1 R Programming

List

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

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)
  - `tapply()`: used to apply function and then group them into a table using grouping index
  - `mapply(func, arg1, arg2, arg3, ...)`: like `sapply` but takes multiple vectors containing arguments to `func`
  - `vapply()`: similar to `sapply` and `lapply` but we specify the output of operation on each element
- `rev()`: reverses elements in a data structure
- `sort()`: sort elements

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
- **[IMPT] Important packages**
  - `lubridate`
  - `zoo`
  - `xts`
- `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()`
- `Sys.time()`: class is POSIXct
- `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

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

- col:

```r
# 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()`

- horiz=TRUE flip y and x axes
- las (under ?par)
- `par()`: **[IMPT]** lists all the default parameters for plots (mar, mfrow etc.)
- How to set graphical param?

```r
# 1 row 2 columns plot
opar <- par(mfrow=c(1,2))
# plot some stuff
par(opar) # to set it back to
   default

```

`hist()`

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

2 Stringr

(to convert numeric to string) Fixed vs scientific format

- Scientific: `1.989e+30` to denote 10^30
- `format(x, scientific=TRUE)` to format number to string by specifying digit numbers etc.
  [IMPT] `digits=` will format the smallest number so that it only has the specified significant digit, and other numbers in the vector follows

```r
format(c(0.0011, 0.011, 1),
 digits=1)
> [1] "0.001" "0.011" "1.000"

```

- `formatC(x, format="f" OR "e" or "g")`
  f stands for fixed, e for scientific, and g for scientific if it saves space

Stringr functions

- `str_c`: concatenate like paste
- `str_length`: find length
- `str_sub`
- `str_detect`: returns boolean vectors
- `str_subset`:
- `str_count`
- `str_split`: n= returns maximum number of n elements, simplify= returns a matrix
  - **[IMPT]** `type=boundary("sentence")`
- `str_match`: returns a matrix with the capture or () regex
- `str_to_upper()`: returns a vec with all uppercase elements
- `str_to_lower()`
- `regex(expr, ignore_case = TRUE)`: tells regex to ignore case

Rebus package

- `install.packages("rebus")` ⇒ `library(rebus)`
- rebus syntax can be used for stringr pattern instead of regex

```r
   pattern = START %R% "a"
   # strings that start with "a"
   # same as regex "^a"
   # END is also possible
   # %R% is read as 'then'
```

- `ANY_CHAR`
- WRD: word, SPC: Space

```r
 # to capture word ending in ING
 one_or_more(WRD) %R% "ING"
```

```
# equals to \w+ING

```

- `or(p1, p2)`: kinda like | in regex
  - `or1(vec)`: pass vec as alternatives instead of arguments
- `char_class("Aa")`: kinda like `"[Aa]"` in regex
- `negated_char_class("aiueoAIUEO")`: self-explanatory
- `optional()`: ? in regex
- `zero_or_more()`: * in regex
- `one_or_more()`: + in regex
- `repeated()`: {m,n} in regex
- `exactly()`: matches exact string
- `capture(pattern)`: group parts of pattern together, which is () in regex format
  *use `REF1`, `REF2`, `REF3` to refer to the capture group (exact match) which is \1, \2 and so on in regular regex

Stringi functions

- `stri_isempty()`: returns boolean

Miscellaneous

- `strftime(date, format)`: string from time object
- `as.POSIXct(date_string, format)`: convert string to Date time
- **Base R String Functions**
  - `grepl(pattern = , x = )`: basically `str_detect`
  - `grep(pattern = , x = )`: basically `str_which`
  - `sub(pattern, replacement, x)`: basically `str_replace`
  - `gsub(pattern, replacement, x)`: basically `str_replace_all`

3 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

4 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()`
- `RfromJSON()??????`
- **[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

4.1 OOP in R

[IMPT] Main purpose: call function the same way (with similar syntax but different behaviour for each class) *e.g.* `plot` works differently for timeseries and vectors **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 `methods(generic.function="plot")`: to list out all the classes a method can be applied to
- `vignette("class")`: for documentation

RC classes

5 Databases

How to connect?

- Install the requisite package on R
- Authenticate to the database server
- Query/Extract the data
- Analyse the data
- Close the connection

5.1 MongoDB

Steps to connect

- **[IMPT]** MongoDB Tutorial Docs
- Code to connect

```
library(mongolite)
library(jsonlite)
# eXXXXX:pwd
credentials <- paste0(readLines("
    mongo_user_pwd.txt", warn=FALSE)
    , collapse=":")
connection_string <- paste0("
    mongodb://",credentials, "
    @rshiny.nus.edu.sg:2717/test")
con2 <- mongo(verbose=TRUE,
    collection="restaurants", url=
    connection_string)
con2$count()

```

Query: Note that for MongoDB query has to be made with JSON object

```
  q1 <- toJSON(list(name="Wendy'S"),
   auto_unbox=TRUE)
  # {"name": "Wendy'S"} # MongoDB takes
    JSON as argument
  q1_out <- con2$find(query=q1, fields=
   '{"borough":1, "cuisine":1}')
```

- `fields=`: only shows the data that are specified as 1 (select only relevant columns and remove those with 0)
- `auto_unbox`: convert arrayed arguments to normal arguments
- **[IMPT]** **Indexed table**: faster to find query results through indexed columns

```
   # How to find indexed columns
   con2$index();
```

- [IMPT] **Paginated Queries**: iterate over the query by batch (especially for large datasets) e.g. download the data by 10% batch
  - To handle error, use try

```r
x <- try(expression);
# let's say x throws an error
if (inherits(x, "try-error")) {
  do stuff
}
```

  - **Systematic sample**: extract 1 row from each batch to see the structure of the data and stuff
- Usually RC style objects are returned
- Remember to close connection

```r
rm(con2)
```

5.2 Data from Web

5.2.1 Download File from Link

- how to download

```r
imda_url <- "https://data.gov.sg/
    dataset/02c1f624-489f-40ad-8fdd
    -5e66e46b2722/download"
return_val <- download.file(imda_
    url, "../data/imda_data.zip")
con <- unz("../data/imda_data.zip",
     "wage-02-size2-annual.csv")
wages_data <- read.csv(con, header=
    TRUE)
```

- download.file(), mode="wb" for Windows
- file.path():
- unz: to unzip

5.2.2 Developer API

- Normal browser $\underset{response}{\overset{request}{\longleftrightarrow}}$ Web server
- request data from server that is continuously running
- [IMPT] Usually for Real-time data
- how to get data?

```r
library(httr)
set_config(verbose())
url <- "https://api.data.gov.sg/v1/
    transport/taxi-availability"
taxi_avail <- GET(url, query=list(
    date_time="2022-08-01T09:00:00")
    )

taxi_data <- content(taxi_avail)

```

Procedure for working with APIs

- Check the Documentation for
  - URL
  - Parameters
  - What it returns
- Check status code (200, 400 etc.)
- Content

5.2.3 Web Scraping With R

- [IMPT] Flukeout for CSS
- [IMPT] Selector Gadget for HTML

Procedure

- Import rvest and xml2

```r
  rbloggers_page <- read_html("
  https://www.r-bloggers.com/")
  nodes <- html_nodes(rbloggers_
  page, "#wppp-3 a")
```

- html_text(): extract text
- html_table(): extract table
- html_structure()

5.3 SQL Databases

Different kinds of SQL:

- MySQL: RMySQL
- PostgresSQL: RPostgresSQL
- Oracle Database: ROracle

```r
install.packages("RMySQL")
library(DBI)
```

How to connect

```r
con <- dbConnect(RMySQL::MySQL(), #
    Construct SQL Driver
                 dbname="company",
                 host = ...
                 port = ..., user= ...,
  password=...)
```

Useful Functions:

- List table names

```r
dbListTables(cons)
```

- Read Table

```r
dbReadTable(con, "employees")
```

- Disconnect

```r
dbDisconnect(con)
```

- Subset

```r
subset(employees,
       subset = started_at > "
    2012-09-01"
       select = col_names)
```

- Subset using SQL Query (More efficient)

```r
dbGetQuery(con, "SELECT name FROM
    employees WHERE ... ")
```

Internal working: (fetching by chunks)

```r
res <- dbSendQuery(con, "query")
while(!dbHasCompleted(res)) {
  chunk <- dbFetch(res, n=2)
  print(chunk)
}
dbDisconnect(res)
```

5.3.1 SQL Queries

- `INNER JOIN`: combine tables
- `CHAR_LENGTH()`

6 Data Manipulation

verb(df/tibble, ...)

- `filter`:

```r
jan1 <- filter(flights, month ==
  1, day == 1)
# or operator
filter(flights, month==11 | month
  ==12)
filter(flights, month %in% c
  (11,12))
```

  - `between(v, val1, val2)`: check if `v` is between the 2 values
  - **[IMPT]** Sometimes a row has `NA` values, and we can include the row to alter the data later using `is.na(x)`
  - How to drop NA values?

```r
df %>% filter(!is.na(col))
```

- `mutate`: create new variables

```r
mutate(flights_sml, air_time_mins=
    air_time/60, .before=...)
```

  - **[IMPT]** `lead()`/`lag()`: allow us to compute running differences / find when a value has changed

```r
# compute running differences
x - lag(x)
# find when a value has changed
x != lag(x)
```

  - **[IMPT]** `cumsum()`
  - **[IMPT]** `cummean()`
  - **[IMPT]** `rank()`: `min_rank()`, `min_rank(desc(x))`, `dense_rank`
  - `col = NULL`: delete a column when doing mutate
- `select`: pick variables (**columns**) by their names

```r
# select by column
select(flights, year, month, day)
# select inclusive columns
select(flights, year:day)
select(flights, !(year:day))
```

  - **[IMPT]** ?select for more operators
  - **[IMPT]** `select(df, where(func))`: `where` will return T/F and only select columns with specified properties (character? numeric?)
- `arrange`: reorder rows

```r
arrange(flights, desc(arr_delay))
```

- `summarise`: collapses many values to a smaller set of summary values
  - Will only return columns that we asked for!
  - similar to `mutate`
  - Use group_by to achieve good results

```r
by_day2 <- group_by(flights, year,
    month, day, origin)
summarise(by_day2, delay= mean(dep_
    delay,na.rm=TRUE), .group="drop"
    )
# .groups drop will drop the groups
     attribute(not grouped anumore)
```

- group_by: splits dataset by values in variable
  - will modify how mutate and filter works
  - Operations take place within the groups

```r
by_day <- group_by(flights, year,
    month, day)
```

  - `n()`: how many obervations in each group
  - `count()`

Other useful functions

- `slice_head()`: similar to `head`
- `slice_max()`: extract max specified values
- `slice_sample()`
- **[IMPT]** `Hmisc::describe()`: more intuitive
- **[IMPT]** `first(dest, order_by=dep_time)`: returns value in a column sorted by another column can only be used inside mutate or summarise
- **[IMPT]** `last()`
- **[IMPT]** `nth()`
- `?n()`: only work in grouped summarise or mutate: number of elements in each group
- `n_distinct`
- `add_tally`: like mutate: add group attributes to original df, useful when need to compare individual data to group data in each row

Miscellaneous

- `across()` apply same functions across a set of columns (something like `apply`)
  can also apply multiple functions (use list to list down the functions!)
- `rowwise()`: group by row and apply functions by row
- `c_across(x:z)`: apply c to the specified columns

6.1 Tidy Data

Ordering variables

- **Fixed variables**: those that describe the experimental design / known in advance
- **Measured variables**: what we actually measure in the study

7 Interesting stuff

- Can lookup location through zipcode