Guia Introductoria: Primeros Pasos en R

José R. Caro Barrera (a partir de Saghir Bashir y Dirk Eddelbuettel)

This version was compiled on 7 de abril de 2021

Esta guía es un intento de facilitar la puesta en marcha en R y Rstudio. El contenido es muy práctico, pues se usan ejemplos y conjuntos de datos reales, la idea es que el alumno se familiarice con los conceptos clave de R, a la hora de manipular, visualizar y resumir un conjunto de datos.

R | Estadística | Química | RStudio

1. Prólogo

Esta guía de "Introducción" es una breve aproximación de lo que se puede hacer con R[1]. Para aprovecharla al máximo es conveniente su lectura y a la vez, ir haciendo los ejemplos y los ejercicios que aquí se presentan usando RStudio[2].

Esta guía es una variante original del documento[3] pero hace hincapié en el uso de Base R junto con una gestión cuidadosa de las dependencias como se explica a continuación.

Experimentar en Modo Seguro. Lo ideal es experimentar con comandos y opciones, ya que es una parte esencial del proceso de aprendizaje. Las cosas pueden salir "mal" (lo harán), como recibir mensajes de error o eliminar cosas que se hayan creado, pero la mayoría de las situaciones pueden recuperarse (por ejemplo, reiniciando R). Para hacer esto "de forma segura", comienza con una sesión en R *nueva* sin ningún otro dato cargado.

2. Introducción

Antes de Comenzar. Es conveniente asegurarse de que:

1. R y RStudio (por ese orden) están instalados.
2. Comprobar que estamos en el directorio de trabajo deseado con `getwd()`.

Comenzando R y RStudio. R arranca automáticamente cuando se abre RStudio (ver Fig. 1). La consola comienza con información sobre el número de versión, la licencia y los colaboradores. La última línea es un mensaje estándar ">" que indica que R está listo y esperando instrucciones para hacer algo.

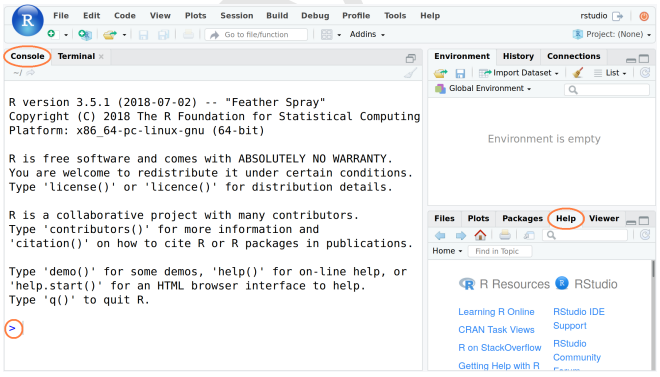


Fig. 1. Captura de pantalla de RStudio con la consola a la izquierda y la pestaña Ayuda en la parte inferior derecha

Saliendo de R y RStudio. Cuando salgamos de RStudio, nos preguntará si deseamos `Guardar espacio de trabajo (Save workspace)` con dos opciones:

- "Yes" – Nuestro espacio de trabajo actual de R (que contiene el trabajo que hemos realizado) será restaurado la próxima vez que se abra RStudio.
- "No" – Comenzaremos con una nueva sesión de R la próxima vez que abramos RStudio. Por ahora, seleccionaremos "*No*" para evitar errores de sesiones anteriores.

3. La Ayuda (Help) en R

RStudio tiene un extenso sistema de ayuda incorporado que suele ser muy útil a la hora de encontrar soluciones a cualquier problema ya sea con comandos, paquetes y/o funciones.

Función `help()`. Desde la "consola" de R se puede usar la función `help()` o `?`. Por ejemplo, los dos comandos siguientes proporcionan el mismo resultado:

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

Búsqueda de palabra clave.. Para buscar una palabra clave se usa la función `apropos()` con la palabra en cuestión entre comillas doble (`"keyword"`) o comillas simple (`'keyword'`). Por ejemplo:

```
apropos("mean")
#   [1] ".colMeans"       ".rowMeans"
#   [3] "colMeans"        "frollmean"
#   [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"
```

Ejemplos de Ayuda. Use la función `example()` para ejecutar los ejemplos al final de la ayuda para una función:

```
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
```

La Ayuda de RStudio. Rstudio proporciona un cuadro de búsqueda en la pestaña "Ayuda" para hacer el trabajo más fácil (ver Fig. 1).

[1] Proyecto R: https://www.r-project.org/
[2] IDE RStudio: https://www.rstudio.com/products/RStudio/
[3] Getting Started with R: https://github.com/saghirb/Getting-Started-in-R

Búsqueda de ayuda para R on-line. Hay muchos recursos on-line que pueden servir de ayuda, sin embargo, copiar y pegar sin más, a veces, no será de mucha ayuda y ralentizará el proceso de aprendizaje y desarrollo. Cuando se haga una búsqueda on-line use `[R]` en el término de búsqueda, (p. ej. "[R] resumen estadísticos por grupo"). A menudo hay más de una solución para un problema en R, por lo que es bueno investigar las diferentes opciones.

Ejercicio. Intente lo siguiente:

1. `help(median)`
2. `?sd`
3. `?max`

Advertencia. Si un comando R no está completo, R mostrará un signo más (`+`) en la segunda línea y en las siguientes hasta que la sintaxis del comando sea correcta.

```
+
```

Para cortar esto, presione la tecla de escape (`ESC`).

Truco. Para recuperar un comando escrito anteriormente, use la tecla de flecha hacia arriba (↑). Para ir entre los comandos escritos anteriormente, use las flechas hacia arriba y hacia abajo (↓). Para modificar o corregir un comando use la flecha izquierda (←) y la derecha (→).

4. Algunos conceptos en R

En lenguaje R, escalares, vectores/variables y datasets se denominan ***objetos***. Para crear objetos (cosas) hay que usar el operador asignación `<-`. Por ejemplo, a continuación, al objeto `altura` se le ha asignado el valor 173 (tecleando `altura` se muestra su valor):

```
altura <- 173
altura
#  [1] 173
```

Advertencia: sobre las mayúsculas. `edad` y `EdaD` son diferentes:

```
edad <- 10
EdaD <- 50
```

```
edad
#  [1] 10
EdaD
#  [1] 50
```

Nuevas líneas. Los comandos en R suelen estar separados por una nueva línea, pero también pueden estarlo por un punto y coma `;`.

```
Nombre <- "Rafael"; Edad <- 20; Ciudad <- "Madrid"
Nombre; Edad; Ciudad
#  [1] "Rafael"
#  [1] 20
#  [1] "Madrid"
```

Comentarios. Es útil incluir comentarios explicativos en el código. Estos comentarios suelen ser de ayuda en el futuro cuando se retome el script. Los comentarios en R comienzan por el símbolo almohadilla (`#`). Todo lo que esté después de este símbolo hasta el final de la línea será ignorado por R.

```
# Este comentario será ignorado cuando se ejecute.
Ciudad     # El texto despues de "#" es ignorado.
#  [1] "Madrid"
```

5. R como calculadora

Es posible usar R como calculadora. Pruebe lo siguiente:

```
2 + 3
#  [1] 5
(5*11)/4 - 7
#  [1] 6.75
# ^ = "potencia de un número"
7^3
#  [1] 343
```

Otras funciones matemáticas. También se pueden utilizar funciones matemáticas estándar que normalmente se encuentran en una calculadora científica.

- Trigonométricas: `sin()`, `cos()`, `tan()`, `acos()`, `asin()`, `atan()`
- Redondeo: `abs()`, `ceiling()`, `floor()`, `round()`, `sign()`, `signif()`, `sqrt()`, `trunc()`
- Logarítmicas y Exponenciales: `exp()`, `log()`, `log10()`, `log2()`

```
# Raíz cuadrada
sqrt(2)
#  [1] 1.414214
# Redondeo a la baja al entero más cercano
floor(8.6178)
#  [1] 8
# Redondeo a dos decimales
round(8.6178, 2)
#  [1] 8.62
```

Ejercicio. Compruebe qué hacen los siguientes pares de comandos:

1. `ceiling(18.33)` y `signif(9488, 2)`
2. `exp(1)` y `log10(1000)`
3. `sign(-2.9)` y `sign(32)`
4. `abs(-27.9)` y `abs(11.9)`

6. Algunos conceptos adicionales en R

Se pueden hacer algunas cosas útiles e interesantes con el operador de asignación "`<-`":

```
longitud <- 7.8
anchura <- 6.4
area <- longitud * anchura
area
#  [1] 49.92
```

Objetos de texto. También podemos asignar un texto a un objeto.

```
saludo <- "Hola Mundo!"
saludo
#  [1] "Hola Mundo!"
```

Vectores. Los objetos presentados hasta ahora han sido todos escalares (valores únicos). Trabajar con vectores es donde R brilla mejor, ya que son los componentes básicos de los conjuntos de datos. Para crear un vector podemos usar la función `c()` (combinar valores en un vector).

```
# Un vector "numérico"
x1 <- c(26, 10, 4, 7, 41, 19)
x1
#  [1] 26 10  4  7 41 19
# Un vector caracter de nombres de países
x2 <- c("Brasil", "Italia", "Cuba", "Ghana")
x2
#  [1] "Brasil" "Italia" "Cuba"   "Ghana"
```

Hay otras muchas formas de crear vectores, por ejemplo, `rep()` (replicate elements) y `seq()` (create sequences):

```
# Repetir el vector (2, 6, 7, 4) tres  veces
r1 <- rep(c(2, 6, 7, 4), times=3)
r1
#   [1] 2 6 7 4 2 6 7 4 2 6 7 4
# Vector desde -2 a 3 incrementado en 0.5
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  2.0
#  [10]  2.5  3.0
```

Operaciones con vectores. También se pueden realizar cálculos con vectores, por ejemplo, usando el vector `x1` creado antes:

```
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
```

Valores perdidos. En R, los valores perdidos son codificados como `NA`. Por ejemplo,

```
x2 <- c(3, -7, NA, 5, 1, 1)
x2
#  [1]  3 -7 NA  5  1  1
x3 <- c("Rata", NA, "Raton", "Hamster")
x3
#  [1] "Rata"    NA        "Raton"   "Hamster"
```

Manejando Objetos. El uso de la función `ls()` permite listar los objetos de nuestro espacio de trabajo. La función `rm()` los elimina (borra).

```
ls()
#   [1] "altura"   "anchura"  "area"     "Ciudad"
#   [5] "edad"     "Edad"     "EdaD"     "longitud"
#   [9] "Nombre"   "op"       "r1"       "s1"
#  [13] "saludo"   "x"        "x1"       "x2"
#  [17] "x3"       "xm"
rm(x1, x2, x3, r1, s1, EdaD, edad)
ls()
#   [1] "altura"   "anchura"  "area"     "Ciudad"
#   [5] "Edad"     "longitud" "Nombre"   "op"
#   [9] "saludo"   "x"        "xm"
```

Ejercicio. Calcular la cantidad `bruta` añadiendo la `tasa` al `neto`.

```
neto <- c(108.99, 291.42, 16.28, 62.29, 31.77)
tasa <- c(22.89, 17.49, 0.98, 13.08, 6.67)
```

7. R Funciones y paquetes

R Funciones. Ya hemos usado algunas funciones en R (p. ej.: `c()`, `mean()`, `rep()`, `sqrt()`, `round()`). La mayoría de los cálculos en R implican el uso de funciones. Una función tiene esencialmente un nombre y una lista de argumentos separados por una coma. Por ejemplo:

```
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
```

El nombre de la función es `seq` y tiene 3 argumentos: `from`, `to` y `by`. Los argumentos `from` y `to` son los valores de comienzo y final de la secuencia que se quieren crear y `by` es el incremento de la secuencia. Las funciones `seq()` tienen otros argumentos los cuales están recogidos en la ayuda. Por ejemplo, podemos usar el argumento `length.out` (en vez de `by`) para fijar la longitud de la secuencia, por ejemplo:

```
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 7.0
#  [12] 7.2 7.4 7.6 7.8 8.0
```

Funciones personalizadas. Podemos crear nuestras propias funciones (usando la palabra clave `function ()`) y así aprovechar la potencia de las funcionalidades de R. Crear funciones propias no es el objetivo del curso aunque a medida que uno se familiarice más con R, es probable que se necesite aprender a hacerl.

Paquetes en R. Con una instalación estándar de R se pueden hacer bastantes cosas pero puede ampliarse utilizando paquetes contribuidos. Los paquetes son como aplicaciones para R, creados por cualquier usuario y pueden contener funciones, datos y documentación.

Base Extensible R. Base R ya viene con más de dos mil funciones versátiles, fiables y estables, lo cual ya es bastante. El código de buenas prácticas de programación en R pasa por resolver un problema con *menos* dependencias externas, por lo que siempre será conveniente pensar detenidamente antes de agregar cualquier tipo de dependencia.

El Universo `tidyverse`. La filosofía de *menos es más* está en el núcleo de tidyverse[4]. Menos dependencias en los paquetes, una instalación más rápida y, aún más importante, menos nodos en el esquema de dependencia.

Por lo tanto, elegir los paquetes adicionales tiene que equilibrarse con la funcionalidad que nos aporta ese paquete con su historial y modelo de desarrollo, el estado de mantenimiento y el historial de cambios y correcciones. Este *es* un tema complejo, pero al agregar otro paquete, siempre abrimos una puerta a cambios de interfaz que ya no controlamos. La funcionalidad agregada es útil a veces, sin embargo, hay que ser consciente de los costes que pueden acumularse como consecuencia. Por lo tanto, la filosofía del curso se basa en que *menos es mejor* y veremos, entre otros, dos

[4] Tidyverse: http://www.tidyverse.org/

paquetes adicionales: `data.table` [5] para la distribución de datos, así como para la entrada/salida, y `ggplot2` [6] para la visualización.

Instalación. Si es necesario, instale estos dos paquetes a través del siguiente comando que debería elegir la versión adecuada para su instalación:

```
install.packages(c("data.table", "ggplot2"))
```

8. El Dataset `Chick Weight`

R viene con muchos datasets instalados[7]. Usaremos el dataset llamado `ChickWeight` para aprender a manipular datos. El sistema de ayuda ofrece un resumen básico del experimento del que se recopilaron los datos:

> *"Los pesos corporales de los polluelos se midieron al nacer y cada dos días, a partir de entonces, hasta el día 20. También se midieron el día 21. Había cuatro grupos de polluelos con diferentes dietas proteicas."*

Se puede obtener más información, incluidas las referencias, escribiendo:

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

Los Datos. Hay 578 observaciones (filas) y 4 variables:

- `Chick` – identificación única para cada polluelo.
- `Diet` – una de las cuatro dietas proteicas.
- `Time` – número de días desde el nacimiento.
- `weight` – peso corporal del polluelo en gramos (recuerde que empieza por minúscula y R es sensible a las mayúsculas).

Objetivo. Investigar el *efecto de la dieta sobre el peso en el tiempo*.

9. Importando los datos

Primero importaremos los datos de un archivo llamado `ChickWeight.csv` usando la función `fread()` del paquete `data.table` que devuelve un objeto `data.table` (mientras que el conjunto de datos integrado en R tiene un formato diferente). Lo primero que se debe hacer, fuera de R, es abrir el archivo `ChickWeight.csv` para verificar qué contiene y si tiene sentido. Ahora podemos importar los datos como sigue:

```
suppressMessages(library(data.table))  # tinyverse
cw <- fread("ChickWeight.csv")
```

Nota. Si funciona, los datos ahora se almacenan en un objeto R llamado `cw`. Si aparece el siguiente mensaje de error, se debe cambiar el directorio de trabajo donde se almacenan los datos.

```
Error: 'ChickWeight.csv' does not exist in current
working directory ...
```

Cambio del directorio de trabajo en Rstudio. En la barra de menú, seleccione "Session - Set Working Directory - Choose Directory..." y luego vaya al directorio donde se almacenan los datos. Alternativamente, dentro de R, puede usar la función `setwd()`[8]. También puede especificar una ruta completa, usando ~ para indicar su directorio de inicio.

[5] data.table: http://r-datatable.com

[6] ggplot2: https://ggplot2.tidyverse.org

[7] Teclee `data()` en la consola de R para ver una lista de todos los datasets.

[8] Use `getwd()` para el directorio actual y `setwd(/to/data/path/data.csv")` para cambiarlo.

10. Mirando el Dataset

Para ver los datos, escriba el nombre del objeto (dataset):

```
cw
#       Chick Diet Time weight
#    1:     1    1    0     42
#    2:     1    1    2     51
#    3:     1    1    4     59
#   ---
#  576:    50    4   18    234
#  577:    50    4   20    264
#  578:    50    4   21    264
```

Varias funciones en base R nos ayudan a inspeccionar los datos: `str()` muestra la estructura de forma compacta, `summary()` nos da un resumen y `head()` y `tail()` muestran el comienzo y el final del conjunto de datos.

```
str(cw)
#  Classes 'data.table' and 'data.frame':
#    578 obs. of  4 variables:
#  $ Chick : int [1:578] 1 1 1 1 1 ...
#  $ Diet : int [1:578] 1 1 1 1 1 ...
#  $ Time : int [1:578] 0 2 4 6 8 ...
#  $ weight: int [1:578] 42 51 59 64 76 ...
#  - attr(*, ".internal.selfref")=<externalptr>
summary(cw)
#       Chick           Diet
#   Min.   : 1.0   Min.   :1.00
#   1st Qu.:13.0   1st Qu.:1.00
#   Median :26.0   Median :2.00
#   Mean   :25.8   Mean   :2.24
#   3rd Qu.:38.0   3rd Qu.:3.00
#   Max.   :50.0   Max.   :4.00
#       Time           weight
#   Min.   : 0.0   Min.   : 35
#   1st Qu.: 4.0   1st Qu.: 63
#   Median :10.0   Median :103
#   Mean   :10.7   Mean   :122
#   3rd Qu.:16.0   3rd Qu.:164
#   Max.   :21.0   Max.   :373
```

Interpretación. Esto muestra que el conjunto de datos tiene 578 observaciones y 4 variables como cabría esperar, y en comparación con el archivo de datos original `ChickWeight.csv`. La función `str()` nos dice los tipos de variables (todos `integer`, es decir, números) y los primeros valores. El panel 'Environment' de RStudio ofrece una vista muy similar.

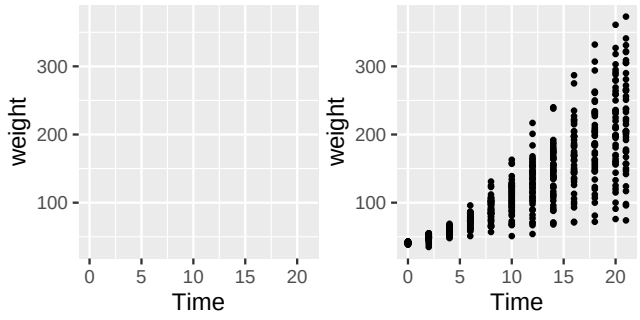
Ejercicio. Es importante mirar las últimas observaciones del conjunto de datos, ya que podría revelar posibles problemas con los mismos. Use la función `tail()` para hacerlo. ¿Es consistente con el archivo original `ChickWeight.csv`?

11. Chick Weight: Visualización de los Datos

El Paquete `ggplot2`. Para visualizar los datos del peso de los polluelos, usaremos el paquete `ggplot2`. Nuestro interés está en ver cómo el *peso de los polluelos cambia con el tiempo por la dieta*. Por el momento solo intentaremos entender y comprender el funcionamiento. Para obtener más información, es bueno intentar cosas diferentes incluso si se recibe un mensaje de error.

Primer gráfico. Let's plot the weight data (vertical axis) over time (horizontal axis).

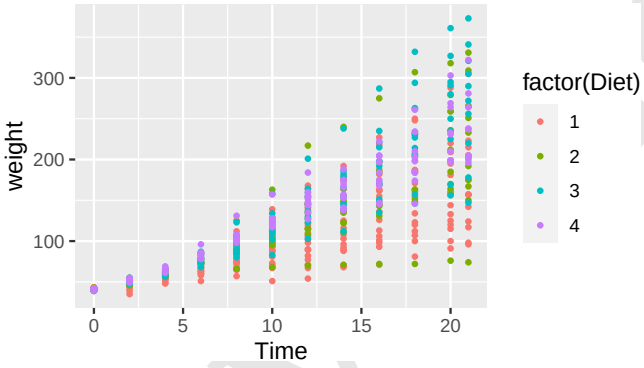
```
# (Silently) load the plotting package
suppressMessages(library(ggplot2))
# An empty plot (the plot on the left)
ggplot(cw, aes(Time, weight))
# With data (the plot on the right)
ggplot(cw, aes(Time, weight)) + geom_point()
```



Exercise. Switch the variables `Time` and `weight` in code used for the plot on the right? What do you think of this new plot compared to the original?

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 one another (with diet 1 underneath and diet 4 at the top).

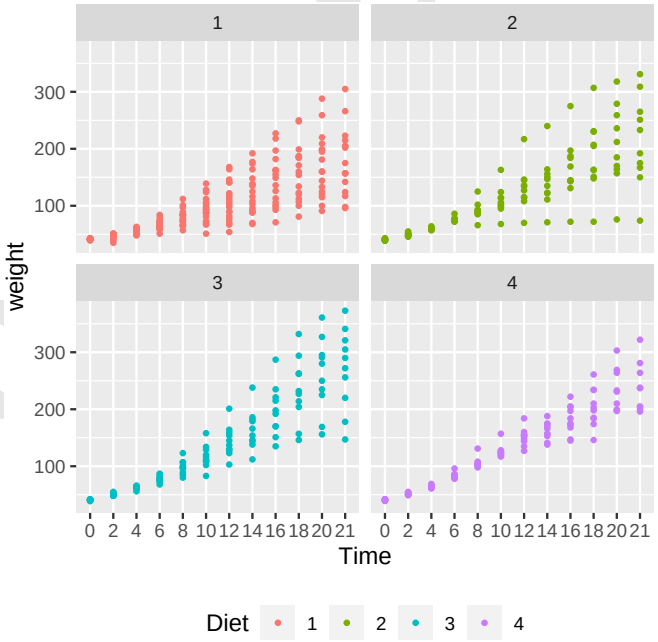
Factor Variables. Before we continue, we have to make an important change to the `cw` dataset by making `Diet` and `Time` *factor variables*. This means that R will treat them as categorical variables instead of continuous variables. It will simplify our coding.

```
cw[, Diet := factor(Diet)]
cw[, Time := factor(Time)]
str(cw)      # notice the difference ?
#  Classes 'data.table' and 'data.frame':
#    578 obs. of  4 variables:
#  $ Chick : int [1:578] 1 1 1 1 1 ...
#  $ Diet : Factor w/ 4 levels "1","2","3","4":
#     1 1 1 1 1 ...
#  $ Time : Factor w/ 12 levels
#     "0","2","4","6",..: 1 2 3 4 5 ...
#  $ weight: int [1:578] 42 51 59 64 76 ...
#  - attr(*, ".internal.selfref")=<externalptr>
```

Notice that the `:=` operator altered the variable "in-place", and no explicit assignment was made. This is a key feature of `data.table` which operated "by reference": changes are made in reference to one instance of the `cw` variable, rather than by creating updated copies. We will revisit this `:=` assignment below.

`facet_wrap()` function. To plot each diet separately in a grid using `facet_wrap()`:

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



Exercise. To overcome the issue of overlapping points we can ***jitter*** the points using `geom_jitter()`. Replace the `geom_point()` above with `geom_jitter()`. What do you observe?

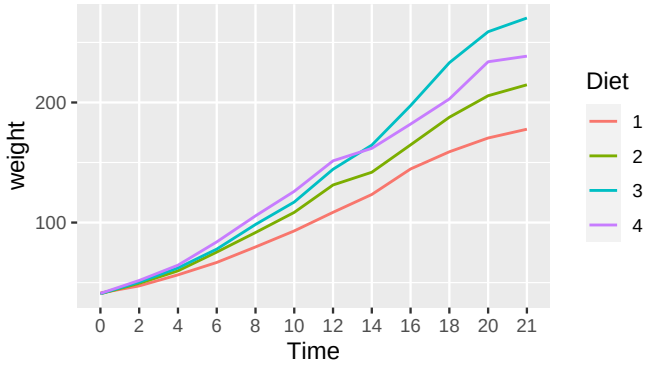
Interpretation. Diet 4 has the least variability but we can't really say anything about the mean effect of each diet although diet 3 seems to have the highest.

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")
```

```
#  Warning: `fun.y` is deprecated. Use `fun` instead.
```

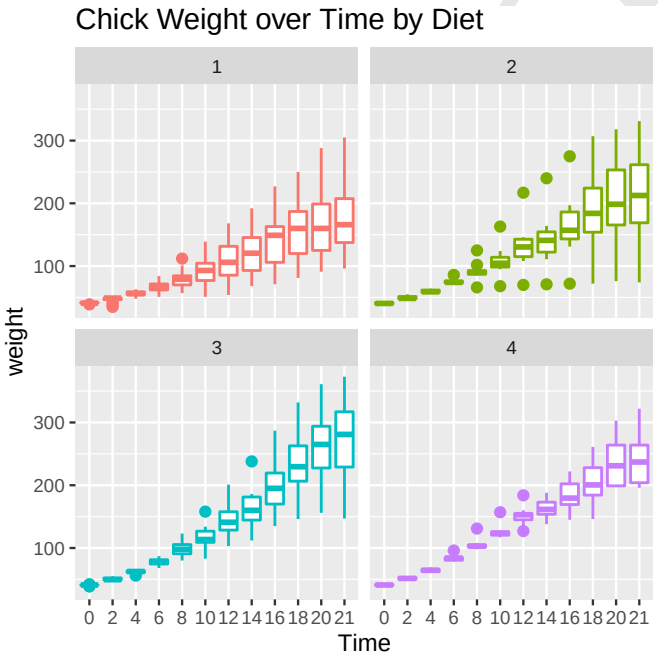


Interpretation. We can see that diet 3 has the highest mean weight gain by the end of the experiment but we don't have any information about the variation (uncertainty) in the data.

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 or after 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 of 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. Diet 3 seems to have the highest "average" weight gain but it has more variation than diet 4 which is consistent with our findings so far.

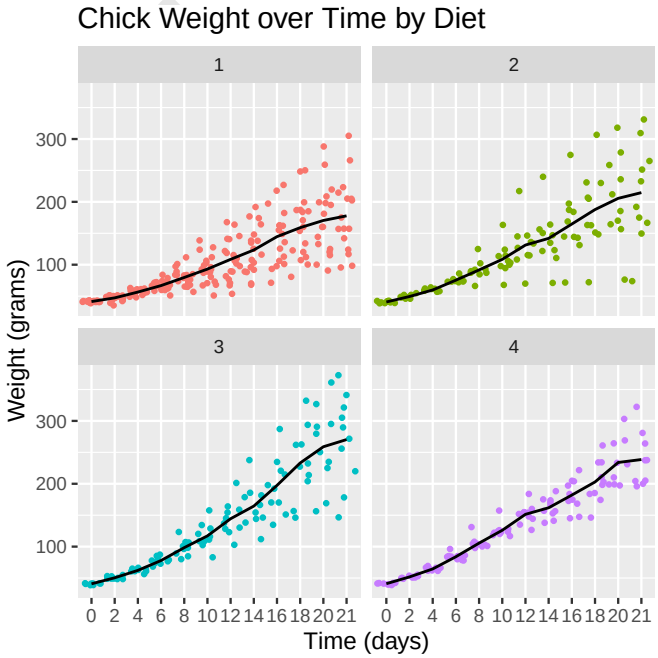
Exercise. Add the following information to the above plot:

- 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)")
```

```
#  Warning: `fun.y` is deprecated. Use `fun` instead.
```



12. `data.table` Data Wrangling Basics

In this section we will learn how to wrangle (manipulate) datasets using the `data.table` package. Conceptually, `data.table` operations can be viewed as `dt[i, j, by]` with some intentional similarity to SQL. Here `i` can select (or subset) rows, `j` is used to select, summarise or mutate columns, and `by` is the grouping operator. Numerous examples follow.

`j` to select (or transform) columns. Adds a new variable (column) or modifies an existing one. We already used this above to create factor variables.

```
cw[, weightKg := weight/1000]     # add a column
cw
#       Chick Diet Time weight weightKg
#    1:     1    1    0     42    0.042
```

```
#    2:     1    1    2     51    0.051
#    3:     1    1    4     59    0.059
#   ---
#  576:    50    4   18    234    0.234
#  577:    50    4   20    264    0.264
#  578:    50    4   21    264    0.264
cw[, Diet := paste0("Diet_", Diet)] # mod col.
cw
#       Chick   Diet Time weight weightKg
#    1:     1 Diet_1    0     42    0.042
#    2:     1 Diet_1    2     51    0.051
#    3:     1 Diet_1    4     59    0.059
#   ---
#  576:    50 Diet_4   18    234    0.234
#  577:    50 Diet_4   20    264    0.264
#  578:    50 Diet_4   21    264    0.264
```

`j` to select (or transform) columns. Keeps, drops or reorders variables.

```
# Keep variables Time, Diet and weightKg
cw[, .(Chick, Time, Diet, weightKg)]
#       Chick Time   Diet weightKg
#    1:     1    0 Diet_1    0.042
#    2:     1    2 Diet_1    0.051
#    3:     1    4 Diet_1    0.059
#   ---
#  576:    50   18 Diet_4    0.234
#  577:    50   20 Diet_4    0.264
#  578:    50   21 Diet_4    0.264
```

`j` to summarise. It can be used to create aggregations, which is particularly handy with the grouping operator. The following example computes means and standard deviations of the 'weight' variable grouped by 'Diet'. Note that the output has been truncated.

```
cw[, .(Mean=mean(weight),SDev=sd(weight)),
   by=.(Diet, Time)]
#        Diet Time     Mean      SDev
#   1: Diet_1    0  41.4000  0.994723
#   2: Diet_1    2  47.2500  4.278157
#   3: Diet_1    4  56.4737  4.128067
#  ---
#  46: Diet_4   18 202.9000 33.557413
#  47: Diet_4   20 233.8889 37.568086
#  48: Diet_4   21 238.5556 43.347754
```

`setnames()` to name or rename. Renames variables whilst keeping all variables.

```
setnames(cw, c("Diet", "weight"),
         c("Group", "Weight"))
cw
#       Chick  Group Time Weight weightKg
#    1:     1 Diet_1    0     42    0.042
#    2:     1 Diet_1    2     51    0.051
#    3:     1 Diet_1    4     59    0.059
#   ---
#  576:    50 Diet_4   18    234    0.234
#  577:    50 Diet_4   20    264    0.264
#  578:    50 Diet_4   21    264    0.264
```

`i` operator. Keeps or drops observations (rows).

```
cw[Time == 21 & Weight > 300]
#     Chick  Group Time Weight weightKg
#  1:     7 Diet_1   21    305    0.305
#  2:    21 Diet_2   21    331    0.331
#  3:    29 Diet_2   21    309    0.309
#  4:    32 Diet_3   21    305    0.305
#  5:    34 Diet_3   21    341    0.341
#  6:    35 Diet_3   21    373    0.373
#  7:    40 Diet_3   21    321    0.321
#  8:    48 Diet_4   21    322    0.322
```

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).

Keying observations. Setting a key changes the order of the observations (rows), and also makes indexing faster.

```
cw[order(Weight)]       # on the fly
#       Chick  Group Time Weight weightKg
#    1:    18 Diet_1    2     35    0.035
#    2:     3 Diet_1    2     39    0.039
#    3:    18 Diet_1    0     39    0.039
#   ---
#  576:    34 Diet_3   21    341    0.341
#  577:    35 Diet_3   20    361    0.361
#  578:    35 Diet_3   21    373    0.373
setkey(cw, Chick, Time)   # setting a key
cw
#       Chick  Group Time Weight weightKg
#    1:     1 Diet_1    0     42    0.042
#    2:     1 Diet_1    2     51    0.051
#    3:     1 Diet_1    4     59    0.059
#   ---
#  576:    50 Diet_4   18    234    0.234
#  577:    50 Diet_4   20    264    0.264
#  578:    50 Diet_4   21    264    0.264
```

Exercise. What does the `order()` do? Try using `order(Time)` and `order(-Time)` in the `i` column.

13. Chaining

You may want to do multiple data wrangling steps at once. This is where the 'chaining' of `data.table` operations (*i.e.*, several sets of commands with square brackets) comes to the rescue:

```
cw21 <- cw[Time %in% c(0,21)][    # i: select rows
  , weight := Weight][            # j: mutate
  , Group := factor(Group)][
  , .(Chick,Group,Time,weight)][  # j: arrange
   order(Chick,Time)][            # i: order
  1:5]                            # i: subset
```

14. Chick Weight: Summary Statistics

From the data visualisations above we concluded that the diet 3 has the highest mean and diet 4 the least variation. In this section, we will quantify the effects of the diets using summary statistics.

We start by looking at the number of observations and the mean of weight grouped by **diet** and **time**.

```
cw[, .(N    = .N,              # .N is nb per group
       Mean = mean(Weight)), # compute mean
   by=.(Group, Time)][       # group by Diet + Time
   1:5]                      # display rows 1 to 5
#      Group Time  N    Mean
#  1: Diet_1    0 20 41.4000
#  2: Diet_1    2 20 47.2500
#  3: Diet_1    4 19 56.4737
#  4: Diet_1    6 19 66.7895
#  5: Diet_1    8 19 79.6842
```

by= argument. For each distinct combination of `Diet` and `Time`, the chick weight data is summarised into the number of observations (`N`, using the internal variable `.N` denoting current group size) and the mean (`Mean`) of `weight`.

Other summaries. We can calculate the standard deviation, median, minimum and maximum values—only at days 0 and 21.

```
cws <- cw[Time %in% c(0,21),
          .(N      = .N,
            Mean   = mean(Weight),
            SDev   = sd(Weight),
            Median = median(Weight),
            Min    = min(Weight),
            Max    = max(Weight)  ),
          by=.(Group, Time)]
cws
#      Group Time  N  Mean   SDev Median Min Max
#  1: Diet_1    0 20  41.4  0.995   41.0  39  43
#  2: Diet_1   21 16 177.8 58.702  166.0  96 305
#  3: Diet_2    0 10  40.7  1.494   40.5  39  43
#  4: Diet_2   21 10 214.7 78.138  212.5  74 331
#  5: Diet_3    0 10  40.8  1.033   41.0  39  42
#  6: Diet_3   21 10 270.3 71.623  281.0 147 373
#  7: Diet_4    0 10  41.0  1.054   41.0  39  42
#  8: Diet_4   21  9 238.6 43.348  237.0 196 322
```

Finally, we can make the summaries "prettier" for a possible report or publication where we format the numeric values as text.

```
cws[, Mean_SD := paste0(format(Mean,digits=1),
                        " (",
                        format(SDev,digits=2),
                        ")")]
cws[, Range := paste(Min, "-", Max)]
prettySum <- cws[ , .(Group, Time, N, Mean_SD,
                      Median, Range)][
                  order(Group, Time)]
prettySum
#      Group Time  N     Mean_SD Median     Range
#  1: Diet_1    0 20  41 ( 0.99)   41.0   39 - 43
#  2: Diet_1   21 16 178 (58.70)  166.0  96 - 305
#  3: Diet_2    0 10  41 ( 1.49)   40.5   39 - 43
#  4: Diet_2   21 10 215 (78.14)  212.5  74 - 331
#  5: Diet_3    0 10  41 ( 1.03)   41.0   39 - 42
#  6: Diet_3   21 10 270 (71.62)  281.0 147 - 373
#  7: Diet_4    0 10  41 ( 1.05)   41.0   39 - 42
#  8: Diet_4   21  9 239 (43.35)  237.0 196 - 322
```

Final Table. Eventually you should be able to produce a publication-ready version such as the following table. Its code uses the `kable` and `kableExtra` packages. While the code is not displayed here for compactness, full details are of course in the sources.

Group	Time	N	Mean_SD	Median	Range
Diet_1	0	20	41 (0.99)	41.0	39 - 43
Diet_1	21	16	178 (58.70)	166.0	96 - 305
Diet_2	0	10	41 (1.49)	40.5	39 - 43
Diet_2	21	10	215 (78.14)	212.5	74 - 331
Diet_3	0	10	41 (1.03)	41.0	39 - 42
Diet_3	21	10	270 (71.62)	281.0	147 - 373
Diet_4	0	10	41 (1.05)	41.0	39 - 42
Diet_4	21	9	239 (43.35)	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[9]. We don't know if the chicks were fairly and appropriately randomised to the diets and whether 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.

16. Acknowledgements

Special thanks to Saghir Bashir for publishing the initial version of *Getting Started with R*, Brodie Gaslam and Matt Dowle for feedback on this version, and to Joshua Ulrich for many discussions about The Tinyverse.

[9] I (ie Saghir) 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."*