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
- `duplicated()`: very useful in deleting second duplicated value
- `case_when()`: more powerful than if-else
- `cut_interval()`: **[IMPT]** cut a numeric vector into closed/half-open intervals (see tutorial 6)

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-expressions') 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 $10^{3}0$
- 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

```
# 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

```r
 # 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 $\xleftrightarrow[response]{request}$ 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)

```
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()

5.4 Other Databases

- SAS (Statistical Analysis Software): used for Business Analytics and Medicine
- STATA (Statistical Data): used for economics: labelled data

```
ontime$airline <- as.character(as_factor(ontime$airline))
```

- SPSS (Software Package for Social Sciences): for FASS

```
library(haven)
library(foreign)

read.dta(file,
        convert.factors = TRUE,
        convert.dates = TRUE,
        missing.type = FALSE) # convert to NA
```

6 Data Manipulation

verb(df/tibble, ...)

- filter:

```
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?

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

- mutate: create new variables

```
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

```
# 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

```
# 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

```
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

```
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

```
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

[IMPT] `vignette("tidy-data")`, `vignette("pivot")`

Definitions

- **Variable**: Contains all values that measure the same underlying attribute (*e.g.* height, temperature, duration)
- **Observation**: contains all values measured on the same unit (*e.g.* a person, a day) across attributes
- **Fixed variables**: those that describe the experimental design / known in advance
- **Measured variables**: what we actually measure in the study

Tidy Data?

- Each variable forms a column
- each observation forms a row
- Each type of observational unit forms a table

6.1.1 pivot_longer

- Column names from the original data go to the year column in the new data
- Column values from the original data go to the cases column in the new data

```
table4a %>%
  pivot_longer(!country, names_to ="
   year", values_to="cases")
```

6.1.2 pivot_wider

- Column names in the reshaped data come from the type column in the original data
- Column values in the reshaped data come from the count column in the original data
- `id_cols`: identifies observational unit (group)
- `names_sep`: separates the last n characters of column name

```
table2 %>%
  pivot_wider(id_cols = country:year,
   names_from="type", values_from ="
   count")
```

6.1.3 separate

- pulls apart one column into multiple columns, by splitting wherever a separator character appears.
- Need to convert again by specifying `convert=TRUE`

```
separate(table3, rate, into=c("cases",
   "population"), convert = TRUE)
```

6.1.4 unite

- combines multiple columns into one using a separator character

6.2 Relational Data

- **primary key**: uniquely identifies an observation in its own table. For example, in the `planes` table, `tailnum` is a primary key
- **foreign key**: uniquely identifies an observation in another table. For example, `planes$tailnum` appears in the `flights` table, where it identifies a unique plane
- Sometimes, the best identifier for an observation is still not unique, so best to double-check if they are indeed unique

```
# test if they are unique
table %>% count(column) %>%
      fiter(n>1)
```

Some operations

- **Mutating joins**: add variables to data frame from matching observations
- **Filtering joins**: filter observations from one data frame based on whether or not they match an observation in the other table (same as mutate join then filter based on new variable?)
- **Set operations**: treat observations as if they were set elements

6.2.1 Joins

- `inner_join`: keeps observation only if they keys exist in both columns
- outer join: keeps observation that appear in at least on of the tables
  - `left_join`: keeps all observation in x
  - `right_join`: keeps all observation in y
  - `full_join`: keeps all observations in x and all in y

[IMPT] What happens if there are duplicates?

- in x: observations in y will be duplicated
- in y: same
- in both x and y: cartesian product (all possible matches will be created)

6.2.2 Filtering Joins

- `semi_join`: keeps all observations in x that have a match in y
- `anti_join`: drops all observations in x that have a match in y (useful for checking mismatches)

6.2.3 [IMPT] Rough Guide

1. Identify the primary keys in each table
2. Check that none of the variables in the primary key are missing
3. Check that foreign keys match primary keys in another table

7 Principles of Visualization

Some references:

- Guides - flowing data

What makes a good graph:

- Show the data
- Induce the viewer to think about the substance rather than about methodology, graphic design, the software used
- avoid distorting what the data have to say
- present many numbers in a small space
- make large data sets coherent
- encourage the eye to compare different pieces of data
- reveal the data at several different levels of detail, from a broad overview to the fine structure
- serve a reasonably clear purpose: description, exploration, tabulation, or decoration
- be closely integrated with the statistical and verbal descriptions of a dataset

Principles of Graphical Excellence

- well-designed presentation of interesting data
- consists of complex ideas communicated with clarity, precision, and efficiency
- is that which gives the viewer the greatest number of ideas in the shortest time with the least ink in the smallest space
- is nearly always multivariate
- requires telling the truth about the data

Biases

- patternicity bias: see pattern alot of times
- storytelling desire: explain data according to our story
- confirmation bias

7.1 What makes a bad graph

- Inconsistent basis of comparison
- Design variation
- Dubious integrity (mistakes!)
- Unnecessary chart: a chart should give insight that you didn't expect to see!
- Never adjust for inflation
- Start from 0 for bar charts

7.2 A Theory of Data Graphics

- Data-ink: strip down the chart to the very bare minimum

7.3 Own Notes

Some pointers

- Aggregation of data might distort result (John Snow Cholera example)
- Labelling/Titles/Annotations
- Colours: good to represent magnitude but cannot be used to compare the numbers *e.g.* how big is red compared to yellow?
- Exploratory data analytics (EDA): after seeing the graph what is the next graph you wanna make?
- Ordering
- Sankey chart: internship, breakdown of budgets
- Smallest effective difference: use smallest difference in colours
  - Use different hues to distinguish different groups
  - Use different intensity but same hue in same group

7.4 Takeaways from Tutorials

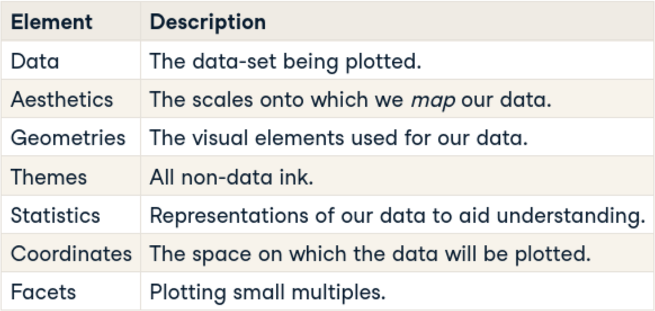
7.4.1 Tutorial 8

- Can plot confidence interval to show confidence in predicting

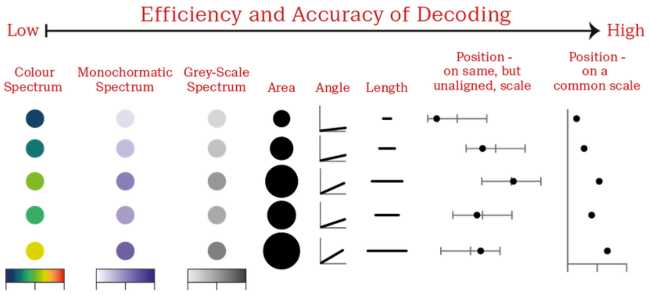
```
geom_errorbar(aes(x=..., ymin=...,
   ymax=...))

```

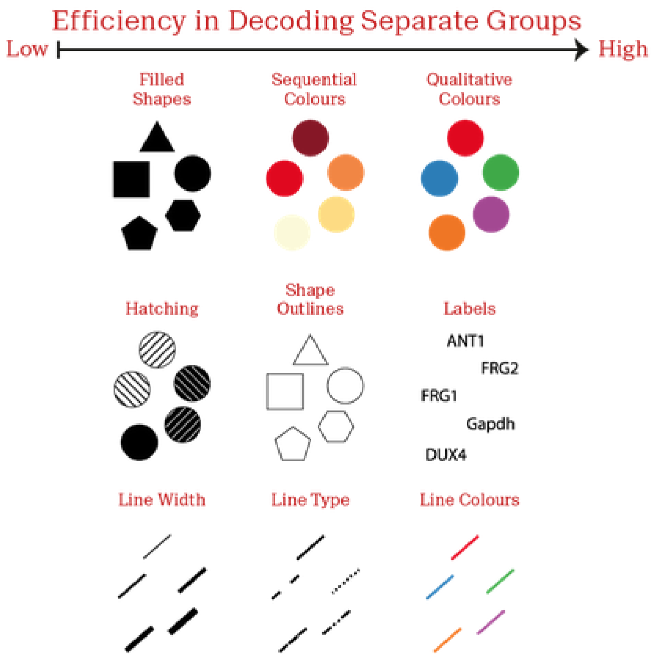
- can plot one prediction on top of another (show tutorial 8 question 2)
- y-axis is usually for dependent, x for independent variables

8 Data Visualization



Element	Description
Data	The data-set being plotted.
Aesthetics	The scales onto which we *map* our data.
Geometries	The visual elements used for our data.
Themes	All non-data ink.
Statistics	Representations of our data to aid understanding.
Coordinates	The space on which the data will be plotted.
Facets	Plotting small multiples.


Efficiency and Accuracy of Decoding
Low
High
Colour Spectrum
Monochormatic Spectrum
Grey-Scale Spectrum
Area
Angle
Length
Position - on same, but unaligned, scale
Position - on a common scale



[IMPT] `vignette("ggplot2-specs")` for help

```
ggplot(data = <DATA>) +
    <GEOM_FUNCTION>(mapping = aes(<
  MAPPINGS>)) + ...
```

- Aesthetics: `aes(x=..., y=...)`, anything that can change according to the value of the data
  - `size`: to determine size of dots
  - `color`: to colour by group
  - `alpha`:
  - `shape`
- Label: `labs()`
  - `title=`
  - `x=, y=`
- **Types of plot**:
  - `geom_point()`: scatter plot
  - `geom_line()`: line plot
  - `geom_col()`: barplot
  - `geom_histogram()`: histogram
  - `geom_boxplot()`: boxplot
- **Scales**:
  - `scale_<AES>_*`: where * is either `discrete` or `continuous` or `manunal`
  - `scale_x_log10()`: log10 scale
    - Alternatively can use `coord_trans(x="log10")`
  - **[IMPT]** `scale_fill_discrete(name = ..., labels=...)`
    - `name is to change legend title`
    - `Labels is to change legend labels`
- `coord_cartesian`: magnifying glass to expand some areas
- `xlim(), ylim()`: to increase/decrease plotting canvas
  - Alternatively can use `scale_x_continuous`
  - Alternatively can also use `coord_cartesian(xlim=c(...))`
- **Aspect ratios**
  - `coord_fixed()` to set ratio `ratio=...`

- **Faceting**:
  - `facet_wrap( ...)`
  - `facet_grid(var1  var2, labeller=...)`
    - `labeller` is for changing the title of each plot
    - `fct_relevel` is to reorder
    - `space=...` and `scales` is to adjust free scales
    - `margins=TRUE`: include all points to be plotted in the margins
- `expand_limits(y=...)`: expand graphing limits
- `coord_flip()`
- `theme()`
- `labs()`: for labels

8.1 Scatter plot

General pointers

- Is there a general trend/correlation?
  - Is it linear?
- Who are the ones that *deviate the most* from the trend?
- Add more variables (using colour) to investigate the trend
- Are there duplicates? check for overplotting?

Interesting pointers

- If we want to change aes globally, must put the variables outside `aes(...)` and inside `geom_point()`
- `scale_<AESTHETIC>_manual`
- Discrete colouring vs Continuous colouring? can use `as.factor()`
- to change colour

```
# recommended to use a named vector
cols <- c("8" = "red", "4" = "blue"
    , "6" = "darkgreen", "10" = "
    orange")
# to change names in legend labels
labs <- c("8" = "Eight", "4" = "
    Four", "6" = "Six", "10" = "Ten"
    )
scale_colour_mannual(values=cols,
    labels=labs)
```

- **Jittering**: points are not plotted at the precise location, but it is useful to solve overplotting

```
geom_point(mapping=..., position="
    jitter")
geom_point(mapping=..., position=
    position_jitter(width=0.5,
    height=0))
```

- **Nudge position**:

8.2 Histogram

General pointers

- Is it symmetric? or skewed?
- Is it bimodal or unimodal??
  - If bimodal then might be good to separate the two groups
- Outliers (interesting data far away from the norm)

- geom_histogram()
  - bins=...
  - binwidth=...
  - boundary=... to set hard x limit for the histogram
  - col=...: to control the outline so that it's easy to separate the bin rectangles
- geom_freqpoly: something like histogram but line (similar to cdf plotting)
- Aesthetics: fill, colour, alpha, x
  - Some computed variables: **[IMPT]** after_stat R will compute statistics for each bin, and the value of each bin is then plotted

```
    geom_histogram(aes(x=..., y=after_stat(density)), col=..., fill=...)
    geom_histogram(aes(x=..., y=after_stat(count)), col=..., fill=...)
```

  - fill=...: will create two histograms stacked on top of each other
  - use position="dodge" to prevent stacking
  - use position="identity" and α to stack histograms on top of another without stacking
  - **[IMPT]** we can also use position = position_dodge(arg=...) to adjust arguments

8.3 Line Plot

- Aesthetic:
  - Line type: dashed? dotted?
  - Line width?
  - color=....
  - group= if we want to separate lines but same color
- Usually need Tidy Data to separate time series into different groups
- **[IMPT]** geom_area(position="fill") is interesting
- **[IMPT]** geom_ribbon() is also interesting

8.4 Geom Text

- mapping = aes(x=..., y=..., label=...)
- Usually used in conjunction with geom_line

8.5 Bar Charts

- geom_col(): normal bar chart
- geom_bar(): maps count of each category
- How to change ordering of bar charts?
  - Use levels in factor!
  - reorder() in mutate!

```
op_budget <- mutate(op+budget, exp_cat = reorder(exp_cat, amount))
```

```
ggplot(tv) +
geom_col(mapping=aes(x=year,y=pct, fill=media_activity),
        position="dodge") +
  facet_wrap(~ age) +
  theme(legend.position="bottom")
```

8.5.1 Facet Wrap

```
# to change scale free and wrap labelling
facet_wrap(~ sex, nrow=2, scales="free_x",
          labeller=as_labeller(c('female'="Female", 'male'="Male")))
```

8.6 Smooth Geom

- geom_smooth(aes(...), method="lm")
- geom_smooth(aes(...), method="loess")
- Is the variability around the line the same??
- **Parameters**
  - span=: to adjust smoothing
  - se=FALSE: to remove SE ribbon

8.7 Geom Rug

geom_rug()

8.8 Boxplots

- **[IMPT]** what to look for: which point/interval has the most common combination!!!
- Shows: minimum, 1st quartile, median, 3rd quartile, max values
- Also shows outliers (values that lie $< 1.5\times$IQR below the lower quartile or values that lie $< 1.5\times$IQR above the upper quartile)
- Useful for identifying outliers, but don't dismiss them so quickly, need to analyze why are there outliers in the first place (are they just following the distribution?)
- **[IMPT]** How to reorder!!!

```
    # order by median
mutate(mpg, class2 = reorder(class, hwy, FUN=median))
```

- Boxplot secrets

```
# how to find and separate outliers
outliers <- boxplot(mpg$hwy, plot=FALSE)$out

# how to hide outliers
geom_boxplot(outlier.shape=NA)
```

8.9 2D-Histograms

```r
ggplot(mpg) +
  geom_bin2d(aes(x=displ, y=hwy), binwidth=c(0.5,5)) +
  #scale_fill_gradient(type = "viridis")
  scale_fill_gradient(name="hmm", low="white", high="red")
  # scale_fill_gradientn(colours = terrain.colors(10))
```

8.9.1 Hex bins

```r
ggplot(mpg) +
  geom_hex(aes(x=displ, y=hwy), binwidth=c(0.5,5))
```

8.9.2 2D-Density

```r
ggplot(mpg) +
  geom_density_2d(aes(x=displ, y=hwy))
```

8.10 2D Heatmaps

YAYYAYYY

```r
ggplot(data, mapping=aes(x=displ, y=hwy, z=ma3)) +
  stat_summary_hex(fun=mean, binwidth=c(0.6,5))
```

8.11 Themes

- **[IMPT]** `?theme` FOR HELP
- `theme(legend.position = "bottom" OR "none")`
  - Alternatively, can position inside the plot where (0,0) represents bottom-left and (1,1) represents top-right
- `axis.ticks`: to remove lines/ticks
- `elements`
  - `element_blank()`: to remove shit
  - `<ELE> = element_<ELE>(args =...)` to adjust arguments
- **[IMPT]** we can also save themes!

```r
t <- theme(...)
ggplot(...) + t
```

- Import themes
  - theme_tufte
  - theme_fivethirtyeight
  - theme_wsj
- `theme_set(t)` to set theme `t` as default theme
- `panel.grid = NA`: remove grid
- **[IMPT]** NICE THEMES

8.12 Grid of Plots (gridExtra)

```r
library(gridExtra)
list_of_plots <- list(p1, p2, p3, p4)
grid.arrange(list_of_plots, nrow=2)
```

8.13 Statistics (stats_)

- `stat_count` is basically `geom_bar`
- `stat_bin` is basically `geom_histogram`
- `stat_smooth` is basically `geom_smooth`
- `stat_summary` is for confidence intervals?

```r
ggplot(iris, aes(x=Species, y=Sepal.Length)) +
  stat_summary(fun.data = mean_sdl,
               fun.args = list(mult=1), # 1 standard deviation
               geom="errorbar",
               width=0.1)

```

- **[IMPT]** QQ plot

```r
iris %>%
  ggplot(aes(sample=Sepal.Length)) +
  stat_qq() +
  geom_qq_line(col="red")
```

9 Exploratory Data Analysis (EDA)

9.1 Exploration

> The greatest value of a picture is when it forces us to notice what we never expected to see

Process

- Generate questions about our data
  - What type of variation occurs within my variables?
  - What type of covariation occurs between my variables?
  - Which values are the most common? Why?
  - Which values are rare?
  - Are there any unusual patterns in the data?
- Search for answers by visualizing, transforming, and modelling our data
- Use what we learn to refine our questions and/or generate new questions (technical knowledge)
- Audience? is it for a tech/non-technical audience?

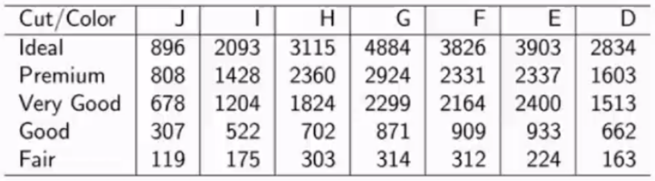
9.1.1 Contingency Table

- Let A and B be two groups with each of them having E or E^C, we want to compute the proportions of E within the groups p_A and p_B
- p_A is estimated by $\hat{p_A}$, likewise $\hat{p_B}$
- Is the difference of p between the two groups significant? in other words, is there any association between the group and event E?
  - Difference of proportion: $\hat{p}_1 - \hat{p}_2$
  - Relative risk: $\hat{p}_1/\hat{p}_2$
  - Let p be the probability of success, and $1-p$ be the probability of failure

$$\text{odds} = \frac{p}{1-p} \in [0, \infty)$$

  - Log-odds ratio: ratio of log odds

- Within different groups, are the distribution/pmf the same?



Cut/Color	J	I	H	G	F	E	D
Ideal	896	2093	3115	4884	3826	3903	2834
Premium	808	1428	2360	2924	2331	2337	1603
Very Good	678	1204	1824	2299	2164	2400	1513
Good	307	522	702	871	909	933	662
Fair	119	175	303	314	312	224	163

- **[IMPT]** `geom_mosaic`!

```r
library(ggmosaic)
ggplot(diamonds) +
  geom_mosaic(aes(x=product(color),
    fill=cut))
```

- Assuming independence,

$$\Pr(AB) = \Pr(A)\Pr(B)$$

but this is not always true, so can calculate the probability of each column/row and compare with the real table
  - Deviation from independence tells us about association!

```r
library(GGally)
ggtable(d, "color", "cut",
        cells="col.prop",
        fill="std.res")
```

9.2 Colours

```r
# table/heatmaps
diamonds %>%
  mutate(mean_price_cat = cut(mean_
    price, c(2000, 3000, 5000, 8000)))
    %>%
  ggplot() +
  geom_tile(aes(x=color, y=cut, fill=
    mean_price_cat)) +
  geom_text(aes(x=color, y=cut, label=
    round(mean_price,0))) +
  scale_fill_brewer(palette="YlOrRd")
```

- Different hues: different groups
- Different value: good for ordering
- `scale_fill_gradientn` or `scale_fill_gradient2`
- Example:

```r
scale_fill_gradient2(low="red", mid
   ="white", high="blue", midpoint
   =5000)
```

- `RColorBrewer` package

```r
?brewer.pal
display.brewer.all()
```

9.3 Correlation Matrices

- `Hmisc::describe()`
- Info correlates to how much information the variable is giving, higher info means more useful
- Gmd is Gini's mean difference to measure dispersion

```r
cor_Cars93 <- select(Cars93, which(
   !sapply(Cars93, is.factor))) %>%
cor(. , use="pair")
# need to convert to tidy data
   format to plot correlation
   matrix
cor_Cars93_df <- as.data.frame(cor=
   Cars93, row.names=NULL) %>%
               mutate(var1=row.
   names(cor_Cars93)) %>%
               pivot_longer(Min.
   Price:Weight, names_to="var2",
   values_to="correlation")
```

9.4 Hierarchical Clustering

Definition: identify groups within the data, such that members within a group are "similar" to one another

- We can cluster variables (the columns) or the observations (rows)

9.4.1 Dissimilarity Measures

Between individual observations: d measures pairwise dissimilarity between individual observations (distance?)

- Euclidean distance is the most common choice

$$d(x_i, x_j) = \sum_{s=1}^{p}(x_{is} - x_{js})^2$$

- $L1$-norm

$$d(x_i, x_j) = \sum_{s=1}^{p}|x_{is} - x_{js}|$$

Between groups: not just between individual points, we also need to generalize dissimilarity/similarity between groups of observations

- Single linkage: dissimilarity is the closest (least dissimilar) pair

$$d_S(G, H) = \min_{i \in G, j \in H} d(x_i, x_j)$$

- Complete linkage: dissimilarity is the furthest (most dissimilar) pair

$$d_C(G, H) = \max_{i \in G, j \in H} d(x_i, x_j)$$

- Average linkage: the average of all pairwise dissimilarities between the groups

$$d_A(G, H) = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{j \in H} d(x_i, x_j)$$

- Ward linkage: returns more compact clusters

Algorithm

1. Assume we start with n observations
2. Get pairwise distances between observations, store it in a $n \times n$ matrix
   - In our case the dissimilarity measure is

$$d(v_1, v_2) = \frac{1 - \text{cor}(v_1, v_2)}{2}, 0 \le d(v_1, v_2) \le 1$$

3. get the minimum dissimilarity measure (distance) in the matrix, and group the two observations together
4. Return a $(n-1)\times(n-1)$ matrix where the two observations are grouped
5. Repeat until we have 1×1 matrix

Dendrogram

- The height is the dissimilarity measure between the two child nodes
- Cut the tree horizontally and get groupings

```r
#split into 5 grps
as.factor(cutree(hc,k=5))
```

```r
# cor_Cars93_df is from section 9.3
## the disimilarity measure used here is (1-correlation)/2
hc <- hclust(as.dist((1-cor_mtcars)/2))
plot(as.dendrogram(hc))

# ordering for plot
ord <- order.dendrogram(as.dendrogram(hc))

cor_Cars93_df2 <- mutate(cor_Cars93_df,
                 var1=factor(var1, levels=row.names(cor_Cars93)[ord]),
                 var2=factor(var2,
                 levels=row.names(cor_Cars93)[ord]))

ggplot(cor_Cars94_df2) +
geom_tile(aes(x=var1, y=var2, fill=correlation)) +
scale_fill_gradient2() +
theme(axis.text.x=element_text(angle=90,vjust=0,hjust=1))
```

9.5 Multidimensional Scaling

- To visualize similarities of high-dimensional data
- With a choice of k (number of dimensions?), we seek values $z_1, z_2, \ldots, z_N \in \mathbb{R}^k$ such that the following function is minimised

$$S(z_1, \ldots z_N) = \left[\sum_{i\neq j}(d(x_i,x_j) - ||z_i - z_j||)^2\right]^{1/2}$$

```r
cars_dist <- as.dist((1-cor_Cars93)/2)
mds2 <- MASS::sammon(cars_dist, k=2)
grps <- as.factor(cutree(hc,k=5))
mds_df <- data.frame(mds2$points) %>%
  mutate(label=row.names(mds2$points), Cluster=grps) %>%
  rename('Var.1' = 'X1', 'Var.2' = 'X2')
# then plot the data with reduced dimension with all the colouring
```

9.6 Scatterplot Stuff

- Use jitter and alpha to reduce overplotting
- To fix heteroscedacity, we need to linearize the thing
- Ladder of transformations

$$-y^{-2}, -y^{-1}, \log y, y, y^2, y^3$$

- How to interpret: (use triangle method)
  - If x is doubled, y will be quadrupled (log-log scale)
  - If x is doubled, y will increase by ... (log-identity scale)

9.7 GGally

- lower/upper: lower/upper triangle
- combo: when one variable is discrete and the other one is continuous
- discrete/continuous

```r
library(GGally)
ggpairs(diamonds[,1:4], upper=list(discrete="colbar"),
        lower=list(combo="facetdensitystrip",
                   discrete="crosstable"))
```

10 Interesting stuff

- Can lookup location through zipcode
- **Which geom to use?**
  - What is the question? if no question, just pick two variables
  - Inspect types of variables
  - Who is your viewer/audience?
  - Do I need to add another variable?
  - Do we need to transform/bin the data differently?
  - Do we need to add a smooth?
  - Why are there so many outliers?
  - Do they follow some patterns that could be explained by another variable?
- **[IMPT]** `library(GGally)`: for correlation plot, useful for EDA

```r
# create correlation plot
ggpairs(data)
```

- `library(plotly)`: for interactive plot

11 Data Cleaning

- Duplicate rows (use id to clean)
- NA values