# R Markdown: An Introduction

*Chris Toffis*

*PSI Conference Tuesday 4th July 2019*

## R Markdown: What is it?

An R Markdown document enables one to mix text with code and associated output. Through the 'Knit' feature within RStudio, you can generate a document (HTML, Word or PDF) that can be shared. It is hoped that this document will give the user some insight and confidence on how to read in and manipulate some dummy data stored in a SAS dataset.

To create a new R Markdown document, within RStudio:

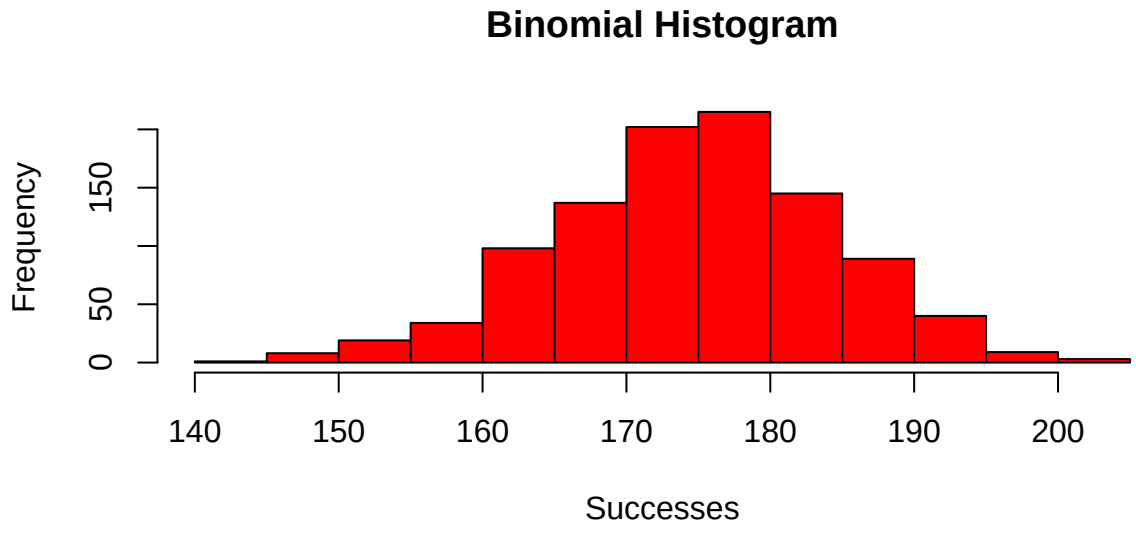
1. Go to File -> New File -> R Markdown...
2. Give the document and title and author
3. Depending on your installation you can decide how the file is output, for most users 'Word' should be sufficient.

This handout was created using RStudio 1.2.1335 with R version 3.6.0. The packages used are the latest as of Wednesday 22nd May 2019.

## Example

Let us say we wanted to plot a histogram of the binomial distribution for 1000 trials with probability of 0.5 and 350 successes. We would insert an R 'code chunk' beneath this plain text and run it, like so:

```
set.seed(20190604) # For reproducibility
nums <- rbinom(n = 1000, size = 350, p = 0.5)
hist(nums, col = "red", xlab = "Successes", main = "Binomial Histogram")
```



## Loading Packages

We use `library()` as below:

```
- library(knitr)
This allows us to create our R Markdown documents.
- library(tidyverse)
This package actually loads many recommended data-wrangling packages available within R.
```

Libraries provide additional functionality to R.

## Functions

We will be exploring several functions to manipulate our data:

- `filter`(*dataset*, [list of conditions])
  Apply conditions to pick certain records; works like the `where` clause in SAS.
- `select`(*dataset*, [list of variables])
  Selects variables in the dataset to retain; works like `keep` clause in SAS.
- `head`(*dataset* [, number of rows to display])
  Similar to applying the `noobs=` option in SAS.
- `distinct`(*dataset*)
  Pick out distinct rows in the dataset.
- `arrange`(*dataset*, [list of variable(s)])
  Sorts the data by the variables provided.
- `mutate`(*dataset*, *newvar* = . . . [, *newvar2* = . . . ])
  Adds additional column(s) to your dataset.
- `group_by`(*dataset*, [list of variables])
  Assigns grouping to the dataset. Grouping is embedded - use `ungroup()` to remove.
- summarise(*dataset*, [stats to use])
  Perform statistical calculations on your data; similar to `PROC FREQ/MEANS`.
- `gather`(*dataset*, *key*, *value*)
  Transposes data from wide-to-long; similar to `PROC TRANSPOSE`.
- `spread`(*dataset*, *key*, *value*)
  Transposes data from long-to-wide; similar to `PROC TRANSPOSE`.

## Manipulating Data: Pipes

The 'traditional' way in R.

```
adlb <- haven::read_sas("adlb.sas7bdat")
t1 <- filter(adlb, fasfl == 'Y' & !(avisitn %in% c(-1, 99)) & paramcd == 'HGB')
t2 <- select(t1, trt01p, avisit, aval)
head(t2, 6)
```

```
## # A tibble: 6 x 3
##   trt01p           avisit   aval
##   <chr>            <chr>   <dbl>
## 1 Comparator 20 mg DAY 1    10.6
## 2 Comparator 20 mg WEEK 2   12
## 3 Comparator 20 mg WEEK 4   11.1
## 4 Comparator 20 mg WEEK 6   11.2
## 5 Comparator 20 mg WEEK 8   12
## 6 Comparator 20 mg WEEK 12   8.6
```

This code is a little verbose; we have to define dataset `t1` in order to create `t2`. However, if we use something called a pipe - `%>%` - we can code in a more natural manner.

```
adlb %>%
  filter(., fasfl == 'Y' & !(avisitn %in% c(-1, 99)) & paramcd == 'HGB') %>%
  select(., trt01p, avisit, aval) %>%
  head(., 6)
```

```
## # A tibble: 6 x 3
##   trt01p           avisit   aval
##   <chr>            <chr>   <dbl>
## 1 Comparator 20 mg DAY 1    10.6
## 2 Comparator 20 mg WEEK 2   12
## 3 Comparator 20 mg WEEK 4   11.1
## 4 Comparator 20 mg WEEK 6   11.2
## 5 Comparator 20 mg WEEK 8   12
## 6 Comparator 20 mg WEEK 12   8.6
```

Which gives the same result. Note the use of periods: `%>%` takes whatever data/result comes before it and passes it to where the period appears in the function call after it. By default, the *first* position is usually where `%>%` passes the result to. Knowing this we can omit the periods (and commas) altogether.

```
adlb %>%
  filter(fasfl == 'Y' & !(avisitn %in% c(-1, 99)) & paramcd == 'HGB') %>%
  select(trt01p, avisit, aval) %>%
  head(6)
```

```
## # A tibble: 6 x 3
##   trt01p           avisit   aval
##   <chr>            <chr>   <dbl>
## 1 Comparator 20 mg DAY 1    10.6
## 2 Comparator 20 mg WEEK 2   12
## 3 Comparator 20 mg WEEK 4   11.1
## 4 Comparator 20 mg WEEK 6   11.2
## 5 Comparator 20 mg WEEK 8   12
## 6 Comparator 20 mg WEEK 12   8.6
```

Again, the result is the same. You may be wondering what the point of the 'period notation' is. It helps make explicit where the result on the left will be passed to. Consider the following

```
set.seed(20190604)
ex1 <- 1:5 %>% cbind(runif(5), .) #period is to the right of `runif(5)`
colnames(ex1)[2] <- "I'm on the right..."
ex1
```

```
##                 I'm on the right...
## [1,] 0.82837933                   1
## [2,] 0.11752394                   2
## [3,] 0.55173900                   3
## [4,] 0.37476756                   4
## [5,] 0.03367091                   5
```

Compare with

```
set.seed(20190604)
ex2 <- 1:5 %>% cbind(runif(5)) # same as cbind (., runif(5))
colnames(ex2)[1] <- "... and now I'm on the left"
ex2
```

```
##      ... and now I'm on the left
## [1,]                           1 0.82837933
## [2,]                           2 0.11752394
## [3,]                           3 0.55173900
## [4,]                           4 0.37476756
## [5,]                           5 0.03367091
```

## Manipulating Data: Intermediate

Let's say we want to replicate the results found in a standard lab output:

| | Comparator (N = xx) | Active (N = xx) | Total (N = xx) |
|---|---|---|---|
| Baseline | | | |
| n | xx | xx | xx |
| Mean (SD) | x.xx (x.xxx) | x.xx (x.xxx) | x.xx (x.xxx) |
| Median | xx.x | xx.x | xx.x |
| Q1, Q3 | xx.x, xx.x | xx.x, xx.x | xx.x, xx.x |
| Min, Max | xx.x, xx.x | xx.x, xx.x | xx.x, xx.x |

However in our data we have 4 treatment groups.

```
adlb %>%
  filter(fasfl == 'Y') %>%
  select(trt01pn, trt01p) %>%
  distinct %>%
  arrange(trt01pn)
```

```
## # A tibble: 4 x 2
##   trt01pn trt01p
##     <dbl> <chr>
## 1       1 Comparator 10 mg
## 2       2 Comparator 20 mg
## 3       3 Active 5 mg
## 4       4 Active 10 mg
```

We shall create a variable that will store whether a subject is on "Comparator" or "Active" using the function `mutate()`. We shall also only keeps rows with scheduled study visits and those pertaining to haemoglobin.

```
hgbdata <- adlb %>%
  filter(fasfl == 'Y' & !(avisitn %in% c(-1, 99)) & paramcd == 'HGB') %>%
  mutate(trt = factor(ifelse(trt01pn %in% c(1,2), "Comparator", "Active"),
                      levels = c("Comparator", "Active"))) %>%
                        #levels applies ordering, Comparator first
  select(avisitn, trt, aval)
head(hgbdata)
```

```
## # A tibble: 6 x 3
##   avisitn trt         aval
##     <dbl> <fct>      <dbl>
## 1       1 Comparator  10.6
## 2       2 Comparator  12
## 3       4 Comparator  11.1
## 4       6 Comparator  11.2
## 5       8 Comparator  12
```

```
## 6      12 Comparator   8.6
```

We can now compute some summary stats. We will need to apply a grouping using `group_by()` then use the `summarise()` function to specify the stats we want. Let's pick out n, mean and median for now.

```
hgbdata %>%
  group_by(avisitn, trt) %>%
  summarise(n      = n(),
            mean   = mean(aval),
            median = median(aval))
```

```
## # A tibble: 12 x 5
## # Groups:   avisitn [6]
##    avisitn trt            n  mean median
##      <dbl> <fct>      <int> <dbl>  <dbl>
##  1       1 Comparator    50  10.8   10.9
##  2       1 Active        49  11.4   11.4
##  3       2 Comparator    50  11.1   10.9
##  4       2 Active        49  10.8   10.7
##  5       4 Comparator    50  10.9   10.9
##  6       4 Active        49  11.0   10.9
##  7       6 Comparator    50  10.9   11.1
##  8       6 Active        49  10.9   11
##  9       8 Comparator    50  10.7   10.6
## 10       8 Active        49  10.9   11
## 11      12 Comparator    46  11.1   11.3
## 12      12 Active        48  10.8   10.9
```

Because we have made trt an ordered factor, the data appears how we would have defined it to appear i.e. Comparator first.

## Visualisation

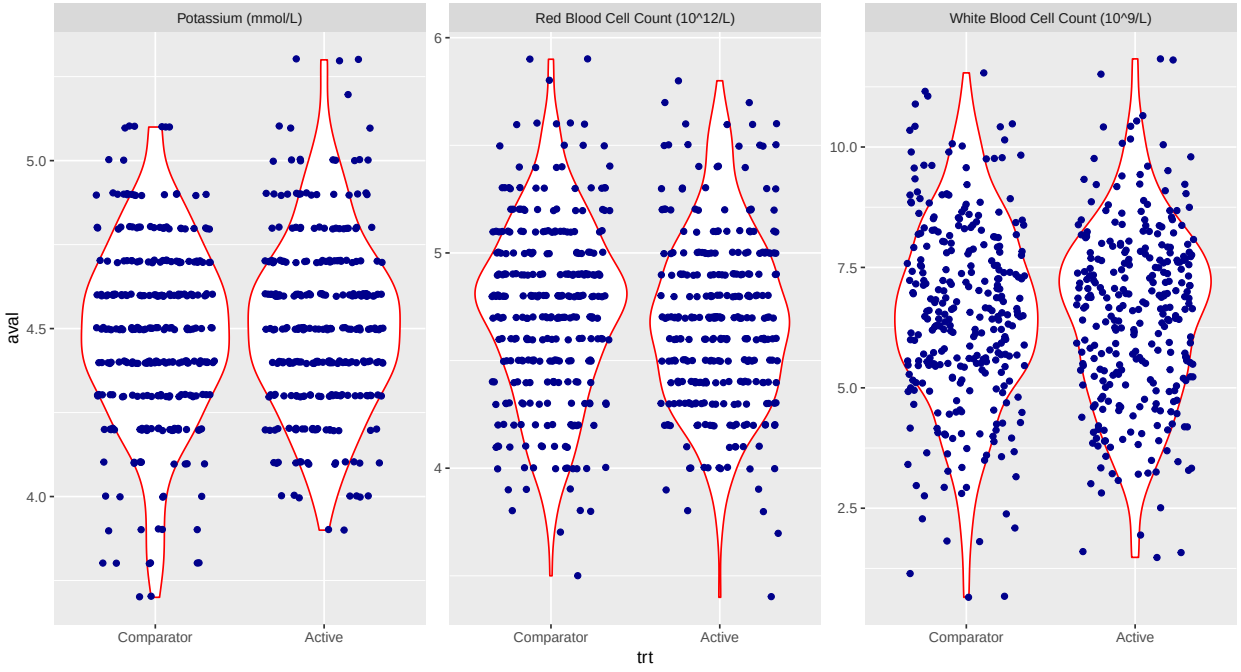
Let's plot the lab values for potassium, red and white blood cells by treatment and lab value. We create a variable called `labs` which will subset ADLB for the full analysis set and scheduled visits. We also collapse the treatment groups again.

```
labs  <- adlb %>%
  filter(fasfl == 'Y' & !(avisitn %in% c(-1, 99)) & paramcd %in% c("K", "RBC", "WBC")) %>%
  mutate(trt = factor(ifelse(trt01pn %in% c(1,2), "Comparator", "Active"),
                      levels = c("Comparator", "Active")))
```

Next we use the powerful plotting function `ggplot`. The syntax is a little different here with `+` used instead of `%>%` but the code should be easy to follow.

The first `ggplot` call specifies which variables to use on the axes. The two lines therafter request what type of plot to use (violin) and to 'jitter' the points so that they do not all overlap. The final line tells `ggplot` which parameter to provide plots for. The scales option allows each plot to have its own scale (the default is fixed, which would force each plot to have the same scale).

```
labs %>%
  ggplot(aes(x = trt, y = aval)) +
  geom_violin(colour = 'red') +
  geom_jitter(width = 0.35, color = 'darkblue') +
  facet_wrap(~param, scales = "free")
```




The violin plots provide insight to the distribution of the lab parameters (RBC for the comparator has a clearly defined peak compard to active) whilst we can see which labs appear to be discrete in nature (eg potassium) as opposed to continuous (eg WBC). NB the data are artificial so the actual interpretation of the real parameters may differ!

## Manipulating Data: More Advanced

Now we shall produce summary stats for hemoglobin and include the 'Total' column. Firstly, let us create a temp data set with all the information we need.

```
d1 <- adlb %>%
  filter(fasfl == 'Y' & !(avisitn %in% c(-1, 99)) & paramcd == "HGB") %>%
  mutate(trt = ifelse(trt01pn %in% c(1,2), "Comparator", "Active"))

d2 <- d1 %>%
  mutate(trt = "Total") %>% #creates the 'Total' column
  bind_rows(d1) %>% #stacks the 'Total' column data with the distinct treatments
  mutate(trt = factor(trt,
                      levels = c("Comparator", "Active", "Total")))
```

Compute all the standard summary stats we need and round the results to 1 decimal place.

```
d3 <- d2 %>%
  group_by(avisit, trt) %>%
  summarise(n            = n(),
            `Mean (SD)`  = paste0(round(mean(aval), 2), " (", round(sd(aval), 3), ")"),
            Median       = round(median(aval), 1),
            `Q1, Q3`     = paste0(round(quantile(aval, p = 0.25), 1), ", ",
                                  round(quantile(aval, p = 0.75), 1)),
            `Min, Max`   = paste0(round(min(aval), 1), ", ",
                                  round(max(aval), 1)))
head(d3, 6)
```

```
## # A tibble: 6 x 7
## # Groups:   avisit [2]
##   avisit  trt            n `Mean (SD)`   Median `Q1, Q3`   `Min, Max`
##   <chr>   <fct>      <int> <chr>          <dbl> <chr>      <chr>
## 1 DAY 1   Comparator    50 10.83 (1.297)   10.9 10.1, 11.5 7.1, 13.6
## 2 DAY 1   Active        49 11.35 (0.989)   11.4 10.7, 11.9 9.2, 13.9
## 3 DAY 1   Total         99 11.09 (1.179)   11.1 10.4, 11.9 7.1, 13.9
## 4 WEEK 12 Comparator    46 11.1 (1.153)    11.3 10.3, 11.8 8.6, 13.7
## 5 WEEK 12 Active        48 10.85 (1.202)   10.9 10.1, 11.7 8.5, 13.2
## 6 WEEK 12 Total         94 10.97 (1.179)   11.1 10.1, 11.9 8.5, 13.7
```

We shall now rotate our data to get it in the form in the table. First, we use `gather()`.

```
d4 <- d3 %>%
  gather(stat, value, n, `Mean (SD)`, Median, `Q1, Q3`, `Min, Max`) %>%
  mutate(stat = factor(stat, levels = c("n", "Mean (SD)", "Median", "Q1, Q3", "Min, Max")))
head(d4, 6)
```

```
## # A tibble: 6 x 4
## # Groups:   avisit [2]
##   avisit  trt        stat  value
##   <chr>   <fct>      <fct> <chr>
## 1 DAY 1   Comparator n     50
## 2 DAY 1   Active     n     49
## 3 DAY 1   Total      n     99
## 4 WEEK 12 Comparator n     46
## 5 WEEK 12 Active     n     48
## 6 WEEK 12 Total      n     94
```

What we have done here is stack all the summary stats into a column variable called `stat` and the corresponding value associated in another column called `value`. `avisitn` and `trt` remain unchanged. The additional line of adding of factor for the summary statistics ensures that they come out in this expected order when we transpose them. Finally, we rotate the data to get `trt` for our column headers with the summary stats populated below using `spread()`.

```
d5 <- d4 %>%
  spread(trt, value)
head(d5, 5)
```

```
## # A tibble: 5 x 5
## # Groups:   avisit [1]
##   avisit stat      Comparator    Active        Total
##   <chr>  <fct>     <chr>         <chr>         <chr>
## 1 DAY 1  n         50            49            99
## 2 DAY 1  Mean (SD) 10.83 (1.297) 11.35 (0.989) 11.09 (1.179)
## 3 DAY 1  Median    10.9          11.4          11.1
## 4 DAY 1  Q1, Q3    10.1, 11.5    10.7, 11.9    10.4, 11.9
## 5 DAY 1  Min, Max  7.1, 13.6     9.2, 13.9     7.1, 13.9
```

## Manipulating Data: More Advanced (1 Page Version)

I have broken this down into chunks for demonstration purposes but you can tie everything together in a couple of dozen lines of code to get the same output (without the need to create d1 to d5). This code provides summary stats for *all* lab parameters:

```
labs <- adlb %>%
  filter(fasfl == 'Y' & !(avisitn %in% c(-1,99))) %>%
  mutate(trt = ifelse(trt01pn %in% c(1,2), "Comparator", "Active"))

labs %>%
  mutate(trt = "Total") %>%
  bind_rows(labs) %>%
  mutate(trt = factor(trt,
                      levels = c("Comparator", "Active", "Total"))) %>%
  group_by(param, avisit, trt) %>%
  summarise(n            = n(),
            `Mean (SD)`  = paste0(round(mean(aval), 2), " (", round(sd(aval), 3), ")"),
            Median       = round(median(aval), 1),
            `Q1, Q3`     = paste0(round(quantile(aval, p = 0.25), 1), ", ",
                                  round(quantile(aval, p = 0.75), 1)),
            `Min, Max`   = paste0(round(min(aval), 1), ", ",
                                  round(max(aval), 1))) %>%
  gather(stat, value, n, `Mean (SD)`, Median, `Q1, Q3`, `Min, Max`) %>%
  mutate(stat = factor(stat, levels = c("n", "Mean (SD)", "Median", "Q1, Q3",
                                        "Min, Max"))) %>%
  spread(trt, value) %>%
  head(5)
```

```
## # A tibble: 5 x 6
## # Groups:   param, avisit [1]
##   param         avisit stat      Comparator    Active       Total
##   <chr>         <chr>  <fct>     <chr>         <chr>        <chr>
## 1 Albumin (g/L) DAY 1  n         50            49           99
## 2 Albumin (g/L) DAY 1  Mean (SD) 39.74 (3.155) 40.1 (3.236) 39.92 (3.184)
## 3 Albumin (g/L) DAY 1  Median    40            40           40
## 4 Albumin (g/L) DAY 1  Q1, Q3    37.2, 41.8    39, 42       38, 42
## 5 Albumin (g/L) DAY 1  Min, Max  34, 46        32, 47       32, 47
```