FE8828 Programming Web Applications in Finance

Week 2: 1. R Programming/2 2. R Shiny/2: Building a web app 3. dplyr/1: Data Manipulation

Dr. Yang Ye yy@runchee.com

Nanyang Business School

Sep 24, 2020

Object

S3 Object System in R

```
# Object
# Define class with attributes.
vanilla_option <- setClass("vanilla_option",
                           slots = c(type = "character",
                                     strike = "numeric",
                                     underlying = "numeric"))

# Create object, either way
opt1 <- new("vanilla_option", type = "c", strike = 100, underlying =
opt2 <- vanilla_option(type = "c", strike = 100, underlying = 100)

# Use @ to visit member. or,
opt1@type
## [1] "c"
slot(opt1, "strike")
## [1] 100
```

Work with objects

```r
# Generate a vector of options
opts <- sapply(1:10000, function(x) {
                             vanilla_option(type = sample(c("c", "p"),
                                            strike = round(runif(1) * 1
                                            underlying = round(runif(1)

# install.packages("fOptions")
library(fOptions)

start <- Sys.time()
# GBSOption also returns an object. We just need its price attribute
res1 <- sapply(opts, function(o) {
  obj <- GBSOption(o@type, o@underlying, o@strike, Time = 1,
             r = 0.01, b = 0, sigma = 0.3)
  obj@price
})
cat(as.numeric(Sys.time() - start))
## 14.50904
head(res1, n = 4)
## [1]  0.3837668  5.7844016  0.0000000 31.8493122
```

Read/Write data

```
# set working directory
setwd("C:/TEMP")
# Save this_is_var1 to a file
saveRDS(this_is_var1, file = "C:/TEMP/DATA/data.Rds")
# Load a variable from a file. `new_loaded` is the name given to it.
new_loaded <- readRDS(file = "C:/TEMP/DATA/data.Rds")
```

- On Windows, use double slashes `\\` or single backslash `/`.
  e.g. `C:\\TEMP\\DATA`, `C:/TEMP/DATA`
- On Mac, use backslash `/Users/.../`

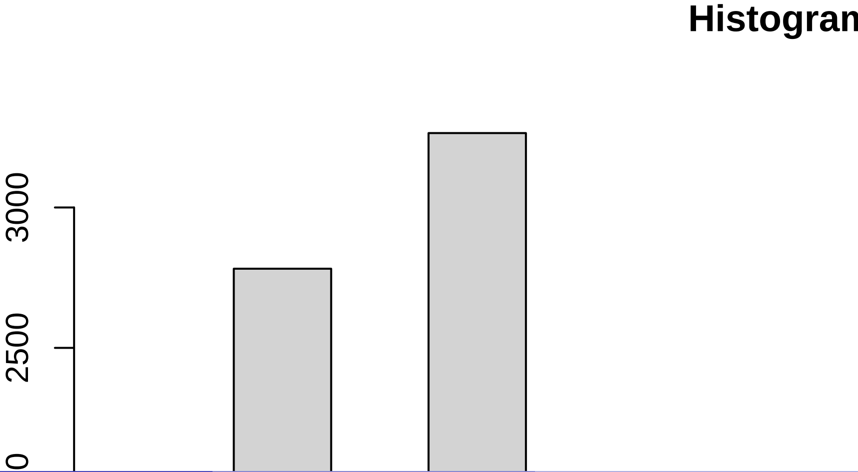
Exercise 1: Fastest Fish Problem

We have ten fishes releases in a very long lane at fixed time interval. They swim at different speed. The fish surpassing the previous fish would eat it. How many fishes would survive on average?

```
## Warning in text.default(res$queue[i], i, paste0("<U+2653> ", i),
## "skyblue3", : conversion failure on 'â" 1' in 'mbcsToSbcs': dot s
## <e2>
## Warning in text.default(res$queue[i], i, paste0("<U+2653> ", i),
## "skyblue3", : conversion failure on 'â" 1' in 'mbcsToSbcs': dot s
## <99>
## Warning in text.default(res$queue[i], i, paste0("<U+2653> ", i),
## "skyblue3", : conversion failure on 'â" 1' in 'mbcsToSbcs': dot s
## <93>
## Warning in text.default(res$queue[i], i, paste0("<U+2653> ", i),
## "skyblue3", : font metrics unknown for Unicode character U+2653
## Warning in text.default(res$queue[i], i, paste0("<U+2653> ", i),
## "skyblue3", : conversion failure on 'â" 2' in 'mbcsToSbcs': dot s
## <e2>
## Warning in text.default(res$queue[i], i, paste0("<U+2653> ", i),
## "skyblue3", : conversion failure on 'â" 2' in 'mbcsToSbcs': dot s
## <99>
```

Histogram of Fishes Alive

```
## res_sim: 2.9425
## res_ana: 2.92896825396825
```



Exercise 2: How far to make a choice?

- Secretary Problem (https://en.wikipedia.org/wiki/Secretary_problem)
- If we have 100 secretaries ranked from best to worst, coming to interview at random order, what's the probability that our picked is the best in the group?
- Our selection strategy, use a small group to establish our selection criteria, for the subsequent ones, we pick the first that's better than our selection criteria. What's the split?

Step 1: make_choice <- function(N, split_number)

1. Generate a list `input_list` of `N` long with integer 1 to N at random position
2. Split the list `input_list` into two: evaluation group and selection group.
3. Remember the best number from evaluaton group and match the first number in selection group, >= than best. Return it.

Run this function for a few (hundred) times and find the probability of getting N.

Step 2: find_optimal(), calls Step 1 a few (hundred) times for each of the split number from 1 to N/2. So we can find the optimal value for the split for the N.

Hint: Find the solution for N = 3, and N = 10, then move on to N = 100.

Section 1

Lecture 5: Shiny/2: R Web Framework

Minimalist

```r
library(shiny)
ui <- fluidPage("Hello World")
server <- function(input, output, session) { }
shinyApp(ui = ui, server = server)
```

Think around Input and Outputs

```r
ui <- fluidPage(
  titlePanel("Hello World with a Histogram"),
  # Input() functions
  numericInput("num", "Number of Sample", value = 30),
  # Output() functions
  plotOutput("hist")
)
```

Input

All input function follow such function signature except for input-specific parameters.

```
inputXXX(inputId = "input name", label = "label to display", ...)
```

- numericInput
- textInput
- passwordInput
- slideInput
- selectInput
- dateInput

Reference: https://shiny.rstudio.com/reference/shiny/1.1.0/

Output

All output functions follow such pattern.

```
yyyOutput(outputId = "output name")
```

- `textOutput("text")`
- `verbatimTextOutput("text_orignal")`
- `tableOutput("t1")`
- `dataTableOutput("t2")`
- `plotOutput(outputId = "hist", width = "400px", height = "400px")`
- `uiOutput("uiX")`

For plotOutput, I suggest to set width and height to fixed size so we need extra parameters. For other kinds of outputs, only `outputId` is good enough.

Server

Sever is to fill the content of output

```
server <- function(input, output, session) {
  # Enable either one of two
  output$hist <- renderPlot({ hist(rnorm(100)) })

  if (FALSE) {
    output$hist <- renderPlot({
      title("a normal random number histogram")
      hist(rnorm(input$num))
    })
  }
}
```

shinyApp = UI + Server

- UI and Server combine to be a ShinyApp.
- UI is to run the same for each browser/client.
- Server is separate between different users.

```
shinyApp(ui, server)
```

Reactivity Kicks In

- Reactivity: `input$num ------> output$p1`
- Reactivity links input to the output like a data flow.

Reactive values work together with reactive functions.

1. Reactive function responds. `input$x => output$y`
2. Reactive value notifies. `input$x => expression() => output$y`

Reactivity - 1

Reactivity is enabled by placing input `inputXXX` inside `renderXXX` function. (shiny-21.R)

```
library(shiny)

ui <- fluidPage(
  numericInput("num", "Num", 100),
  # numericInput("mean", "Mean", 5),
  # numericInput("sd", "SD", 3),
  numericInput("lambda", "Lambda", 1),
  plotOutput("p1")
)

server <- function(input, output, session) {
  output$p1 <- renderPlot({
    # hist(rnorm(input$num, mean = input$mean, sd = input$sd))
    hist(rpois(n = input$num, lambda = input$lambda))
  })
}

shinyApp(ui, server)
```

Reactivity - 2

- Button represents a manual trigger of the action.
- We use `observeEvent` to observe button action, and `isolate` to cut down the link of `inputXXX` in `renderXXX`, so button can work.
- If we remove `isolate`? (shiny-22.R)

```
library(shiny)

ui <- fluidPage(
  numericInput("num", "Num", 10),
  actionButton("go", "Go"),
  plotOutput("p1")
)

server <- function(input, output, session) {
  observeEvent(input$go, {
    output$p1 <- renderPlot({
      # hist(rnorm(isolate(input$num)))
      # To make code in good clarity, I re-write above one line into
      # with additional variable input_num to hold the value from in
      input_num <- isolate(input$num)
      hist(rnorm(input_num))
```

Reactivity - 3

We can add a reactiveValue with `eventReactive`. (shiny-23.R)

```
library(shiny)

ui <- fluidPage(
  numericInput("num", "Num", 10),
  actionButton("go", "Go"),
  plotOutput("p1")
)

server <- function(input, output, session) {
  data <- eventReactive(input$go, {
    hist(rnorm(input$num))
  })
  # Variable data becomes a reactive variable.
  # What changes to it will trigger the output.
  output$p1 <- renderPlot({ data() })
}

shinyApp(ui, server)
```

Output

For tableOutput

```
output$t1 <- renderTable(iris)

output$t1 <- renderTable({
  some input..
  output is a data frame.
})
```

For dataTableOutput (Dynamic table)

```
output$t2 <- renderDataTable(iris)
```

For plotOutput

```
output$p2 <- renderPlot({ plot(runif(1000), runif(1000)) })
```

For textOutput and verbatimTextOutput

```
output$t3 <- renderText({ "foo" })
output$t4 <- renderPrint({
  print("foo")
  print("bar")
```

Example: (Shiny-24.R)

```
library(shiny)
library(DT)

ui <- fluidPage(
  h3("t1"),
  tableOutput("t1"),
  hr(),
  fluidRow(
    column(9, h3("dt1"),
           dataTableOutput("dt1")),
    column(3,   h3("x4"),
           verbatimTextOutput("x4"))),
  hr(),
  fluidRow(
    column(8, h3("dt2"),
           dataTableOutput("dt2")),
    column(4, h3("p5"),
              plotOutput("p5")))
)
```

Debug Shiny

- Debug in R Studio
- Clear all variable to run Shiny in R Studio
- debugSource, if you use other source code

Shiny Summary

- Reactive is about wiring input and output
- Connect from receiver: plot/tabulate for data
- Connect from trigger: button, isolate to create a Chinese wall

Shiny Assignment

1. For Shiny-24.R, add a selectInput for different color names, returned from `colors()`.

```r
plot(1:10, pch = 19, cex = 1, col = "skyblue1")
```

2. Create a Bond Schedule

   - Inputs: start date, tenor, coupon rate, coupon frequency, and yield to maturity.
   - Output: coupon schedule (ignore public holidays), amount in table and plot. NPV

$$NPV = \frac{Cashflow1}{(1+yield)^1} + \frac{Cashflow2}{(1+yield)^2} + ... + \frac{LastCashflow}{(1+yield)^n}$$

For a Bond with fixed coupon

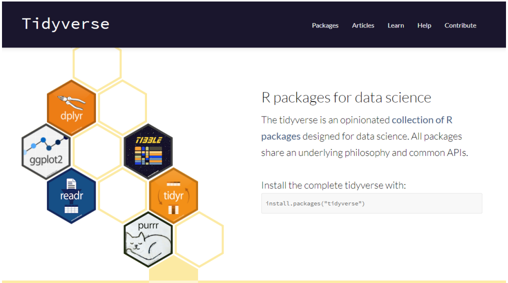
$$BondPrice = Coupon * \frac{1-(\frac{1}{(1+yield)^n})}{yield} + \left[MaturityValue * \frac{1}{(1+yield)^n} \right]$$

Section 2

Lecture 6: Data Manipulation and EDA (Exploratory Data Analysis) / 1

Tidyverse

install.packages("tidyverse")



SQL

- It was invented by Edgar Codd
- It first appeared in 1974, which is 46 years ago.

Edgar F. Codd

From Wikipedia, the free encyclopedia

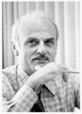
Edgar Frank "Ted" Codd (19 August 1923 – 18 April 2003) was an English computer scientist who, while working for IBM, invented the relational model for database management, the theoretical basis for relational databases and relational database management systems. He made other valuable contributions to computer science, but the relational model, a very influential general theory of data management, remains his most mentioned, analyzed and celebrated achievement.[6][7]

Contents [hide]

Biography [edit]

Edgar Frank Codd was born in Fortuneswell, on the Isle of Portland in Dorset, England. After attending Poole Grammar School, he studied mathematics and chemistry at Exeter College, Oxford, before serving as a pilot in the RAF Coastal Command during the Second World War, flying Sunderlands.[8] In 1948, he moved to New York to work for IBM as a mathematical programmer. In 1953, angered by Senator Joseph McCarthy, Codd moved to Ottawa, Ontario, Canada. In 1957 he returned to the US working for IBM and from 1961–1965 pursuing his doctorate in computer science at the

Edgar "Ted" Codd



Born	Edgar Frank Codd 19 August 1923[1][2] Fortuneswell, Dorset, England
Died	18 April 2003 (aged 79) Williams Island, Aventura, Florida, USA
Alma mater	Exeter College, Oxford University of Michigan
Known for	OLAP Relational model[3] Codd's cellular automaton Codd's 12 rules

CRUD: Create | Read | Update | Delete

Data engineering was born around 70s with SQL.

SQL does CRUD

```
# Select everything from Shops.
SELECT * FROM Shops;

# Select with a filter
SELECT * FROM Shops WHERE size = "Big";

# Select with a filter and order
SELECT * FROM Shops WHERE size = "Big" ORDER BY Name;

# Select with a filter, order, group and summary function `sum`
SELECT Region, sum(Sales) FROM Shops WHERE size = "Medium" GROUP BY

# Insert a new record to Shops.
INSERT into Shops (Name, Region, Sales) VALUES ("Costco", "North", 1

# Update a field
UPDATE Shops SET Sales = Sales + 1000 WHERE Name = "Costco";

# Delete from Shops with a filter
DELETE from Shops WHERE Sales < 1000
```

Data frame does CRUD

```r
df <- data.frame(a = 1:10, b = 10:1)
# Select (aka Filter)
df[which(df$a == 3 | df$b == 3), , drop = T]
df[match(3, df$a), , drop = T]
df[, match("b", colnames(df)), drop = T]

# Insert
rbind(df, df)

# Delete
df[-(which(df$a == 3 | df$b == 3)), , drop = T]

# Update
df[which(df$a == 3 | df$b == 3), 2] <- 3
```

dplyr

dplyr package from tidyverse is a high-performance package to manipulate data in data frame.

```
# tidyverse is a bundle of packages.
# I usually load them all with library(tidyverse, instead of librar
library(tidyverse)
# -- Attaching packages --------------------------------------- tid
# v ggplot2 3.2.1     v purrr   0.3.2
# v tibble  2.1.3     v dplyr   0.8.3
# v tidyr   0.8.3     v stringr 1.4.0
# v readr   1.3.1     v forcats 0.4.0
# -- Conflicts ------------------------------------------ tidyverse_
# x dplyr::filter() masks stats::filter()
# x dplyr::lag()    masks stats::lag()
```

Use dplyr::lag and dplyr::filter when it doesn't work.

How dplyr works

`dplyr` provides functions in "verbs", which is functions that does one thing only. We will learn to use the following.

- Key
  - select: return a subset of the columns of a data frame
  - filter: extract a subset of rows based on logical conditions
  - arrange: reorder rows
  - rename: rename variables
  - mutate: add new variables/columns or transform existing variables
- Group
  - group_by / rowwise / ungroup: stratify the data
  - summarise / summarize: generate summary statistics of different variables in the data frame, possibly within strata
  - do: process data within the strata
- Combine
  - left_join / right_join / anti_join / full_join
  - bind_rows / bind_cols
- Helpers
  - %>%: the "pipe" operator is used to connect multiple verb actions together into a pipeline
  - ifelse / case_when
  - lag/distinct

Sample dataset

```
A data-driven approach to predict the success of telemarketing
Author: Sérgio Moroa; Paulo Cortezb; Paulo Ritaa
<http://dx.doi.org/10.1016/j.dss.2014.03.001>
```

I chose this data set of a Portuguese retail bank clients profile.

- Real data collected from a Portuguese retailbank, from May 2008 to June 2013, in a total of 52,944 phone contacts.

A data-driven approach to predict the success of bank telemarketing

Sérgio Moro [a,*], Paulo Cortez [b], Paulo Rita [a]

[a] ISCTE-IUL, Business Research Unit (BRU-IUL), Lisboa, Portugal
[b] ALGORITMI Research Centre, Univ. of Minho, 4800-058 Guimarães, Portugal

ARTICLE INFO

Article history:
Received 1 November 2013
Received in revised form 28 February 2014
Accepted 4 March 2014
Available online 13 March 2014

Keywords:
Bank deposits
Telemarketing
Savings
Classification
Neural networks
Variable selection

ABSTRACT

We propose a data mining (DM) approach to predict the success of telemarketing calls for selling bank long-term deposits. A Portuguese retail bank was addressed, with data collected from 2008 to 2013, thus including the effects of the recent financial crisis. We analyzed a large set of 150 features related with bank client, product and social-economic attributes. A semi-automatic feature selection was explored in the modeling phase, performed with the data prior to July 2012 and that allowed to select a reduced set of 22 features. We also compared four DM models: logistic regression, decision trees (DTs), neural network (NN) and support vector machine. Using two metrics, area of the receiver operating characteristic curve (AUC) and area of the LIFT cumulative curve (ALIFT), the four models were tested on an evaluation set, using the most recent data (after July 2012) and a rolling window scheme. The NN presented the best results (AUC = 0.8 and ALIFT = 0.7), allowing to reach 79% of the subscribers by selecting the half better classified clients. Also, two knowledge extraction methods, a sensitivity analysis and a DT, were applied to the NN model and revealed several key attributes (e.g., Euribor rate, direction of the call and bank agent experience). Such knowledge extraction confirmed the obtained model as credible and valuable for telemarketing campaign managers.

Sample dataset columns (also called variable, field or feature)

- Personal profile
- 1 - age (numeric)
- 2 - job : type of job (categorical: "admin.","unknown","unemployed","management","housemaid","entrepreneur","stud "blue-collar","self-employed","retired","technician","services")
- 3 - marital : marital status (categorical: "married","divorced","single"; note: "divorced" means divorced or widowed)
- 4 - education (categorical: "unknown","secondary","primary","tertiary")
- 5 - default: has credit in default? (binary: "yes","no")
- 6 - balance: average yearly balance, in euros (numeric)
- 7 - housing: has housing loan? (binary: "yes","no")
- 8 - loan: has personal loan? (binary: "yes","no")
- Related with the last contact of the current campaign:
- 9 - contact: contact communication type (categorical:

Read data

Use RStudio's File -> Import Dataset, you may choose either "From Text (base)" or "From Text (readr)". Either way loads the data.

`base` comes with R. `readr` is a package from tidyverse that provides more options and functionality. Copy the generated code to your script file.

I place it at https://goo.gl/PBQnBt (for direct use), https://goo.gl/fFQAAm (for Download).

You may download it and save it to local.

```
# Use base
bank <- read.csv("example/data-bank/bank.csv", sep=";") # or,
bank <- read.csv("https://goo.gl/PBQnBt", sep = ";")

# use readr
library(readr)
bank <- read_delim("example/data-bank/bank.csv",
                   ";", escape_double = FALSE, trim_ws = TRUE)
## Parsed with column specification:
## cols(
##   age = col_double(),
```

`select`

`select(df, ...)`, `...` can be

- variable name
- numeric to indicate nth column (`-` means exclude)
- a range
- a function

select - Examples

```r
subset <- select(bank, marital)
subset <- select(bank, 1)
subset <- select(bank, -1)
subset <- select(bank, -job)
subset <- select(bank, -(job:education))
subset <- select(bank, starts_with("p"))
subset <- select(bank, ends_with("p"))
subset <- select(bank, contains("p"))
```

`select` as a re-arrangement of columns.

```r
job_first <- select(bank, job, everything())
```

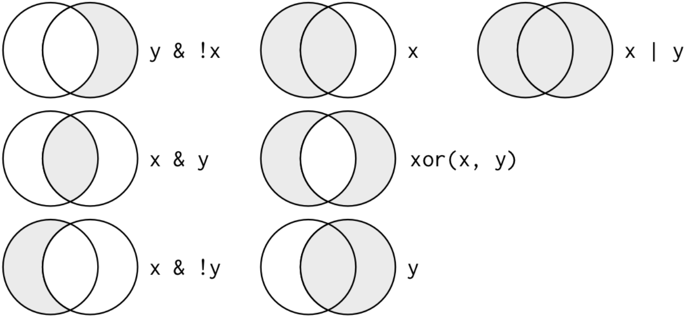
filter

```
colnames(bank)
##  [1] "age"       "job"       "marital"   "education" "default"
##  [7] "housing"   "loan"      "contact"   "day"       "month"
## [13] "campaign"  "pdays"     "previous"  "poutcome"  "y"

young <- dplyr::filter(bank, age < 40)
another_young <- dplyr::filter(bank, age < 20 & marital == "married'
just_young <- dplyr::filter(bank, age < 20 & marital == "single")

young2 <- dplyr::filter(bank, age >= 20 & age < 30)
another_young2 <- dplyr::filter(bank, age >= 20 & age < 30 & marital
just_young2 <- dplyr::filter(bank, age >= 20 & age < 30 & marital ==
```

`filter` - logic operators



filter - string operations

```r
# %in% to match multiple
second_upper <- dplyr::filter(bank, education %in% c("tertiary", "se

# filter out NA value.
no_na <- dplyr::filter(bank, !is.na(balance) & balance > 0)
```

Exercise

- How many bank client have a loan while doesn't have a housing?
- How many bank client have a job between 20 to 40?

rename

```r
# rename(new name = old)
# Use tick to quote special strings.
df <- rename(bank, young_age = age)
df <- rename(bank, `Age in Bank` = age)
```

arrange

```
# arrange is sort
arrange(bank, job)
arrange(bank, default, job)

# descending for day
arrange(bank, desc(day))
arrange(bank, desc(as.Date(day, format="%d", origin = Sys.Date())))
```

NB: Missing values are always sorted at the end.

Exercise

- How could you use arrange() to sort all missing values to the start? (Hint: use `is.na()`).

```
arrange(bank, !is.na(a), a)
```

- Find the longest duration?
- Find the eldest?

mutate

```r
# Replace existing
# ifelse is to check condition.
df1 <- mutate(bank, y = ifelse(y == "yes", T, F))

# Add a new column.
df2 <- mutate(bank, duration_diff = duration - mean(duration, na.rm

# case_when is a function to deal multiple choices.
df2_age_group <- mutate(bank, age_group = case_when(
  age < 20 ~ "youth",
  age < 40 ~ "middle-age",
  age < 50 ~ "senior",
  TRUE ~ "happy"
))

df2_age_group_res <-
  group_by(df2_age_group, age_group) %>%
  summarise(mean_age = mean(age)) %>%
  transmute(mean_age_diff = mean_age - lag(mean_age))
## `summarise()` ungrouping output (override with `.groups` argumen
```

mutate 2

```
firstup <- function(x) {
  substr(x, 1, 1) <- toupper(substr(x, 1, 1))
  x
}

# month.abb is a built-in array of month names.
df3 <- mutate(bank, month_name = factor(firstup(as.character(month))

# transmute would remove all other columns after mutation, only keep
df5 <- transmute(bank,
                 duration_trend = duration - mean(duration, na.rm =
                 balance_trend = balance - mean(balance, na.rm = TR
```

What you can do with `mutate`

- +, -, *, /: ordinary arithmetic operator
- %/% (integer division) and %% (remainder), where x == y * (x %/% y) + (x %% y)
- x / sum(x): compute the proportion of all things
- y - mean(y): computes the difference from the mean.
- log2(), log(), log10():
- lead(), lag(): compute running differences (e.g. x - lag(x)) or find when values change (x != lag(x)
- rolling sum, prod, min, max: cumsum(), cumprod(), cummin(), cummax(); and dplyr provides cummean()
- row_number()/min_rank()/ntile(,n)

```
y <- c(1, 2, 2, NA, 3, 4)
row_number(y)
## [1]  1  2  3 NA  4  5
min_rank(y)
## [1]  1  2  2 NA  4  5
ntile(y, 2)
## [1]  1  1  1 NA  2  2
```

Summary

- We learned the key "verbs" from dplyr. Let's pick up the rest next week.

Pipe: %>%

We may write such code.

```r
df <- select(df, x)
df <- mutate(df, a = 1)
df <- rename(df, a = b)
df <- arrange(df, x)

# This is effectively,
arrange(rename(mutate(select(df, x), a = 1), a = b), x)

third(second(first(x)))
```

How about this?

```r
df %>% select %>% mutate %>% rename %>% arrange
```

%>% Benefits

%>% operator allows you to transform the flow from nesting to left-to-right fashion, i.e.

```
first(x) %>% second() %>% third()

x %>% first() %>% second() %>% third() # this could also do.

x %>% first(.) %>% second(.) %>% third(.) # . represents the input
```

What's the output of below?

```
c(1, 3, 7, 9) %>% {
  print(.)
  mean(.)
} %>% { . * 3 } %>% {
  print(.)
  sample(round(., 0))
}
## [1] 1 3 7 9
## [1] 15
##  [1]  9 14  5  3  4  8 10 15  1 12  2  7 11  6 13
```

Work with Pipe

%>% … %>%

```
# Feed the data for multiple processing
{
  v <- .
  cn <- colnames(v)

  v <- select(v, u, z)
  colnames(v) <- cn[1:3]
  v
}

# How to return multiple value

%>% {
  assign("new_data", filter(., group == "1"), envir = parent.env(env
  filter(., group == "2")
} %>% {
  select(., z < 0.4) # on group 2
  select(new_data, z > 0.4) # on group 1
}
```

Code pattern with Pipe

```
df %>%
... %>%
... %>%
... %>%
{
  v <- .
  ggplot(data = v) +
    # full data is used here
    geom_line(data = v) +
    # partial data needs to be hightlighted.
    geom_line(data = filter(., some condition), color = "red")
}
```

Use of Caution for Pipe (%>%)

Pros:

- We don't need to keep intermediate result, sames memory and also variable names.

Cons:

- Difficult to debug, to find something in the middle of the chain.
- Use `{ print(.); filter(., ...) }` to print intermediate resuls.
- Separate the long pipes into shorter pipes, adding more intermediate variables.
- Your pipes are longer than (say) ten steps. In that case, create intermediate objects with meaningful names. That will make debugging easier, because you can more easily check the intermediate results, and it makes it easier to understand your code, because the variable names can help communicate intent.
- You have multiple inputs or outputs. If two or more objects being combined together, don't use the pipe.
- Pipes are fundamentally linear and expressing complex relationships with them will typically yield confusing code.

Environment

Environment is where your data resides. Use `local()` to isolate.

```
# local stores the data wihtin the boundary of {}
x <- 3
local({
  print(x)
  x <- 1
  print(x)
})
## [1] 3
## [1] 1
print(x)
## [1] 3
```

```
# local stores the nearest environment
x <- 3
{
  print(x)
  x <- 1
  print(x)
}
```

Environment

Use `assign()` to do space-jump.

```r
# assign data to global environment
x <- 1
pass_out_global <- function() {
  assign("x", 3, envir = .GlobalEnv)
}

# assign data to just one level up
pass_out <- function(env) {
  print(env)
  assign("x", 2, envir = env)
}
```

```r
x <- 1
pass_out(environment())
## <environment: R_GlobalEnv>
x
## [1] 2

# assign data to pass it out of function
```