1 R Programming

List

- `[[idx]]`: get element in a list
- `str(ls)`: get **str**ucture of a list (similar to summary)
- `saveRDS` and `loadRDS`

Recycling Rule

- shorter vectors are recycled until they match the length of the longest vector
- the length of the longest vector must be a multiple of the shorter vector in arithmetic operations!

Useful functions

- `sample(x, size, replace, prob)`
  - `size`: length of output vector
  - `replace`: if `TRUE`, then sampling is with replacement
  - `prob`: a vector of probability weights
- `any(duplicated(vec))`: returns true or false if there are any duplicated elements in a vector
- `rep(x, times, length.out)`
- `table()`
- `args(func)`: list the arguments of a function
- `seq(from, to, by, length)`
- `paste(v1, v2, sep)`: concatenate vectors after converting them to characters
  - `sep`: separator between elements of `v1` and `v2`
  - The recycling rule applies when `length(v1) != length(v2)`
- apply function family: apply function to each row (1) or column (2)
  - `apply(X, margin, func, ...)`
    - Note that `X` must be a **matrix** or **df** in `apply`
  - `sapply` returns a vector or a matrix, **input must be 1 dimensional!**
  - `lapply` returns a list, useful when the output of the function may not be all of the same length/-type, **input must be 1 dimensional!**
  - `replicate(n, func)`: replicate anonymous function n number of times (especially useful for random number generations)

Function debugging

- `cat("...")`: used to print statements
- `browser()`: debugging with breakpoint

Important classes

Strings

- Start by importing `tidyverse` and `stringr`
- Library functions
  - `str_length`: returns vector of string lengths
  - `str_c(..., sep)`: concatenate strings with optional separator
  - `str_sub(string, start, end)`: returns vector of substrings
- Regular expressions (`str_view()` to test out regex), *Tidyverse Article*
  - to match an **a** at the beginning of a string

    `str_view(x, "^a")`
  - to match an **a** at the end of a string

    `str_view(x, "a$")`
  - to match an **a** or **e** at the end of a string

    `str_view(x, "[ae]$")`
  - to match a string of 3 chars with **a** in the middle

    `str_view(x, ".a.")`
- `str_detect(vec, regex)`: returns a boolean vector
  - | : means or

    `str_detect(street_names, "Jurong|Boon Lay")`
  - + : means modifier (pattern detected 1 or more times)
  - (): to group stuff
  - `\\w`: any word
  - `[0-9]`: can be 0 to 9
  - `\\d`: any number
    - `\\d{3,6}` to search for digits repeating between 3 and 6 times
  - **[IMPT]** `?about_search_regex` for help
  - **[IMPT]** `?base::regex` :help for regex from R base package; `[:punct:]`, `[:digit:]`, `[:space:]`
- `str_extract(vec, regex)`: returns a vector of strings, particularly helpful for `".a."` regex

```r
# To find the number of eggs given a sentence
str_extract(sent, "[0-9]+(?= eggs)")
# ?= is a look behind operator
# ?<= is a look ahead operator

```

- `str_trim`: to trim trailing whitespaces
- `str_split`
- `str_replace`

```r
# to remove duplicate words
str_replace(sent_type, "\\b(\\w+)\\b \\1", "\\1")
```

  Note that `\\b` means word boundary and `\\1` means group boundary 1
- `str_match`

[IMPT] USE `vignette('stringr')` and `vignette('regular-expre`
for help

- `devtools::install_github("gadenbuie/regexplain")` to install regexplain GUI, need to install `devtools` library first
- Also Tools → Addins → Browse Addins.. → regexplain (cheatsheet/GUI)

Factors

`factor(vec, levels=c(...))`: convert vec to factors with fixes levels
`unique(vec)`: returns a vector with unique values

Date

- **[IMPT]** ?strftime for help page
- `as.Date(x, format)`: convert string x to Date object
  e.g. `as.Date("2014/02/22", "%Y/%m/%d")`
- `months(d)`: what month of the year is the date in?
- `weekdays(d)`: what day of the week is the date on?
- `Sys.Date()`
- `cut(x, breaks, labels)`: usually used to group dates that fall into a month/week/quarter
  - `breaks`: numeric vector/string (`"month"`, `"week"`)
  - `labels`: if TRUE, return a label vector
- `seq(d,d+365,by="1 week" or "1 quarter")`

Basic Plotting

`plot()`

- pch: abbr. for plotting character

```
# show all pch characters
example(pch)
```

- col:

```
# show all preset colours
colours()
# set custom colour, alpha is
  transparency
col <- rgb(..., alpha=?)
```

- cex: abbr. for character expansion
- bty: change box borders
- **[IMPT]** ?par shows all parameters for `plot()`
- use `points()` or `lines()` to add more stuff to an existing plot
  - `segments(x_)`

`barplot()`

`hist()`

- `freq`: makes the y-axis a proportion of all the total shit (count/total), not total count using integer

2 R Markdown (RMD)

- `.yaml` header

```
title:"..."
output:
  html-document:
  toc: true #table of content
  toc_float: true # floating TOC
 at the left side of the window
    collapsed: true
    smooth_scroll: true
  toc_depth: 2
  number_sections:true/false
date: `r format(Sys.time(), "%d %
 B %Y")`
params:
    country: Indonesia

```

  - how to reference?? ⇒ I want die liao `r params$country`
  - Referencing is important as it allows more control over the report, don't need to manually change the name of every variable if we want something else
- R Setup **[IMPT]** , will apply settings globally

~~~
```{r setup, include=FALSE}
knitr::opts_chunk$set(fig.align='
  center', echo=TRUE)
```
~~~

- Use `r var` to insert inline code and ask R to run it
- Figure
  - `include=FALSE/TRUE`: to include the output or not
  - `fig.width, fig.height, fig.dim = c(w,h), out.width="XX%"`
  - `fig.align='left'/'centre'`
  - `fig.cap` for captions
- Bulleted list: just indent and use '-'
- Dsiplay table: use `kable(df, col.names=c(...))`
  - Important parameters: `caption, align="ccc"` or `"lll"` for text alignment inside boxes

Code Chunk Settings

- `include=FALSE` doesn't print the code
- `echo=FALSE` usually for plots, don't include the actual code but just runs it
- `eval=FALSE` code chunk is not run/evaluated
- `collapse=TRUE` combines text output and source code in single block
- `message=FALSE`
- `warning=FALSE`
- `error=TRUE` will continue to knit the file even when there are errors and will include error messages in the file

3 Importing Data

[IMPT] use `read.delim` or `readLines` if none is working

CSV Files

`read.csv()`: main arguments:
- `file`: filename/path
- `skip`: skip lines?
- `header`: default is `TRUE`
- `row.names`
- `stringsAsFactors`
- `na.strings`: what are the `NA` values
- `colClasses`: what classes are the columns (in terms of class names vector)

Procedure when dealing with CSV:
- `apply(salaries, 2, function(x) sum(is.na(x)))` **[IMPT]** (check if any column has missing values)

- if `read.csv` doesn't work, can try `readLines` and `str_split` to split commas

Excel Files

- import `readxl`, data is in the form of a tibble
- `read_excel(path, sheet=?)`: sheet parameter can be string or integer
- `sheet_names(path)`: to retrieve sheet names

JSON Files

- import `jsonlite`
- `fromJSON(txt)`: takes up text/string object as an argument
- `readLines(path)`: returns a string **[IMPT]** line break will count as another element of a vector
- `prettify()`
- **[IMPT]** How to convert list to data frame?
  1. create a function `ls_to_df` which returns `data.frame` given an element of a list
  2. `lapply` the list to return a list of dataframes
  3. use `do.call` to combine the individual dataframes into one single dataframe

```
df_row_list <- lapply(list, ls_to_df)
# combine repeatedly
do.call(rbind, df_row_list)
```

- Some thoughts **[IMPT]** Are there missing data for any observation?? if yes then remove

OOP in R

S3 classes

- `methods`: to search for available methods
- `summary`

```
studentBio <- list(studentName = "Harry Potter", studentAge = 19, studentContact="London")
class(studentBio) <- "StudentInfo"

# how to assign method
contact <- function(object) {
  UseMethod("contact")
}
contact.StudentInfo <- function(object) {
  cat("Your contact is", object$studentContact, "\n")
}
# can just call contact(studentBio) without .StudentInfo
```

S4 classes

```
# How to set Class with slots
setClass("employee", slots=list(name="character", id="numeric", contact="character"))

# Constructor
obj <- new("employee",name="Steven", id=1002, contact="West Avenue")
```

[IMPT] How to add method?

```
setMethod("show",
signature(object="employee"),
definition=function(object) {
  # do stuff
})
```

[IMPT] Tips for dealing with S4 data

- `isS4(obj)`: check if obj is S4
- `slotNames(obj)` list all the attributes/slots
- `methods(class="????")`: to list out all the methods
- `vignette("class")`: for documentation

RC classes