

Day_1_session_02

Module III:

Working with dataframe in R

- Installing & using packages
- To understand how to import external data set
- Construct data structures to store external data in R.
- Inspect data structures in R
- Labeling variables, Filtering data, data transformation
- Demonstrate how to subset data from data structures,
- Exporting data to the directory.

Installing & using packages

R is a modular environment that is extended by the use of packages. Packages are collections of functions or commands that are designed to perform specific tasks.

Using an R package is a two step process:

- Install the package onto your computer using the **install.packages()** function. This only needs to be done the first time you use the package.
- Load the package into your R session's search path using the **library()** function. This needs to be done each time you use the package.

```
# Installing package

install.packages("package name")

##Call/Loading required packages into R-Studio environment
library(tidyverse)
library(ggplot2)
library(readxl)
library(foreign)
require(AER)
require(wooldridge)
library(plotrix)
library(gapminder)
library(highcharter)
library(gridExtra)
library(mgcv)
library(car)
library(stargazer)
```

Built in data set in R

```
data()## see all the available data sets
data(ChickWeight)
View(ChickWeight)
data(Titanic)
View(Titanic)

## Transform values into a dataframe

df <- as.data.frame(Titanic)
```

Regardless of the specific analysis in R we are performing, we usually need to bring data in for the analysis. The function in R we use will depend on the type of data file we are bringing in (e.g. text, Stata, SPSS, SAS, Excel, etc.) and how the data in that file are separated, or delimited. The table below lists functions that can be used to import data from common file formats.

Data type	Extension	Function	Package
Comma separated values	csv	read.csv()	utils (default)
		read_csv()	readr (tidyverse)
Tab separated values	tsv	read_tsv()	readr
Other delimited formats	txt	read.table()	utils
		read_table()	readr
		read_delim()	readr
Stata version 13-14	dta	readddta()	haven
Stata version 7-12	dta	read.dta()	foreign
SPSS	sav	read.spss()	foreign
SAS	sas7bdat	read.sas7bdat()	sas7bdat
Excel	xlsx, xls	read_excel()	readxl (tidyverse)

Import csv Data Set

```
#Import the brand Choice.csv dataset (df1)

df1_brandChoices_method1<- read.csv(file.choose(), header=T)

df1_brandChoices_method2<- read.csv(file = "brandChoices.csv", header = TRUE,
stringsAsFactors=FALSE)
```

Import xlsx Data Set

Now we import *xlsx data* set using readxl/tidyverse package.

```
# Import the flower xls|xlsx dataset (df2 and df3)
df2_xls <- read_excel("flower.xls")
df3_xlsx <- read_excel("flower2.xlsx")
```

Import txt Data Set

```
# Import the flower.txt dataset
```

```
df4_flowers <- read.table(file = 'flower.txt', header = TRUE, sep = "\t",
stringsAsFactors = TRUE)
```

```
or
df <- read.table("dataset.txt", header=TRUE, sep=",")
or
df <- read.table("dataset.txt", header=T, strings=F)
```

Importing datasets that are STATA format:

```
# Read Stata data into R
df5_stata <- read.dta("affairs.dta")
```

Importing datasets that are SPSS format:

```
# Read the SPSS data
df6_spss <- read.spss("bodyfat.sav", to.data.frame=TRUE,
                      use.value.labels=FALSE)

df7_spss <- read.spss("foodcompany.sav", to.data.frame=TRUE,
                      use.value.labels=FALSE)
```

Importing datasets that are SAS format:

```
# Activate the `sas7bdat` library
library(sas7bdat)
# Read in the SAS data
df8_sas <- read.sas7bdat("dataFileName.sas7bdat")
```

Importing dataset from online data archive:

```
#Import data from online

df9_url <- read.dta("http://fmwww.bc.edu/ec-p/data/wooldridge/affairs.dta")

df10_url <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data",header = FALSE)
View(df10_url)

location <- "https://wiki.q-researchsoftware.com/images/3/35/Technology_2018.sav"
df11_url <- read_spss(location)

library (RCurl)
download <- getURL("https://data.kingcounty.gov/api/views/yaa-
7frk/rows.csv?accessType=DOWNLOAD")
df12_url <- read.csv (text = download)
```

Importing dataset from GitHub data archive:

```
library (readr)
```

```
urlfile <- "https://raw.githubusercontent.com/masud-alam/ECO274LAB/main/ret_final.csv"

df12_github <- read_csv(url(urlfile))
```

Inspecting data structures

There is a wide selection of base functions in R that are useful for inspecting your data and summarizing it.

List of functions for data inspection

Here is a non-exhaustive list of functions to get a sense of the content/structure of data.

- All data structures - content display:
 - **str()**: compact display of data contents (env.)
 - **class()**: data type (e.g. character, numeric, etc.) of vectors and data structure of dataframes, matrices, and lists.
 - **head()**: will print the beginning entries for the variable
 - **tail()**: will print the end entries for the variable
- Vector and factor variables:
 - **length()**: returns the number of elements in the vector or factor
- Dataframe and matrix variables:
 - **dim()**: returns dimensions of the dataset
 - **nrow()**: returns the number of rows in the dataset
 - **ncol()**: returns the number of columns in the dataset
 - **rownames()**: returns the row names in the dataset
 - **colnames()**: returns the column names in the dataset

Example1: Labeling variables and values, and data transformation

```
##Import Technology 2018 data set

Technology_2018 <- read_sav("Technology_2018.sav")
View(Technology_2018)
attributes(Technology_2018)

# Labeling variables and values

str(Technology_2018$Q1)

# Step 1: Rename the Q1 column label
colnames(Technology_2018)[colnames(Technology_2018) == "Q1"] <- "gender"

# Step 2: Convert numerical values to categorical variables
Technology_2018$fact_gender <- factor(Technology_2018$gender, levels = c(1, 2),
labels = c("Male", "Female"))
```

```
Technology_2018$fact1 <- ifelse(Technology_2018$gender == 1, "Male", "Female")

## Convert from character to numeric

Technology_2018$gender2 <- ifelse(Technology_2018$fact1 == "Male", 1, 2)

# Creating dummy variables from a factor variable
Technology_2018$dummy_gender <- model.matrix(~ Technology_2018$fact1 - 1)

# Printing the dummy variables
print(Technology_2018$dummy_gender)
```

Example 2: From continuous to categorical

```
## From continuous to categorical

df13 <- Technology_2018[,1:5]
str(df13$Q2)
attributes(df13$Q2)

# Assuming you have the dplyr package installed
library(dplyr)

df13$age_range <- df13$Q2

# Create the new column age_cat based on age_range values
# Simplify Your Code with %>% (ctrl+shift+M):Pipe operator
df13 <- df13 %>%
  mutate(age_cat = case_when(
    age_range == 1 ~ "Less than 18",
    age_range == 2 ~ "18 to 24",
    age_range == 3 ~ "25 to 29",
    age_range == 4 ~ "30 to 34",
    age_range == 5 ~ "35 to 39",
    age_range == 6 ~ "40 to 44",
    age_range == 7 ~ "45 to 49",
    age_range == 8 ~ "50 to 54",
    age_range == 9 ~ "55 to 64",
    age_range == 10 ~ "65 or more",
  ))
```

Example3: Data Transformation

Let's use the affairs data that we imported above to test out data import functions.

```
## Step I
#download affairs data from the web

affairs <- read.dta("http://fmwww.bc.edu/ec-p/data/wooldridge/affairs.dta")

#alternatively you can download data from the GitHub page
# or you can use package Wooldridge
# require(wooldridge)
# data(package="wooldridge")
# data("affairs")
# View(affairs)
View(affairs)

head(affairs)
tails(affairs)

str(affairs)
View(affairs$kids)

class(affairs$kids)
```

```
## Step II: create factors for kids and for marriage and attach labels

haskids <- factor(affairs$kids,labels = c("no","yes"))

#for ratmarr collum, create five labels and convert the col values

mlab <- c("very unhappy","unhappy","average","happy","very happy")

marriage <- factor(affairs$ratemarr,labels = mlab)
marriage

table(haskids)#frequencies for kids

prop.table(table(marriage)) #marriage ratings and check the share/proportions

#Now make a contingency table and counts(display and store variables)
```

```
(countstab <- table(marriage,haskids))

#now we will see the share with in marriage,i.e with in a row (1)
prop.table(countstab,margin = 1)

#next check share within "haskids",i.e with in a column
prop.table(countstab,margin = 2)
```

```
## Step III: #let's make some graph to depict above information

pie(table(marriage),col = c("blue","green","yellow","red","grey"),main =
"Proportion of marriage couple")
table(marriage)

# x <- c(16,66,93,194,232)
# library(plotrix) # you need this package to draw 3D plot. it looks cool!!
# pie3D(x,labels=mlab,explode=0.1,
#       main="Distribution of marriage status ")

barplot(table(marriage),horiz = F,las=1,
        main = "Distribution of happiness",ylim = c(0,180), col =
c("blue","green","yellow","red","purple"))

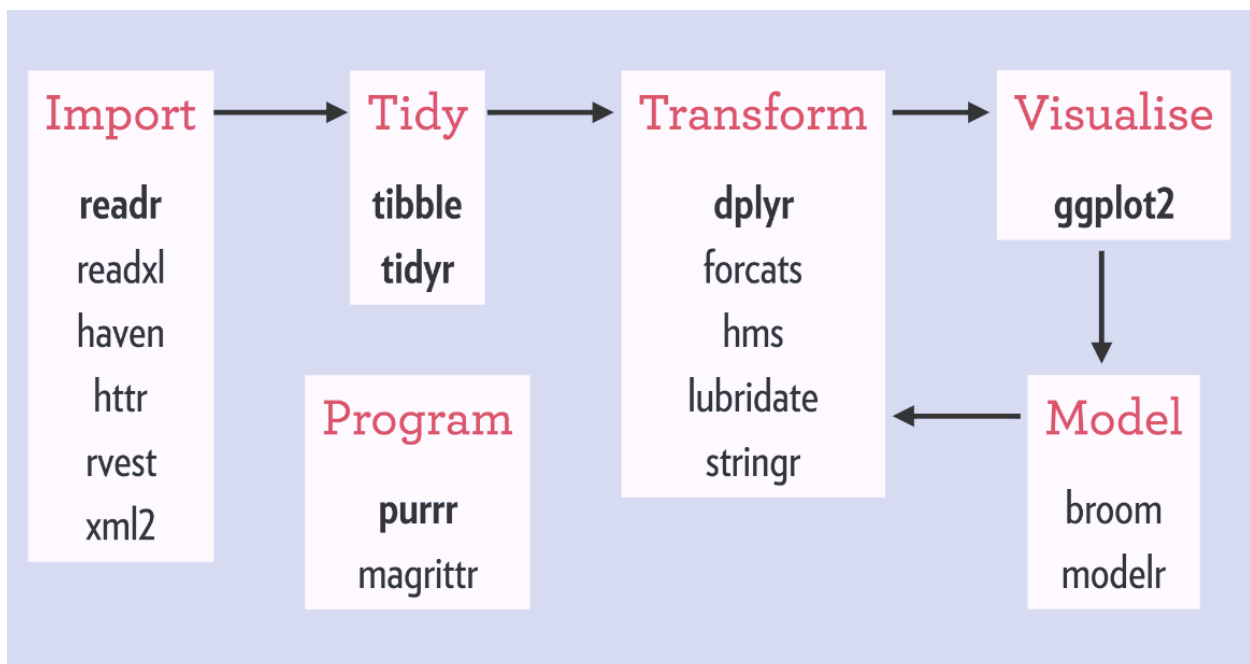
barplot(table(haskids, marriage),horiz = T,las=1,
        legend=T, args.legend = c(x="bottomright"),
        main = "Happiness by kids",col = c("green","purple"))

barplot(table(haskids, marriage),beside = T,las=2,
        legend=T, args.legend = c(x="topleft"),
        main = "Happiness by kids",col = c("green","purple"))
```

Data filtering

This section is primarily based on tidyverse package. Welcome to the wonderful world of Tidyverse!

It is the most powerful collection of R packages for preparing, wrangling and visualizing data. Tidyverse has completely changed the way we work with messy data – it has actually made data cleaning and massaging fun! The tidyverse is a coherent system of packages for data manipulation, exploration and visualization that share a common design philosophy. These were mostly developed by Hadley Wickham himself, but they are now being expanded by several contributors. Tidyverse packages are intended to make statisticians and data scientists more productive by guiding them through workflows that facilitate communication, and result in reproducible work products. Fundamentally, the tidyverse is about the connections between the tools that make the workflow possible.



First, let's talk about PIPE operator. Simplify Your Code with %>% (ctrl+shift+M) (read then)

In R, the pipe operator is, as you have already seen, %>% . It takes the output of one function and passes it into another function as an argument. This allows us to link a sequence of analysis steps.

Example 4: Data filtering

```
### Using pipe operator
```



```

library(gapminder)
library(tidyverse)

# Load gapminder dataset
data("gapminder")
View(gapminder)

# Filter and display data for Oman (first 10 rows)
gapminder %>%
  filter(country == "Oman") %>%
  head(10)

# Filter and display data for Oman, years 1981 to 2000 (first 10 rows)
gapminder %>%
  filter(country == "Oman" & year > 1980 & year <= 2000) %>%
  head()

gapminder %>% filter(country=="Oman") %>% head(10)
gapminder %>% filter(country=="Oman" & year>1980 & year<=2000
) %>% head()

```

Example 5: Data filtering

```

data("starwars")
View(starwars)

starwars %>% filter(height>150 & mass<200) %>%
  mutate(height_in_meters=height/100) %>%
  select(height_in_meters,mass) %>%
  arrange(mass) %>%
  View()

## Select variables

starwars %>%
  select(name,height,mass)

starwars %>% select(1:3)

## Changing variable name
starwars %>%
  rename("characters"="name") %>%
  head()

## filter rows
starwars %>%
  select(mass,sex) %>%
  filter(mass<55 & sex=="male")

##Recode data

```

```

starwars %>%
  select(sex) %>%
  mutate(sex=recode(sex, "male"="man", "female"="women"))

## Dealing with missing data

mean(starwars$height, na.rm = T)

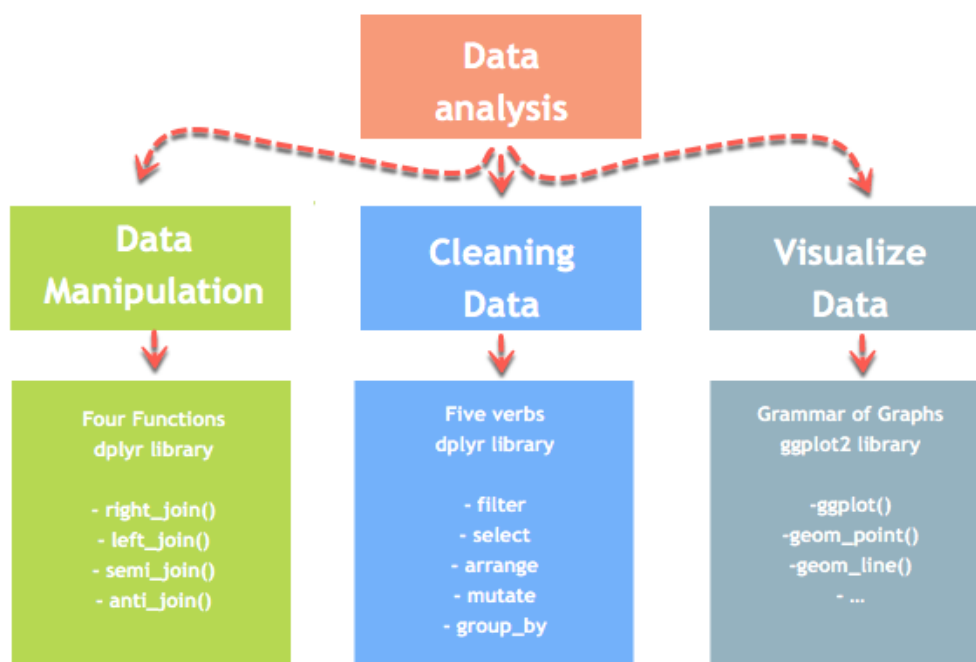
# Remove NA values and calculate the mean of height
mean_height <- starwars %>%
  na.omit() %>%          # Remove rows with NA values
  summarise(mean_height = mean(height))

## create or change a new variable
starwars %>%
  mutate(height_m=height/100) %>%
  select(name,height,height_m)

## conditional statement

starwars %>%
  mutate(height_m=height/100) %>%
  select(name,height,height_m) %>%
  mutate(tallness=if_else(height_m<1,"short","tall"))

```




Example 6: Reshape data into a long or wide format

Reshaping data from wide to long

Gather() `gather(year, growth, q1_2017:q4_2018)`

Messy

country	q1_2017	q2_2017	q3_2017	q4_2017
A	0.03	0.05	0.04	0.03
B	0.05	0.07	0.05	0.02
C	0.01	0.02	0.01	0.04



country	time	growth
A	q1_2017	0.03
B	q1_2017	0.05
C	q1_2017	0.01
A	q2_2017	0.05
B	q2_2017	0.07
C	q2_2017	0.02
A	q3_2017	0.04
B	q3_2017	0.05
C	q3_2017	0.01
A	q4_2017	0.03

```
## Let's create wide data sets

library(tidyr)
# Create a wide messy dataset
messy_df <- data.frame(
  country = c("A", "B", "C"),
  q1_2017 = c(0.03, 0.05, 0.01),
  q2_2017 = c(0.05, 0.07, 0.02),
  q3_2017 = c(0.04, 0.05, 0.01),
  q4_2017 = c(0.03, 0.02, 0.04))
messy_df

# Reshape the data from wide to tidy long data
tidy_df <- messy_df %>%
  gather(quarter, growth, q1_2017:q4_2017)
tidy_df

#The spread() function does the opposite of gather
# from long to wide data
```

```
# Reshape the data
messy_wide <- tidy_df %>%
  spread(quarter, growth)
messy_wide
```

separate()

The `separate()` function splits a column into two according to a separator. This function is helpful in some situations where the variable is a date.

```
separate_tidier <- tidy_df %>%
  separate(quarter, c("Qrt", "year"), sep = "_")
head(separate_tidier)
```

unite()

The `unite()` function concatenates two columns into one.

```
unit_tidier <- separate_tidier %>%
  unite(Quarter, Qrt, year, sep = "_")
head(unit_tidier)
```

Reshaping Wide Data to Long Data using `pivot_longer()` and `pivot_wider()` functions

```
## Create a data frame

students <- data.frame(
  name = c("Alice", "Bob", "Charlie"),
  math = c(90, 85, 92),
  science = c(95, 88, 91),
  history = c(87, 92, 78)
)

# Reshape wide data to long data
long_data <- students %>%
  pivot_longer(cols = c(math, science, history),
               names_to = "subject", values_to = "score")

long_data

# Reshape long data to wide data
wide_data_back <- long_data %>%
  pivot_wider(names_from = "subject", values_from = "score")

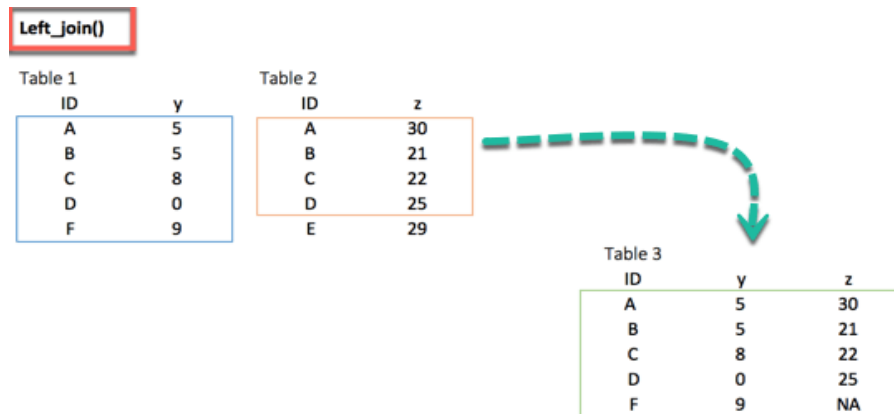
# Display wide data
```

```
print(wide_data_back)
```

Example 7: Merging data set

left_join(): Merge two datasets. Keep all observations from the origin table

With the `left_join()`, we will keep all the variables in the original table and don't consider the variables that do not have a key-paired in the destination table. In our example, the variable E does not exist in table 1. Therefore, the row will be dropped. The variable F comes from the origin table; it will be kept after the `left_join()` and return NA in the column z.



```
## Let's create two data sets

df_1 <- data_frame(id = c("a", "b", "c", "d", "f"), y = c(5, 5, 8, 0, 9))
df_2 <- data_frame(id = c("a", "b", "c", "d", "e"), y = c(15, 7, 3, 10, 19))

left_join(df_1, df_2, by = 'id')
```

right_join()

The `right_join()` function works exactly like `left_join()`. The only difference is the row dropped. The value E, available in the destination data frame, exists in the new table and takes the value NA for the column y.

right_join()

Table 1

ID	y
A	5
B	5
C	8
D	0
F	9

Table 2

ID	z
A	30
B	21
C	22
D	25
E	29



Table 3

ID	y	z
A	5	30
B	5	21
C	8	22
D	0	25
E	NA	29

```
right_join(df_1, df_2, by = 'id')
```

inner_join()

When we are 100% sure that the two datasets won't match, we can consider to return only rows existing in both dataset. This is possible when we need a clean dataset or when we don't want to impute missing values with the mean or median.

inner_join()

Table 1

ID	y
A	5
B	5
C	8
D	0
F	9

Table 2

ID	z
A	30
B	21
C	22
D	25
E	29



Table 3

ID	y	z
A	5	30
B	5	21
C	8	22
D	0	25

```
inner_join(df_1, df_2, by = 'id')
```

full_join()

Finally, the full_join() function keeps all observations and replace missing values with NA.

full_join()

Table 1

ID	y
A	5
B	5
C	8
D	0
F	9

Table 2

ID	z
A	30
B	21
C	22
D	25
E	29



Table 3

ID	y	z
A	5	30
B	5	21
C	8	22
D	0	25
E	NA	29
F	9	NA

```
full_join(df_1, df_2, by = 'id')
```

Also, check other useful function: [semi_join](#), [anti_join](#), [nest_join](#)

Method 2: Using merge command.

```
data("sleep")
View(sleep)
A <- subset(sleep, group==1, select=c("ID","extra"))
B <- subset(sleep, group==2, select=c("ID","extra"))

# for clarity, rename `extra`
names(A) <- c("ID", "extra_A")
names(B) <- c("ID", "extra_B")

## Merge when observation matched
merge(A,B)

merge(A, B, by="ID")

## Merge when observations do not match
#Now suppose that not every observation has a match.

A <- subset(sleep, group==1, select=c("ID","extra"))
B <- subset(sleep, group==2, select=c("ID","extra"))

# just some observations from B
B <- B[1:5, ]

merge(A, B, by="ID", all=TRUE)
```

```
## Append data

data("mtcars")
manual_disp <- subset(mtcars, am==1, select=c("mpg", "disp"))
automatic_hp <- subset(mtcars, am==0, select=c("mpg", "hp"))

head(manual_disp) # to show row.names
morecars1 <- merge(manual_disp, automatic_hp, by="row.names", all=TRUE)
head(morecars1, 10)

morecars2 <- merge(manual_disp, automatic_hp, by=c("row.names", "mpg"),
all=TRUE)
head(morecars2, 10)
```

Ref:

- Hadley Wickham's Advanced R book
- DataCamp's Intermediate R course
- Coursera's R Programming course
- Data Carpentry (<http://datacarpentry.org/>), data camp, data quest, Kaggle
- The Book of R: A First Course in Programming and Statistics by Tilman M. Davies
- <https://www.tidyverse.org/learn/>