Best Practices for Creating Reproducible Analytics within Teams Using R

Li Li Chia

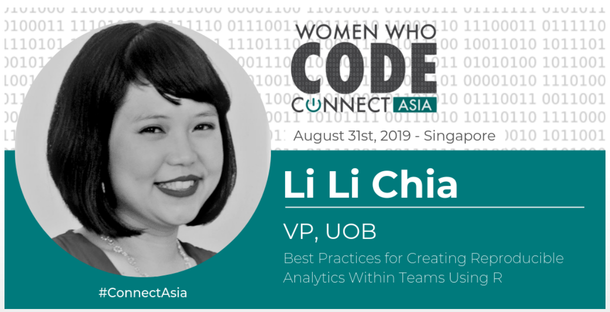
31 August 2019

Speaker Bio: Li Li Chia

Li Li is a VP from UOB's Group Compliance - Analytics, Automation and AI team. Among others, the work involves analytics for regime-specific sanctions look-back on payment and trades, analysing transactions patterns and optimising efficiency of name screening systems.

Prior to UOB, Li Li has led projects in the Singapore community care and healthcare space. She has also implemented turnkey solutions and business intelligence applications in light manufacturing, fast moving consumer goods and property development. She has over 10 years of experience in technology and project management.

Li Li holds a Masters of Technology in Knowledge Engineering from the National University of Singapore.



Agenda

1. Introduction to R and R studio
2. Usage of R markdown
3. Importing Data
4. Data Summarization
5. Data Transformation
6. Data Visualization
7. User Defined Functions
8. R Studio and Version Control

Introduction to R

is a programming language and free software environment for statistical computing and graphics which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues.

The R language is widely used among statisticians and data miners for interpreting and visualizing data. Below is a list of some high profile companies using R.

- Microsoft, Google, Facebook
- BBC , New York Times
- Grab, Uber

References:

1. The R Foundation for Statistical Computing, https://www.r-project.org/about.html
2. Revolution Analytics, https://blog.revolutionanalytics.com/2013/05/companies-using-open-source-r-in-2013.html
3. Revolution Analytics, https://bbc.github.io/rcookbook/
4. Tech In Asia https://www.techinasia.com/grab-runs-data-science-team

Introduction to R Studio

RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

RStudio is available in open source and commercial editions and runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio Server Pro (Debian/Ubuntu, RedHat/CentOS, and SUSE Linux).

RStudio https://www.rstudio.com/products/rstudio/

Usage of R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. This makes analytics **easily reproducible**! You can embed a code chunk like this.

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

Data Import

- Connect to your database and import data via ODBC.

```
library(odbc)
odbc <- dbConnect(odbc::odbc(), dsn = "PostgreSQL")
odbc_result <- dbReadTable(odbc, "flights")
```

- Otherwise, flat file data import from CSV also works.
- Parameterize the data source in the RMarkdown header.

```
params$dataSourceFile
```

```
## [1] "data/total-air-passenger-arrivals-by-country.csv"
```

```
df_arrivals<-read.csv(params$dataSourceFile)
```

Source: Data.gov.sg, https://data.gov.sg/dataset/air-passenger-arrivals-total-by-region-and-selected-country-of-embarkation

Data Summarization, Transformation and Visualization

We use the following libraries and their dependencies for summarization, transformation, visualization

```
# For data reshaping and summarization
library(dplyr)
# For graphical visualization
library(ggplot2)
# For prettier standardized graphics
library(ggthemes)
# For more fonts
library(extrafont)
# For geo-spatial visualization
library(maps)
```

Data Summarization

Summarize imported data for easy reconciliation

```
dim(df_arrivals) # dim provides the number of rows and columns of data
```

```
## [1] 7722    5
```

```
str(df_arrivals) # str gives the structure of the data frame
```

```
## 'data.frame':    7722 obs. of  5 variables:
##  $ month  : Factor w/ 702 levels "1961-01","1961-02",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ level_1: Factor w/ 1 level "Number Of Air Passenger Arrivals": 1 1 1 1 1 1 1 1 1 1 ...
##  $ level_2: Factor w/ 3 levels "Europe","North East Asia",..: 3 3 3 3 3 2 2 2 1 1 ...
##  $ level_3: Factor w/ 11 levels "China","France",..: 7 5 9 8 11 1 4 6 10 2 ...
##  $ value  : Factor w/ 6133 levels "-","1000","100029",..: 6133 1697 507 1093 6133 6133 5959 120
```

Data Transformation

Transforming the data structure

- stringsAsFactors=TRUE is R default behaviour
- We can change the setting when importing data or converting to dataframe.

```
df_arrivals <- data.frame(df_arrivals,stringsAsFactors = FALSE)
df_arrivals <- read.csv(params$dataSourceFile,stringsAsFactors = FALSE)

# Check the structure after setting stringsAsFactors=FALSE
str(df_arrivals)
```

```
## 'data.frame':    7722 obs. of  5 variables:
##  $ month  : chr  "1961-01" "1961-01" "1961-01" "1961-01" ...
##  $ level_1: chr  "Number Of Air Passenger Arrivals" "Number Of Air Passenger Arrivals" "Number C
##  $ level_2: chr  "South East Asia" "South East Asia" "South East Asia" "South East Asia" ...
##  $ level_3: chr  "Malaysia" "Indonesia" "Thailand" "Philippines" ...
##  $ value  : chr  "na" "1687" "1172" "139" ...
```

Data Transformation

Transforming the data structure

- Converting to numeric
- Handling of NA values

```
# Unsuccessfully converted values become NA, so we get the index
na.index <- (is.na(as.numeric(df_arrivals$value)))
```

```
## Warning: NAs introduced by coercion
```

```
# Visual check the values of data that became NA
table(df_arrivals$value[na.index])
```

```
##
##     -   na
##     1 1368
```

Data Transformation

- Converting to numeric
- Handling of NA values

```
# We are OK with the values that became NA, so we convert to numeric
df_arrivals$value <- (as.numeric(df_arrivals$value))
```

```
## Warning: NAs introduced by coercion
```

```
# Check the df structure
str(df_arrivals$value)
```

```
##  num [1:7722] NA 1687 1172 139 NA ...
```

Data Transformation

Converting to date format. Why do it?

- Consistency of calculations
- Easier plotting of monthly/yearly totals
- Use *help(as.Date)* for more info on formatting

```
# Update to YYYY-MM-DD format by adding -01
df_arrivals$month <- paste(df_arrivals$month,'01',sep='-')

# Change the format to  Date format
df_arrivals$month <- as.Date(df_arrivals$month, "%Y-%m-%d")

# Check the df structure
str(df_arrivals$month)
```

```
##  Date[1:7722], format: "1961-01-01" "1961-01-01" "1961-01-01" "1961-01-01" "1961-01-01" ...
```

Data Transformation

Labeling Data

It's important to label the data and column with sensible names for easy readability. When people find it easier to understand your scripts, the results will be easier to reproduce.

```
colnames(df_arrivals)[colnames(df_arrivals)=="level_2"] <- "region"
colnames(df_arrivals)[colnames(df_arrivals)=="level_3"] <- "country"

#After renaming, check the names.
names(df_arrivals)
```

```
## [1] "month"   "level_1" "region"  "country" "value"
```

Data Transformation

We use **dplyr** library **select** to select columns to analyse

```
# Alternate code in base R , same result
#   df_arrivals[,c("month","region","country","value")]

df_arrivals<-dplyr::select(df_arrivals, # the first parameter is the df
                       month, region, country, value) # folowed by  columns

#Check only the selected columns remain
names(df_arrivals)
```

```
## [1] "month"   "region"  "country" "value"
```

Data Transformation

Today, we want to analyze the most recent 12 months of data.

First, we find the most recent month in the dataset *month_max*

```
month_max <- max(df_arrivals$month)
month_max
```

```
## [1] "2019-06-01"
```

Then, we generate a sequence *month_seq* starting from month_max with length 2, decreasing by 12 months. **Good reason to do date conversion!**

```
month_seq <- seq(month_max,length=2, by="-12 months")
month_seq
```

```
## [1] "2019-06-01" "2018-06-01"
```

Data Transformation

Subsequently, we take the second value of *month_seq* to be the lower bound of the date range for extraction.

Finally, we use **dplyr::filter** to filter the rows we want to analyse

```
# The second item in the sequence will be the lower bound
month_lowerbound <- month_seq[2]
month_lowerbound
```

```
## [1] "2018-06-01"
```

```
# filter the latest 12 months of data in the arrivals
df_arrivals <- filter(df_arrivals, month > month_lowerbound)
```

Data Transformation

We use **dplyr::count** to verify filtering is successful

```
dplyr::count(df_arrivals,month)
```

```
## # A tibble: 12 x 2
##    month          n
##    <date>     <int>
##  1 2018-07-01    11
##  2 2018-08-01    11
##  3 2018-09-01    11
##  4 2018-10-01    11
##  5 2018-11-01    11
##  6 2018-12-01    11
##  7 2019-01-01    11
##  8 2019-02-01    11
##  9 2019-03-01    11
## 10 2019-04-01    11
## 11 2019-05-01    11
## 12 2019-06-01    11
```

Data Visualization

We use **dplyr::group_by** to group the data by month, subsequently, we use **dplyr::summarise** to sum up the value of each month (ie: grouped variable)

```
# We use the %>% for easier readability, actually it is the same as
#       summarise(month=format(month,"%Y-%m"),value=sum(value))
#
df_arrivals %>%
 group_by(month=format(month,"%Y-%m"))
```

```
## # A tibble: 132 x 4
## # Groups:   month [12]
##    month   region          country          value
##    <chr>   <chr>           <chr>            <dbl>
##  1 2018-07 South East Asia Malaysia        294720
##  2 2018-07 South East Asia Indonesia       360295
##  3 2018-07 South East Asia Thailand        257334
##  4 2018-07 South East Asia Philippines     118953
##  5 2018-07 South East Asia Vietnam         125391
##  6 2018-07 North East Asia China           330589
```

Data Visualization

We use **dplyr::group_by** to group the data by country, subsequently, we use **dplyr::summarise** to sum up the value of each country (ie: grouped variable). Then, we use **dplyr::arrange(desc())** to arrange the summed values in desc order. Finally, we use *head(5)* to display the Top 5 summed values

```
# We use the %>% for easier readability, actually it is the same as
#      summarise(group_by(df_arrivals,country),value=sum(value))
#
df_arrivals %>% group_by(country) %>% summarise(value=sum(value)) %>%
                      arrange(desc(value)) %>% head(5)
```

```
## # A tibble: 5 x 2
##   country     value
##   <chr>       <dbl>
## 1 Indonesia 4017371
## 2 China     3516166
## 3 Malaysia  3503182
```

Data Visualization

Use **ggtheme::theme** to set a standard theme for ggplot

- corporate colour
- corporate font family
- other elements similar to css/html

```
theme_WWCPlot <-
  theme(axis.text.y = element_text(colour = c("#007a7c"),
        face = "bold", size = 8),
        axis.text.x = element_text(colour = c("#007a7c"),
        face = "bold",  size = 8,
        angle = 90, hjust = 1),
        text = element_text(family = "Microsoft Sans Serif",
        size = 8),
        legend.position="right")
```

Data Visualization

How **ggplot2::ggplot** grammar works…

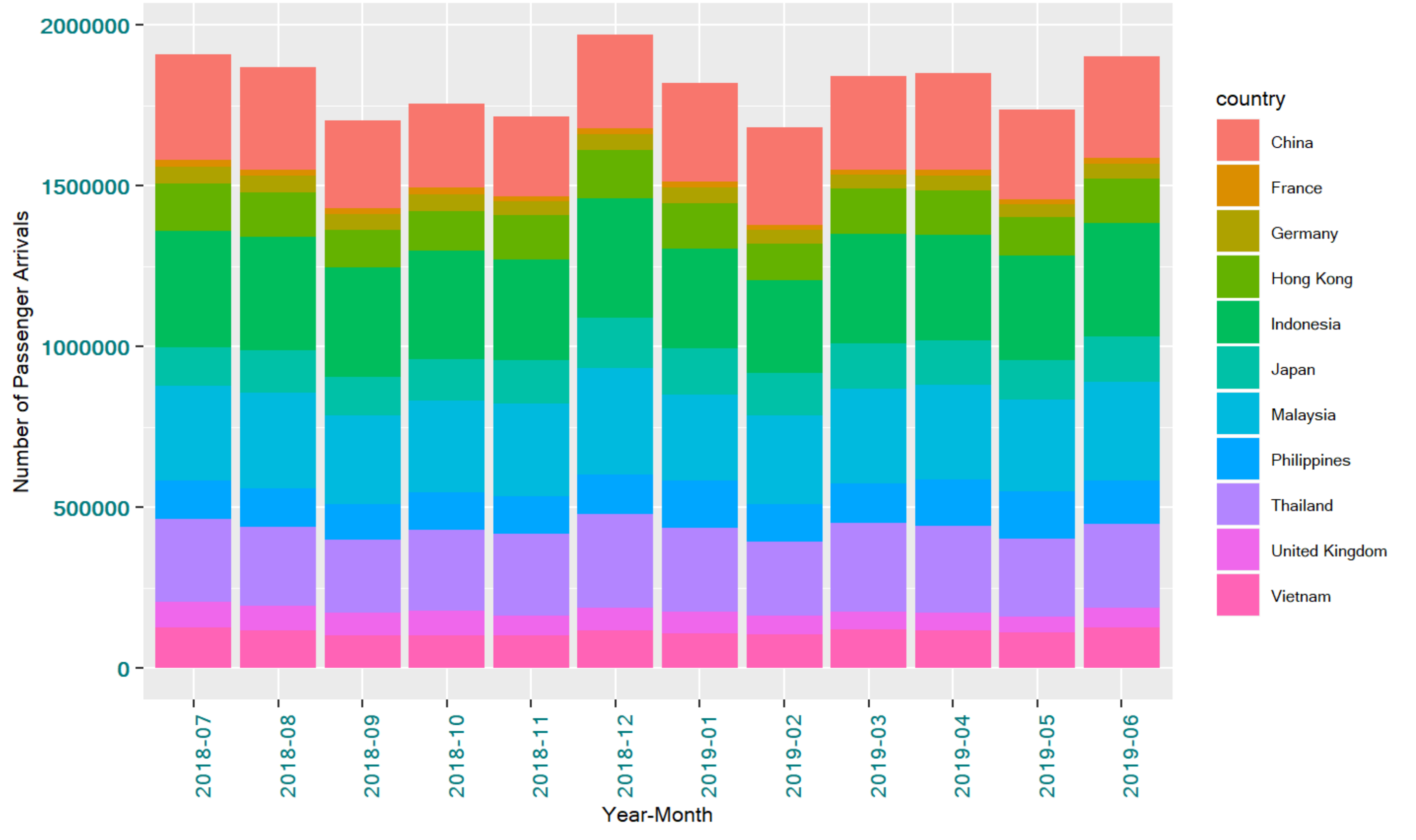
- First we supply a *data* set
- Second, we set aesthetic mapping using *aes()*
- Then, we add layers *geom_bar(stat="identity")* for the barchart
- And labels *xlab* and *ylab*
- Bar charts are automatically stacked when multiple bars are placed together

```
plot1 <- ggplot(data=df_arrivals) +
                aes(x=format(month,"%Y-%m"), y=value, fill=country) +
                geom_bar(stat="identity") +
                xlab('Year-Month') +
                ylab('Number of Passenger Arrivals')
```

Data Visualization

Finally, we add the theme_WWCPlot and display the bar plot.

```
plot1 + theme_WWCPlot
```



Data Visualization

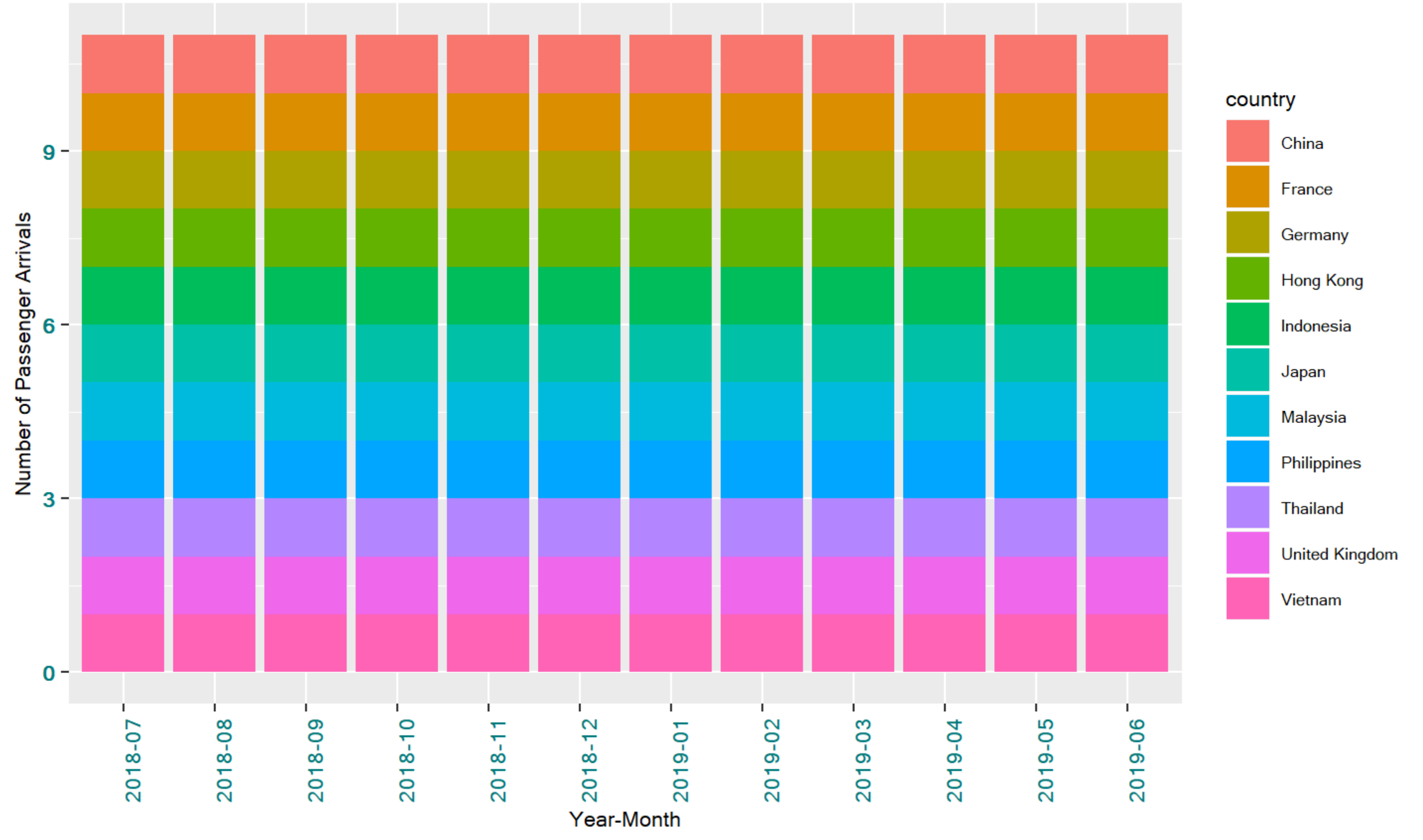
- Puzzled about the geom_bar(stat="identity")?
- The default value of y is the frequency
- When you see the next chart, it will make more sense why we change the default setting

```
plot12 <- ggplot(data=df_arrivals) +
                aes(x=format(month,"%Y-%m"), fill=country) +
                geom_bar() +
                xlab('Year-Month') +
                ylab('Number of Passenger Arrivals')
```

Data Visualization

All countries/regions appear as one observation per month.

```
plot12 + theme_WWCPlot
```



Data Visualization

Let's repeat the **ggplot2::ggplot** grammar again…

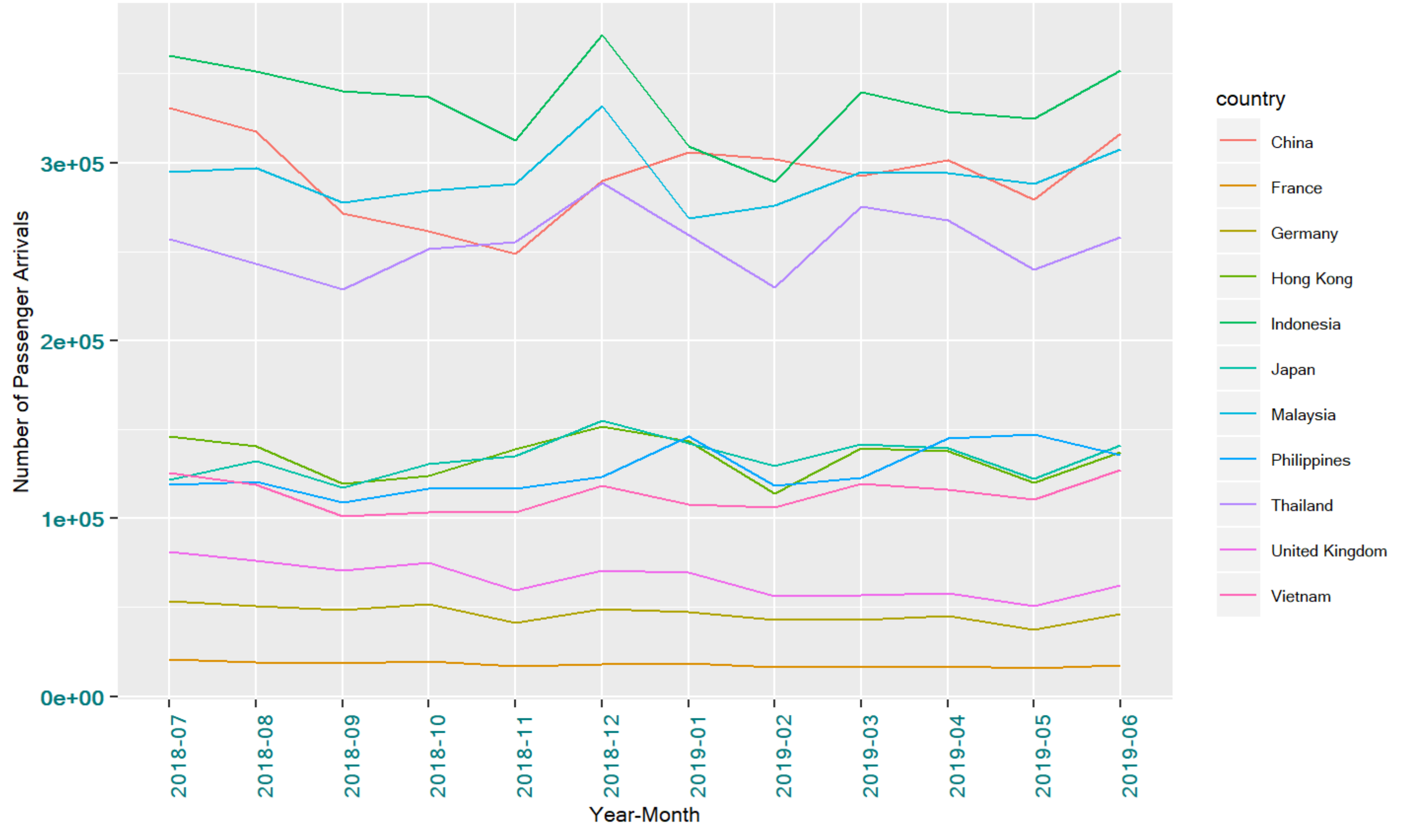
- First we supply a *data* set
- Second, we set aesthetic mapping using *aes()*
- Then, we add layers *geom_line(stat="identity")* for the line chart
- And finally labels *xlab* and *ylab*

```
plot2 <- ggplot(data=df_arrivals)+
                aes(x=format(as.Date(month),"%Y-%m"),
                    y=value, group=country,color=country) +
                stat_summary(fun.y=sum,geom="line") +
                xlab('Year-Month') + ylab('Number of Passenger Arrivals')
```

Data Visualization

Finally, we add the theme_WWCPlot and display the Line Plot

```
plot2 + theme_WWCPlot
```



Data Visualization for Map

Retrieve maps::map_data and review the structure

```r
world_map <- map_data("world") # retrieve world map data
str(world_map) #see the structure
```

```
## 'data.frame':    99338 obs. of  6 variables:
##  $ long     : num  -69.9 -69.9 -69.9 -70 -70.1 ...
##  $ lat      : num  12.5 12.4 12.4 12.5 12.5 ...
##  $ group    : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ order    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ region   : chr  "Aruba" "Aruba" "Aruba" "Aruba" ...
##  $ subregion: chr  NA NA NA NA ...
```

```r
world_map # see first few rows
```

```
##                  long           lat group order
## 1       -69.89912415   12.452001572     1     1
## 2       -69.89570618   12.422998428     1     2
```

Data Transformation for Map

We then take country statistics, normalize the country/region information and join

```
# We save the count we did previously to a variable
count_ctry <- df_arrivals %>% group_by(country) %>% summarise(value=sum(value))
# We are familiar with the world_map data so we update UK value
if("United Kingdom" %in% count_ctry$country)
{   index.unitedkingdom <- (count_ctry$country)=="United Kingdom"
    count_ctry$country[index.unitedkingdom]<-"UK"}
# join the count and world map data
arrivals.map<-left_join(count_ctry,world_map,by=c("country"="region"))
# see the first few rows
head(arrivals.map)
```

```
## # A tibble: 6 x 7
##   country   value  long   lat group order subregion
##   <chr>     <dbl> <dbl> <dbl> <dbl> <int> <chr>
## 1 China   3516166  111.  20.0   418 28698 1
## 2 China   3516166  111.  19.9   418 28699 1
```

Data Visualization for Map

ggplot2::ggplot grammar for maps is similar

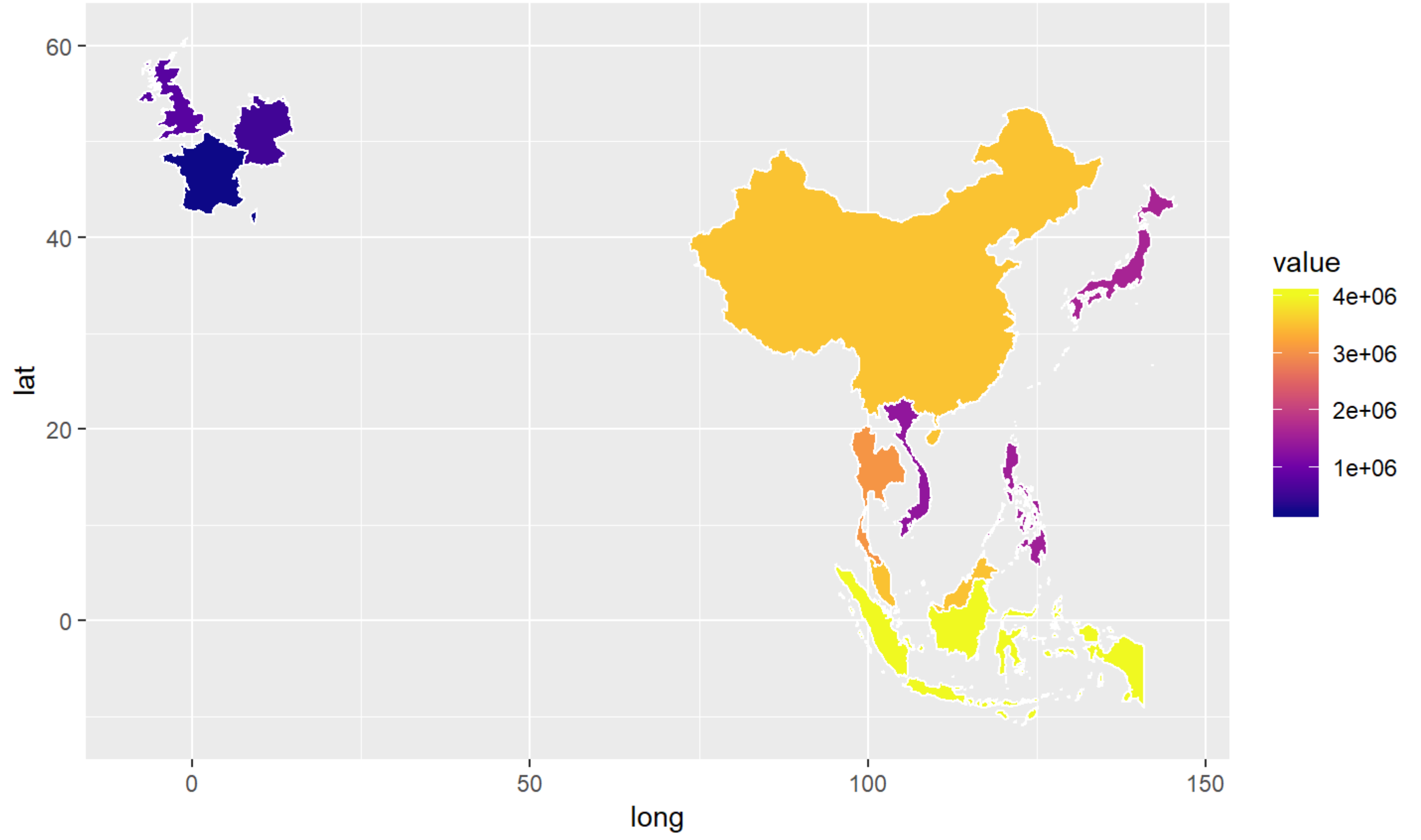
- First we supply a *data* set
- Second, we set aesthetic mapping using *aes()*
- Then, we add layers *geom_polygon* to create a chloroplethmap
- *scale_fill_viridis_c* layers sets the scale color

```
mapplot1 <- ggplot(arrivals.map) +
              aes(long, lat, group = group) +
                geom_polygon(aes(fill = value), color = "white")+
                scale_fill_viridis_c(option = "C")
```

Data Visualization for Map

Let's display the map!

```
mapplot1
```



User Defined Functions

Good candidate for UDF

- improve readability
- ease maintenance
- likely to be reusable

```r
# We get the maximum month values
month_max <- max(df_arrivals$month)
# We generate a sequence starting from month_max with length 2
month_seq <- seq(month_max,length=2, by="-12 months")
# The second item in the sequence will be the lower bound
month_lowerbound <- month_seq[2]
# Let's view the final variable
month_lowerbound
```

```
## [1] "2018-06-01"
```

User Defined Functions

Creating the UDF

```r
getTwelveMonthAgoDate <- function(date_data){
    # We get the maximum month values
    month_max <- max(date_data)
    # We generate a sequence starting from month_max with length 2
    month_seq <- seq(month_max,length=2, by="-12 months")
    # The second item in the sequence will be the lower bound
    month_lowerbound <- month_seq[2]
    # Garbage collection
    rm(month_max, month_seq)
    return(month_lowerbound)
}

#Call the function
getTwelveMonthAgoDate(df_arrivals$month)
```

```
## [1] "2018-06-01"
```

Combining R Markdown and UDF

General good practices

- Readability is key
- Ease of maintenance and reusability is another bonus
- Store functions in central folder
- Call all common functions using **source("folderpath")**

R Studio and Version Control

- R Studio is compatible with code management tools *git* and *SVN* with minimal configuration
- It's possible to create a Project from Version Control directly from R Studio
- Similarly, you can *commit* or *revert* changes as well as view the *log* or *diff*
- AVOID doing version control using file naming convention.

https://support.rstudio.com/hc/en-us/articles/200532077-Version-Control-with-Git-and-SVN

Bonus: Social Network Analysis

```
# Load libraries
library(igraph)
library(networkD3)

# Create connection dataframe - List of flows with intensity for each flow
links <- data.frame(
  source=c("group_A","group_A", "group_C", "group_C", "group_G"),
  target=c("group_C","group_D", "group_F", "group_G", "group_A"),
  value=c(2,3, 2, 3, 1) )

# From these flows we need to create a node data frame:
#                 it lists every entities involved in the flow
nodes <- data.frame(
  name=c(as.character(links$source),
  as.character(links$target)) %>% unique()
)
```

Reference: https://www.r-graph-gallery.com/321-introduction-to-interactive-sankey-diagram-2.html

Bonus: Social Network Analysis

```
# Turn it into igraph object
network <- graph_from_data_frame(d=links,vertices=nodes,directed=T)
#Make the plot
plot(network)
```

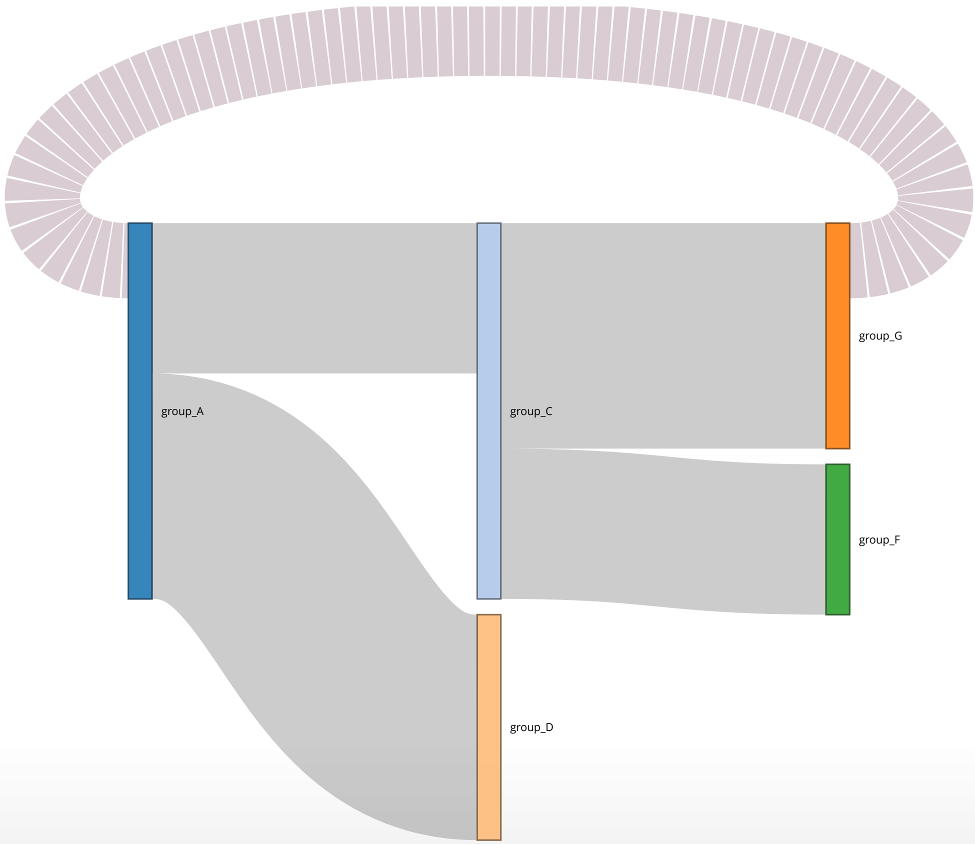
Bonus Visualization: Sankey Diagram

```
# With networkD3, connection must be provided using id, not using real name
# like in the links dataframe.. So we need to reformat it.
links$IDsource <- match(links$source, nodes$name)-1
links$IDtarget <- match(links$target, nodes$name)-1

# Make the Network
p <- sankeyNetwork(Links = links, Nodes = nodes,
              Source = "IDsource", Target = "IDtarget",
              Value = "value", NodeID = "name",
              sinksRight=FALSE)
```

Bonus Visualization: Sankey Diagram

```
# Display the network
p
```



Bonus Visualization: Word Cloud

Load libraries and dependencies

```
# Text Mining
library(tm)
# wordcloud
library(wordcloud)
# colour palette
library(RColorBrewer)
```

Load data

```
filePath <- "data/ndr_2019_fulltext.txt"
text <- readLines(filePath,warn=FALSE)
```

Source: Prime Minister's Office Singapore, https://www.pmo.gov.sg/Newsroom/National-Day-Rally-2019

Bonus Visualization: Word Cloud

Prepare the data using *library(tm)*

```
# Load the data as a corpus
docs <- Corpus(VectorSource(text))
# Convert the text to lower case
docs <- tm_map(docs, content_transformer(tolower))
# Remove numbers
docs <- tm_map(docs, removeNumbers)
# Remove english common stopwords
docs <- tm_map(docs, removeWords, stopwords("english"))
#docs <-tm_map(docs,removeWords,c("hello"))
# Remove punctuations
docs <- tm_map(docs, removePunctuation)
# Eliminate extra white spaces
docs <- tm_map(docs, stripWhitespace)
```

Bonus Visualization: Word Cloud

Generate using *library(wordcloud)*

```
wordcloud(docs, scale = c(5, 0.5),max.words = 70,random.order = FALSE,
               use.r.layout = FALSE, colors = brewer.pal(6,"Dark2"))
```