copy of TP53")
str_replace_all(copy_number, c("1" = "one", "2" = "two", "4" = "four"))
```

```
## [1] "two copies of PTEN"   "four copies of GSK3B" "one copy of TP53"
```

```
sentences <- c("Lung cancer is largely due to tobacco smoking (active and passive).", "Sun exposure increases the
 risk for melanoma.", "Acute lymphoblastic leukemia is largely a diagnosis of the pediatric age group.")
sentences %>%
  str_replace("([^ ]+) ([^ ]+) ([^ ]+)", "\\1 \\3 \\2")
```

```
## [1] "Lung is cancer largely due to tobacco smoking (active and passive)."
## [2] "Sun increases exposure the risk for melanoma."
## [3] "Acute leukemia lymphoblastic is largely a diagnosis of the pediatric age group."
```

Note. This flips the order of the 2nd and 3rd word.

Splitting

Let's explore str_split() function, which returns a list separated by a white space (as specified below)

```
sentences %>%
  str_split(" ")
```

```
## [[1]]
##  [1] "Lung"      "cancer"    "is"        "largely"   "due"
##  [6] "to"        "tobacco"   "smoking"   "(active"   "and"
## [11] "passive)."
##
## [[2]]
## [1] "Sun"       "exposure"  "increases" "the"       "risk"      "for"
## [7] "melanoma."
##
## [[3]]
##  [1] "Acute"         "lymphoblastic" "leukemia"      "is"
##  [5] "largely"       "a"             "diagnosis"     "of"
##  [9] "the"           "pediatric"     "age"           "group."
```

```
"breast|brain|lung|pancreas" %>%
  str_split("\\|") %>%
  .[[1]]
```

```
## [1] "breast"   "brain"    "lung"     "pancreas"
```

Note. Since the vector length is one, it's easier to extract the 1st element of the list.

In a matrix, simplify = TRUE can be used:

```
sentences %>%
  str_split(" ", simplify = TRUE)
```

```
##      [,1]    [,2]            [,3]        [,4]      [,5]      [,6]
## [1,] "Lung"  "cancer"        "is"        "largely" "due"     "to"
## [2,] "Sun"   "exposure"      "increases" "the"     "risk"    "for"
## [3,] "Acute" "lymphoblastic" "leukemia"  "is"      "largely" "a"
##      [,7]        [,8]      [,9]      [,10]       [,11]       [,12]
## [1,] "tobacco"   "smoking" "(active" "and"       "passive)." ""
## [2,] "melanoma." ""        ""        ""          ""          ""
## [3,] "diagnosis" "of"      "the"     "pediatric" "age"       "group."
```

Note. This puts the sentences in a “3 by 12” matrix, each cell containing 1 word (or nothing is the sentence is shorter than the longest sentence).

For example, we can isolate the word Japanese.

```
Japanese_disorders <- HumanDO$name[grep(x=HumanDO$name, pattern = "Japanese")]
Japanese_disorders %>% str_split(" ", n = 2, simplify = TRUE)
```

```
##      [,1]       [,2]
## [1,] "Japanese" "spotted fever"
## [2,] "Japanese" "encephalitis"
```

Highlights all the words or sentences using the boundary argument.

```
str_view_all(Japanese_disorders, boundary("word"))
```

Japanese spotted fever

Japanese encephalitis

```
str_view_all(Japanese_disorders, boundary("sentence"))
```

Japanese spotted fever

Japanese encephalitis

This only takes the first element of the list.

```
str_split(Japanese_disorders, boundary("word"))[[1]]
```

```
## [1] "Japanese" "spotted"  "fever"
```

Find matches

With regex() !

```
encephalitis_disorders <- HumanDO$name[grep(x=HumanDO$name, pattern = "encephalitis")]
str_view(encephalitis_disorders, regex("vir"))
```

Rocio virus encephalitis

Listeria meningoencephalitis

La Crosse encephalitis

tuberculous encephalitis

Tahyna virus encephalitis

Jamestown Canyon encephalitis

snowshoe hare encephalitis

trivittatus encephalitis

inkoo encephalitis

Kunjin encephalitis

tick-borne encephalitis

Powassan encephalitis

Colorado tick fever encephalitis

Herpes simplex virus encephalitis

Varicella-zoster virus encephalitis

Epstein-Barr virus encephalitis

Measles virus encephalitis

Cytomegalovirus encephalitis

Rubella virus encephalitis

coxsackievirus encephalitis

polioencephalitis

adenovirus encephalitis

influenza virus encephalitis

Nipah virus encephalitis

Lymphocytic choriomeningitis virus encephalitis

primary amebic meningoencephalitis

granulomatous amebic encephalitis

tertiary syphilitic encephalitis

Banna virus encephalitis

Lymphocytic choriomeningitis virus meningoencephalitis

syphilitic encephalitis

Mumps virus encephalitis

meningoencephalitis

congenital syphilitic encephalitis

Eastern equine encephalitis

Murray Valley encephalitis

Western equine encephalitis

Japanese encephalitis

St. Louis encephalitis

acute hemorrhagic leukoencephalitis

postinfectious encephalitis

post-vaccinal encephalitis

meningococcal encephalitis

West Nile encephalitis

acute necrotizing encephalitis

acute hemorrhagic encephalitis

viral encephalitis

subacute sclerosing panencephalitis

Herpes simplex virus meningoencephalitis

Venezuelan equine encephalitis

equine encephalitis

encephalitis

Note. This allows use to highligth words containing “vir”, so here “viral” and “virus”.

ignore_case = TRUE can be useful!

```
fever_disorders <- HumanDO$name[grep(x=HumanDO$name, pattern = "fever")]
str_view(fever_disorders, "rocky")
```

African tick-bite fever

Astrakhan spotted fever

Far Eastern spotted fever

Flinders Island spotted fever

Japanese spotted fever

Rickettsia parkeri spotted fever

Rocky Mountain spotted fever

Rickettsia honei spotted fever

Pontiac fever

Colorado tick fever encephalitis

Argentine hemorrhagic fever

Bolivian hemorrhagic fever

Venezuelan hemorrhagic fever

Brazilian hemorrhagic fever

Chapare hemorrhagic fever

Whitewater Arroyo hemorrhagic fever

Korean hemorrhagic fever

lujo hemorrhagic fever

yellow fever hepatitis

Alkhurma hemorrhagic fever

Rickettsia aeschlimannii spotted fever

aneruptive fever

sennetsu fever

O'nyong'nyong fever

Ross River fever

Oropouche fever

Balkan hemorrhagic fever

Zika fever

autosomal dominant familial periodic fever

Q fever

trench fever

spotted fever

hemorrhagic fever with renal syndrome

Phlebotomus fever

tickborne fever

rat-bite fever

dengue hemorrhagic fever

Crimean-Congo hemorrhagic fever

relapsing fever

louse-borne relapsing fever

tick-borne relapsing fever

Haverhill fever

typhoid fever

Rift Valley fever

uveoparotid fever

pharyngoconjunctival fever

blackwater fever

boutonneuse fever

rheumatic fever

West Nile fever

Yellow fever virus infectious disease

familial Mediterranean fever

paratyphoid fever

Ebola hemorrhagic fever

Marburg hemorrhagic fever

Colorado tick fever

ephemeral fever

African swine fever

classical swine fever

scarlet fever

Arenavirus hemorrhagic fever

Lassa fever

yellow fever

Omsk hemorrhagic fever

```
str_view(fever_disorders, regex("rocky", ignore_case = TRUE))
```

African tick-bite fever

Astrakhan spotted fever

Far Eastern spotted fever

Flinders Island spotted fever

Japanese spotted fever

Rickettsia parkeri spotted fever

Rocky Mountain spotted fever

Rickettsia honei spotted fever

Pontiac fever

Colorado tick fever encephalitis

Argentine hemorrhagic fever

Bolivian hemorrhagic fever

Venezuelan hemorrhagic fever

Brazilian hemorrhagic fever

Chapare hemorrhagic fever

Whitewater Arroyo hemorrhagic fever

Korean hemorrhagic fever

lujo hemorrhagic fever

yellow fever hepatitis

Alkhurma hemorrhagic fever

Rickettsia aeschlimannii spotted fever

aneruptive fever

sennetsu fever

O'nyong'nyong fever

Ross River fever

Oropouche fever

Balkan hemorrhagic fever

Zika fever

autosomal dominant familial periodic fever

Q fever

trench fever

spotted fever

hemorrhagic fever with renal syndrome

Phlebotomus fever

tickborne fever

```
rat-bite fever
dengue hemorrhagic fever
Crimean-Congo hemorrhagic fever
relapsing fever
louse-borne relapsing fever
tick-borne relapsing fever
Haverhill fever
typhoid fever
Rift Valley fever
uveoparotid fever
pharyngoconjunctival fever
blackwater fever
boutonneuse fever
rheumatic fever
West Nile fever
Yellow fever virus infectious disease
familial Mediterranean fever
paratyphoid fever
Ebola hemorrhagic fever
Marburg hemorrhagic fever
Colorado tick fever
ephemeral fever
African swine fever
classical swine fever
scarlet fever
Arenavirus hemorrhagic fever
Lassa fever
yellow fever
Omsk hemorrhagic fever
```

Note. In the first example, the word “Rocky” is not found because the first letter is capitalized, but using regex with ignore_case = TRUE resolves this issues.

Another useful one: multiline = TRUE allows ^ and $ to match the start and end of each line

```
x <- "BRCA1 1\nPTEN 2\nTP53 3\nBRCA2 4"
str_extract_all(x, "^BRCA")[[1]]
```

```
## [1] "BRCA"
```

```
str_extract_all(x, regex("^BRCA", multiline = TRUE))[[1]]
```

```
## [1] "BRCA" "BRCA"
```

As mentioned in the tutorial (http://r4ds.had.co.nz/strings.html), comments = TRUE allows you to use comments and white space to make complex regular expressions more understandable. Spaces are ignored, as is everything after #. To match a literal space, you’ll need to escape it: “\ ”

If we use genomic coordinates for example, with regex, you can put optional format so that the function can accept either a specific chromosomal location or a chromosomal region range.

```
genomic_position <- regex("
  (^chr)   # specify the chromosome number will be indicated
  (\\d{1}) # chromosome number
  (:)      # nomenclature
  (\\d{7}) # five numbers
  [ -]?    # optional dash (for genomic range)
  (\\d{7})?
  ", comments = TRUE)
str_match("chr1:1234567", genomic_position)
```

```
##      [,1]           [,2]  [,3] [,4] [,5]      [,6]
## [1,] "chr1:1234567" "chr" "1"  ":"  "1234567" NA
```

```
str_match("chr1:1234567-3217654", genomic_position)
```

```
##      [,1]                   [,2]  [,3] [,4] [,5]      [,6]
## [1,] "chr1:1234567-3217654" "chr" "1"  ":"  "1234567" "3217654"
```

Note. dotall = TRUE allows . to match everything, including .

```
genomic_position2 <- regex("
  (^chr)   # specify the chromosome number will be indicated
  (\\d{1}) # chromosome number
  (.)      # nomenclature
  (\\d{7}) # five numbers
  [-]?     # optional dash (for genomic range)
  (\\d{7})?
  ", comments = TRUE,
  dotall = TRUE)
str_match("chr1:1234567", genomic_position2)
```

```
##      [,1]           [,2]  [,3] [,4] [,5]      [,6]
## [1,] "chr1:1234567" "chr" "1"  ":"  "1234567" NA
```

```
str_match("chr1:1234567-3217654", genomic_position2)
```

```
##      [,1]                   [,2]  [,3] [,4] [,5]      [,6]
## [1,] "chr1:1234567-3217654" "chr" "1"  ":"  "1234567" "3217654"
```

```
str_match("chr1*1234567-3217654", genomic_position2)
```

```
##      [,1]                   [,2]  [,3] [,4] [,5]      [,6]
## [1,] "chr1*1234567-3217654" "chr" "1"  "*"  "1234567" "3217654"
```

Note. In the example above, I replace the ":" by "." and then regardless of the character at that position, the function will take it.

Three other options for "regex-like" tasks.

Option 1 - fixed(), the fast one, ignores all special regular expressions.

The microbenchmark package is able to evaluate how much rime expressions are taking to run.

```
#install.packages("microbenchmark")
library(microbenchmark)
microbenchmark::microbenchmark(
  fixed = str_detect(sentences, fixed("the")),
  regex = str_detect(sentences, "the"),
  times = 20
)
```

```
## Unit: microseconds
##   expr    min      lq     mean median      uq     max neval cld
##  fixed 31.269 34.8335 55.34975 36.165 52.2115 259.798    20   b
##  regex 17.064 19.8370 29.45770 21.411 27.4620  72.985    20  a
```

Note.1. As you can see, the fixed() function takes about twice as much time to run than regex. Note.2. It can not be used with non-English data. As there are often multiple ways of representing the same character, this can cause problem. For example, there are two ways to define “á”: either as a single character or as an “a” plus an accent:

```
a1 <- "\u00e1"
a2 <- "a\u0301"
c(a1, a2)
```

```
## [1] "á" "á"
```

```
a1 == a2
```

```
## [1] FALSE
```

Option 2 - coll()

```
str_detect(a1, fixed(a2)) # Regex does not identify that the two terms are identical, because it looks at the lit
teral form and they are encoded differently
```

```
## [1] FALSE
```

```
str_detect(a1, coll(a2)) # coll is able to identify it's the same letter
```

```
## [1] TRUE
```

Note.1. So I guess I won’t be using French with regex then ;) Note.2. coll() though compares strings using standard collation rules, which is useful for doing case insensitive matching, and it takes a locale parameter.

```
i <- c("I", "İ", "i", "ı")
i
```

```
## [1] "I" "İ" "i" "ı"
```

```
str_subset(i, coll("i", ignore_case = TRUE)) # reads the Turkish "İ" as "I"
```

```
## [1] "I" "i"
```

```
str_subset(i, coll("i", ignore_case = TRUE, locale = "tr")) # presents the Turkish "İ"
```

```
## [1] "İ" "i"
```

This allows you to identify your default locale (English Canada here).

```
stringi::stri_locale_info()
```

```
## $Language
## [1] "en"
##
## $Country
## [1] "CA"
##
## $Variant
## [1] ""
##
## $Name
## [1] "en_CA"
```

The boundary argument used in str_split can also be used in the other stringr package functions. ### Other uses of regular expressions

This function apropos() searches all objects available from the global environment, which is particularly helpful when you can’t quite remember the object/function name but you know it contains a word like “replace”.

```
apropos("replace")
```

```
## [1] "%+replace%"       "replace"          "replace_na"
## [4] "setReplaceMethod" "str_replace"      "str_replace_all"
## [7] "str_replace_na"   "theme_replace"
```

Note. I find it funny to see that the English language borrowed the term “apropos” from French. However, in the French language, this expression is actually written “à propos” and it can have two different meanings, as seen in the Collins dictionary here (https://www.collinsdictionary.com/dictionary/french-english/%C3%A0-propos): “to show presence of mind, to do the right thing” or “suitably, aptly”.

List all the files in the directory with dir()

```
dir()
```

```
## [1] "GO"
## [2] "HumanDiseaseOntology_git"
## [3] "README.md"
## [4] "scratch-space"
## [5] "stat547-hw06-thibodeau-mylinh.html"
## [6] "stat547-hw06-thibodeau-mylinh.Rmd"
```

stringi

Talking of ontology, it appears that the stringi package is actually an ancestor of stringr, which was built on top of stringi.

We can use the getNamespaceExports() to see what functions stringi has.

```
library(stringi)
getNamespaceExports("stringi")
```

```
##   [1] "stri_width"                    "stri_replace_last_charclass"
##   [3] "stri_replace_all_charclass"    "stri_sub<-"
##   [5] "stri_numbytes"                 "stri_extract_first_fixed"
##   [7] "stri_enc_isutf32be"            "stri_join_list"
##   [9] "stri_replace_first_charclass"  "stri_enc_isutf16be"
##  [11] "stri_trans_list"               "stri_extract_all_fixed"
##  [13] "%s===%"                        "stri_locate_first_boundaries"
##  [15] "stri_read_lines"               "stri_opts_regex"
##  [17] "stri_subset_fixed"             "stri_trans_nfkc"
##  [19] "stri_endswith"                 "stri_trans_nfkd"
##  [21] "stri_replace_all_fixed"        "stri_cmp_nequiv"
##  [23] "stri_match_all_regex"          "stri_replace_last_fixed"
##  [25] "stri_stats_general"            "stri_replace_all_coll"
##  [27] "stri_count_boundaries"         "stri_locale_list"
##  [29] "stri_locale_set"               "stri_enc_set"
##  [31] "stri_trim_both"                "stri_timezone_list"
##  [33] "stri_extract_first_coll"       "%stri!==%"
##  [35] "stri_enc_detect2"              "%s+%"
##  [37] "stri_rand_strings"             "stri_escape_unicode"
##  [39] "stri_pad_right"                "stri_locate_last_coll"
##  [41] "stri_trans_tolower"            "stri_extract_last_coll"
##  [43] "stri_timezone_set"             "stri_reverse"
##  [45] "stri_enc_tonative"             "stri_extract_last_boundaries"
##  [47] "stri_compare"                  "stri_write_lines"
##  [49] "stri_split_lines"              "stri_trans_isnfkc_casefold"
##  [51] "stri_unescape_unicode"         "stri_split_lines1"
##  [53] "stri_stats_latex"              "stri_extract_last_regex"
##  [55] "stri_trim"                     "stri_locate_all_words"
##  [57] "%stri<=%"                      "stri_extract_all_words"
##  [59] "stri_enc_toascii"              "stri_trim_right"
##  [61] "stri_duplicated"               "stri_rand_lipsum"
##  [63] "stri_subset_fixed<-"           "stri_unique"
##  [65] "stri_enc_list"                 "stri_enc_mark"
##  [67] "%stri+%"                       "stri_locate_last_charclass"
##  [69] "stri_datetime_add"             "stri_enc_fromutf32"
##  [71] "stri_duplicated_any"           "stri_extract_first_charclass"
##  [73] "stri_list2matrix"              "stri_replace_na"
##  [75] "stri_dup"                      "stri_trans_nfc"
##  [77] "stri_wrap"                     "stri_trans_nfd"
##  [79] "stri_pad"                      "stri_cmp_equiv"
##  [81] "%stri<%"                       "stri_startswith_fixed"
##  [83] "stri_extract_all_charclass"    "stri_trans_toupper"
##  [85] "stri_cmp_neq"                  "stri_extract_all_coll"
##  [87] "stri_datetime_now"             "stri_endswith_fixed"
##  [89] "stri_split_boundaries"         "%s<=%"
##  [91] "stri_length"                   "stri_enc_get"
##  [93] "stri_extract_first_regex"      "stri_extract_first_boundaries"
##  [95] "stri_enc_toutf32"              "stri_opts_brkiter"
##  [97] "stri_count_fixed"              "stri_count_charclass"
##  [99] "stri_locate_last_boundaries"   "stri_trans_isnfc"
## [101] "stri_timezone_info"            "stri_locate_all_regex"
## [103] "stri_trans_isnfd"              "stri_locate_first_words"
## [105] "stri_extract"                  "stri_conv"
## [107] "stri_subset_charclass<-"       "stri_trans_totitle"
## [109] "stri_locate_all_fixed"         "%stri===%"
## [111] "stri_locate_last_regex"        "stri_timezone_get"
## [113] "stri_c_list"                   "stri_endswith_coll"
## [115] "stri_encode"                   "stri_match"
## [117] "stri_endswith_charclass"       "%s<%"
## [119] "stri_datetime_parse"           "stri_match_first_regex"
## [121] "stri_detect_coll"              "stri_extract_all_boundaries"
## [123] "stri_datetime_fstr"            "stri_pad_both"
## [125] "stri_enc_isascii"              "stri_replace"
## [127] "stri_c"                        "stri_locate"
## [129] "stri_pad_left"                 "stri_locale_info"
## [131] "stri_flatten"                  "stri_trans_char"
## [133] "stri_replace_first_regex"      "stri_paste"
## [135] "stri_locate_last"              "stri_locate_last_fixed"
## [137] "stri_replace_all"              "stri_detect_charclass"
## [139] "stri_cmp_gt"                   "stri_split_fixed"
## [141] "stri_datetime_format"          "stri_count"
```

```
## [143] "stri_join"                     "stri_sort"
## [145] "stri_trans_isnfkc"             "%s!=%"
## [147] "stri_trans_isnfkd"             "stri_opts_collator"
## [149] "%stri!=%"                      "%s>%"
## [151] "stri_subset_coll<-"            "stri_replace_last"
## [153] "%s==%"                         "stri_subset"
## [155] "stri_startswith_charclass"     "stri_datetime_add<-"
## [157] "stri_detect_regex"             "stri_extract_first_words"
## [159] "%stri==%"                      "stri_split_coll"
## [161] "stri_locate_first_charclass"   "stri_isempty"
## [163] "stri_paste_list"               "stri_cmp_ge"
## [165] "%s!==%"                        "stri_trim_left"
## [167] "stri_locale_get"               "stri_order"
## [169] "stri_subset<-"                 "stri_datetime_fields"
## [171] "stri_locate_all_charclass"     "stri_enc_isutf8"
## [173] "stri_locate_first_coll"        "stri_locate_all_boundaries"
## [175] "stri_enc_detect"               "stri_locate_all"
## [177] "stri_cmp_eq"                   "stri_extract_last_words"
## [179] "stri_replace_first_fixed"      "stri_match_all"
## [181] "stri_trans_general"            "stri_replace_last_regex"
## [183] "stri_startswith"               "stri_enc_isutf32le"
## [185] "stri_cmp_lt"                   "stri_subset_regex"
## [187] "stri_match_last_regex"         "stri_subset_regex<-"
## [189] "stri_enc_toutf8"               "stri_extract_first"
## [191] "stri_enc_isutf16le"            "stri_locate_first_regex"
## [193] "stri_opts_fixed"               "stri_sub"
## [195] "stri_split_regex"              "stri_enc_info"
## [197] "stri_extract_all"              "stri_match_first"
## [199] "stri_read_raw"                 "stri_cmp"
## [201] "%stri>%"                       "stri_detect"
## [203] "stri_trans_nfkc_casefold"      "stri_rand_shuffle"
## [205] "stri_count_coll"               "stri_info"
## [207] "stri_locate_last_words"        "stri_match_last"
## [209] "stri_datetime_create"          "stri_subset_charclass"
## [211] "stri_cmp_le"                   "stri_extract_last_fixed"
## [213] "stri_replace_last_coll"        "stri_detect_fixed"
## [215] "stri_count_words"              "stri_datetime_symbols"
## [217] "stri_split_charclass"          "stri_locate_first_fixed"
## [219] "stri_extract_all_regex"        "stri_replace_all_regex"
## [221] "stri_replace_first"            "stri_extract_last"
## [223] "stri_subset_coll"              "stri_count_regex"
## [225] "stri_locate_all_coll"          "stri_replace_first_coll"
## [227] "%s>=%"                         "stri_startswith_coll"
## [229] "stri_split"                    "%stri>=%"
## [231] "stri_extract_last_charclass"   "stri_locate_first"
```

```
stri_join("BRCA1", "BRCA2")
```

```
## [1] "BRCA1BRCA2"
```

```
stri_compare("BRCA1", "BRCA2") # the only difference between these 2 words is a position -1
```

```
## [1] -1
```

```
stri_info() # Get the default settings used by the ICU library.
```

```
## $Unicode.version
## [1] "7.0"
##
## $ICU.version
## [1] "55.1"
##
## $Locale
## $Locale$Language
## [1] "en"
##
## $Locale$Country
## [1] "CA"
##
## $Locale$Variant
## [1] ""
##
## $Locale$Name
## [1] "en_CA"
##
##
## $Charset.internal
## [1] "UTF-8"  "UTF-16"
##
## $Charset.native
## $Charset.native$Name.friendly
## [1] "UTF-8"
##
## $Charset.native$Name.ICU
## [1] "UTF-8"
##
## $Charset.native$Name.UTR22
## [1] NA
##
## $Charset.native$Name.IBM
## [1] "ibm-1208"
##
## $Charset.native$Name.WINDOWS
## [1] "windows-65001"
##
## $Charset.native$Name.JAVA
## [1] "UTF-8"
##
## $Charset.native$Name.IANA
## [1] "UTF-8"
##
## $Charset.native$Name.MIME
## [1] "UTF-8"
##
## $Charset.native$ASCII.subset
## [1] TRUE
##
## $Charset.native$Unicode.1to1
## [1] NA
##
## $Charset.native$CharSize.8bit
## [1] FALSE
##
## $Charset.native$CharSize.min
## [1] 1
##
## $Charset.native$CharSize.max
## [1] 3
##
##
## $ICU.system
## [1] FALSE
```

Note. Here, ICU stands for the International Components for Unicode, which Wikipedia here (https://en.wikipedia.org/wiki/International_Components_for_Unicode) tells me are a set of opend source C/C++ and Java libraries. However, my brain always thinks about Intensive Care Unit (ICU) when I read it because of my clinical training ;)

Wow, this took a lot of time !! Of course, not all the lines have code and I tried something different, but still, reaching 600 lines for the first part of the homework, that is a a bit intense.

---

5. Work with a list