

# Getting Started in R

Saghir Bashir

This version was compiled on August 16, 2018

Are you curious to learn what R can do for you? Do you want to see how it works? If so, then this “Getting Started” guide is for you which introduces you to some key concepts using realistic examples. Using a real life dataset, it will also show you how to produce graphs and summary statistics using the “tidyverse”.

R | Tiyverse | Statistics | Data Science

## 1. Preface

This “Getting Started” guide will give you a flavour of what is R<sup>1</sup> and the tidyverse can do for you. To get the most out of this guide, read it whilst doing the examples and exercises using RStudio<sup>2</sup>.

**Experiment Safely.** Be brave and experiment with commands and options as it is an essential part of the learning process. Yes things can (and will) go “wrong”, like, getting error messages or deleting things that you created from using this guide. You can recover from most situations (e.g. by restarting R). To do this “safely” start with a *fresh* R session without any other data loaded (otherwise you could lose it).

## 2. Introduction

**Before Starting.** Make sure that you have:

1. R and RStudio installed
2. Files downloaded from: <https://ilustat.com/resources/> [UPDATE]
3. Double click on the file "GettingStartedinR.Rproj" which will open RStudio automatically with the right set up to work though this guide.

**Starting R & RStudio.** R starts automatically when you open RStudio (see Figure 1). The console starts with some information include about the version number, license and contributors. The last line is a standard prompt “>” that indicates R is ready and expecting a instructions to do something.

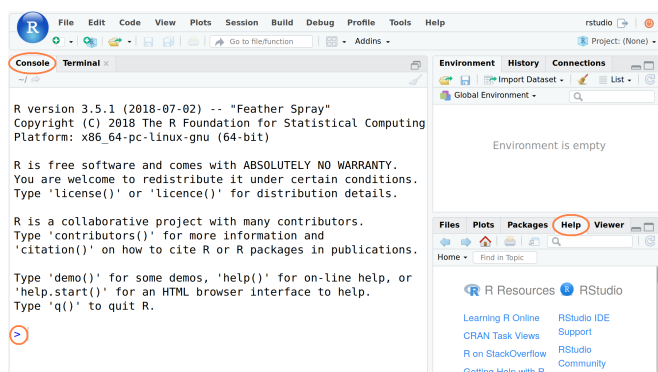


Fig. 1. RStudio Screenshot with Console on the left and Help tab in the bottom right

**Quitting R & RStudio.** When you quit RStudio you will be asked whether to Save workspace with two options:

- “Yes” – Your current R workspace will be restored next time you open RStudio.

<sup>1</sup>R project: <https://www.r-project.org/>

<sup>2</sup>RStudio IDE: <https://www.rstudio.com/products/RStudio/>

- “No” – You will start with a fresh R session next time you open RStudio. For now select “No” to prevent errors being carried over from previous sessions).

## 3. R Help

I strongly recommend that you learn how to use R’s useful and extensive built in help system as it is an essential part of finding solutions to your R programming problems. The easiest way is to use RStudio’s “Help” tab which has a search box (see Figure 1).

**help() function.** From the R “Console” you can use the `help()` function or `?`. For example, try the following two commands (which give the same thing):

```
help(mean)
?mean
```

**Keyword search.** To do a keyword search use the function `apropos()` with the keyword in double quotes ("keyword") or single quote ('keyword'). For example:

```
apropos("mean")
```

```
# [1] ".colMeans"      ".rowMeans"
# [3] "colMeans"       "cummean"
# [5] "kmeans"         "mean"
# [7] "mean_cl_boot"   "mean_cl_normal"
# [9] "mean_sdl"       "mean_se"
# [11] "mean.Date"      "mean.default"
# [13] "mean.difftime"  "mean.POSIXct"
# [15] "mean.POSIXlt"   "rowMeans"
# [17] "weighted.mean"
```

**Help Examples.** Use the `example()` function to run the examples at the end of the help for a function:

```
example(mean)
```

```
#
# mean> x <- c(0:10, 50)
#
# mean> xm <- mean(x)
#
# mean> c(xm, mean(x, trim = 0.10))
# [1] 8.75 5.50
```

**Searching On-line For R Help.** There are a lot of on-line resources that can help. However you must understand that blindly copying and pasting could be harmful and further it won’t help you to learn and develop. When you search on-line use [R] in your search term (e.g. “[R] summary statistics by group”). Note that often there is more than one solution to your problem. It is good to investigate the different options.

**Exercise.** Try the following (without worrying about what the help means):

- `help(median)`
- `?sd`
- `help(unique)`

## 4. Some R Concepts

In R speak, scalars, vectors/variables and datasets are called **objects**. To create objects (things) we have to use the assignment operator “<-”. For example, below, object `height` is assigned a value of 173 (type `height` shows its value):

```
height <- 173
height
```

```
# [1] 173
```

**Warning: R is case sensitive.** `age`, `Age` and `AGE` are different:

```
age <- 10
Age <- 50
AGE <- 99
```

```
age
```

```
# [1] 10
```

```
Age
```

```
# [1] 50
```

```
AGE
```

```
# [1] 99
```

**New lines.** R commands are usually separated by a new line but they can also be separated by a semicolon “;”.

```
Name <- "Leo"; Age <- 25; City <- "Lisbon"
Name; Age; City
```

```
# [1] "Leo"
```

```
# [1] 25
```

```
# [1] "Lisbon"
```

**Comments.** It is useful to put human readable comments in your programs that R ignores. These comments could help the future you when you go back to your program. R comments start with a hash sign (#). Everything after the hash to the end of the line will be ignored by R.

```
# This comment line will be ignored when run.
AGE      # Text after "#" is ignored.
```

```
# [1] 99
```

**Warning.** If an R command is not complete then R will show a plus sign (“+”) prompt on second and subsequent lines until the command syntax is correct.

```
+
```

To break out this, press the escape key (ESC).

**Recalling previous commands.** To recall a previously typed commands use the up arrow key (↑). To go between previously typed commands use the up and down arrow (↓) keys. Once a command is recalled, it can be modified/corrected using the left (←) and right arrow (→) keys.

## 5. R as a Calculator

You can use R as a calculator. Try the following:

```
2 + 3
```

```
# [1] 5
```

```
(5*6) - 7
```

```
# [1] 23
```

```
(39/7) - 6.9
```

```
# [1] -1.328571
```

```
# ^ = "to the power of"
7^3
```

```
# [1] 343
```

**Other math functions.** You can also use standard mathematical functions that are typically found on a scientific calculator.

- Trigonometric: `sin()`, `cos()`, `tan()`, `acos()`, `asin()`, `atan()`
- Rounding: `abs()`, `ceiling()`, `floor()`, `round()`, `sign()`, `signif()`, `sqrt()`, `trunc()`
- Logarithms & Exponentials: `exp()`, `log()`, `log10()`, `log2()`

```
# Square root
sqrt(2)
```

```
# [1] 1.414214
```

```
# Round down to nearest integer
floor(8.6178)
```

```
# [1] 8
```

```
# Round up to nearest integer
ceiling(8.6178)
```

```
# [1] 9
```

```
# Round pi to 3 decimal places
round(8.6178, 2)
```

```
# [1] 8.62
```

**Exercise.** Try out five of the mathematical functions above, using the help page where necessary.

## 6. Some More R Concepts

You can do some clever and useful things with using the assignment operator “<-”:

```
roomLength <- 7.8
roomWidth <- 6.4
roomArea <- roomLength * roomWidth
roomArea
```

```
# [1] 49.92
```

**Text objects.** You can also assign text to a objects too.

```
Greeting <- "Hello World!"
Greeting
```

```
# [1] "Hello World!"
```

**Vectors.** The objects presented so far have all been scalars (single values). Working with vectors is where R shines best as they are the basic building blocks of datasets. To create a vector we can use the `c()` (combine values into a vector) function.

```
# A "numeric" vector
x1 <- c(26, 10, 4, 7, 41, 19)
x1
```

```
# [1] 26 10 4 7 41 19
```

```
# A "character" vector of country names
x2 <- c("Peru", "Italy", "Cuba", "Ghana")
x2
```

```
# [1] "Peru" "Italy" "Cuba" "Ghana"
```

There are many other ways to create vectors, for example, `rep()` (replicate elements) and `seq()` (create sequences):

```
# Repeat vector (2, 6, 7, 4) three times
r1 <- rep(c(2, 6, 7, 4), times=3)
r1
```

```
# [1] 2 6 7 4 2 6 7 4 2 6 7 4
```

```
# Vector from -2 to 3 incremented by half
s1 <- seq(from=-2, to=3, by=0.5)
s1
```

```
# [1] -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5
# [9] 2.0 2.5 3.0
```

**Vector operations.** You can do also calculations on vectors, for example using `x1` from above:

```
x1 * 2
```

```
# [1] 52 20 8 14 82 38
```

```
round(sqrt(x1*2.6), 2)
```

```
# [1] 8.22 5.10 3.22 4.27 10.32 7.03
```

**Missing Values.** Missing values are coded as `NA` in R. For example,

```
x2 <- c(3, -7, NA, 5, 1, 1)
x2
```

```
# [1] 3 -7 NA 5 1 1
```

```
x3 <- c("Rat", NA, "Mouse", "Hamster")
x3
```

```
# [1] "Rat" NA "Mouse" "Hamster"
```

**Managing Objects.** Use function `ls()` to list the objects in your workspace. The `rm()` function removes (delete) them.

```
ls()
```

```
# [1] "age" "Age" "AGE"
# [4] "City" "Greeting" "height"
# [7] "Name" "r1" "roomArea"
# [10] "roomLength" "roomWidth" "s1"
# [13] "x" "x1" "x2"
# [16] "x3" "xm"
```

```
rm(x, x1, x2, x3, xm, r1, s1, AGE, age)
ls()
```

```
# [1] "Age" "City" "Greeting"
# [4] "height" "Name" "roomArea"
# [7] "roomLength" "roomWidth"
```

## 7. R Functions and Packages

**R Functions.** We have already used some R functions (e.g. `c()`, `mean()`, `rep()`, `sqrt()`, `round()`). Most of the computations in R involves using functions. A function essentially has a name and a list of arguments separated by a comma. Let's have look at an example:

```
seq(from = 5, to = 8, by = 0.4)
```

```
# [1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8
```

The function name is `seq` and it has three arguments `from`, `to` and `by`. The arguments `from` and `to` are the start and end values of a sequence that you want to create, and `by` is the increment of the sequence. The `seq()` functions has other arguments that you could use which are documented in the help page. For example, we could use the argument `length.out` (instead of `by`) to fix the length of the sequence as follows:

```
seq(from = 5, to = 8, length.out = 16)
```

```
# [1] 5.0 5.2 5.4 5.6 5.8 6.0 6.2 6.4 6.6 6.8
# [11] 7.0 7.2 7.4 7.6 7.8 8.0
```

**Custom Functions.** You can create your own functions (using the `function()` function) which is a very powerful way to extend R. Writing your own functions is outside the scope of this guide. As you get more and more familiar with R it is very likely that you will need to learn do so but for now you don't need to.

**R Packages.** You can do many things with a standard R installation and it can be extended using contributed packages. Packages are like apps for R. They can contain functions, data and documentation. We will use the `tidyverse` package<sup>3</sup> to see this in practice.

**tidyverse package.** \*\*\* EXPLAIN TIDYVERSE HERE\*\*\*

**Must do.** You must have the “tidyverse” package installed to continue. Please install it now using:

```
install.packages("tidyverse")
```

**Package usage.** To be able to use the `tidyverse` (or any other) package you have to use the `library()` function to load it. Please run the following in R:

```
library(tidyverse)
```

## 8. Chick Weight Data

R comes with many datasets installed<sup>4</sup>. We will use the `ChickWeight` dataset to learn about the tidyverse. The help system gives a basic summary of the experiment from which the data was collected:

*“The body weights of the chicks were measured at birth and every second day thereafter until day 20. They were also measured on day 21. There were four groups of chicks on different protein diets.”*

You can get more information, including references by typing:

```
help("ChickWeight")
```

**The Data.** There are 578 observations (rows) and 4 variables:

- `Chick` – unique ID for each chick.
- `Diet` – one of four protein diets.
- `Time` – number of days since birth.
- `weight` – body weight of chick in grams.

**Note.** `weight` has a lower case `w` (recall R is case sensitive).

**Objective.** Investigate the *effect of diet on the weight over time*.

## 9. Importing The Data

First we will import the data from a file called `ChickWeight.csv` using the `read_csv()` function from the `readr` package (part of the `tidyverse`). The first thing to do, outside of R, is to open the file `ChickWeight.csv` to check what it contains and that it makes sense. Now we can import the data as follows:

```
CW <- read_csv("ChickWeight.csv")
```

```
# Parsed with column specification:
# cols(
#   Chick = col_double(),
#   Diet = col_double(),
#   Time = col_double(),
#   weight = col_double()
# )
```

All columns (variables) have been read in as numeric values (i.e. `col_double()`)

<sup>3</sup>Tidyverse: <https://www.tidyverse.org/>

<sup>4</sup>Type `data()` in the R console to see a list of the datasets.

**Important Note.** If all goes well then the data is now stored in an R object called `CW`. If you get the following error message then you need to change the working directory to where the data is stored.

Error: ‘ChickWeight.csv’ does not exist in current working directory ...

**Change the working directory in RStudio.** From the menu bar select “Session - Set Working Directory - Choose Directory...” then go to the directory where the data is stored. Alternatively, within R, you could use the function `setwd()`<sup>5</sup>.

## 10. Looking at the Dataset

To look at the data type just type the object (dataset) name:

```
CW
```

```
# # A tibble: 578 x 4
#   Chick Diet Time weight
#   <dbl> <dbl> <dbl> <dbl>
# 1    18 1      0     39
# 2    18 1      2     35
# 3    16 1      0     41
# 4    16 1      2     45
# 5    16 1      4     49
# 6    16 1      6     51
# 7    16 1      8     57
# 8    16 1     10     51
# 9    16 1     12     54
# 10   15 1      0     41
# # ... with 568 more rows
```

**glimpse() function.** If there are too many variables then not all of them may be printed. To overcome this issue we can use the `glimpse()` function which makes it possible to see every column in your dataset (called a “data frame” in R speak).

```
glimpse(CW)
```

```
# Observations: 578
# Variables: 4
# $ Chick <dbl> 18, 18, 16, 16, 16, 16, 1...
# $ Diet <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1...
# $ Time <dbl> 0, 2, 0, 2, 4, 6, 8, 10, ...
# $ weight <dbl> 39, 35, 41, 45, 49, 51, 5...
```

**Interpretation.** Both of these show that the dataset has 578 observations and 4 variables as we would expect and as compared to the original data file `ChickWeight.csv`. So a good start.

**Exercise.** It is important to look at the last observations of the dataset as it could reveal potential data issues. Use the `tail()` function to do this. Is it consistent with the original data file `ChickWeight.csv`?

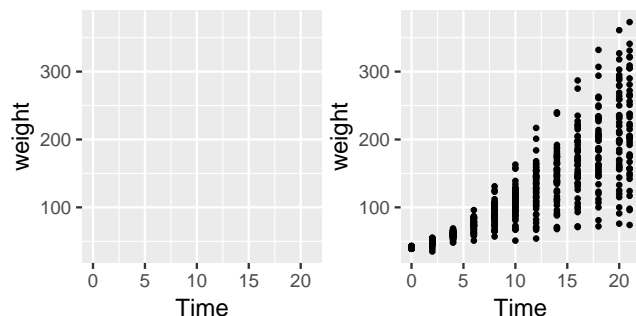
## 11. Chick Weight: Data Visualisation

**ggplot2 Package.** To visualise the chick weight data, we will use the `ggplot2` package (included in the `tidyverse`). Our interest is in seeing how the *weight changes over time for the chicks by diet*. For the moment don’t worry too much about the details just try to build your own understanding and logic. To learn more try different things even if you get an error message.

<sup>5</sup>Use `getwd()` to see the current working directory and `setwd("/to/data/path/data.csv")` to change it (important to use / even for Microsoft Windows).

**First plot.** Let's plot the weight data (vertical axis) over time (horizontal axis).

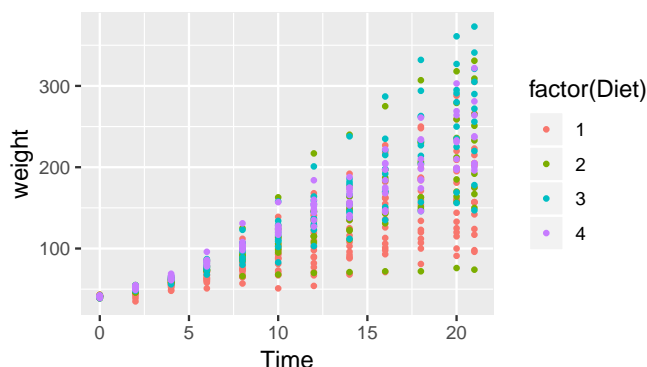
```
# An empty plot (the plot on the left)
ggplot(CW, aes(Time, weight))
# With data (the plot on the right)
ggplot(CW, aes(Time, weight)) + geom_point()
```



**Exercise.** Switch the variables Time and weight in code used for the second plot (on the right)? What is your opinion on this new plot compared to the one above?

**Add colour for Diet.** The graph above does not differentiate between the diets. Let's use a different colour for each diet.

```
# Adding colour for diet
ggplot(CW, aes(Time, weight, colour=factor(Diet))) +
  geom_point()
```



**Interpretation.** It is difficult to conclude anything from this graph as the points are printed on top of each other with diet 1 at the bottom and diet 4 on top.

**Factor Variables.** Before we continue and we will make an important change to the CW dataset by making Diet and Time factor variables. This means that R will see them as variables that have different categories (see the <fct> variables below). It will simplify our coding. The next section will explain the mutate() function.

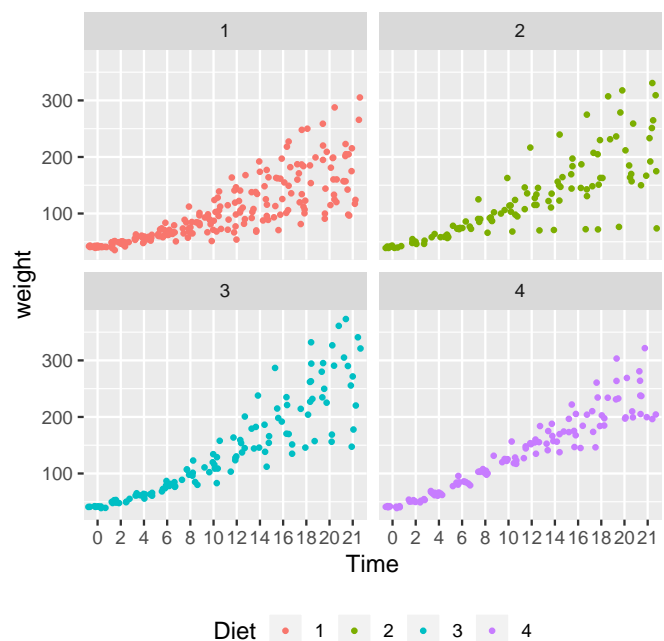
```
CW <- mutate(CW, Diet = factor(Diet))
CW <- mutate(CW, Time = factor(Time))
glimpse(CW)
```

```
# Observations: 578
# Variables: 4
# $ Chick <dbl> 18, 18, 16, 16, 16, 16, 1...
# $ Diet <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1...
# $ Time <fct> 0, 2, 0, 2, 4, 6, 8, 10, ...
```

```
# $ weight <dbl> 39, 35, 41, 45, 49, 51, 5...
```

**geom\_jitter() function.** To overcome the issue of overlapping points we can jitter the points using geom\_jitter() and plot each diet separately using facet\_wrap():

```
# Adding jitter to the points
ggplot(CW, aes(Time, weight, colour=Diet)) +
  geom_jitter() +
  facet_wrap(~Diet) +
  theme(legend.position = "bottom")
```

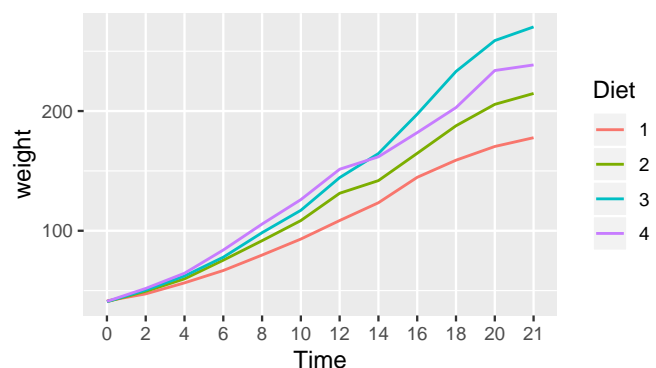


**Interpretation.** Diet 4 has the least variability but we can't really say anything about the mean effect of each diet.

**Exercise.** For the legend.position try using "top", "left" and "none". Do we really need a legend for this plot?

**Mean line plot.** Next we will plot the mean changes over time for each diet using the stat\_summary() function:

```
ggplot(CW, aes(Time, weight,
  group=Diet, colour=Diet)) +
  stat_summary(fun.y="mean", geom="line")
```

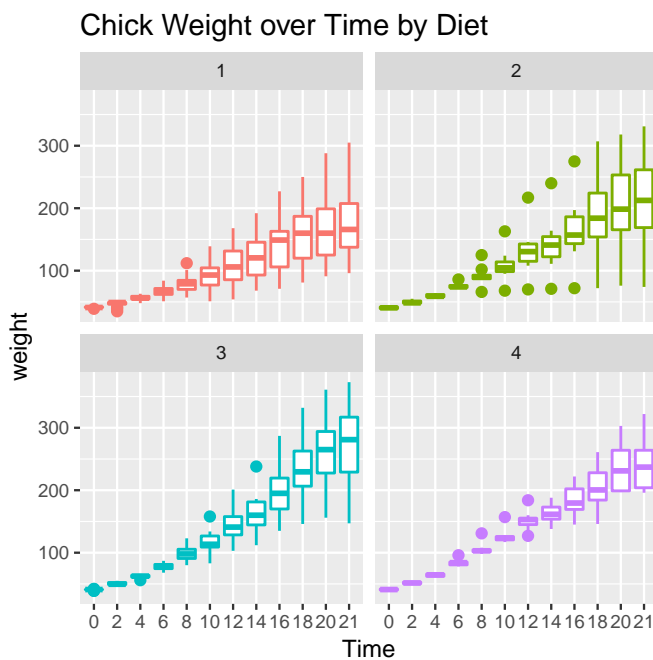


**Interpretation.** This shows that diet 3 group of chicks has the highest mean weight gains by the end of the experiment. However we don't see the variation (uncertainty) in the data.

**Exercise.** What happens when you add `geom_point()` to the plot above? Don't forget the `+`. Does it make a difference if you put it before the `stat_summary(...)` line? Hint: Look very carefully at how the graph is plotted.

**Box-whisker plot.** To see variation between the different diets we use `geom_boxplot()` to plot a box-whisker plot. A note caution is that the number of chicks per diet is relatively low to produce this plot.

```
ggplot(CW, aes(Time, weight, colour=Diet)) +
  facet_wrap(~Diet) +
  geom_boxplot() +
  theme(legend.position = "none") +
  ggtitle("Chick Weight over Time by Diet")
```



**Interpretation.** Looking at the graphs above we can say that diet 3 seems to have the most “average” weight gain but it has more variation than diet 4. We will have to look at the summary statistics to confirm this.

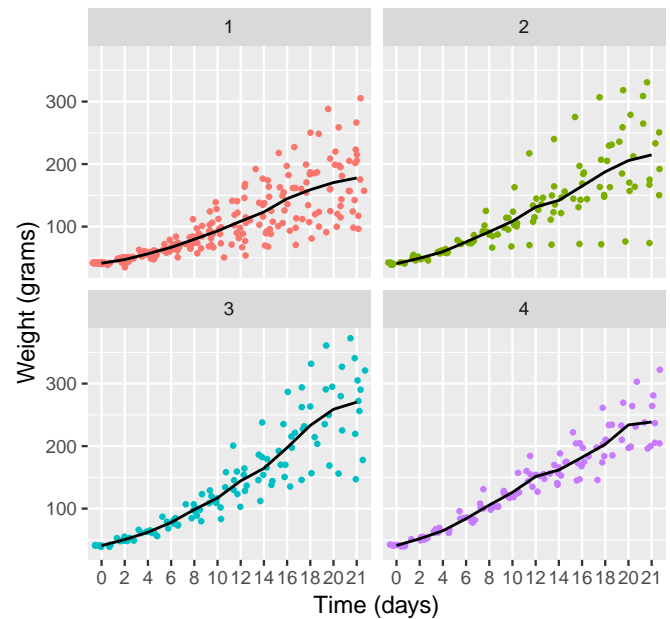
**Exercise.** Add the following information to the the above plot:

- x-axis label (use `xlab()`): “Time (days)”
- y-axis label (use `ylab()`): “Weight (grams)”

**Final Plot.** Let's finish with a plot that you might include in a publication.

```
ggplot(CW, aes(Time, weight, group=Diet,
  colour=Diet)) +
  facet_wrap(~Diet) +
  geom_jitter() +
  stat_summary(fun.y="mean", geom="line",
    colour="black") +
  theme(legend.position = "none") +
  ggtitle("Chick Weight over Time by Diet") +
  xlab("Time (days)") +
  ylab("Weight (grams)")
```

Chick Weight over Time by Diet



## 12. Tidyverse: Data Wrangling Basics

In this section we will learning how wrangle (manipulate) datasets using the tidyverse. We will look at the `mutate()`, `select()`, `rename()`, `filter()` and `arrange()` functions.

**`mutate()`.** Adds a new variable (column) or modifies an existing one. We already used this above to create factor variables.

```
# Added a column
CWm1 <- mutate(CW, weightKg = weight/1000)
CWm1
```

```
# # A tibble: 578 x 5
#   Chick Diet Time weight weightKg
#   <dbl> <fct> <fct> <dbl> <dbl>
# 1    18 1    0      39    0.039
# 2    18 1    2      35    0.035
# 3    16 1    0      41    0.041
# # ... with 575 more rows
```

```
# Modify an existing column
CWm2 <- mutate(CW, Diet = str_c("Diet ", Diet))
CWm2
```

```
# # A tibble: 578 x 4
#   Chick Diet Time weight
#   <dbl> <chr> <fct> <dbl>
# 1    18 Diet 1 0      39
# 2    18 Diet 1 2      35
# 3    16 Diet 1 0      41
# # ... with 575 more rows
```

**`select()`.** Keeps, drops or reorders variables.

```
# Drop the weight variable from CWm1 using minus
select(CWm1, -weight)
```



```
# # A tibble: 578 x 4
#   Chick Diet Time weightKg
#   <dbl> <fct> <fct>   <dbl>
# 1    18  1     0       0.039
# 2    18  1     2       0.035
# 3    16  1     0       0.041
# # ... with 575 more rows
```

```
# Keep variables Time, Diet and weightKg
select(CWm1, Chick, Time, Diet, weightKg)
```

```
# # A tibble: 578 x 4
#   Chick Time Diet weightKg
#   <dbl> <fct> <fct>   <dbl>
# 1    18  0     1       0.039
# 2    18  2     1       0.035
# 3    16  0     1       0.041
# # ... with 575 more rows
```

`rename()`. Renames variables whilst keeping all variables.

```
rename(CW, Group = Diet, Weight = weight)
```

```
# # A tibble: 578 x 4
#   Chick Group Time Weight
#   <dbl> <fct> <fct>   <dbl>
# 1    18  1     0       39
# 2    18  1     2       35
# 3    16  1     0       41
# # ... with 575 more rows
```

`filter()`. Keeps or drops observations (rows).

```
filter(CW, Time==21 & weight>300)
```

```
# # A tibble: 8 x 4
#   Chick Diet Time weight
#   <dbl> <fct> <fct>   <dbl>
# 1     7  1    21    305
# 2    29  2    21    309
# 3    21  2    21    331
# # ... with 5 more rows
```

For comparing values in vectors use: < (less than), > (greater than), <= (less than and equal to), >= (greater than and equal to), == (equal to) and != (not equal to). These can be combined logically using & (and) and | (or).

`arrange()`. Changes the order of the observations (rows).

```
arrange(CW, Chick, Time)
```

```
# # A tibble: 578 x 4
#   Chick Diet Time weight
#   <dbl> <fct> <fct>   <dbl>
# 1     1  1     0       42
# 2     1  1     2       51
# 3     1  1     4       59
# # ... with 575 more rows
```

```
arrange(CW, desc(weight))
```

```
# # A tibble: 578 x 4
#   Chick Diet Time weight
#   <dbl> <fct> <fct>   <dbl>
# 1    35  3    21    373
# 2    35  3    20    361
# 3    34  3    21    341
# # ... with 575 more rows
```

**Exercise.** What does the `desc()` do? Try using `desc(Time)`.

### 13. Tidyverse: Pipe operator %>%

In reality you will end up doing multiple data wrangling steps that you want to save. Neither are optimal as they have their own issues. This is where the pipe operator, %>% comes to the rescue:

```
CW21 <- CW %>%
  filter(Time %in% c(0, 21)) %>%
  rename(Weight = weight) %>%
  mutate(Group = factor(str_c("Diet ", Diet))) %>%
  select(Chick, Group, Time, Weight) %>%
  arrange(Chick, Time)
CW21
```

```
# # A tibble: 95 x 4
#   Chick Group Time Weight
#   <dbl> <fct> <fct>   <dbl>
# 1     1 Diet 1 0       42
# 2     1 Diet 1 21    205
# 3     2 Diet 1 0       40
# # ... with 92 more rows
```

**Read %>% as “then”.** To understand the code above we should read the pipe operator %>% as “then”.

Create a new dataset (object) called CW21 using dataset CW **then** keep the data for days 0 and 21 **then** rename variable weight to Weight **then** create a variable called Group **then** keep variables Chick, Group, Time and Weight and **then** finally arrange the data by variables Chick and Time.

**In practice.**

```
CW21 <- CW %>%
  filter(., Time %in% c(0, 21)) %>%
  rename(., Weight = weight) %>%
  mutate(., Group=factor(str_c("Diet ",Diet))) %>%
  select(., Chick, Group, Time, Weight) %>%
  arrange(., Chick, Time)
```

The pipe operator, %>%, replaces the dots (.) with whatever is returned from code preceding it. For example, the dot in `filter(., Time %in% c(0, 21))` is replaced by CW. The output of the `filter(...)` then replaces the dot in `rename(., Weight = weight)` and so on. Think of it as a data assembly line with each function doing its thing and passing it to the next.

### 14. Chick Weight: Summary Statistics

Having done some data visualisations we will quantify the effects of the diets using summary statistics. First we will look at the number of observations and the mean by diet and time.

```
mnsdCW <- CW %>%
  group_by(Diet, Time) %>%
  summarise(N = n(), Mean = mean(weight)) %>%
  arrange(Diet, Time)
mnsdCW
```

```
# # A tibble: 48 x 4
# # Groups:   Diet [4]
#   Diet Time     N Mean
#   <fct> <fct> <int> <dbl>
# 1 1      0      20  41.4
# 2 1      2      20  47.2
# 3 1      4      19  56.5
# # ... with 45 more rows
```

For each distinct combination of Diet and Time, the chick weight data is summarised into the number of observations (N) and the mean (Mean) of weight.

Let's also calculate the standard deviation, median, minimum and maximum values but only at days 0 (zero) and 21.

```
sumCW <- CW %>%
  filter(Time %in% c(0, 21)) %>%
  group_by(Diet, Time) %>%
  summarise(N = n(),
            Mean = mean(weight),
            SD = sd(weight),
            Median = median(weight),
            Min = min(weight),
            Max = max(weight)) %>%
  arrange(Diet, Time)
sumCW
```

```
# # A tibble: 8 x 8
# # Groups:   Diet [4]
#   Diet Time     N Mean     SD Median  Min
#   <fct> <fct> <int> <dbl> <dbl> <dbl> <dbl>
# 1 1      0      20  41.4  0.995   41    39
# 2 1      21     16 178.  58.7  166    96
# 3 2      0      10  40.7  1.49   40.5   39
# # ... with 5 more rows, and 1 more variable:
# #   Max <dbl>
```

Let's make the summaries "prettier", say, for a report or publication.

```
prettySumCW <- sumCW %>%
  mutate(Mean_SD = str_c(format(Mean, digits=1),
    " (", format(SD, digits=2), ")")) %>%
  mutate(Range = str_c(Min, " - ", Max)) %>%
  select(Diet, Time, N, Mean_SD, Median, Range) %>%
  arrange(Diet, Time)
prettySumCW
```

```
# # A tibble: 8 x 6
# # Groups:   Diet [4]
#   Diet Time     N Mean_SD      Median Range
#   <fct> <fct> <int> <chr>      <dbl> <chr>
# 1 1      0      20 " 41 ( 0.~   41    39 - ~
# 2 1      21     16 178 (58.7~  166    96 - ~
# 3 2      0      10 " 41 ( 1.~   40.5  39 - ~
# # ... with 5 more rows
```

**Final Table.** Eventually you should be able to produce<sup>6</sup> a publication ready version as follows:

Diet	Time	N	Mean_SD	Median	Range
1	0	20	41 ( 0.99)	41.0	39 - 43
1	21	16	178 (58.70)	166.0	96 - 305
2	0	10	41 ( 1.5)	40.5	39 - 43
2	21	10	215 (78.1)	212.5	74 - 331
3	0	10	41 ( 1)	41.0	39 - 42
3	21	10	270 (72)	281.0	147 - 373
4	0	10	41 ( 1.1)	41.0	39 - 42
4	21	9	239 (43.3)	237.0	196 - 322

**Interpretation.** This summary table offers the same interpretation as before, namely that diet 3 has the highest mean and median weights at day 21 but a higher variation than group 4. However it should be noted that at day 21, diet 1 lost 4 chicks from 20 that started and diet 4 lost 1 from 10. This could be a sign of some issues (e.g. safety).

**Limitations of data.** Information on bias reduction measures is not given and is not available either<sup>7</sup>. We don't know if the chicks were fairly and appropriately randomised to the diets and that the groups are comparable (e.g., same breed of chicks, sex (gender) balance). Hence we should be very cautious with drawing conclusion and taking actions with this data.

## 15. Conclusion

This "Getting Started in R" guide introduced you to some of the basic concepts underlying R and used a real life dataset to produce some graphs and summary statistics. It is only a flavour of what R can do but hopefully you have seen some of power of R and its potential.

**What next.** There are plenty of R courses, books and on-line resources that you can learn from. It is hard to recommend any in particular as it depends on how you learn best. Find things that work for you (paying attention to the quality) and don't be afraid to make mistakes or ask questions. Most importantly have fun.

**Acknowledgments.** [Names] for reviewing the draft versions of this document.

<sup>6</sup>Using the `kable()` function from the `knitr` package with functions from the `kableExtra` package.

<sup>7</sup>I contacted the source authors and kindly received the following reply "They were mainly undergraduate projects, final-year, rather than theses, so, unfortunately, it's unlikely that any record remains, particularly after so many years."