Introduction to Data Science (11372 & G 11516)

Semester 1 2021

UNIVERSITY OF CANBERRA

INTRODUCTION TO DATA SCIENCE

Lecture 5

Dr. Ibrahim Radwan

DISTINCTIVE BY DESIGN

1

OUTLINE

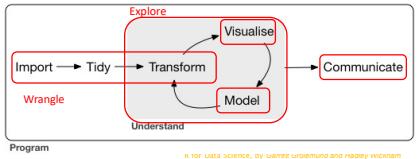
UNIVERSITY OF CANBERRA

- Data Science, a Practical View
- Data Sources and Data Formats
- Tibbles vs. Data Frames
- Data Import with `readr` Package
- Variable Specifications and Parsing

2

DATA SCIENCE; A PRACTICAL VIEW

UNIVERSITY OF CANBERRA



R for Data Science, by Garrett Grolemund and Hadley Wickham

Program Steps

1- Reading Data
2- Data Wrangling
3- Data Exploratory
4- Modelling
5- Result Communication

3

DATA SOURCES

UNIVERSITY OF CANBERRA

Data scientist deals with a diverse set of data sources (e.g. sensors, Internet, etc.)

Internet Sensors Manually created or collected

https://www.kdnuggets.com/2017/12/big-data-free-sources.html

4

DATA SOURCES (2)

UNIVERSITY OF CANBERRA

- DataHub
  - https://datahub.io/
- OpenML
  - https://www.openml.org
- Australia Open Data
  - https://data.gov.au/
- GitHub repositories, manually collected work
  - https://github.com/awesomedata/awesome-public-datasets

5

DATA FORMATS

UNIVERSITY OF CANBERRA

Data scientist also deals with different types of content such as:

Texts

Images

Sounds

This means that the data scientist spends lot of time on pre-processing these data into some data structure that is manageable by the data analysis techniques.

6

DATA FORMATS (2)

UNIVERSITY OF CANBERRA

- The most important and the commonly used data structure is the **2-D data table**, which is known as data frame in R
- Characteristics of the data tables:
  - Each row represents an entity (e.g. product, person, etc.) and the columns represent the properties (e.g. name, age, temperature, etc.) we have measured for the entities.
  - The entities are frequently referred to as objects, tuples, records, examples or feature-vectors.
  - The properties are often called features, attributes, variables, dimensions or fields.

7

DATA FORMATS (3)

UNIVERSITY OF CANBERRA

- Characteristics of the data tables (continued):
  - Rows of a dataset can be either:
    1. independent from each others; or
    2. There is some dependency among them.
       - For examples, dependencies in the form of time order (e.g. measurements of a set of variables in successive time steps).
  - Columns of a dataset can be either:
    1. Quantitative, or
    2. Categorical.
  - Columns also can be independent or correlated

8

DATA FORMATS (4)

UNIVERSITY OF CANBERRA

- Characteristics of the data tables (continued):
  - A finer categorization for the types of the columns can be:
    1. **Interval**: quantitative variables like for instance dates.
    2. **Ratio**: quantitative variables like for instance height of a person or price of a product.
    3. **Nominal**: these are categorical variables whose values are some sort of labels without any ordering among them (e.g. colors).
    4. **Ordinal**: again categorical variables but this time with some implicit ordering among their finite set of values (e.g. small, medium and large).

9

DATA FORMATS (5)

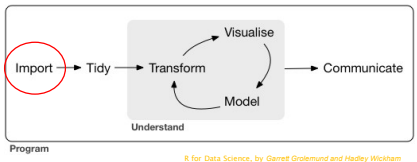
UNIVERSITY OF CANBERRA

- Most of the data scientists are dealing with the following data sources:
  1. Text files, separated by some delimiters (e.g. spaces, commas (i.e. csv), etc.)
  2. Spreadsheets (eg, *.xls, .xlsx,* etc.)
  3. Databases (MySQL, SQL, Oracle, etc.)
  4. Other software-specific formats
- In IDS unit, we will mostly deal with the plain text, **rectangular** data files such as .txt, .csv, etc.
- To handle these data, we need to read them in **data frames**

10

READING DATA

UNIVERSITY OF CANBERRA



R for Data Science, by *Garrett Grolemund and Hadley Wickham*

Program Steps

1- Reading Data
2- Data Wrangling
3- Data Exploratory
4- Modelling
5- Result Communication

11

DATA FRAMES, A RECAP

UNIVERSITY OF CANBERRA

ID, Name, Age
23424, Ana, 45
11234, Charles, 23
77654, Susanne, 76

data.csv

How to read this using the base functions in R?

Data Frame

	X	Y	Z
1	...	...	...
2	...	...	...
3	...	...	...
4	...	...	...
5	...	...	...
6	...	...	...

Observations

How many?

Variables

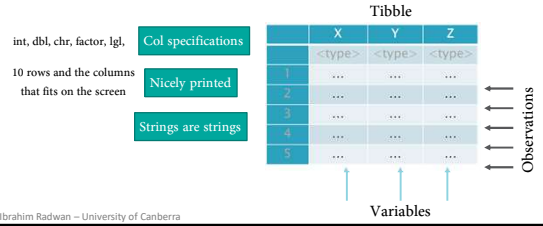
Types
Names

12

TIBBLES – A CUSTOMIZED DATA FRAMES

UNIVERSITY OF CANBERRA

- Tibbles are enhanced data frame objects where some of the features have been added and some others were dropped to make the life a bit easier when dealing with big datasets.



TIBBLES

UNIVERSITY OF CANBERRA

- Tibbles are part of the **`tidyverse`** library that has been developed by *Hadley Wickham.*
- The **`tidyverse`** library provides an effective way to read, visualize and to clean the data. It has packages such as:
  - `readr` to import data
  - `tidyr` and `dplyr` to wrangle the data
  - `ggplot2` to visualize the data

TIBBLES VS. DATA FRAMES

UNIVERSITY OF CANBERRA

```
# To use tibble, we first need to load it
# It is part of the `tidyverse` package
if_(!("tidyverse" %in% rownames(installed.packages())){
  install.packages("tidyverse")
}

library("tidyverse")
# creating a tibble from existing data frame
my_data <- iris
my_data # how is it printed?


# convert it to tibble
my_data2 <- as_tibble(my_data)
my_data2 # and now

##############################################33
# another way to create a tibble
tbl_1 <- tibble(
  x=1:5,
  y=1, # recycling feature is allowed when it is just one element in a column
  z=x^2+y)

tbl_1

# also with a transposed tibble, `tribble`
tbl_2 <- tribble(
  ~x, ~y, ~z,
  "a", 2, 3.6,
  "b", 1, 8.5
)
tbl_2
```

TIBBLES VS. DATA FRAMES (2)

UNIVERSITY OF CANBERRA

```r
# tibbles vs. dataframes
# 1- printing (first 10 rows, all cols that fits with the screen, type of column)
tbl_3 <- tibble(
  a = lubridate::now() + runif(1e3) * 86400,
  b = lubridate::today() + runif(1e3) * 30,
  c = 1:1e3,
  d = runif(1e3),
  e = sample(letters, 1e3, replace = TRUE)
)
tbl_3
df_3 <- data.frame(
  a = lubridate::now() + runif(1e3) * 86400,
  b = lubridate::today() + runif(1e3) * 30,
  c = 1:1e3,
  d = runif(1e3),
  e = sample(letters, 1e3, replace = TRUE)
)
df_3
# If you want to print more rows, n = ?
print(tbl_3, n=20, width=Inf)
# also you can use the view
view(tbl_3)
# 2- subsetting
df4 <- data.frame(x = 1:3, y = 3:1)
class(df4[, 1:2])
class(df4[, 1])
tbl4 <- tibble(x = 1:3, y = 3:1)
class(tbl4[, 1:2])
class(tbl4[,1])
class(tbl4[[1]])
class(tbl4$x)
# Tibbles are also stricter with $. Tibbles never do partial matching.
df5 <- data.frame(abc = 1)
df5$a
tbl5 <- tibble(abc = 1)
tbl5$a
```

©Dr. Ibrahim Radwan – University of Canberra

16

DATA IMPORT

UNIVERSITY OF CANBERRA

- Reading flat files such as CSV or text files
- Suppose we have the following CSV file:
  - We will use the `***readr***' library to read the csv files
  - Then, we can use the read_csv("filename.csv") function to read the contents.
  - The returned object is a tibble.
- It reads the data with 10x faster than the base R functions.

ID, Name, Age
23424, Ana, 45
11234, Charles, 23
77654, Susanne, 76

```r
> read_csv("data_sample.csv")
Parsed with column specification:
cols(
  ID = col_double(),
  Name = col_character(),
  Age = col_double()
)
# A tibble: 3 x 3
     ID Name      Age
  <dbl> <chr>   <dbl>
1 23424 Ana        45
2 11234 Charles    23
3 77654 Susanne    76
>
```

©Dr. Ibrahim Radwan – University of Canberra

17

DATA IMPORT (2)

UNIVERSITY OF CANBERRA

read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive())

a,b,c
1,2,3
4,5,NA

A	B	C
1	2	3
4	5	NA

Comma Delimited Files
read_csv("file.csv")

a b c
1 2 3
4 5 NA

A	B	C
1	2	3
4	5	NA

Tab Delimited Files
read_tsv("file.tsv")
Also **read_table()**

a;b;c
1;2;3
4;5;NA

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files
read_csv2("file2.csv")

a|b|c
1|2|3
4|5|NA

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter
read_delim("file.txt", delim = "|")

To save data into csv or txt file

Comma delimited file
write_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**
File with arbitrary delimiter
write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append**)**

©Dr. Ibrahim Radwan – University of Canberra

18

DATA IMPORT (3)

UNIVERSITY OF CANBERRA

```r
# Import data into R

# It is part of the `tidyverse` package
if (!("tidyverse" %in% rownames(installed.packages()))){
  install.packages("tidyverse")
}

library("tidyverse")

heights <- read_csv("data/heights.csv")

# supply inline csv file
read_csv("a,b,c
1,2,3
4,5,6")

# skip first two lines, i.e. meta data the beginning of a file
read_csv("The first line of metadata
  The second line of metadata
  x,y,z
  1,2,3", skip = 2)
# or skip when comment
read_csv("# A comment I want to skip
  x,y,z
  1,2,3", comment = "#")

# by default the first row is for column, we can skip this rule
read_csv("1,2,3\n4,5,6", col_names = FALSE) # \n is convenient shortcut for new line
# pass column names
read_csv("1,2,3\n4,5,6", col_names = c("x", "y", "z"))

# deal with na values
read_csv("a,b,c\n1,2,.", na = ".")
```


19

DATA IMPORT – COL SPECIFICATIONS

UNIVERSITY OF CANBERRA

- The functions of `readr` package **guess** the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically), a message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer

sex is a character

earn is a double (numeric)

The solution is to use problems() function

For each col_* function, there is a parse_* function

col_guess function uses the first 1000 rows from each column to guess the column type.

What about if:

- The first 1000 rows are just special case and the rest are different?
- The first 1000 rows are `na` ?

20

DATA IMPORT – COL SPECIFICATIONS (2)

UNIVERSITY OF CANBERRA

```r
# col specifications
# logicals
lgl <- parse_logical(c("TRUE", "FALSE", "NA"))
str(lgl)
# integers
intgr <- parse_integer(c("1", "2", "3"))
str(intgr)
# date
dt <- parse_date(c("2010-01-01", "1979-10-14"))
str(dt)
# which strings should be specified as missing
intgr_missing <- parse_integer(c("1", "231", ".", "456"), na = ".")
str(intgr_missing)
# sometimes the parsing fails, NAs will be used when failure is occuring
x <- parse_integer(c("123", "345", "abc", "123.45", "221"))
x
# here you can use problems
problems(x)

# parse numbers with , or . or $ or %
dbl <- parse_double("1.24")
str(dbl)
num <- parse_number("$123,456,789")
num
num <- parse_number("$123,456,789")
num
num <- parse_number("$123.456.789", locale=locale(grouping_mark = '.'))
num
num <- parse_number("123'456'789", locale = locale(grouping_mark = "'"))
num
```


21

DATA IMPORT – COL SPECIFICATIONS (3)

UNIVERSITY OF CANBERRA

- Using the *problems()* function with the read_* functions from the `readr` package to check the problems in parsing and guessing the column types.

problems(tibble_object)

22

DATA IMPORT – COL SPECIFICATIONS (4)

UNIVERSITY OF CANBERRA

```r
# base function to read data.frame
df <- read.csv(readr_example("challenge.csv"))
df
View(df)
df[1001,]
str(df)
problems(df) # no problems as everything are chars
##################################
# readr functions to read csv
tbl <- read_csv(readr_example("challenge.csv"))
tbl
View(tbl)
tbl[1001,]
str(tbl)
##################################
# use problems to check the problems in the tbl parsing
problems(tbl)

# how to fix the problems
# first let us fix the problem by hard-specifying the column types
tbl2 <- read_csv(
  readr_example("challenge.csv"),
  col_types = cols(
    x = col_double(),
    y = col_character()
  )
)
tbl2
tail(tbl2) # there are dates stored with this column
# change the character to date
tbl3 <- read_csv(
  readr_example("challenge.csv"),
  col_types = cols(
    x = col_double(),
    y = col_date()
  )
)
```


23

KEY TAKEAWAYS

UNIVERSITY OF CANBERRA

- As a data scientist, you will deal with different data formats and data sources
- Reading the rectangular data into tibbles are much efficient than using the data.frame
- Tibble is a customized and enhanced version of the basic data frame, where the features that resist the test of the time have been kept and the frustrating ones have been dropped
- Using the functions inside readr package to read and to write flat data are much faster and efficient than using the basic functions in R

24

RECOMMENDED READING

UNIVERSITY OF CANBERRA

- You are recommended to read chapters 10 & 11 from the *"R for Data Science"* book:
  - https://r4ds.had.co.nz/tibbles.html
  - https://r4ds.had.co.nz/data-import.html