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)

Put in a dataframe?

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 32.297 35.1850 55.1257 41.6140 51.5395 292.387    20   b
##  regex 15.828 19.4055 25.4871 21.0245 26.4080  72.312    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.pdf"
## [7] "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") %>% head()
```

```
## [1] "stri_width"                  "stri_replace_last_charclass"
## [3] "stri_replace_all_charclass"  "stri_sub<-"
## [5] "stri_numbytes"               "stri_extract_first_fixed"
```

```
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

This exercise is using the GitHub GenomicDataCommons tutorial here (https://github.com/seandavi/GenomicDataCommons#filtering) which shows a step by step process to obtain some cancer genomic data.

The tasks completed are based on the STAT545/547 purrr tutorial (Simplifying data from a list of GitHub users) here (https://jennybc.github.io/purrr-tutorial/ls02_map-extraction-advanced.html), the general purrr tutorial here (https://jennybc.github.io/purrr-tutorial/index.html), the Trump tweets tutorial here (https://jennybc.github.io/purrr-tutorial/ls08_trump-tweets.html) and the class notes from the STAT545 (http://stat545.com/syllabus.html) course this Fall 2017.

DATA AND BASIC WRANGLING

From the Genomic Data Commons (GDC) website (https://gdc.cancer.gov/about-gdc):

The National Cancer Institute's (NCI's) Genomic Data Commons (GDC) is a data sharing platform that promotes precision medicine in oncology. It is not just a database or a tool; it is an expandable knowledge network supporting the import and standardization of genomic and clinical data from cancer research programs.

I installed a few bioconductor packages. References: Bioconductor packages here (https://bioconductor.org/install/#install-bioconductor-packages) and here (https://bioconductor.org/packages/release/bioc/html/GenomicDataCommons.html), and pdf information on GenomicDataCommons here (https://www.biorxiv.org/content/biorxiv/early/2017/04/04/117200.full.pdf).

```
library(magrittr)
library(devtools)
#source("https://bioconductor.org/biocLite.R")
#biocLite(c("GenomicFeatures", "AnnotationDbi", "GenomeInfoDbData"))
#biocLite("GenomicDataCommons")
library(GenomicDataCommons)
```

```
## Warning: package 'GenomicDataCommons' was built under R version 3.4.2
```

```
GenomicDataCommons::status()
```

```
## $commit
## [1] "a38d9114206f253599cfcb12e454fc10582be38d"
##
## $data_release
## [1] "Data Release 9.0 - October 24, 2017"
##
## $status
## [1] "OK"
##
## $tag
## [1] "1.10.0"
##
## $version
## [1] 1
```

```
library(GenomicDataCommons)
library(magrittr)
ge_manifest = files() %>%
    GenomicDataCommons::filter( ~ cases.project.project_id == 'TCGA-OV' &
                type == 'gene_expression') %>%
    manifest()
```

```
#library(BiocParallel)
#register(MulticoreParam())
#destdir = tempdir()
#fnames = bplapply(ge_manifest$id, gdcdata,
                  #destination_dir=destdir,
                  #BPPARAM = MulticoreParam(progressbar=TRUE))
```

Note. At this stage, I kept having error messages, for example:

```
Error: BiocParallel errors element index: 1, 2, 3, 4, 5, 6, ... first error: SSL certificate problem: Invalid certificate cha
```

- I have tries to troubleshoot this SSL certificate issue, but after several hours, I had to give up as I was getting nowhere. I kept the code chunk just in case I am able to resolve this SSL issue at a later time.

Let's just make a simpler nested data.frame with information about the patients, diagnoses, samples, etc.

```
qfiles = files() %>%
  GenomicDataCommons::filter( ~ cases.project.project_id == 'TCGA-DLBC' &
                              type == 'gene_expression' &
                              analysis.workflow_type == 'HTSeq - Counts')
manifest_df = qfiles %>% manifest()
nrow(manifest_df)
```

```
## [1] 48
```

```
str(manifest_df , max.level = 1)
```

```
## Classes 'gdc_manifest', 'tbl_df', 'tbl' and 'data.frame':    48 obs. of  5 variables:
##  $ id      : chr  "8d8b0e13-fb54-45fe-855e-718dbf3bc219" "bf9bae2b-2f0f-4250-a2bd-2d1e80aa9b0f" "e2c43df6-95a8
-4190-a488-7cf186627840" "36393056-9f92-4a9f-bf18-fe69765c8f1b" ...
##  $ filename: chr  "ce79c39f-0752-4789-a46e-dde431b886da.htseq.counts.gz" "3ed4d69f-25bc-43b5-aa05-749ec88030a
3.htseq.counts.gz" "38b4af20-2579-4efd-8657-e93c55035e24.htseq.counts.gz" "a39253da-8dc4-46c2-bd29-491a61ea4471.h
tseq.counts.gz" ...
##  $ md5     : chr  "7dd6c5e9bc20a31c1eee57e7a7c24703" "c9ea596aff0f76922e6205c016e665fd" "09695d040afb27cbc9f4c
96136c74216" "aedd5ddc4620fe5a54ac4d04f52042bd" ...
##  $ size    : int  246274 251197 246390 245569 247532 240827 245332 250856 251303 248814 ...
##  $ state   : chr  "live" "live" "live" "live" ...
##  - attr(*, "spec")=List of 2
##   ..- attr(*, "class")= chr "col_spec"
```

Creating a data query

```
pquery = projects()
presults = pquery %>% results()
```

```
# total number of files of a specific type
res = files() %>% facet(c('type','data_type')) %>% aggregations()
res$type
```

```
##                            key doc_count
## 1   annotated_somatic_mutation     63581
## 2      simple_somatic_mutation     63581
## 3                 aligned_reads     45988
## 4          copy_number_segment     44752
## 5              gene_expression     34722
## 6             mirna_expression     22976
## 7       methylation_beta_value     12359
## 8       biospecimen_supplement     11370
## 9          clinical_supplement     11211
## 10 aggregated_somatic_mutation       186
## 11     masked_somatic_mutation       132
```

Select only he gene_expression data.

```
qfiles = files() %>% filter(~ type == 'gene_expression')
# here is what the filter looks like after translation
str(get_filter(qfiles))
```

```
## List of 2
##  $ op     :Classes 'scalar', 'character'  chr "="
##  $ content:List of 2
##   ..$ field: chr "type"
##   ..$ value: chr "gene_expression"
```

```
grep('pro',available_fields('files'),value=TRUE)
```

```
##  [1] "cases.diagnoses.progression_free_survival"
##  [2] "cases.diagnoses.progression_free_survival_event"
##  [3] "cases.diagnoses.progression_or_recurrence"
##  [4] "cases.project.dbgap_accession_number"
##  [5] "cases.project.disease_type"
##  [6] "cases.project.intended_release_date"
##  [7] "cases.project.name"
##  [8] "cases.project.primary_site"
##  [9] "cases.project.program.dbgap_accession_number"
## [10] "cases.project.program.name"
## [11] "cases.project.program.program_id"
## [12] "cases.project.project_id"
## [13] "cases.project.releasable"
## [14] "cases.project.released"
## [15] "cases.project.state"
## [16] "cases.samples.days_to_sample_procurement"
## [17] "cases.samples.method_of_sample_procurement"
## [18] "cases.samples.portions.slides.number_proliferating_cells"
## [19] "cases.tissue_source_site.project"
```

The aggregation function will group data of the same type together.

```
files() %>% facet('cases.project.project_id') %>% aggregations()
```

```
## $cases.project.project_id
##            key doc_count
## 1        FM-AD     36134
## 2    TCGA-BRCA     27207
## 3    TCGA-LUAD     14804
## 4    TCGA-UCEC     13604
## 5    TCGA-LUSC     13124
## 6    TCGA-HNSC     12895
## 7     TCGA-LGG     12603
## 8    TCGA-THCA     12703
## 9      TCGA-OV     13054
## 10   TCGA-PRAD     12568
## 11   TCGA-COAD     11824
## 12   TCGA-SKCM     11265
## 13   TCGA-KIRC     12272
## 14   TCGA-STAD     10731
## 15   TCGA-BLCA     10193
## 16    TCGA-GBM      9657
## 17   TCGA-LIHC      9511
## 18   TCGA-CESC      7349
## 19   TCGA-KIRP      7368
## 20   TCGA-SARC      6282
## 21   TCGA-ESCA      4473
## 22   TCGA-PAAD      4433
## 23   TCGA-PCPG      4422
## 24   TCGA-READ      4012
## 25   TCGA-LAML      3954
## 26   TCGA-TGCT      3636
## 27  TARGET-NBL      2806
## 28   TCGA-THYM      2974
## 29  TARGET-AML      1873
## 30    TCGA-ACC      2108
## 31   TARGET-WT      1324
## 32   TCGA-MESO      2050
## 33    TCGA-UVM      1928
## 34   TCGA-KICH      1853
## 35    TCGA-UCS      1364
## 36   TCGA-CHOL      1157
## 37   TCGA-DLBC      1163
## 38   TARGET-OS         4
## 39   TARGET-RT       174
## 40 TARGET-CCSK         2
```

```
files() %>% facet('cases.project.project_id') %>% aggregations()  %>% str(max.level = 1)
```

```
## List of 1
##  $ cases.project.project_id:'data.frame':    40 obs. of  2 variables:
```

I only want o select the TCGA-DLBC subset of data.

```
qfiles = files() %>% filter( ~ cases.project.project_id == 'TCGA-DLBC' & type == 'gene_expression')
str(get_filter(qfiles))
```

```
## List of 2
##  $ op     :Classes 'scalar', 'character'  chr "and"
##  $ content:List of 2
##   ..$ :List of 2
##   .. ..$ op     :Classes 'scalar', 'character'  chr "="
##   .. ..$ content:List of 2
##   .. .. ..$ field: chr "cases.project.project_id"
##   .. .. ..$ value: chr "TCGA-DLBC"
##   ..$ :List of 2
##   .. ..$ op     :Classes 'scalar', 'character'  chr "="
##   .. ..$ content:List of 2
##   .. .. ..$ field: chr "type"
##   .. .. ..$ value: chr "gene_expression"
```

```
qfiles %>% count()
```

```
## [1] 144
```

```
qfiles %>% str(max.level = 1)
```

```
## List of 5
##  $ fields : chr [1:23] "access" "acl" "created_datetime" "data_category" ...
##  $ filters:List of 2
##  $ facets : NULL
##  $ legacy : logi FALSE
##  $ expand : NULL
##  - attr(*, "class")= chr [1:3] "gdc_files" "GDCQuery" "list"
```

```
manifest_df = qfiles %>% manifest()
head(manifest_df)
```

```
## # A tibble: 6 x 5
##                                     id
##                                  <chr>
## 1 8d8b0e13-fb54-45fe-855e-718dbf3bc219
## 2 bf9bae2b-2f0f-4250-a2bd-2d1e80aa9b0f
## 3 e2c43df6-95a8-4190-a488-7cf186627840
## 4 36393056-9f92-4a9f-bf18-fe69765c8f1b
## 5 7f073ffc-1f5c-44d0-9fa7-8f22631a16f0
## 6 a8b51e82-05e0-44fc-8da7-b59d4351409f
## # ... with 4 more variables: filename <chr>, md5 <chr>, size <int>,
## #   state <chr>
```

```
qfiles = files() %>% filter( ~ cases.project.project_id == 'TCGA-DLBC' &
                             type == 'gene_expression' &
                             analysis.workflow_type == 'HTSeq - Counts')
manifest_df = qfiles %>% manifest()
nrow(manifest_df)
```

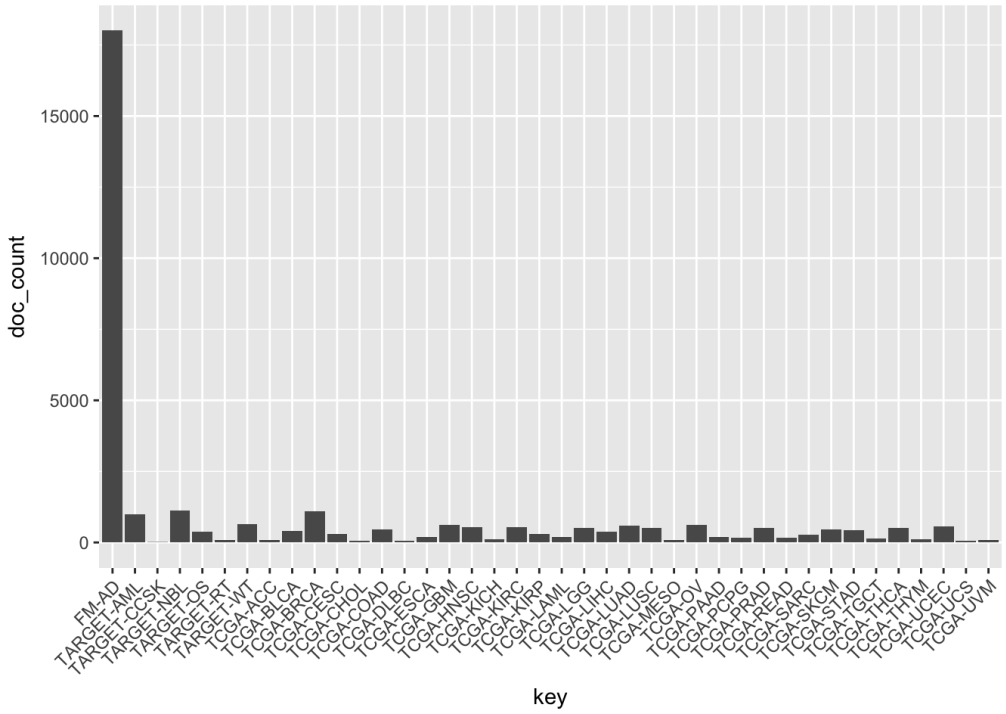
```
## [1] 48
```

```
fnames = gdcdata(manifest_df$id[1:2],progress=FALSE)
```

```
res = cases() %>% facet("project.project_id") %>% aggregations()
head(res)
```

```
## $project.project_id
##            key doc_count
## 1        FM-AD     18004
## 2   TARGET-NBL      1127
## 3    TCGA-BRCA      1098
## 4   TARGET-AML       988
## 5    TARGET-WT       652
## 6     TCGA-GBM       617
## 7      TCGA-OV       608
## 8    TCGA-LUAD       585
## 9    TCGA-UCEC       560
## 10   TCGA-KIRC       537
## 11   TCGA-HNSC       528
## 12    TCGA-LGG       516
## 13   TCGA-THCA       507
## 14   TCGA-LUSC       504
## 15   TCGA-PRAD       500
## 16   TCGA-SKCM       470
## 17   TCGA-COAD       461
## 18   TCGA-STAD       443
## 19   TCGA-BLCA       412
## 20   TARGET-OS       381
## 21   TCGA-LIHC       377
## 22   TCGA-CESC       307
## 23   TCGA-KIRP       291
## 24   TCGA-SARC       261
## 25   TCGA-LAML       200
## 26   TCGA-ESCA       185
## 27   TCGA-PAAD       185
## 28   TCGA-PCPG       179
## 29   TCGA-READ       172
## 30   TCGA-TGCT       150
## 31   TCGA-THYM       124
## 32   TCGA-KICH       113
## 33    TCGA-ACC        92
## 34   TCGA-MESO        87
## 35    TCGA-UVM        80
## 36   TARGET-RT        75
## 37   TCGA-DLBC        58
## 38    TCGA-UCS        57
## 39   TCGA-CHOL        51
## 40 TARGET-CCSK        13
```

```
library(ggplot2)
ggplot(res$project.project_id,aes(x = key, y = doc_count)) +
    geom_bar(stat='identity') +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Purrr tutorial exercises

Part 1. Create a recursive list with multiple levels (analogous to gh_users)

In this section, I will create 3 nested recursive (d2, d4, d6) containing data information of 3 TCGA datasets (DLBC, GBM, LUAD) which all have the same format, and then I will make a list of these 3 lists called “list_d”.

```
d1 = cases() %>% GenomicDataCommons::filter(~ project.project_id=='TCGA-DLBC') %>%
    GenomicDataCommons::select(c(default_fields(cases()),'samples.sample_type')) %>%
    response_all()
d2 = d1 %>% results()
str(d1[1],list.len=1)
```

```
## List of 1
##  $ results:'data.frame': 58 obs. of  19 variables:
##   ..$ updated_datetime     : chr [1:58] "2017-03-04T16:39:19.244769-06:00" "2017-03-04T16:39:19.244769-06:00"
"2017-03-04T16:39:19.244769-06:00" "2017-03-04T16:39:19.244769-06:00" ...
##   .. [list output truncated]
```

```
head(ids(d1))
```

```
## [1] "3622cf29-600f-4410-84d4-a9afeb41c475"
## [2] "31bbad4e-3789-42ec-9faa-1cb86970f723"
## [3] "c1c06604-5ae2-4a53-b9c0-eb210d38e3f0"
## [4] "f855dad1-6ffc-493e-ba6c-970874bc9210"
## [5] "29aff186-c321-4ff9-b81b-105e27e620ff"
## [6] "4263949c-f962-40dd-9998-02ad3fba4537"
```

```
str(d1, max.level = 1)
```

```
## List of 4
##  $ results     :'data.frame':    58 obs. of  19 variables:
##  $ query       :List of 5
##   ..- attr(*, "class")= chr [1:3] "gdc_cases" "GDCQuery" "list"
##  $ pages       :List of 7
##  $ aggregations: list()
##  - attr(*, "class")= chr [1:3] "GDCcasesResponse" "GDCResponse" "list"
```

```
#View(d1)
#View(d2)
str(d2, max.level = 1)
```

```
## List of 19
##  $ updated_datetime     : chr [1:58] "2017-03-04T16:39:19.244769-06:00" "2017-03-04T16:39:19.244769-06:00" "20
17-03-04T16:39:19.244769-06:00" "2017-03-04T16:39:19.244769-06:00" ...
##  $ submitter_analyte_ids:List of 58
##  $ analyte_ids          :List of 58
##  $ submitter_id         : chr [1:58] "TCGA-G8-6914" "TCGA-G8-6907" "TCGA-GR-A4D6" "TCGA-GR-A4D5" ...
##  $ case_id              : chr [1:58] "3622cf29-600f-4410-84d4-a9afeb41c475" "31bbad4e-3789-42ec-9faa-1cb86970f
723" "c1c06604-5ae2-4a53-b9c0-eb210d38e3f0" "f855dad1-6ffc-493e-ba6c-970874bc9210" ...
##  $ id                   : chr [1:58] "3622cf29-600f-4410-84d4-a9afeb41c475" "31bbad4e-3789-42ec-9faa-1cb86970f
723" "c1c06604-5ae2-4a53-b9c0-eb210d38e3f0" "f855dad1-6ffc-493e-ba6c-970874bc9210" ...
##  $ disease_type         : chr [1:58] "Lymphoid Neoplasm Diffuse Large B-cell Lymphoma" "Lymphoid Neoplasm Diff
use Large B-cell Lymphoma" "Lymphoid Neoplasm Diffuse Large B-cell Lymphoma" "Lymphoid Neoplasm Diffuse Large B-c
ell Lymphoma" ...
##  $ sample_ids           :List of 58
##  $ portion_ids          :List of 58
##  $ submitter_portion_ids:List of 58
##  $ created_datetime     : logi [1:58] NA NA NA NA NA NA ...
##  $ slide_ids            :List of 58
##  $ state                : chr [1:58] "live" "live" "live" "live" ...
##  $ aliquot_ids          :List of 58
##  $ primary_site         : chr [1:58] "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" ...
##  $ samples              :List of 58
##  $ submitter_aliquot_ids:List of 58
##  $ submitter_sample_ids :List of 58
##  $ submitter_slide_ids  :List of 58
##  - attr(*, "row.names")= int [1:58] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "class")= chr [1:3] "GDCcasesResults" "GDCResults" "list"
```

```
typeof(d2)
```

```
## [1] "list"
```

```
str(d2, list.len = 1)
```

```
## List of 19
##  $ updated_datetime     : chr [1:58] "2017-03-04T16:39:19.244769-06:00" "2017-03-04T16:39:19.244769-06:00" "20
17-03-04T16:39:19.244769-06:00" "2017-03-04T16:39:19.244769-06:00" ...
##   [list output truncated]
##  - attr(*, "row.names")= int [1:58] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "class")= chr [1:3] "GDCcasesResults" "GDCResults" "list"
```

I will make another nested list d4.

```
d3 = cases() %>% GenomicDataCommons::filter(~ project.project_id=='TCGA-GBM') %>%
    GenomicDataCommons::select(c(default_fields(cases()),'samples.sample_type')) %>%
    response_all()
d4 = d3 %>% results()
typeof(d4)
```

```
## [1] "list"
```

```
d5 = cases() %>% GenomicDataCommons::filter(~ project.project_id=='TCGA-LUAD') %>%
    GenomicDataCommons::select(c(default_fields(cases()),'samples.sample_type')) %>%
    response_all()
d6 = d5 %>% results()
typeof(d6)
```

```
## [1] "list"
```

```
list_d <- list(d2, d4, d6)
typeof(list_d)
```

```
## [1] "list"
```

```
#View(list_d)
```

Part 2. Name and position shortcuts

For simplicity, let's start by looking at d2 only

```
str(d2[[1]], list.len = 1) # This is the first list in this nested data.frame, and it the values are from "update
d_datetime"
```

```
##  chr [1:58] "2017-03-04T16:39:19.244769-06:00" ...
```

```
map(d2, 1)
```

```
## $updated_datetime
## [1] "2017-03-04T16:39:19.244769-06:00"
##
## $submitter_analyte_ids
## [1] "TCGA-G8-6914-01A-11R" "TCGA-G8-6914-14A-01D" "TCGA-G8-6914-14A-01W"
## [4] "TCGA-G8-6914-01A-11W" "TCGA-G8-6914-01A-11D"
##
## $analyte_ids
## [1] "9cbced97-9fff-4612-8129-fb72ad85633d"
## [2] "b793a5fc-0002-4710-af92-0322e43dade6"
## [3] "fbb9365b-8023-4da8-a7ec-13e720b7a095"
## [4] "89cb105d-12e3-444d-8d70-dd833688da54"
## [5] "2248b334-0ffa-4f55-af13-48e6b44c5b88"
##
## $submitter_id
## [1] "TCGA-G8-6914"
##
## $case_id
## [1] "3622cf29-600f-4410-84d4-a9afeb41c475"
##
## $id
## [1] "3622cf29-600f-4410-84d4-a9afeb41c475"
##
## $disease_type
## [1] "Lymphoid Neoplasm Diffuse Large B-cell Lymphoma"
##
## $sample_ids
## [1] "717bdd50-8a66-4652-b2e3-d8bb5738b4c2"
## [2] "a72c401f-9712-4aff-842a-dd9ae500b4cd"
##
## $portion_ids
## [1] "46301a26-2459-423e-9f8d-f0470032911e"
## [2] "ed9f2b1a-05bc-442a-8ed5-bbee278e2817"
## [3] "e176261a-7212-428d-9db9-99126092e0e5"
##
## $submitter_portion_ids
## [1] "TCGA-G8-6914-14A-01"         "TCGA-G8-6914-01A-11"
## [3] "TCGA-G8-6914-01A-21-A45K-20"
##
## $created_datetime
## [1] NA
##
## $slide_ids
## [1] "d02f3d16-ec64-416e-86d7-5413603d6a39"
## [2] "5b13e1ea-ee78-40c7-b162-91591b39bb54"
##
## $state
## [1] "live"
##
## $aliquot_ids
##  [1] "5ac8fa6a-2d61-44e4-b0ef-0d9fcbf7c53a"
##  [2] "8f8eb4b4-64cf-4199-8d89-0bcb1357b9c9"
##  [3] "6f2b10b9-ceb2-45c8-96b8-4a49f4d62d4c"
##  [4] "d4c6b197-60f8-468b-a4ba-3c219d451171"
##  [5] "366ac237-58d3-45ae-bf82-9f85fc562612"
##  [6] "5431207a-4e82-408c-91b6-f5437cb7f5b5"
##  [7] "23beea99-6187-4600-ab43-3d7452f6a26c"
##  [8] "16ea266a-69ef-42a9-81c7-affe59adb1ce"
##  [9] "cbfb4849-20fa-4429-8943-dcf106bb37c6"
## [10] "d15202c0-099f-46c9-99c0-cc6bdf3e82b1"
## [11] "1f8c5cb0-4040-49b8-b51d-da8cf0529a6a"
##
## $primary_site
## [1] "Lymph Nodes"
##
## $samples
##          sample_type
## 1 Bone Marrow Normal
## 2      Primary Tumor
##
## $submitter_aliquot_ids
```

```
##  [1] "TCGA-G8-6914-01A-11R-2212-13" "TCGA-G8-6914-01A-11W-2233-10"
##  [3] "TCGA-G8-6914-14A-01W-2233-10" "TCGA-G8-6914-01A-11D-2210-10"
##  [5] "TCGA-G8-6914-14A-01D-2208-26" "TCGA-G8-6914-14A-01D-2210-10"
##  [7] "TCGA-G8-6914-01A-11D-2208-26" "TCGA-G8-6914-01A-11R-2213-07"
##  [9] "TCGA-G8-6914-14A-01D-2209-01" "TCGA-G8-6914-01A-11D-2209-01"
## [11] "TCGA-G8-6914-01A-11D-2211-05"
##
## $submitter_sample_ids
## [1] "TCGA-G8-6914-14A" "TCGA-G8-6914-01A"
##
## $submitter_slide_ids
## [1] "TCGA-G8-6914-01A-01-TS1" "TCGA-G8-6914-01A-01-BS1"
```

Note. This provides the first element of each list in the d2 data.frame, but since there are multiple levels to the data.frame, it

For example, if we were to manually pull out the first element of “primary_site” and “submitter_aliquot_ids”, these functions illustrated below would be equivalent.

```
map(d2["primary_site"], 1)
```

```
## $primary_site
## [1] "Lymph Nodes"
```

```
d2[["primary_site"]][[1]]
```

```
## [1] "Lymph Nodes"
```

```
map(d2["submitter_aliquot_ids"], 1)
```

```
## $submitter_aliquot_ids
##  [1] "TCGA-G8-6914-01A-11R-2212-13" "TCGA-G8-6914-01A-11W-2233-10"
##  [3] "TCGA-G8-6914-14A-01W-2233-10" "TCGA-G8-6914-01A-11D-2210-10"
##  [5] "TCGA-G8-6914-14A-01D-2208-26" "TCGA-G8-6914-14A-01D-2210-10"
##  [7] "TCGA-G8-6914-01A-11D-2208-26" "TCGA-G8-6914-01A-11R-2213-07"
##  [9] "TCGA-G8-6914-14A-01D-2209-01" "TCGA-G8-6914-01A-11D-2209-01"
## [11] "TCGA-G8-6914-01A-11D-2211-05"
```

```
d2[["submitter_aliquot_ids"]][[1]]
```

```
##  [1] "TCGA-G8-6914-01A-11R-2212-13" "TCGA-G8-6914-01A-11W-2233-10"
##  [3] "TCGA-G8-6914-14A-01W-2233-10" "TCGA-G8-6914-01A-11D-2210-10"
##  [5] "TCGA-G8-6914-14A-01D-2208-26" "TCGA-G8-6914-14A-01D-2210-10"
##  [7] "TCGA-G8-6914-01A-11D-2208-26" "TCGA-G8-6914-01A-11R-2213-07"
##  [9] "TCGA-G8-6914-14A-01D-2209-01" "TCGA-G8-6914-01A-11D-2209-01"
## [11] "TCGA-G8-6914-01A-11D-2211-05"
```

Let’s move on to the recursive list_d

Let’s look at the first level of our nested list_d.

```
str(list_d, max.level = 1)
```

```
## List of 3
##  $ :List of 19
##   ..- attr(*, "row.names")= int [1:58] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "class")= chr [1:3] "GDCcasesResults" "GDCResults" "list"
##  $ :List of 19
##   ..- attr(*, "row.names")= int [1:617] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "class")= chr [1:3] "GDCcasesResults" "GDCResults" "list"
##  $ :List of 19
##   ..- attr(*, "row.names")= int [1:585] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "class")= chr [1:3] "GDCcasesResults" "GDCResults" "list"
```

```
str(list_d[[3]], list.len = 1) # Look at the first level (updated_datetime) of the 3rd list in list_d, which is t
he LUAD dataset.
```

```
## List of 19
##  $ updated_datetime     : chr [1:585] "2017-03-04T16:39:19.244769-06:00" "2017-03-04T16:39:19.244769-06:00" "2
017-03-09T09:44:12.300985-06:00" "2017-03-04T16:39:19.244769-06:00" ...
##   [list output truncated]
##  - attr(*, "row.names")= int [1:585] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "class")= chr [1:3] "GDCcasesResults" "GDCResults" "list"
```

```
#list_d[[1]][c("samples")]
map(list_d[["updated_time"]][[1]], 1)
```

```
## list()
```

```
list_d[["submitter_aliquot_ids"]][[1]]
```

```
## NULL
```

We can extract multiple values of the LUAD (3rd list of list_d) using map

```
str(list_d[[1]][c("state", "primary_site")], max.level =1) # extract the state and primary site of the first list
 (TCGA-DLBC)
```

```
## List of 2
##  $ state       : chr [1:58] "live" "live" "live" "live" ...
##  $ primary_site: chr [1:58] "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" ...
```

```
x <- map(list_d, `[`, c("state", "primary_site")) # extrate the state and primary site for all lists
str(x[1:2]) # shows the 1st and 2nd list attribute
```

```
## List of 2
##  $ :List of 2
##   ..$ state       : chr [1:58] "live" "live" "live" "live" ...
##   ..$ primary_site: chr [1:58] "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" ...
##  $ :List of 2
##   ..$ state       : chr [1:617] "live" "live" "live" "live" ...
##   ..$ primary_site: chr [1:617] "Brain" "Brain" "Brain" "Brain" ...
```

The extract function of magrittr can also be used, which provides the same result.

```
x <- map(list_d, magrittr::extract, c("state", "primary_site"))
str(x[1:2])
```

```
## List of 2
##  $ :List of 2
##   ..$ state       : chr [1:58] "live" "live" "live" "live" ...
##   ..$ primary_site: chr [1:58] "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" ...
##  $ :List of 2
##   ..$ state       : chr [1:617] "live" "live" "live" "live" ...
##   ..$ primary_site: chr [1:617] "Brain" "Brain" "Brain" "Brain" ...
```

Exercise. Use your list inspection skills to determine the position of the elements named “state” and “primary_site”. Map [over the lists, requesting elements by position instead of name. “state” and “primary_site” are at position 13 and 15 respectively in each list.

```
str(list_d[[1]][c(13, 15)], max.level =1) # only for TCGA-DLBC
```

```
## List of 2
##  $ state       : chr [1:58] "live" "live" "live" "live" ...
##  $ primary_site: chr [1:58] "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" ...
```

```
x <- map(list_d, `[`, c(13, 15)) # for all 3 TCGA lists
str(x[1:3])
```

```
## List of 3
##  $ :List of 2
##   ..$ state       : chr [1:58] "live" "live" "live" "live" ...
##   ..$ primary_site: chr [1:58] "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" ...
##  $ :List of 2
##   ..$ state       : chr [1:617] "live" "live" "live" "live" ...
##   ..$ primary_site: chr [1:617] "Brain" "Brain" "Brain" "Brain" ...
##  $ :List of 2
##   ..$ state       : chr [1:585] "live" "live" "live" "live" ...
##   ..$ primary_site: chr [1:585] "Lung" "Lung" "Lung" "Lung" ...
```

We can also use piping with these functions !

```
list_d %>%
  map(`[`, c(13, 15)) %>%
  str()
```

```
## List of 3
##  $ :List of 2
##   ..$ state       : chr [1:58] "live" "live" "live" "live" ...
##   ..$ primary_site: chr [1:58] "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" "Lymph Nodes" ...
##  $ :List of 2
##   ..$ state       : chr [1:617] "live" "live" "live" "live" ...
##   ..$ primary_site: chr [1:617] "Brain" "Brain" "Brain" "Brain" ...
##  $ :List of 2
##   ..$ state       : chr [1:585] "live" "live" "live" "live" ...
##   ..$ primary_site: chr [1:585] "Lung" "Lung" "Lung" "Lung" ...
```

Part 3. Type-specific map

At this stage, it's worth exploring if the lists only have character data, or other types of data (e.g. numeric). We can take a quick peak:

```
sapply(list_d[[1]], class)
```

```
##      updated_datetime submitter_analyte_ids           analyte_ids
##           "character"                "list"                "list"
##          submitter_id               case_id                    id
##           "character"           "character"           "character"
##          disease_type            sample_ids           portion_ids
##           "character"                "list"                "list"
## submitter_portion_ids      created_datetime             slide_ids
##                "list"             "logical"                "list"
##                 state           aliquot_ids          primary_site
##           "character"                "list"           "character"
##               samples submitter_aliquot_ids  submitter_sample_ids
##                "list"                "list"                "list"
##   submitter_slide_ids
##                "list"
```

Note. It seems like at least the top levels are characters, but it is good to keep in mind that if we use specific map function. I checked if map_chr works the same here, but it received and error message (`Error: Result 1 is not a length 1 atomic vector`) because not only the list_d contains character, but it also contains vector, and therefore, using the general map() funciton is more appropriate here.

```
list_d %>%
  map_chr(`[`, c(13, 15)) %>%
  str()
```

Part 4. Data frame output

Let's stack up some of these lists on top of each other with map_df().

```
map_df(list_d, `[`, c(5, 13, 15))  # here I am selection the case_id, state and primary_site
```

```
## # A tibble: 1,260 x 3
##                                 case_id state primary_site
##                                   <chr> <chr>        <chr>
##  1 3622cf29-600f-4410-84d4-a9afeb41c475  live  Lymph Nodes
##  2 31bbad4e-3789-42ec-9faa-1cb86970f723  live  Lymph Nodes
##  3 c1c06604-5ae2-4a53-b9c0-eb210d38e3f0  live  Lymph Nodes
##  4 f855dad1-6ffc-493e-ba6c-970874bc9210  live  Lymph Nodes
##  5 29aff186-c321-4ff9-b81b-105e27e620ff  live  Lymph Nodes
##  6 4263949c-f962-40dd-9998-02ad3fba4537  live  Lymph Nodes
##  7 e6365b38-bc44-400c-b4aa-18ce8ff5bfce  live  Lymph Nodes
##  8 58e66976-4507-4552-ac53-83a49a142dde  live  Lymph Nodes
##  9 eda9496e-be80-4a13-bf06-89f0cc9e937f  live  Lymph Nodes
## 10 ea54dbad-1b23-41cc-9378-d4002a8fca51  live  Lymph Nodes
## # ... with 1,250 more rows
```

I tried to create the same stacked up list with the explicit function, but since my elements (e.g. “case_id”) are lists, I was only able to produce a data.frame with nested lists.

```
library(dplyr)
library(tibble)
df_list_d <- list_d %>% {
  tibble(
      case_id = map(., "case_id"),
      state = map(., "state"),
      primary_site = map(., "primary_site"))
}
class(df_list_d)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

Part 5. “Repositories for each user”

So in this exercise, we are suppose to switch from gh_users to gh_repos, the latter having multiple nested lists (at least 4 levels from what I see).

I will keep the same nested object (called list_d) because although my data is less layered and smaller, there is one item (called “samples”) which has a 4th level of modest size, with each sample id containing a list of 2 items: what type of sample was used for normal DNA and for cancer DNA studies.

Let’s extract some specific data values:

Task: submitter_analyte_ids is the *2nd* item in each list. For each list, retrieve the *1st* sample listed in the *3rd* submitter_analyte_ids.

```
list_d %>%
  map(c(2, 3, 1))
```

```
## [[1]]
## [1] "TCGA-GR-A4D6-10A-01W"
##
## [[2]]
## [1] "TCGA-02-0321-01A-01R"
##
## [[3]]
## [1] "TCGA-95-7043-01A-11H"
```

Part 6. List inside a data frame

As mentioned previously, we do have a data frame with lists inside: df_list_d.

However, it would be nice to have the TCGA cancer type (DLBC, GBM, LUAD) also in that data frame. Interestingly enough, the name of the TCGA dataset is not stored anywhere in the data, so we will simply select the primary_site of cancer then.

```
(primary_site_cancer <- map(list_d, c(15, 1, 1)))
```

```
## [[1]]
## [1] "Lymph Nodes"
##
## [[2]]
## [1] "Brain"
##
## [[3]]
## [1] "Lung"
```

```
(cancer_df_list_d <- list_d %>%
    set_names(primary_site_cancer) %>%
    enframe("primary_site_cancer", "list_d"))
```

```
## # A tibble: 3 x 2
##   primary_site_cancer               list_d
##                 <chr>               <list>
## 1         Lymph Nodes <S3: GDCcasesResults>
## 2               Brain <S3: GDCcasesResults>
## 3                Lung <S3: GDCcasesResults>
```

You might have noticed the presence of *S3:* preceding the list. There is actually 3 levels of information here.

```
sapply(list_d, class)
```

```
##      [,1]              [,2]              [,3]
## [1,] "GDCcasesResults" "GDCcasesResults" "GDCcasesResults"
## [2,] "GDCResults"      "GDCResults"      "GDCResults"
## [3,] "list"            "list"            "list"
```

This is actually refering to a more advanced object in R, which I will not dive into here, but if you would like more information, please click here (http://adv-r.had.co.nz/S3.html).

How do we create “one row’s worth” of data for one cancer dataset? How do we do that for all lists for a single cancer dataset?

Let’s start with a simpler example. We will select the 3rd cancer type (LUAD = Lung adenocarcinoma) and only one element of each list submitter_analyte_ids. Then we will create a 4 rows tibble with each row representing the values corresponding to the first element of the list submitter_analyte_ids.

```
one_cancer <- cancer_df_list_d$list_d[[3]]
View(one_cancer)
one_submitter_analyte_ids <- one_cancer$submitter_analyte_ids[[1]][1:3]
one_submitter_analyte_ids
```

```
## [1] "TCGA-97-A4M5-01A-11D" "TCGA-97-A4M5-10A-01W" "TCGA-97-A4M5-01A-11H"
```

```
#one_cancer$submitter_analyte_ids <- one_cancer$submitter_analyte_ids[[2]][1:3]
View(one_cancer)
str(one_cancer, max.level=1)
```

```
## List of 19
##  $ updated_datetime     : chr [1:585] "2017-03-04T16:39:19.244769-06:00" "2017-03-04T16:39:19.244769-06:00" "2
017-03-09T09:44:12.300985-06:00" "2017-03-04T16:39:19.244769-06:00" ...
##  $ submitter_analyte_ids:List of 585
##  $ analyte_ids          :List of 585
##  $ submitter_id         : chr [1:585] "TCGA-97-A4M5" "TCGA-44-2657" "TCGA-95-7043" "TCGA-44-A47B" ...
##  $ case_id              : chr [1:585] "5fe77d4a-a8a5-4c90-8ff2-9c3bbbb309ef" "f40301ba-831e-4afd-9ce8-5f3c1a05
ff7e" "c650b1ff-8a4c-4ee9-b7c1-268c28c83827" "967d6548-5a84-4b7e-bc3f-2e522859fce6" ...
##  $ id                   : chr [1:585] "5fe77d4a-a8a5-4c90-8ff2-9c3bbbb309ef" "f40301ba-831e-4afd-9ce8-5f3c1a05
ff7e" "c650b1ff-8a4c-4ee9-b7c1-268c28c83827" "967d6548-5a84-4b7e-bc3f-2e522859fce6" ...
##  $ disease_type         : chr [1:585] "Lung Adenocarcinoma" "Lung Adenocarcinoma" "Lung Adenocarcinoma" "Lung
Adenocarcinoma" ...
##  $ sample_ids           :List of 585
##  $ portion_ids          :List of 585
##  $ submitter_portion_ids:List of 585
##  $ created_datetime     : logi [1:585] NA NA NA NA NA NA ...
##  $ slide_ids            :List of 585
##  $ state                : chr [1:585] "live" "live" "live" "live" ...
##  $ aliquot_ids          :List of 585
##  $ primary_site         : chr [1:585] "Lung" "Lung" "Lung" "Lung" ...
##  $ samples              :List of 585
##  $ submitter_aliquot_ids:List of 585
##  $ submitter_sample_ids :List of 585
##  $ submitter_slide_ids  :List of 585
##  - attr(*, "row.names")= int [1:585] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "class")= chr [1:3] "GDCcasesResults" "GDCResults" "list"
```

```
str(one_submitter_analyte_ids, max.level = 1, list.len = 1)
```

```
##  chr [1:3] "TCGA-97-A4M5-01A-11D" "TCGA-97-A4M5-10A-01W" ...
```

```
one_submitter_analyte_ids[c(1,3,4)] # We present exactly 3 submitter_analyte and the corresponding TCGA sample id
entifiers
```

```
## [1] "TCGA-97-A4M5-01A-11D" "TCGA-97-A4M5-01A-11H" NA
```

```
map_df(one_cancer, `[`, c(1,3,4))
```

```
## # A tibble: 3 x 19
##                   updated_datetime submitter_analyte_ids analyte_ids
##                              <chr>               <list>      <list>
## 1 2017-03-04T16:39:19.244769-06:00            <chr [6]>   <chr [6]>
## 2 2017-03-09T09:44:12.300985-06:00            <chr [6]>   <chr [6]>
## 3 2017-03-04T16:39:19.244769-06:00            <chr [6]>   <chr [6]>
## # ... with 16 more variables: submitter_id <chr>, case_id <chr>, id <chr>,
## #   disease_type <chr>, sample_ids <list>, portion_ids <list>,
## #   submitter_portion_ids <list>, created_datetime <lgl>,
## #   slide_ids <list>, state <chr>, aliquot_ids <list>, primary_site <chr>,
## #   samples <list>, submitter_aliquot_ids <list>,
## #   submitter_sample_ids <list>, submitter_slide_ids <list>
```

We can scale it up for all 3 cancer dataset (DLBC, GBM and LUAD), and see that the rows represent one single case.

```
d3 <- cancer_df_list_d %>%
  mutate(cancer_info = list_d %>%
           map(. %>% map_df(`[`, c(1,3,4))))
d3$cancer_info[c(1,2,3)]
```

```
## [[1]]
## # A tibble: 3 x 19
##                   updated_datetime submitter_analyte_ids analyte_ids
##                              <chr>                <list>      <list>
## 1 2017-03-04T16:39:19.244769-06:00             <chr [5]>   <chr [5]>
## 2 2017-03-04T16:39:19.244769-06:00             <chr [5]>   <chr [5]>
## 3 2017-03-04T16:39:19.244769-06:00             <chr [5]>   <chr [5]>
## # ... with 16 more variables: submitter_id <chr>, case_id <chr>, id <chr>,
## #   disease_type <chr>, sample_ids <list>, portion_ids <list>,
## #   submitter_portion_ids <list>, created_datetime <lgl>,
## #   slide_ids <list>, state <chr>, aliquot_ids <list>, primary_site <chr>,
## #   samples <list>, submitter_aliquot_ids <list>,
## #   submitter_sample_ids <list>, submitter_slide_ids <list>
##
## [[2]]
## # A tibble: 3 x 19
##                   updated_datetime submitter_analyte_ids analyte_ids
##                              <chr>                <list>      <list>
## 1 2017-03-04T16:39:19.244769-06:00             <chr [6]>   <chr [6]>
## 2 2017-03-04T16:39:19.244769-06:00             <chr [3]>   <chr [3]>
## 3 2017-03-04T16:39:19.244769-06:00             <chr [6]>   <chr [6]>
## # ... with 16 more variables: submitter_id <chr>, case_id <chr>, id <chr>,
## #   disease_type <chr>, sample_ids <list>, portion_ids <list>,
## #   submitter_portion_ids <list>, created_datetime <lgl>,
## #   slide_ids <list>, state <chr>, aliquot_ids <list>, primary_site <chr>,
## #   samples <list>, submitter_aliquot_ids <list>,
## #   submitter_sample_ids <list>, submitter_slide_ids <list>
##
## [[3]]
## # A tibble: 3 x 19
##                   updated_datetime submitter_analyte_ids analyte_ids
##                              <chr>                <list>      <list>
## 1 2017-03-04T16:39:19.244769-06:00             <chr [6]>   <chr [6]>
## 2 2017-03-09T09:44:12.300985-06:00             <chr [6]>   <chr [6]>
## 3 2017-03-04T16:39:19.244769-06:00             <chr [6]>   <chr [6]>
## # ... with 16 more variables: submitter_id <chr>, case_id <chr>, id <chr>,
## #   disease_type <chr>, sample_ids <list>, portion_ids <list>,
## #   submitter_portion_ids <list>, created_datetime <lgl>,
## #   slide_ids <list>, state <chr>, aliquot_ids <list>, primary_site <chr>,
## #   samples <list>, submitter_aliquot_ids <list>,
## #   submitter_sample_ids <list>, submitter_slide_ids <list>
```

THE END - Finally ;)

I have done everything I could (and more) to make this homework both relevant for my learning and to fulfill the requirements of the course, but this has been an extremly difficult task. I hope you will understand that as this is my 14th year of postsecondary schooling and at this point in my educational path, I am trying to focus on learning things that have direct applications to my research.