

| MQIM | R Workshop | Part 3

Casey T Li

Tuesday, August 31, 2018

- 1 Outline
- 2 R Objects and Functions
- 3 Data manipulation and Visualization
- 4 R Markdown, R Projects and Github
- 5 Using R for k-Nearest Neighbors (KNN)

Outline

Table of Contents

- R Objects and Functions
- Data Manipulation and Visualization (tidyverse package)
- Advanced topics: R Markdown, Rstudio Projects and Github
- Using R for k-Nearest Neighbors (KNN)
- Introduction to Python/Malab

R Objects and Functions

R Objects (frequently used)

- Vectors

Type	Example
Doubles	<code>die <- c(1:6)</code>
Characters	<code>text <- c('R', 'Workshop')</code>
Logicals	<code>logic <- c(TRUE, FALSE, TRUE)</code>

- Matrices

```
> die <- c(1:6)
> mtx <- matrix(die, nrow = 2, byrow = TRUE)
```

- Arrays

```
> ary <- array(mtx, dim=c(2,3,3))
```

R Objects (frequently used)

- Lists

```
> list <- list(die, mtx, ary)
```

- DataFrame is the two-dimensional version of a list. It is a very useful storage structure for data analysis. You can think of a dataframe as R's equivalent to the Excel spreadsheet.

```
> df <- data.frame(face = c("ace", "king", "queen"),  
+                  suit = c("heart", "spades", "diamonds"),  
+                  value = c(1, 13, 12))
```

Try for yourself by running *rObj.r*.

R function

Let's build a simple slot machine that you can play by running an R function. When you're finished, you'll be able to play it like this:

```
> play()
```

```
## [1] "DD" "0"  "0"
```

```
## [1] "You win $0"
```

The **play** function will need to do two things. First, it will need to randomly generate three symbols; and, second, it will need to calculate a prize based on those symbols.

R function

- 1 Define your slot machine symbols and then randomly generate three symbols with the **sample** function.

```
> get_symbols <- function(){  
+   wheel <- c("DD","7","BBB","BB","B","0")  
+   sample(wheel, size = 3, replace = TRUE,  
+         prob = c(0.03, 0.03, 0.06 ,0.1, 0.27, 0.51))  
+ }  
> get_symbols()
```

```
## [1] "0" "B" "B"
```

R function

- 2 Write a program that can take the output of `get_symbols` and calculate the correct prize based on the prize rule, the payout scheme is set as:

Combination	Prize
DD DD DD	100
7 7 7	80
BBB BBB BBB	40
BB BB BB	25
B B B	10
Any combo of bars	5

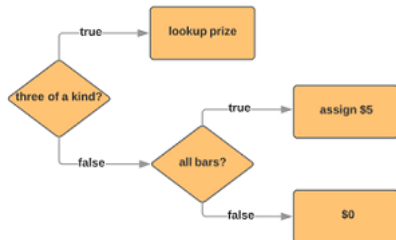
We call this function **score**, such that `score(c("DD", "DD", "DD"))` will give you \$100.

R function

- 3 Put them together to create the full slot machine, like:

```
> play <- function(){  
+   symbols <- get_symbols()  
+   print(symbols)  
+   score(symbols)  
+ }
```

Now let's write the score function, the flow chart below describes the payout scenarios, diamond shape symbolize an *if else* decision.



R function

```

> score <- function(symbols){
+   # identify case
+   same <- symbols[1] == symbols[2] && symbols[2] == symbols[3]
+   bars <- symbols %in% c("B","BB","BBB")
+   # get prize
+   if(same){
+     payouts <- c("DD" = 100, "7" = 80, "BBB" = 40, "BB" = 25,
+                  "B" = 10, "C" = 10, "0" = 0)
+     prize <- unname(payouts[symbols[1]])
+   } else if(all(bars)){
+     prize <- 5
+   } else {
+     prize <- 0
+   }
+   # return prize
+   prize <- paste0("You Win $", prize)
+   return(prize)
+ }

```

R function

Once the **score** function is defined, the **play** function will work as well:

```
> play <- function(){  
+   symbols <- get_symbols()  
+   print(symbols)  
+   score(symbols)  
+ }  
> play()
```

```
## [1] "BB" "0"  "0"
```

```
## [1] "You Win $0"
```

```
> play()
```

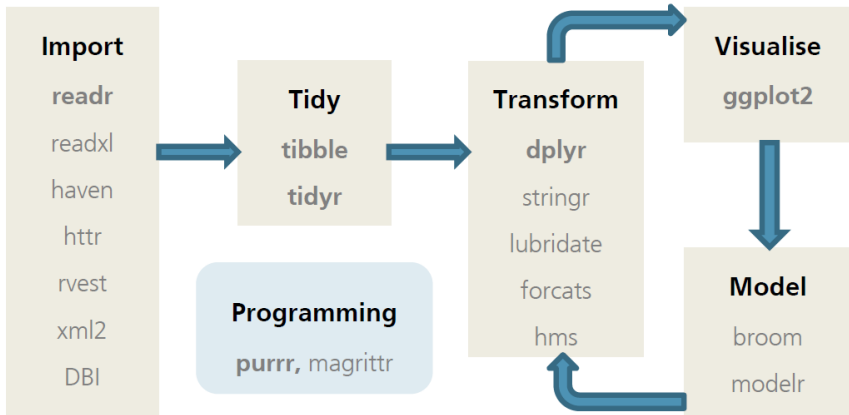
```
## [1] "0" "0" "0"
```

```
## [1] "You Win $0"
```

Data manipulation and Visualization

tidyverse

The *tidyverse* is a set of packages for doing data science, it loads six “core” libraries that provide tools for importing(*readr*), tidying(*tidyr*,*tibble*), manipulation(*dplyr*) and visualizing(*ggplot*) data, as well as support for functional programming(*purrr*).



For more details on using tidyverse for data science see <http://r4ds.had.co.nz/>.

Data Manipulation(*dplyr*)

key verbs:

- **select** for subsetting variables (columns)
- **mutate** for creating new variables from existing ones
- **filter** for subsetting observations (rows)
- **arrange** for rearranging/ordering observations
- **summarise** for computing various summary statistics

Along with the **group_by**, combining multiple simple pieces with the pipe operator `%>%` is a powerful way of solving complex problems.

Let's see a simple example by loading `data_df.RData`, which contains information about 40 large-cap companies.

```
> library(tidyverse)
> load("data_df.RData")
```


Data Manipulation(*dplyr*)

Suppose we have the following task: *Calculate the mean ROE by sector and 12-month momentum (positive or negative), making sure that there are at least 3 observations per group. Finally, arrange the data in descending order of ROE.* What would you do it without advanced package except base R?

```
> data_df$Momentum <- ifelse(data_df$Return12m > 0, "Positive",
+                             "Negative")
> data_df <- transform(data_df, freq = as.numeric(as.character(
+ ave(Sector, Sector, Momentum, FUN = length))))
> data_df_filtered <- subset(data_df, freq >= 3)
> mean_roe <- aggregate(ROE ~ Sector + Momentum,
+                        data = data_df_filtered,
+ FUN = mean, na.rm = T)
> mean_roe <- mean_roe[order(mean_roe$ROE, decreasing = TRUE), ]
> head(mean_roe,3)
```

```
##              Sector Momentum      ROE
## 3      Consumer Staples Positive 59.76600
## 2 Consumer Discretionary Positive 39.30333
```

Data Manipulation(*dplyr*)

It's more clean and straight forward by using *dplyr*

```
> mean_roe <- data_df %>%
+ mutate(Momentum = ifelse(Return12m > 0, "Positive",
+                           "Negative")) %>%
+ group_by(Sector, Momentum) %>%
+ filter(n() >= 3) %>% # n() returns row count in each group
+ summarise(ROE = mean(ROE, na.rm = TRUE)) %>%
+ arrange(desc(ROE))
> head(mean_roe,3)
```

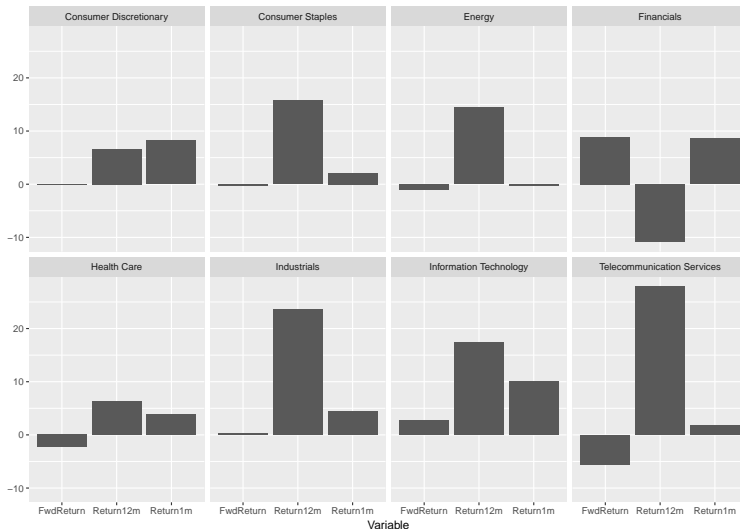
```
## # A tibble: 3 x 3
## # Groups:   Sector [3]
##   Sector                Momentum    ROE
##   <chr>                <chr>    <dbl>
## 1 Consumer Staples      Positive  59.8
## 2 Consumer Discretionary Positive  39.3
## 3 Information Technology Positive  25.2
```

Data Visualization(*ggplot*)

Charts are great tools to help you understand the data from the initial data research. If you don't want to spend too much time writing your own data manipulating and plotting functions, there is a nice and quick way to do it by combining *dplyr* and *ggplot*. Here is an example to visualize the sector mean returns of the data:

```
> myplot <- data_df %>%  
+   group_by(Sector) %>%  
+   summarise_at(vars(contains("Return")), funs(mean)) %>%  
+   tidyr::gather(Variable, Return, -Sector) %>%  
+   ggplot(aes(x = Variable, y = Return)) +  
+   geom_bar(stat = "identity") + ylab("") +  
+   facet_wrap(~ Sector, ncol = 4)
```

Data Visualization(*ggplot*)



R Markdown, R Projects and Github

R Markdown

Why R Markdown?

- R Markdown provides various great formats for communicating, presenting as well as publishing research results.
- It enables the final report to include code chunks and output (table, plot etc.) while executing the codes.
- In addition, it makes typing complicated formulas and equations much less painful.
- It also ensures reproducibility and consistency. You can keep your code, notes, graphs and relevant links all in one place.
- Of course, a great way to submit your assignment.

R Markdown

How it works?

- create *.Rmd* file, select *File > New File > R Markdown*. in the menubar. RStudio will launch a wizard that you can use to pre-populate your file with useful content that reminds you how the key features of R Markdown work.
- **knitr** the document, R Markdown sends the *.Rmd* file to knitr, <http://yihui.name/knitr/>. R Markdown executes all of the code chunks and creates a new markdown (*.md*) document which includes the code and its output.
- The markdown file generated by *knitr* is then processed by **pandoc**, <http://pandoc.org/>. R Markdown is responsible for creating the final file.



Figure 1

R Markdown

R Markdown formats:

- Documents:
 - html
 - pdf
 - word
 - github document (designed for sharing on GitHub)
- Presentations:
 - ioslides - HTML presentation with ioslides
 - slidy - HTML presentation with W3C Slidy
 - beamer - PDF presentation with LaTeX Beamer.

Let's see a couple of examples.

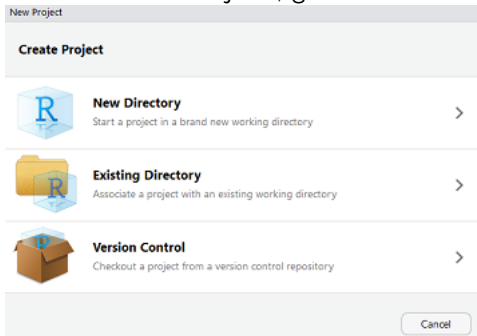
Rstudio Projects

It is a good practice to create a projects for an ongoing research process with kept developing codes. The beauty of using projects are:

- Holds all the files relevant to that particular piece of work in a folder in your computer in a desired project directory. - Set unite working directory to Project directory, save the trouble for typing `*setwd("C:/Users/path)*` and `rm(list = ls())` for each r script.
- Dedicated R process. File browser pointed at Project directory.
- The way to create RStudio Project is quite flexible, you can create it in a new folder, in a existing folder, or link it to a version control repository (we will talk about GitHub here).

Rstudio Projects

To create a new Rstudio Projects, go to *RStudio -> New Project*, you will see



options:

Click **New Directory** if you'd like to make a new folder as project, or **Existing Directory** to make existing folder into an RStudio Project.

GitHub

What is it?

- Git is an open-source version control system that was started by Linus Torvalds, whom created Linux.
- When developers create something (an app, for example), they make constant changes to the code, releasing new versions up to and after the first official release.
- Version control systems keep these revisions straight, storing the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.
- People who have nothing to do with the development of a project can still download the files and use them.
- “Hub” part in GitHub is <https://github.com/>


GitHub

- Repository: a repository(“repo”) is a location where all the files for a particular project are stored. Each project has its own repo, and you can access it with a unique URL.
- Forking a Repo
 - create a new project based off of another project that already exists.
 - fork the repo that you'd like to contribute to, make the changes you like and release the revised project as a new repo.
- You can fork the **R_Workshop** repo from my github to yours to access some R files and data.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 git4casey ▾

Repository name

r_Workshop_2018 ✓

Great repository names are short and memorable. Need inspiration? How about [jubilant-computing-machine](#).

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾



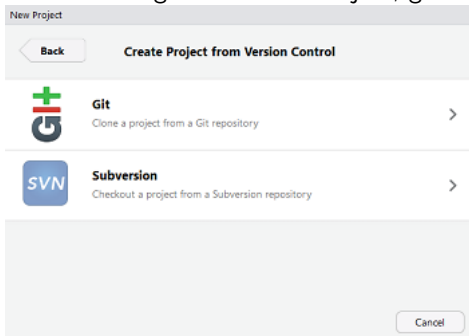
Create repository

Figure 2

Link Rstudio Project to Github

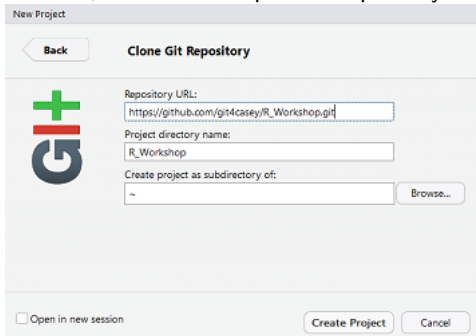
How to use it along with Rstudio Project?

- When creating the Rstudio Project, go to the third option **Version Control**



Link Rstudio Project to Github

- Select **Git**, and paste the repo URL you'd like to work on. For example, it can be the forked **R_Workshop** repo on your github.
- Select the preferred the directory in your computer by *Browse...*, then click *Create Project*. Now you should be able to see all the files in the repo from Rstudio, and edit and push to update your github repo.



The screenshot shows the 'New Project' dialog box in RStudio, specifically the 'Clone Git Repository' tab. On the left is the R logo. The main area contains the following fields and buttons:

- Repository URL:** A text box containing `https://github.com/git4casey/R_Workshop.git`.
- Project directory name:** A text box containing `R_Workshop`.
- Create project as subdirectory of:** A text box with a small icon, followed by a **Browse...** button.
- Buttons:** A **Back** button at the top left, and **Create Project** and **Cancel** buttons at the bottom right.
- Checkbox:** An unchecked checkbox labeled 'Open in new session' at the bottom left.

GitHub

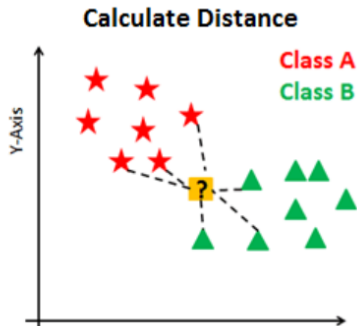


Figure 3

Using R for k-Nearest Neighbors (KNN)

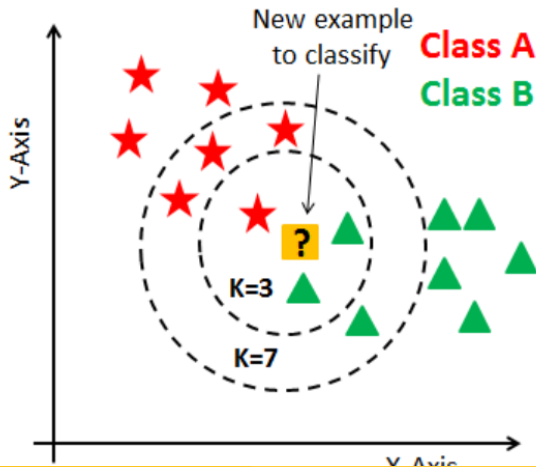
KNN concepts

- The KNN or k-nearest neighbors algorithm is one of the simplest machine learning algorithms and is an example of instance-based learning, where new data are classified based on stored, labeled instances.
- More specifically, the distance between the stored data and the new instance is calculated by means of some kind of a similarity measure.
- This similarity measure is typically expressed by a distance measure such as the Euclidean distance, cosine similarity or the Manhattan distance.



KNN

The number of neighbors(K) in KNN is a hyperparameter that you need choose at the time of model building. Look at the case below: The question is how to choose the optimal number of k neighbors?



KNN

No unified answer that suits all kind of data sets. In general, requires test on different k and validate the performance. Research has also shown that

- a small amount of neighbors are most flexible fit which will have low bias but high variance
- a large number of neighbors will have a smoother decision boundary, which implies lower variance but higher bias.

KNN Iris Example in R

Let's use famous iris dataset to understand how knn works in R. To inspect the data, some simple summary:

```
##      Sepal.Length      Sepal.Width      Petal.Length      Petal.Width
## Min.      :4.300      Min.      :2.000      Min.      :1.000      Min.      :0.100
## 1st Qu.:5.100      1st Qu.:2.800      1st Qu.:1.600      1st Qu.:0.300
## Median :5.800      Median :3.000      Median :4.350      Median :1.300
## Mean    :5.843      Mean    :3.057      Mean    :3.758      Mean    :1.199
## 3rd Qu.:6.400      3rd Qu.:3.300      3rd Qu.:5.100      3rd Qu.:1.800
## Max.    :7.900      Max.    :4.400      Max.    :6.900      Max.    :2.500
##
##      Species
## setosa      :50
## versicolor:50
## virginica   :50
##
##
##
```

KNN Iris Example in R

To visualize if there is any correlation between variables.

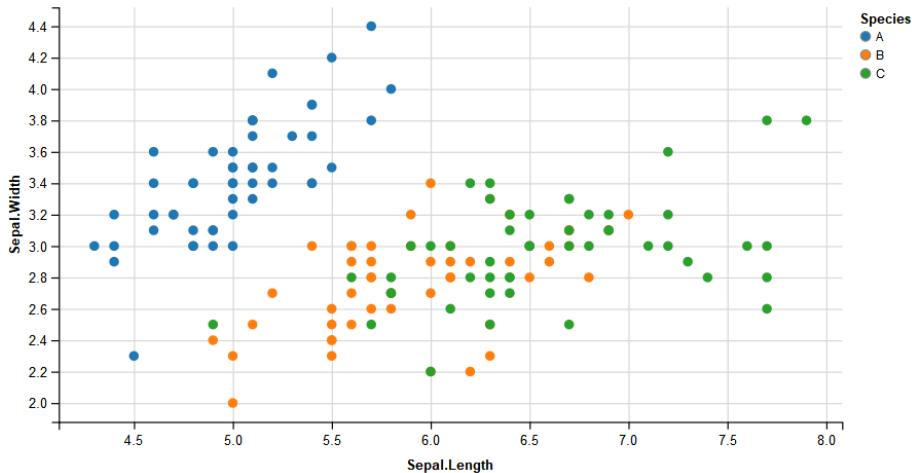


Figure 5: flowchart

KNN Iris Example in R

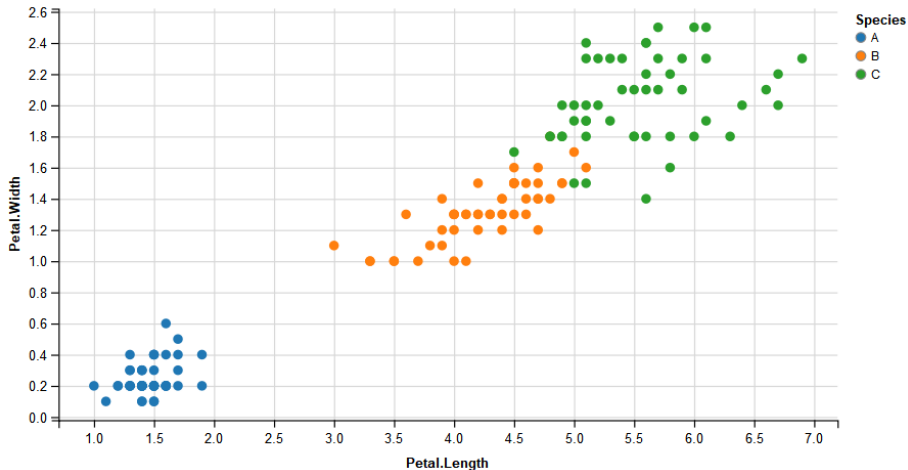


Figure 6: flowchart

KNN Iris Example in R

- Divide the data set into two parts: a training set and a test set. (2/3, 1/3 split)
- Using the `knn()` function, which uses the Euclidian distance measure in order to find the k-nearest neighbours to your new, unknown instance.

```
> library(class)
> iris_pred <- knn(train = iris_training, test = iris_test, cl = iris_training$cl)
> iris_pred
```

```
## [1] A A A A A A A A A A A A A A A B B B B B B B B B C C B B B
## [36] C C C C C C C C C C C C C C
## Levels: A B C
```


KNN Iris Example in R

- Compare the predicted class to the test labels.

```
##      test_lable pred_lable diff
## 1          A          A      1
## 2          A          A      1
## 3          A          A      1
## 4          A          A      1
## 5          A          A      1
## 6          A          A      1

## [1] "Accuracy: 91.6666666666667%"
```