R Coding Session: A Journey into Data Manipulation and Visualisation

Day 1-2: Quick Intro to R & Tidyverse Environment

Indira Puteri Kinasih

June 6, 2023

Universiti Brunei Darussalam

Sessions

Sessions

What is R?

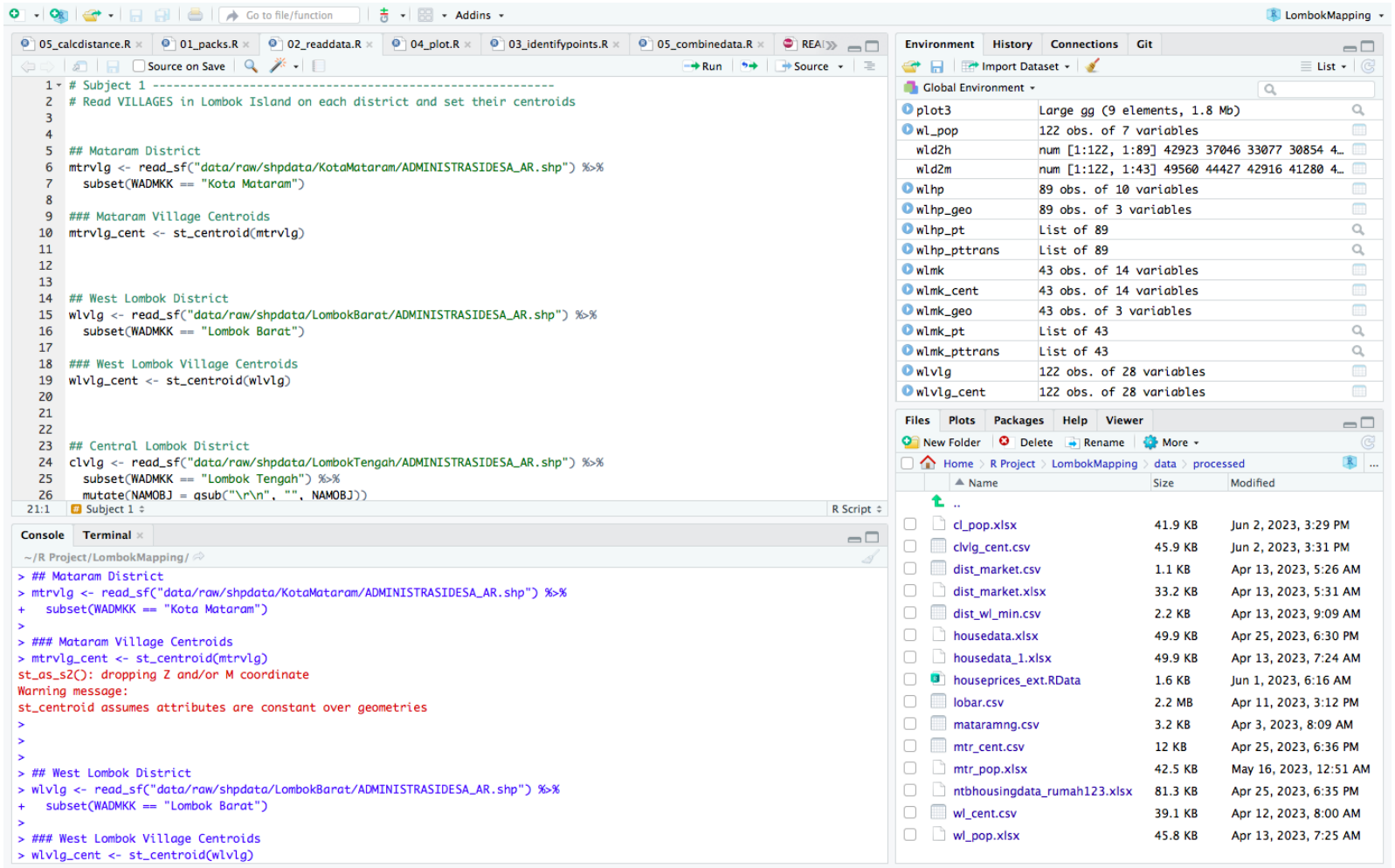
- a programming language and software environment for statistical computing and graphics;
- provides a wide variety of statistical and graphical techniques;
- widely used in academia, industry, and data analysis fields

R and R Studio

- R Studio can be considered as an extension for the programming language R, offering a more intuitive interface that includes the **R Console**, a **script editor**, and additional valuable features such as **R markdown** and integration with GitHub;

Source: Douglas et al., 2023

R Studio Orientation



R Studio Orientation

- **Script**: In the top-left quadrant, there is a simple text editor for writing your R code. It highlights the code, and allows you to easily run a section of your code (highlight a section and hit `command` and `enter`);
- **Console**: In the bottom-left quadrant, there is a console for entering R commands. This closely resembles a command-line interface where you can execute individual lines, and the console will display the corresponding results. Note, you can use the up arrow ↑ to easily access previously executed lines of code

R Studio Orientation

- **Environment**: In the top-right quadrant, the environment displays information that you have stored inside of variables. When working with scripts, it is common to create numerous variables, and the environment area assists in maintaining a record of the stored values and their respective variables;
- **Plots**, **Packages**, etc.: Within the lower-right section, there are several tabs available to access diverse information. This area serves as the rendering space for visualizations and allows you to access documentation and view the packages you have loaded

Getting Started

we can try some simple basic arithmetic expressions, using script or directly in R console, ex:

```
# Arithmetic/Number Operation
2 + 2
[1] 4
```

The other operators are $-$, $*$, / for subtraction, multiplication and division respectively;

Math functions include: $log()$, $log10()$, $exp()$, $sqrt()$. ***Please give it a try :)***

Getting Started

besides number and arithmetic expression, we also can give R a **character** input ex:

```
# Character Input
``Hi, how are you doing?''
[1] ``Hi, how are you doing?''
```

make sure to put “ ” between the characters you want to input

Objects in R

- "everything in R is an **object**";
- it can be single number, character string, plot output, a summary of your statistical analysis, or a set of R commands that perform a specific task
- to view the object's value, type the name of the object

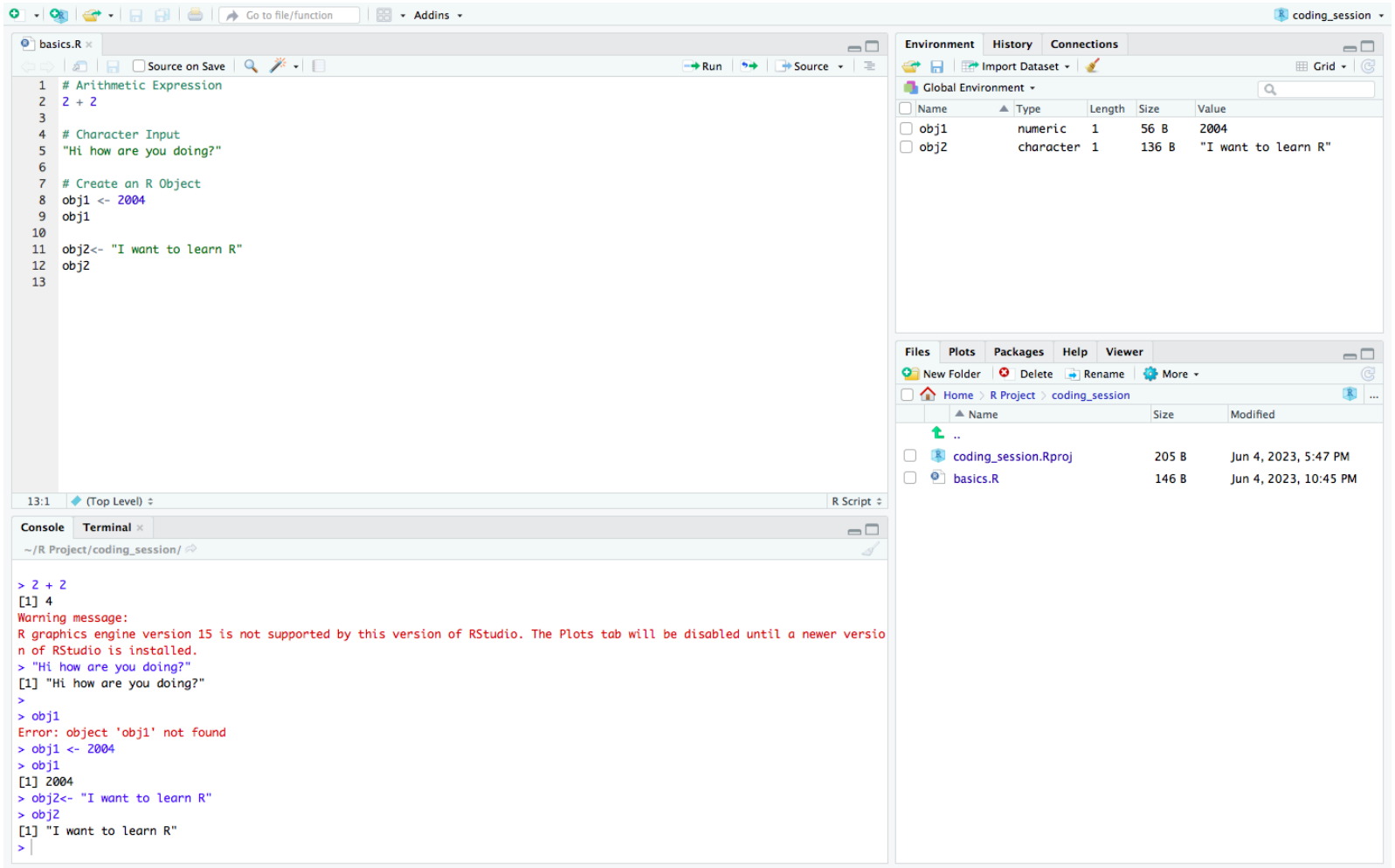
Creating Objects

- create an object = give a name to that object;
- assign a value to this object using the assignment operator <-

```
# Create an Object
obj1 <- 2003
obj1
[1] 2003
```

```
obj2 <- ''I want to learn R''
obj2
[1] ''I want to learn R''
```

Creating Objects



Creating Objects

we can do several stuff with our objects;

```
# Summing two objects
obj3 <- 1982 obj4 <- obj1 + obj3
obj4
[1] 3986
```

Sometimes, we will find an `Error` message

```
obj5 <- obj4 - obj0
obj5
Error object 'obj0' not found
```

why? because we have not created/defined the object `obj0` yet

Functions in R

- A function is a construct/object that holds a set of **instructions** to carry out a particular action or task;
- The base installation of R comes with many functions already defined;
- It is also possible for us to create our own functions to accomplish tasks that are tailored to our objectives.

Functions in R

Example of a function in R is `c()`. It is a function, which stands for *concatenate*, is utilized to **combine multiple values and store them in a vector**, which is a data structure for holding sequential elements.

```
shoes_sz <- c(41, 39, 36, 40, 38, 42, 39)
shoes_sz
[1] 41 39 36 40 38 42 39
```

Functions in R

Now, can you try to calculate the descriptive statistics of `shoes_sz` using functions `mean()`, `var()`, `sd()`, `length()`, and also `summary()` ?

Sessions

Introduction to Tidyverse

- A compilation of packages that specifically target data science
- has significantly advanced the field of R programming.

Source: Roye, 2020

Introduction to Tidyverse

Package	Description
ggplot2	Grammar for creating graphics
purrr	R functional programming
tibble	Modern and effective table system
dplyr	Grammar for data manipulation
tidyr	Set of functions to create tidy data
stringr	Function set to work with characters
readr	An easy and fast way to import data
forcats	Tools to easily work with factors

Table 1: Important Packages in Tidyverse

Style Guide

- **Avoid using more than 80 characters** per line to allow reading the complete code.
- Always use a space after a comma, never before.
- The operators (==, +, −, < −, %>%, etc.) must have a space before and after.
- **No space** between the name of a function and the first parenthesis, nor between the last argument and the final parenthesis of a function.

Style Guide

- **Avoid** reusing names of functions and common variables. Ex: `c <- 5` vs. `c()`;
- Sort the script separating the parts with the comment form `#` and `---`
- **Avoid** accent marks or special symbols in names, files, routes, etc.
- Object names must follow a constant structure: `day_one`, `day_1`.

Pipe Operator

- Using proper indentation is recommended when working with multiple arguments of a function or when chaining functions together using the `pipe` operator (%>%);
- It allows the output of a function applied to the first argument to be passed as the input to the next function.

Pipe Operator

```
# Example of a simple pipe application
c(41, 39, 36, 40, 38, 42, 39) %>% mean()
[1] 39.28571
```

Read & Write Data

Function	Description
`read_csv()` or `read_csv2()`	coma or semicolon (CSV)
`read_delim()`	general separator
`read_table()`	whitespace-separated

Table 2: Functions to Read Data

Read & Write Data

```
# Load Package
library(tidyverse)
gmob_data <-
read_csv("data/gmobility_report.csv")
gmob_data
```

Read & Write Data

```
> gmob_data <- read_csv("data/gmobility_report.csv")
Rows: 516697 Columns: 13
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr  (6): country_region_code, country_region, sub_region_1, sub_region_2, iso_3166_2_code, census_fips_code
dbl  (6): retail_and_recreation_percent_change_from_baseline, grocery_and_pharmacy_percent_change_from_baseline, pa...
date (1): date

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
> gmob_data
# A tibble: 516,697 × 13
   country_region_code country_region       sub_region_1 sub_region_2 iso_3166_2_code census_fips_code date
   <chr>               <chr>                <chr>        <chr>        <chr>           <chr>           <date>
 1 AE                  United Arab Emirates NA           NA           NA              NA              2020-02-15
 2 AE                  United Arab Emirates NA           NA           NA              NA              2020-02-16
 3 AE                  United Arab Emirates NA           NA           NA              NA              2020-02-17
 4 AE                  United Arab Emirates NA           NA           NA              NA              2020-02-18
 5 AE                  United Arab Emirates NA           NA           NA              NA              2020-02-19
 6 AE                  United Arab Emirates NA           NA           NA              NA              2020-02-20
 7 AE                  United Arab Emirates NA           NA           NA              NA              2020-02-21
 8 AE                  United Arab Emirates NA           NA           NA              NA              2020-02-22
 9 AE                  United Arab Emirates NA           NA           NA              NA              2020-02-23
10 AE                  United Arab Emirates NA           NA           NA              NA              2020-02-24
# ℹ 516,687 more rows
# ℹ 6 more variables: retail_and_recreation_percent_change_from_baseline <dbl>,
#   grocery_and_pharmacy_percent_change_from_baseline <dbl>, parks_percent_change_from_baseline <dbl>,
#   transit_stations_percent_change_from_baseline <dbl>, workplaces_percent_change_from_baseline <dbl>,
#   residential_percent_change_from_baseline <dbl>
# ℹ Use `print(n = ...)` to see more rows
>
```

Character Manipulations

For working with strings we use the `stringr` package, whose functions always start with `str_*` followed by a verb and the first argument.

Function	Description
`str_replace()`	replace patterns
`str_c()`	combine characters
`str_detect()`	detect patterns
`str_extract()`	extract pattern
`str_sub()`	extract by position
`str_length()`	length of string

Table 3: Functions to Read Data

Character Manipulations

```
# Replace 'ember' at the end with empty space
str_replace(month.name, "ember$", "")
```

```
# Combine Characters
a <- str_c(month.name, 1:12, sep = "_")
a
```

```
# Collapse Combination
str_c(month.name, collapse ", ")
```

Character Manipulations

```
# Detect Patterns
str_detect(a, "_[6-12]1")
```

```
# Extract Patterns
str_extract(a, "_[1-5]1,2")
```

```
# Extract the Characters Between Position 1
and 2
str_sub(month.name, 1, 2)
```

Character Manipulations

```
# String Length of Each Month
str_length(month.name)
```

```
# the '.'  represents the object passed by the
pipe operator
str_length(month.name)%>%
str_c(month.name, ., sep ".")
```

Table & Vector Manipulation

The `dplyr` and `tidyr` packages provide us with a data manipulation grammar, a set of useful verbs to solve common problems

Function	Description
`mutate()` or `read_csv2()`	add new variables or modify existing ones
`select()`	select variables
`filter()`	filter
`summarise()` or `read_csv2()`	summarize/reduce
`arrange()`	sort
`group_by()`	grouped
`rename()`	rename column

Table 4: Functions to Read Data

Select & Rename

```
new_gmob <- select(gmob_data,
country_region_code:sub_region_1,
date,
residential_percent_change_from_baseline ) %>%
rename(resi = 5)
```

Filter & Sort

```
ny_data <-
read_csv("data/bym_nyc_study.csv")%>%
filter(med_hhincome > 50000)%>%
arrange(-med_hhincome)
ny_data
```

```
filter(new_gmob,
country_region_code == "US")
```

```
filter(new_gmob,
country_region_code == "US",
sub_region_1 == "New York")
```

Group & Summarise

"On March 12, 2020, where can we observe the highest level of variation among regions within each country?"

```
resvar <- new_gmob %>%
          filter(date == ymd("2020-03-12"),
          !is.na(sub_region_1))%>%
              group_by(country_region)%>%
     summarise(mx = max(resi, na.rm = TRUE),
               min = min(resi, na.rm = TRUE),
               range = abs(mx) - abs(min))

arrange(resvar, -range)
```

"we want to get a subset of Europe"

```
# Package of Spatial Vectorial Data
install.packages("rnaturalearth")
library(rnaturalearth)
# world limits
wld <- ne_countries(returnclass = "sf")
# filter the countries with iso code and
select the two columns of interest
wld <- filter(wld, !is.na(iso_a2)) %>%
        select(iso_a2, subregion)

plot(wld)
```

Join Table

- Sometimes, we need to join two datasets together based on their common column. We can employ `*_join` in `dplyr` functions;
- There are `left_join`, `right_join`, and also `full_join`;
- `by` argument is not necessary as long as both tables have a column in common;

Join Table

- The `forcats` package of `tidyverse` has many useful functions for handling categorical variables (factors), variables that have a fixed and known set of possible values ;
- All `forcats` functions have the prefix `fct_*`;
- in this case we use `fct_reorder()` to reorder the country labels in order of the maximum based on the residential mobility records;
- a new column called `resi_real` is generated to modify the baseline or reference value from 0 to 100

Join Table

```
sub_eur <- filter(new_gmob,
                  is.na(sub_region_1),
                  !is.na(resi)) %>%
left_join(wld, by = c("country_region"="iso_a2")) %>%
      filter(subregion %in% c("Northern Europe",
                  "Southern Europe",
                  "Western Europe",
                  "Eastern Europe")) %>%
      mutate(resi_real = resi + 100,
                  region = fct_reorder(country_region,
                        resi,
                        .fun = "max",
                        .desc = FALSE)) %>%
      select(-geometry, -sub_region_1)

str(sub_eur)
```

Long & Wide Table

A table is tidy when:

- each variable is a column
- each observation/case is a row
- each type of observational unit forms a table

```
gmob_slc <- select(sub_eur,
                   country_region_code,
                   date:resi)

gmob_slc
```

Long & Wide Table

```
# wide table
mob_wide <- pivot_wider(gmob_slc,
          names_from = country_region,
          values_from = resi)

mob_wide
```

```
# back to long table
pivot_longer(mob_wide,
           2:36,
           names_to = "country",
           values_to = "resi")
```

Visualise Data: `ggplot2`

- A contemporary data visualisation system offering an extensive range of choices;
- The grammar of graphics (`gg`) consists of the sum of several independent layers or objects that are combined using + to construct the final graph

Visualise Data: `ggplot2`

- data: our dataset (data.frame or tibble);
- aesthetics: with the `aes()` function we indicate the variables that correspond to the x, y, z, ... axes, or when it is intended to apply graphic parameters (color, size, shape) according to a variable. It is possible to include `aes()` in `ggplot()` or in the corresponding function to a geometry `geom_*`

Visualise Data: `ggplot2`

- geometries: are `geom_*` objects that indicate the geometry to be used, (eg: `geom_point()`, `geom_line()`, `geom boxplot()`, etc.);
- scales: are objects of type `scales_*` (eg, `scales_x_continous()`, `scales_colour_manual()`) to manipulate axes, define colors, etc.
- statistics: are `stat_*` objects (eg, `stat_density()`) that allow to apply statistical transformations.

Visualise Data: `ggplot2`

see: this for more details and this for more ideas

Line & Scatter Plot

```
# Create subset
ita <- filter(gmob_data,
              country_region == "Italy",
              is.na(sub_region_1)) %>%
       mutate(resi = residential_percent_change_from_baseline/100,
       parks = parks_percent_change_from_baseline/100/100)

# line plot
ggplot(ita,
              aes(date, resi)) +
              geom_line()
```

Line & Scatter Plot

```
# Scatter Plot
ggplot(ita,
       aes(parks, resi)) +
       geom_point() +
       geom_smooth(method = "lm")
```

Sessions

References I

Douglas, A., Roos, D., Mancini, F., Couto, A., & Lusseau, D. (2023). *An introduction to r*. https://intro2r.com/chap1.html

Roye, D. (2020). *A very short introduction to tidyverse*. https://dominicroye.github.io/en/2020/a-very-short-introduction-to-tidyverse/#pipe