Statistical Computing with R
Masters in Data Science 503 (S5)
First Batch, SMS, TU, 2021

Shital Bhandary

Associate Professor

Statistics/Bio-statistics, Demography and Public Health Informatics

Patan Academy of Health Sciences, Lalitpur, Nepal

Faculty, Data Analysis and Decision Modeling, MBA, Pokhara University, Nepal

Faculty, FAIMER Fellowship in Health Professions Education, India/USA.

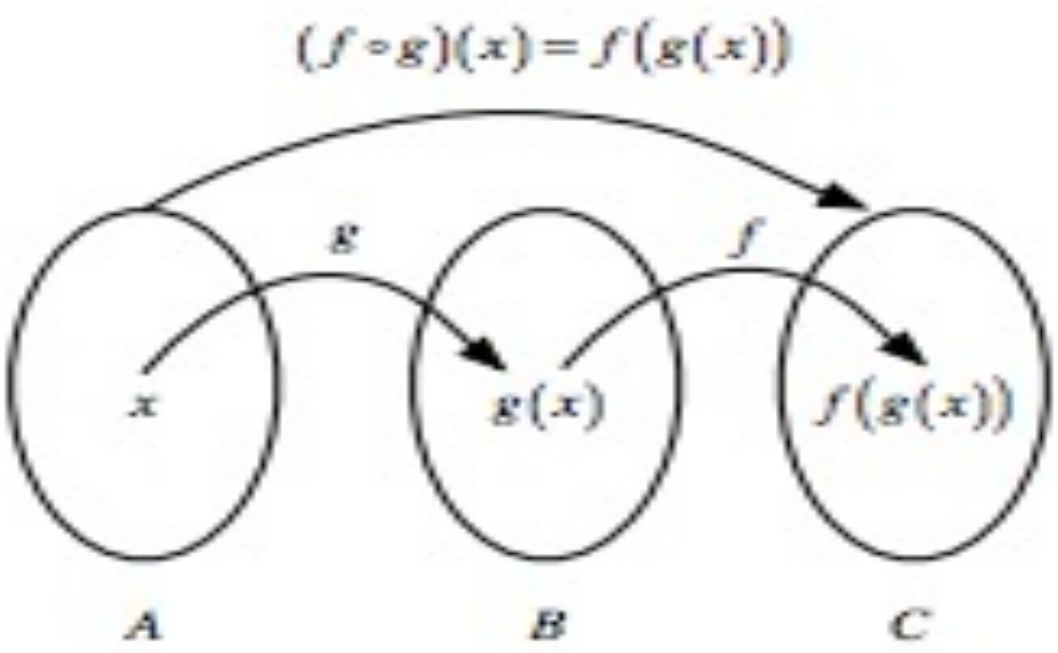
Review Preview

- Vectors in R
  - Vector operations in R
- Function in R
  - Built-in function
  - User defined function
- Pipe operator in R
  - magrittr %>%
  - dplyr %>%
  - base R from R 4.1.0 with |>

- Reproducible/Dynamic Reports
  - Markdown
  - YAML
  - R Markdown
  - Knitr
  - Pandoc

Pipe operator: Chain the functions together

- If f: B -> C and g: A -> B then we can chain these functions together by taking the output of one function and inserting in into the next



https://www.datacamp.com/community/tutorials/pipe-r-tutorial

Why to use pipe?

- R is a functional language
- It requires lots of parenthesis e.g. (and) while coding for data science
- When we have complex code, this often means we have to nest those parenthesis together for statistical computing
- This makes R codes hard to read and understand
- Here’s where pipe operator (%>%) comes handy and can rescue us too!

Example of a problem without pipe:

- # Initialize `x`

```
x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)
```

- # Compute the logarithm of `x`,
- # return suitably lagged and iterated differences
- # compute the exponential function and round the result

```
round(exp(diff(log(x))), 1)
```

Example with pipes:

- # Initialize `x`

```
x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)
```

Load the library and use the pipe operator:

```
library(magrittr)
x %>% log() %>%          # f(x) is written as x %>% f then %>% to chain
    diff() %>%           # diff function then %>% to chain
    exp() %>%            # exp function then %>% to chain
    round(1)             # round function
```

The other pipe operators:

- %<>% #Compound assignment operator
  - x <- rnorm(100)
  - x %<>% abs %>% sort #Update the value of ‘x’ and assign it to ‘x’

- %T>% #The tee operator
  - rnorm(200)
  - matrix(ncol = 2) %T>%
  - plot%>%
  - colSums

- %$% #Exposition pipe operator
  - data.frame(z = rnorm(100)) %$%
  - ts.plot(z)

More examples:

- #Load the package, install if require!

```
install.packages("babynames")
library(babynames)
library(dplyr)
```

- #Load the data:

```
data(babynames)
```

- # Count how many young boys with the name "Taylor" are born

Without and With pipe operator:

- Without:

```
sum(select(filter(babynames,sex=="M",name=="Taylor"),n))
```

- With:

```
babynames%>%filter(sex=="M",name=="Taylor")%>%
select(n)%>%
sum
```

Assigning new variable with pipe operator:

- # Load in the Iris data from internet:

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

# Add column names to the Iris data
names(iris) <- c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "Species")

# Compute the square root of `iris$Sepal.Length` and assign it to the new variable
iris$Sepal.Length.SQRT <-
        iris$Sepal.Length %>%
        sqrt()
```

Using compound assignment with pipes:

- # Compute the square root of `iris$Sepal.Length` and assign it to the **same** variable

```
iris$Sepal.Length %<>% sqrt
```

```
# Return `Sepal.Length` iris$Sepal.Length
```

The “tee” pipe operator “%T%”:

```
set.seed(123)
rnorm(200) %>%
matrix(ncol = 2) %T>%
plot %>%
colSums
```

- Normally, code ends after plot command but the “tee” pipe operator allows it to continue for the next arguement

The exposing pipe operator “%$%”:

```
iris %>%
        subset(Sepal.Length > mean(Sepal.Length)) %$%
        cor(Sepal.Length, Sepal.Width)
```

The %$% operation comes handy for functions where “data” argument is not required/used like built-in “cor” function of R!

The “dplyr” package (of tidyverse package!)

- It was built around five verbs to do data manipulation:
- “select”, “filter”, “arrange”, “mutate” and “summarize”
- These five verbs can do majority of data manipulation for data science projects and thus they are used widely
- Pipe operators are used in other tidyverse packages like ggplot2 too!

With “group_by” function of “base” but Without “dplyr” package and pipe operators:

```
library(hflights)        #Install the package if required!
grouped_flights <- group_by(hflights, Year, Month, DayofMonth)

flights_data <- select(grouped_flights, Year:DayofMonth, ArrDelay, DepDelay)

summarized_flights <- summarise(flights_data,
        arr = mean(ArrDelay, na.rm = TRUE),       #Remove missing data!
        dep = mean(DepDelay, na.rm = TRUE))     #Remove missing data!

final_result <- filter(summarized_flights, arr > 30 | dep > 30)
final_result
```

Missing values in R: VERY IMPORTANT!

- In R, missing values are represented by the symbol **NA** (not available).
- is.na(x) # Checking na in R: returns TRUE if x is missing
  y <- c(1,2,3,NA) # y variable with one missing data
  is.na(y) # returns a vector (F F F T) as 4th data is missing (T)
- # recode 99 to missing for variable v1
  # select rows where v1 is 99 and recode column v1
  mydata$v1[mydata$v1==99] <- NA
- x <- c(1,2,NA,3)
  mean(x) # returns NA (mean of 1, 2, NA and 3)
  mean(x, na.rm=TRUE) # returns 2 (mean of 1, 2 and 3)

With SELECT, SUMMARIZE and FILTER of “dplyr” package and pipe operators:

```
hflights %>% group_by(Year, Month, DayofMonth) %>%

select(Year:DayofMonth, ArrDelay, DepDelay) %>%

summarise(arr = mean(ArrDelay, na.rm = TRUE), dep = mean(DepDelay, na.rm = TRUE)) %>%

filter(arr > 30 | dep > 30)
```

ARRANGE: Sort data (after select / filter)!

```
iris %>%
        select(starts_with('Sepal')) %>%
        fliter(Sepal.Length >=70) %>%
        arrange(Sepal.Length)                    #Sort data in ascending order

iris %>%
        select(starts_with('Sepal')) %>%
        fliter(Sepal.Length >=70) %>%
        arrange(desc(Sepal.Length)               #Sort data in descending order
```

MUTATE: Make new variable

```
iris %>%
        select(contains('Sepal')) %>%
        mutate(Sepal.Area = Sepal.Lenght * Sepal.Width)

iris %>%
        select(end_with('Length')) %>%
        mutate(Length.Diff = Sepal.Length – Petal.Length)
```

https://uoftcoders.github.io/studyGroup/lessons/r/dplyrmagrittr/lesson/

What is different here?

```
iris %>%
        select(end_with('Length'), Species) %>%
        rowwise() %>%
        mutate(Length.Diff = Sepal.Length – Petal.Length)

iris %>%
        select(contains('Sepal'), Species) %>%
        transmute(Sepal.Area = Sepal.Length * Sepal.Width)
```

When NOT to use pipes?

- In chapter 18 of the web version of the text book “R for Data Science”, the authors have given four suggestions:

  - Your pipes are longer than (say) ten steps
  - You have multiple inputs or outputs
  - You are starting to think about a directed graph with a complex dependency structure
  - You're doing internal package development

More here: https://stackoverflow.com/questions/38880352/should-i-avoid-programming-packages-with-pipe-operators

Naming convention is R?

- It is not properly defined!

- This article "The State of Naming Conventions in R" talks about:

  - alllowercase e.g. adjustcolor
  - period.separated e.g. plot.new
  - **underscore_separated e.g. numeric_version**
  - lowerCamelCase e.g. addTaskCallback
  - UpperCamelCase e.g. SignatureMethod

Also check this out:
https://www.r-bloggers.com/2014/07/consistent-naming-conventions-in-r/
https://bookdown.org/content/d1e53ac9-28ce-472f-bc2c-f499f18264a3/names.html

Reproducible outputs: Markdown and YAML

- **Markdown** is described as: "*Text-to-HTML conversion tool/syntax*".
- Markdown is two things: (1) a plain text formatting syntax; and (2) a software tool, written in Perl, that converts the plain text formatting to HTML.
- On the other hand, **YAML** is detailed as "*A straightforward machine parsable data serialization format designed for human readability and interaction*".
- **YAML** is a human-readable data-serialization language. It is commonly used for configuration files, but could be used in many applications where data is being stored or transmitted.

R Markdown and knitr: Dynamic Report Generation

- You cannot execute any R code in a plain Markdown document
- You can embed the R code in plain Markdown using syntax for fenced code block ```r i.e. without curly braces but it will not be executed!
- You can embed R code chunks (```{r}) in an R Markdown document
- More here:
  - https://cran.r-project.org/web/packages/rmarkdown/index.html
  - https://sachsmc.github.io/knit-git-markr-guide/knitr/knit.html
  - https://github.com/rstudio/bookdown

Example:

```` {r my-first-chuck, results “asis”}
## code goes here
```

```` {r mtcars-example}
lm(mpg ~ hp + wt, data = mtcars)
```

```` {r mt-plot}
library(ggplot2)
ggplot(mtcars, aes(y = mpg, x = wt, size = hp)) + geom_point() + stat_smooth(method = “lm”, se = FALSE)
```

# Controlling outputs with knitr:

```
``` {r kable, results = “asis”}
kable (head(mtcars), digits = 2, align = c(“1”, 4), rep(“c”, 4), rep(“r”,4))
```
```

```
``` {r xtable, results = “asis”}
library(xtable)
print(xtable(head(mtcars)),  type = “html”)
```
```

https://sachsmc.github.io/knit-git-markr-guide/knitr/knit.html

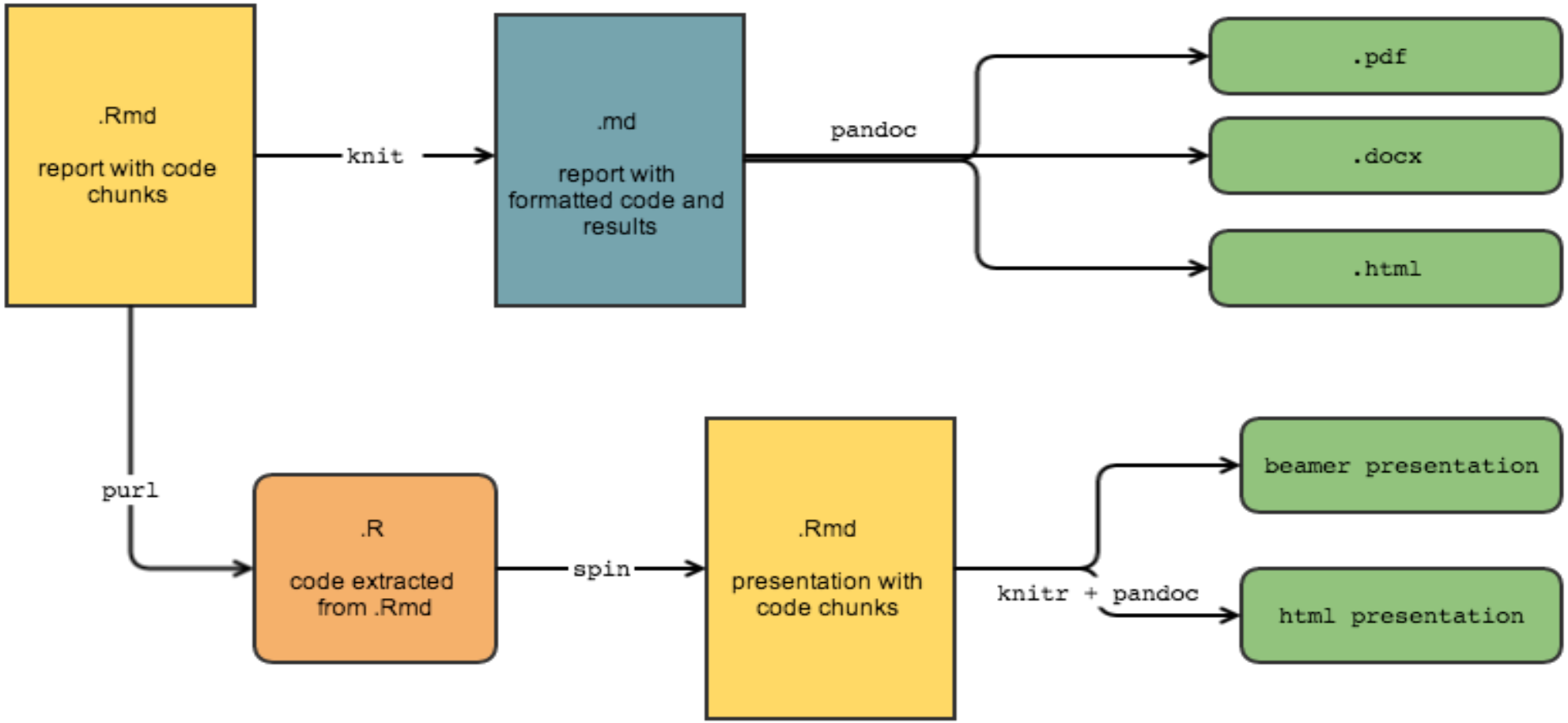
# Extending knitr: PANDOC in R Studio

- R studio have bundled knitr with pandoc, a universal document converter.

- Pandoc allows us to take markdown documents and covert them to any file format: docx, pdf, html and much more.

- We can also convert markdown to Tex (LaTex), which comes handy for journal submission that requires “Tex” format

# Workflow:




https://sachsmc.github.io/knit-git-markr-guide/knitr/knit.html

# Question/Queries?

# Thank you!

@shitalbhandary