BIO 4022. Manipulación de datos e investigación reproducible en R

Derek Corcoran

2018-08-06

Contents

Parte I

Requerimientos

Para comenzar el trabajo se necesita la última versión de R y RStudio (R Core Team, 2018).También se requiere de los paquetes *pacman*, *rmarkdown*, *tidyverse* y *tinytex*. Si no se ha usado R o RStudio anteriormente, el siguiente video muestra cómo instalar ambos programas y los paquetes necesarios para este curso en el siguiente link.

El código para la instalación de esos paquetes es el siguiente:

```
install.packages("pacman", "rmarkdown", "tidyverse", "tinytex")
```

En caso de necesitar ayuda para la instalación, contactarse con el instructor del curso.

0.1 Antes de comenzar

Si nunca se ha trabajado con `R` antes de este curso, una buena herramienta es provista por el paquete Swirl (Kross et al., 2017). Para comenzar la práctica, realizar los primeros 7 modulos del programa *R Programming: The basics of programming in R* que incluye:

- Basic Building Blocks
- Workspace and Files
- Sequences of Numbers
- Vectors
- Missing Values
- Subsetting Vectors
- Matrices and Data Frames

El siguiente link muestra un video explicativo de cómo usar el paquete swirl Video

0.2 Descripción del curso

Este curso está enfocado en entregar principios básicos de investigación reproducible en R, con énfasis en la recopilación y/o lectura de datos de forma reproducible y automatizada. Para esto se trabajará con bases de datos complejas, las cuales deberán ser transformadas y organizadas para optimizar su análisis. Se generarán documentos reproducibles integrando en un documento: código, bibliografía, exploración y análisis de datos. Se culminará el curso con la generación de un manuscrito, una presentación y/o un documento interactivo reproducible.

0.3 Objetivos del curso

1. Conocer y entender el concepto de investigación reproducible como una forma y filosofía de trabajo que permite que las investigaciones sean más ordenadas y replicables, desde la toma de datos hasta la escritura de resultados.

2. Conocer y aplicar el concepto de pipeline, el cual permite generar una modularidad desde la toma de datos hasta la escritura de resultados, donde la corrección independiente de un paso tiene un efecto cascada sobre el resultado final.

3. Aprender buenas prácticas de recolección y estandarización de bases de datos, con la finalidad de optimizar el análisis de datos y la revisión de éstas por pares.

4. Realizar análisis críticos de la naturaleza de los datos al realizar análisis exploratorios, que permitirán determinar la mejor forma de comprobar hipótesis asociadas a estas bases de datos.

0.4 Contenidos

- Capítulo 1 *Tidy Data*: En este capítulo se aprenderá a cómo optimizar una de base de datos, sobre la limpieza y transformación de bases de datos, qué es una base de datos *tidy* y cómo manipular estas bases de datos con el paquete *dplyr* (Wickham et al., 2018).

- Capítulo 2 *Investigación reproducible*: En este capítulo se trabajará en la confección de un documento que combine códigos de `R` y texto para generar documentos reproducibles utilizando el paquete *rmarkdown* (Allaire et al., 2018). Además, se verá cómo al usar RStudio se pueden guardar los proyectos en un repositorio de github.

- Capítulo 3 *El tidyverso* y el concepto de pipeline:En este capítulo se aprenderá sobre la limpieza de datos complejos.

4. Visualización de datos, visualizar datos vs. visualizar modelos. Insertar gráficos con leyenda en un documento Rmd

5. Creación de funciones propias y loops. Generación de funciones propias en R y loops

6. Escritura de manuscritos en R, transformación de documentos Rmd en un manuscrito

7. Presentaciones en R y generar documentos interactivos. Transformación de datos en una presentación o en una Shiny app. Realizar una presentación o aplicación en R.

0.5 Metodología

Todas las clases estarán divididas en dos partes: I. Clases expositivas de principios y herramientas, donde se presentarán los principios de investigación reproducible y tidy data, junto con las herramientas actuales más utilizadas, y II. Clases prácticas donde cada estudiante trabajará con datos propios para desarrollar un documento reproducible. Los estudiantes que no cuenten con datos propios podrán acceder a sets de datos para su trabajo o podrán simularlos, dependiendo del caso.

Además, se deberán generar informes y presentaciones siguiendo los principios de investigación reproducible, en base al trabajo con sus datos. Se realizará un informe final, en el cual se espera un trabajo que compile los conociminetos adquiridos durante el curso.

0.6 Evaluación

- Evaluación 1: Informe exploratorio de base de datos 25%
- Evaluación 2: Presentación 25%
- Evaluación 3: Informe final 50%

0.7 Libros de consulta

Los principios de este curso están explicados en los siguientes libros gratuitos.

- Gandrud, Christopher. Reproducible Research with R and R Studio. CRC Press, 2013. Available for free in the following link
- Stodden, Victoria, Friedrich Leisch, and Roger D. Peng, eds. Implementing reproducible research. CRC Press, 2014. Available for free in the following link

0.8 Bibliografía

Chapter 1

Tidy Data y manipulación de datos

1.1 Paquetes necesarios para este capítulo

Para este capitulo necesitas tener instalado el paquete *tidyverse*

En este capítulo se explicará qué es una base de datos *tidy* (Wickham et al., 2014) y se aprenderá a usar funciones del paquete *dplyr* (Wickham et al., 2018) para manipular datos.

Dado que este libro es un apoyo para el curso BIO4022, esta clase del curso puede también ser seguida en este link. El video de la clase se encuentra disponible en este link.

1.2 Tidy data

Una base de datos tidy es una base de datos en la cuál (modificado de (Leek, 2015)):

- Cada vararible que se medida debe estar en una columna.
- Cada observación distinta de esa variable debe estar en una fila diferente.

En general, la forma en que representaríamos una base de datos *tidy* en `R` es usando un *data frame*.

1.3 dplyr

El paquete *dplyr* es definido por sus autores como una gramática para la manipulación de datos. De este modo sus funciones son conocidas como verbos. Un resumen útil de muchas de estas funciones se encuentra en este link.

Este paquete tiene un gran número de verbos y sería difícil ver todos en una clase, en este capítulo nos enfocaremos en sus funciones más utilizadas, las cuales son:

- *group_by* (agrupa datos)
- *summarize* (resume datos agrupados)
- *mutate* (genera variables nuevas)
- *%>%* (pipeline)
- *filter* (encuentra filas con ciertas condiciones)
- *select* junto a *starts_with*, *ends_with* o *contains*

Table 1.1: una tabla con 10 filas de la base de datos iris.

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.8	4.0	1.2	0.2	setosa
4.7	3.2	1.6	0.2	setosa
5.1	3.8	1.9	0.4	setosa
5.2	2.7	3.9	1.4	versicolor
6.4	2.9	4.3	1.3	versicolor
5.5	2.5	4.0	1.3	versicolor
6.5	3.0	5.8	2.2	virginica
6.0	2.2	5.0	1.5	virginica
6.1	2.6	5.6	1.4	virginica
5.9	3.0	5.1	1.8	virginica

Table 1.2: Resumen del promedio y desviación estándar del largo de pétalo de las flores del generi Iris.

Mean.Petal.Length	SD.Petal.Length
3.758	1.765298

1.3.1 summarize

La función `summarize` toma los datos de un data frame y los resume. Para usar esta función, el primer argumento que tomaríamos sería un data frame, se continúa del nombre que queremos darle a una variable resumen, seguida del signo = y luego la fórmula a aplicar a una o mas columnas. COmo un ejemplo se utilizará la base de datos `iris` (Anderson, 1935) que viene en `R` y de las cual podemos ver parte de sus datos en la tabla 1.1

Si quisieramos resumir esa tabla y generar un par de variables que fueran la media y la desviación estándar del largo del pétalo, lo haríamos con el siguiente código:

```
library(tidyverse)
Summary.Petal <- summarize(iris, Mean.Petal.Length = mean(Petal.Length),
    SD.Petal.Length = sd(Petal.Length))
```

El resultado se puedde ver en la tabla 1.2, en el cuál se obtienen los promedios y desviaciones estándar de los largos de los pétalos. Es importante notar que al usar summarize, todas las otras variables desapareceran de la tabla.

1.3.2 group_by

La función `group_by` por si sola no genera cambios visibles en las bases de datos. Sin embargo, al ser utilizada en conjunto con `summarize` permite resumir una variable agrupada (usualmente) basada en una o más variables categóricas.

Se puede ver que para el ejemplo con el caso de las plantas del género *Iris*, el resumen que se obtiene en el caso de la tabla 1.2 no es tan útil considerando que tenemos tres especies presentes. Si se quiere ver el promedio del largo del pétalo por especie, se debe ocupar la función `group_by` de la siguiente forma:

Table 1.3: Resumen del promedio y desviación estándar del largo de pétalo de las flores del generi Iris.

Species	Mean.Petal.Length	SD.Petal.Length
setosa	1.462	0.1736640
versicolor	4.260	0.4699110
virginica	5.552	0.5518947

Table 1.4: Millas por galón promedio en vehiculos automáticos (am = 0) y manuales (am = 1), con los distintos tipos de cilindros

cyl	am	Eficiencia
4	0	22.90000
4	1	28.07500
6	0	19.12500
6	1	20.56667
8	0	15.05000
8	1	15.40000

```
BySpecies <- group_by(iris, Species)
Summary.Byspecies <- summarize(BySpecies, Mean.Petal.Length = mean(Petal.Length),
    SD.Petal.Length = sd(Petal.Length))
```

Esto dá como resultado la tabla 1.3, con la cuál se puede ver que *Iris setosa* tiene pétalos mucho más cortos que las otras dos especies del mismo género.

1.3.2.1 group_by en más de una variable

Se puede usar la función `group_by` en más de una variable, y esto generaría un resumen anidado. Como ejemplo se usará la base de datos `mtcars` presente en R (Henderson and Velleman, 1981). Esta base de datos presenta una variable llamada *mpg* (miles per gallon) y una medida de eficiencia de combustible. Se resumirá la información en base a la variable *am* (que se refiere al tipo de transmisión, donde 0 es automático y 1 es manual) y al número de cilindros del motor. Para eso se utilizará el siguiente código:

```
Grouped <- group_by(mtcars, cyl, am)
Eficiencia <- summarize(Grouped, Eficiencia = mean(mpg))
```

Como puede verse en la tabla 1.4, en todos los casos los autos con cambios manuales tienen mejor eficiencia de combustible. Se podría probar el cambiar el orden de las variables con las cuales agrupar y observar los distintos resultados que se pueden obtener.

1.3.3 mutate

Esta función tiene como objetivo crear variables nuevas basadas en otras variables. Es muy facil de usar, como argumento se usa el nombre de la variable nueva que se quiere crear y se realiza una operación con variables que ya estan ahí. Por ejemplo, si se continúa el trabajo con la base de datos *Iris*, al crear una nueva variable que sea la razón entre el largo del pétalo y el del sépalo, resulta lo siguiente:

Table 1.5: Tabla con diez de las observaciones de la nueva base de datos con la variable nueva creada con mutate

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Petal.Sepal.Ratio
5.8	4.0	1.2	0.2	setosa	0.21
4.7	3.2	1.6	0.2	setosa	0.34
5.1	3.8	1.9	0.4	setosa	0.37
5.2	2.7	3.9	1.4	versicolor	0.75
6.4	2.9	4.3	1.3	versicolor	0.67
5.5	2.5	4.0	1.3	versicolor	0.73
6.5	3.0	5.8	2.2	virginica	0.89
6.0	2.2	5.0	1.5	virginica	0.83
6.1	2.6	5.6	1.4	virginica	0.92
5.9	3.0	5.1	1.8	virginica	0.86

```
DF <- mutate(iris, Petal.Sepal.Ratio = Petal.Length/Sepal.Length)
```

El resultado de esta operación es la tabla 1.5. Siempre la variable que se acaba de crear aparecerá al final del data frame.

1.3.4 Pipeline (%>%)

El pipeline es un simbolo operatorio `%>%` que sirve para realizar varias operaciones de forma secuencial sin recurrir a parentesis anidados o a sobrescribir muúltiples bases de datos.

Para ver como funciona esto como un vector, supongamos que se tiene una variable a la cual se quiere primero obtener su logaritmo, luego su raíz cuadrada y finalmente su promedio con dos cifras significativas. Para realizar esto se debe seguir lo siguiente:

```
x <- c(1, 4, 6, 8)
y <- round(mean(sqrt(log(x))), 2)
```

Si se utiliza pipeline, el código sería mucho más ordenado. En ese caso, se partiría por el objeto a procesar y luego cada una de las funciones con sus argumentos si es necesario:

```
x <- c(1, 4, 6, 8)
y <- x %>% log() %>% sqrt() %>% mean() %>% round(2)
```

```
## [1] 0.99
```

El código con pipeline es mucho más fácil de interpretar a primera vista ya que se lee de izquierda a derecha y no de adentro hacia afuera. EL uso de pipeli se hace aun más importante cuando se usa con un *Data frame*, como se ve en el siguiente ejemplo:

1.3.4.1 El pipeline en data frames

POr ejemplo se quiere resumir la variable recien creada de la razón entre el sépalo y el petalo. Para hacer esto, si se partiera desde la base de datos original, tomaría varias líneas de código y la creación de múltiples bases de datos intermedias

Table 1.6: Razón pétalo sépalo promedio para las tres especies de Iris

Species	MEAN	SD
setosa	0.2927557	0.0347958
versicolor	0.7177285	0.0536255
virginica	0.8437495	0.0438064

Table 1.7: Símbolos lógicos de R y su significado

simbolo	significado	simbolo_cont	significado_cont
>	Mayor que	!=	distinto a
<	Menor que	%in%	dentro del grupo
==	Igual a	is.na	es NA
>=	mayor o igual a	!is.na	no es NA
<=	menor o igual a	| &	o, y

```
DF <- mutate(iris, Petal.Sepal.Ratio = Petal.Length/Sepal.Length)
BySpecies <- group_by(DF, Species)
Summary.Byspecies <- summarize(BySpecies, MEAN = mean(Petal.Sepal.Ratio),
    SD = sd(Petal.Sepal.Ratio))
```

Otra opción es usar paréntesis anidados, lo que se traduce en el siguiente código:

```
Summary.Byspecies <- summarize(group_by(mutate(iris, Petal.Sepal.Ratio = Petal.Length/Sepal.Length),
    Species), MEAN = mean(Petal.Sepal.Ratio), SD = sd(Petal.Sepal.Ratio))
```

Esto se simplifica mucho más al usar el pipeline, lo cual permite partir en un *Data Frame* y luego usar el pipeline. Esto permite obtener el mismo resultado que en las operaciones anteriores con el siguiente código:

```
Summary.Byspecies <- iris %>% mutate(Petal.Sepal.Ratio = Petal.Length/Sepal.Length) %>%
    group_by(Species) %>% summarize(MEAN = mean(Petal.Sepal.Ratio),
    SD = sd(Petal.Sepal.Ratio))
```

Estos tres códigos son correctos (tabla 1.6), pero definitivamente el uso del pipeline da el código más conciso y fácil de interpretar sin pasos intermedios.

1.3.5 filter

Esta función permite seleccionar filas que cumplen con ciertas condiciones, como tener un valor mayor a un umbral o pertenecer a cierta clase Los símbolos más típicos a usar en este caso son los que se ven en la tabla 1.7.

Por ejemplo si se quiere estudiar las características florales de las plantas del género *Iris*, pero no tomar en cuenta a la especie *Iris versicolor* se deberá usar el siguiente código:

```
data("iris")
DF <- iris %>% filter(Species != "versicolor") %>% group_by(Species) %>%
    summarise_all(mean)
```

Table 1.8: Resumen de la media de todas las características florales de las especies Iris setosa e Iris virginica

Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	5.006	3.428	1.462	0.246
virginica	6.588	2.974	5.552	2.026

Table 1.9: Número de plantas de cada especie con un largo de pétalo mayor a 4 y un largo de sépalo mayor a 5 centímetros

Species	N
versicolor	39
virginica	49

De esta forma se obtiene como resultado la tabla 1.8. En este caso se introduce la función `summarize_all` de `summarize`, la cual aplica la función que se le da como argumento a todas las variables de la base de datos.

Por otro lado si se quiere estudiar cuántas plantas de cada especie tienen un largo de pétalo mayor a 4 y un largo de sépalo mayor a 5 se deberá usar el siguiente código:

```
DF <- iris %>% filter(Petal.Length >= 4 & Sepal.Length >= 5) %>%
    group_by(Species) %>% summarise(N = n())
```

En la tabla tabla 1.9 se ve que con este filtro desaparecen de la base de datos todas las plantas de *Iris setosa* y que todas menos una planta de *Iris virginica* tienen ambas características.

1.3.6 select

Esta función permite seleccionar las variables a utilizar dado que en muchos casos nos encontraremos con bases de datos con demasiadas variables y por lo tanto, se querrá reducirlas para solo trabajar en una tabla con las variables necesarias.

Con select hay varias formas de trabajar, por un lado se puede escribir las variables que se utilizarán, o restar las que no. En ese sentido estos cuatro códigos dan exactamente el mismo resultado. Esto se puede ver en la tabla 1.10

```
iris %>% group_by(Species) %>% select(Petal.Length, Petal.Width) %>%
    summarize_all(mean)
```

```
iris %>% group_by(Species) %>% select(-Sepal.Length, -Sepal.Width) %>%
    summarize_all(mean)
```

```
iris %>% group_by(Species) %>% select(contains("Petal")) %>%
    summarize_all(mean)
```

```
iris %>% group_by(Species) %>% select(-contains("Sepal")) %>%
    summarize_all(mean)
```

Table 1.10: Promedio de largo de pétalo y ancho de pétalo para las especies del genero Iris

Species	Petal.Length	Petal.Width
setosa	1.462	0.246
versicolor	4.260	1.326
virginica	5.552	2.026

1.3.7 Ejercicios

1.3.7.1 Ejercicio 1

Usando la base de datos `storm` del paquete *dplyr*, calcular la velocidad promedio y diámetro promedio (hu_diameter) de las tormentas que han sido declaradas huracanes para cada año.

1.3.7.2 Ejercicio 2

La base de datos `mpg` del paquete ggplot2 tiene datos de eficiencia vehicular en millas por galón en ciudad (*cty*) en varios vehículos. Obtener los datos de vehículos del año 2004 en adelante que sean compactos y transformar la eficiencia Km/litro (1 km = 1.609 millas; 1 galón = 3.78541 litros)

Las soluciones a estos ejercicios se encuentran en el capítulo 8

Chapter 2

Investigación reproducible

2.1 Paquetes necesarios para este capítulo

Para este capitulo necesitas tener instalado los paquetes *rmarkdown*, *knitr* y *stargazer*

En este capítulo explicaremos que es la investigación reproducible, y como aplicarla usando github, y los paquetes *rmarkdown* (Allaire et al., 2018) y *knitr* (Xie, 2015). Además aprenderemos a usar tablas usando *knitr* (Xie, 2015) y *stargazer* (Hlavac, 2018)

Recuerda que este libro es un apoyo para el curso BIO4022, puedes seguir la clase de este curso en este link, y en cuanto el video de la clase este disponible encontrarás un link aca.

2.2 Investigación reproducible

La investigación reproducible no es lo mismo que la investigación replicable. La replicabilidad implica que experimentos o estudios llevados a cabo en condiciones similares nos llevarán a conclusiones similares, mientras que la investigación reproducible implica que desde los mismos datos, y/o el mismo código.

En la figura 2.1 vemos el continuo de replicabilidad (Peng, 2011). En este continuo tenemos el ejemplo de no reproducibilidad como una publicación sin código. Pasando de menos a más reproducible por la publicación y el código que genero los resultados y gráficos; seguido por la publicación, el código y los datos que generan los resultados y gráficos; y por último código, datos y texto entrelazados de forma tal que al correr el código obtenemos exactamente la mismma publicación que leimos.

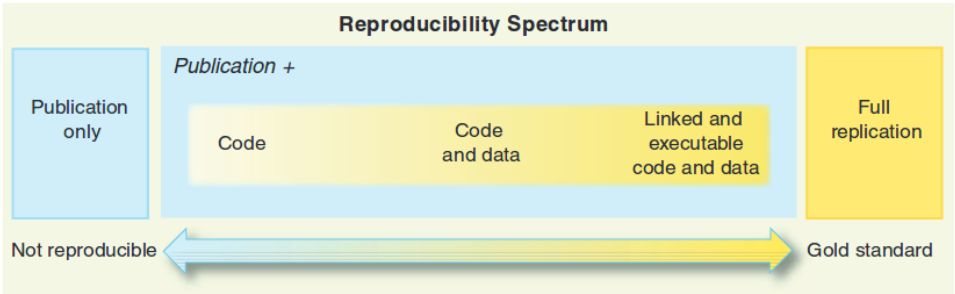

Figure 2.1: Continuo de reproducibilidad (extraido de Peng 2011)

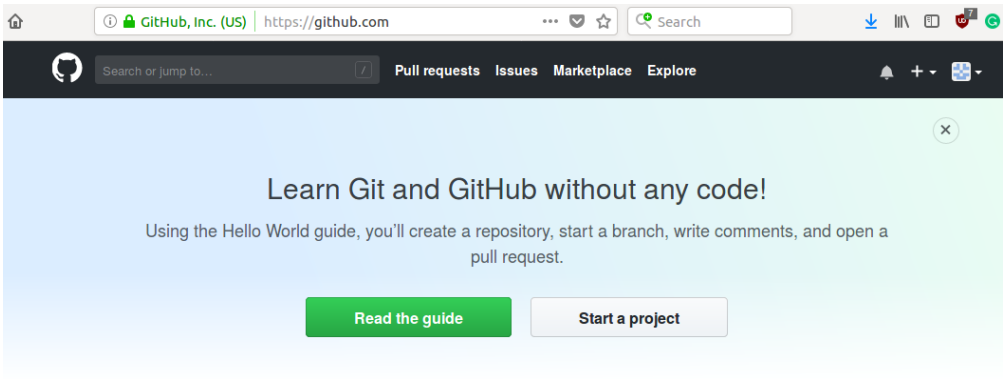

Figure 2.2: Para empezar un projecto en github, debes presionar Start a project en tu página de inicio

Esto tiene muchas ventajas, incluyendo el que es más fácil aplicar exactamente los mismos metodos a otra base de datos, basta poner la nueva base de datos en el formato que tenía el autor de la primera publicación y podremos comparar los resultados.

Además en un momento en que la ciencia está basada cada vez más en bases de datos, se puede poner en el código la recolección y/o muestreo de datos.

2.3 Guardando nuestro proyecto en github

2.3.1 Que es github?

Github es una suerte de dropbox o google drive pensado para la investigación reproducible, en github cada proyecto es un *repositorio*. La mayoría de los investigadores que trabajan en investigación reproducible dejan todo su trabajo documentado en sus repositorios, lo cual permite interactuar con otros autores.

2.3.2 creando un proyecto de github en RStudio

Para crear un proyecto en github presionamos **start a project** en la pagina inicial de nuestra cuenta como vemos en la figura 2.2

Luego crea un nombre unico, y sin cambiar nada más presiona **create repository** en el botón verde como vemos en la figura 2.3.

Esto te llevará a una página donde te aparecera una url de tu nuevo repositorio como en la figura 2.4

Para incorporar tu proyecto en tu repositorio, lo primero que debes hacer es generar un proyecto en RStudio, para esto debes ir en el menú superior de *Rstudio* a *File* > *New Project* > *Git* como se ve en las figuras 2.5 y 2.5.

Luego, seleccionar la ubicación del proyecto nuevo y pegar el url que aparece en la figura 2.4 en el espacio que dice **Repository URL:** como muestra en la figura 2.7.

Cuando tu proyecto de R ya este siguiendo los cambios en github, te aparecera una pestaña git dentro de la ventana superior derecha de tu sesión de RStudio, tal como vemos en la figura 2.8

2.3.3 Los tres principales pasos de un repositorio

Github es todo un mundo, con muchas funciones, hay expertos en el uso de github, pero en este curso, nos enfocaremos en los 3 pasos principales de un repositorio, *Add*, *commit* y *push*. Para entender bien que

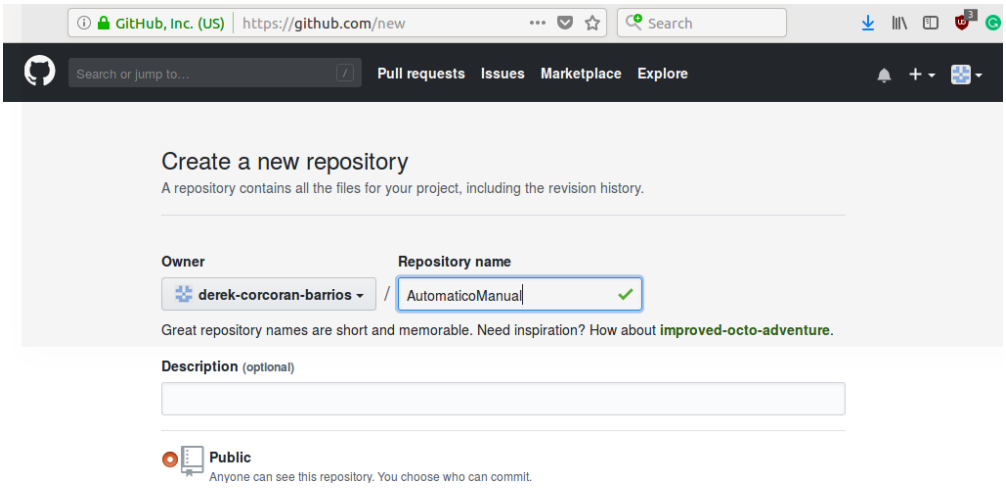

Figure 2.3: Crea el nombre de tu repositorio y apreta el boton create repository

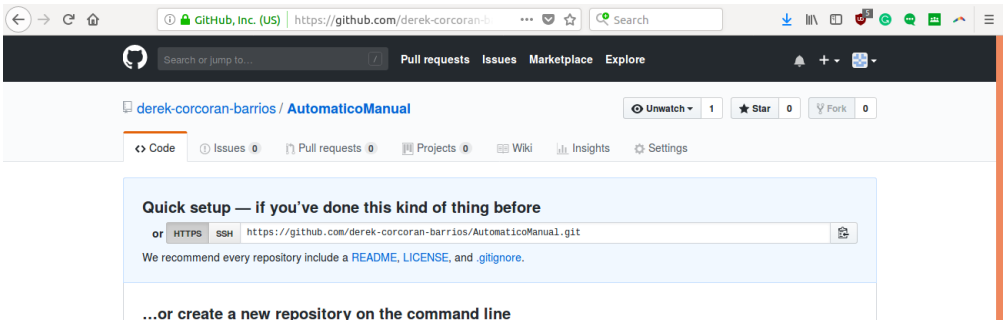

Figure 2.4: El contenido del cuadro en el cual dice ssh es la url de tu repisitorio

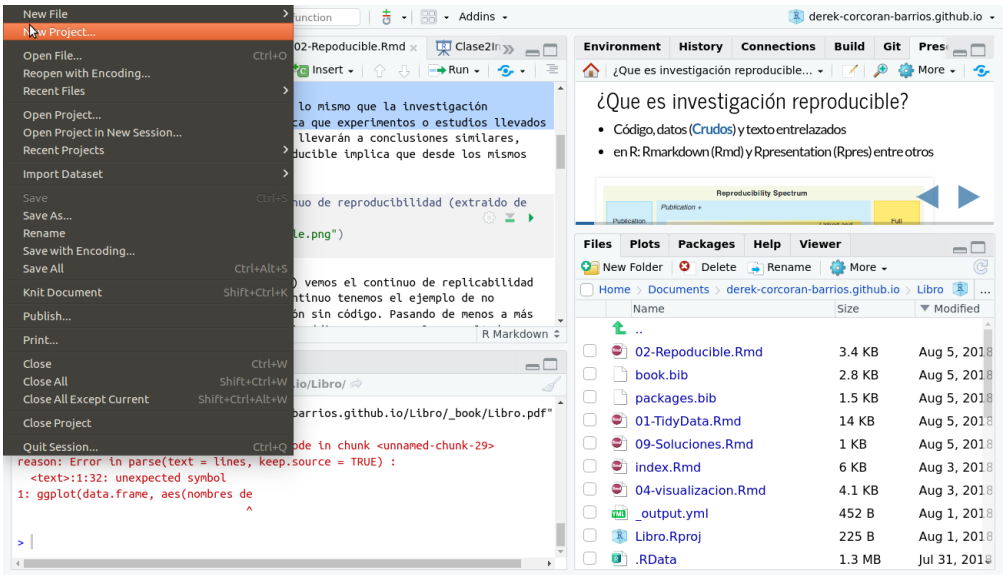

Figure 2.5: Menú para crear un proyecto nuevo

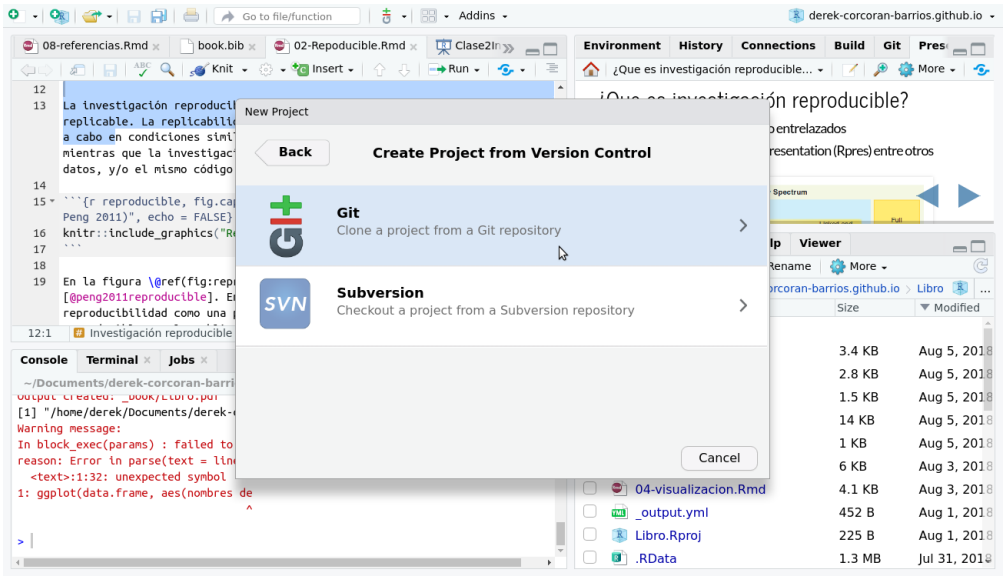

Figure 2.6: Seleccionar git dentro de las opciones

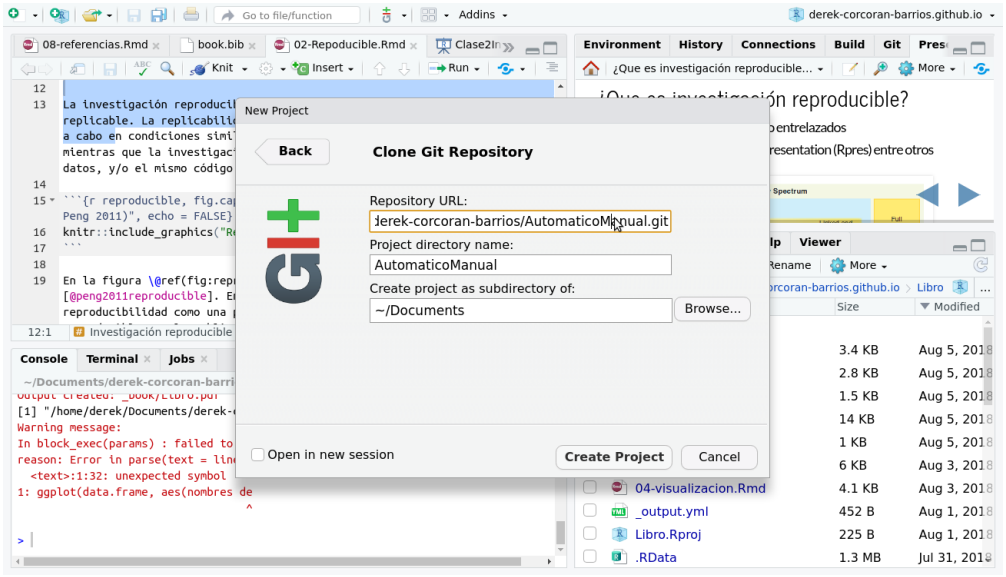

Figure 2.7: Pegar el url del repositorio en el cuadro de dialogo Repository URL:

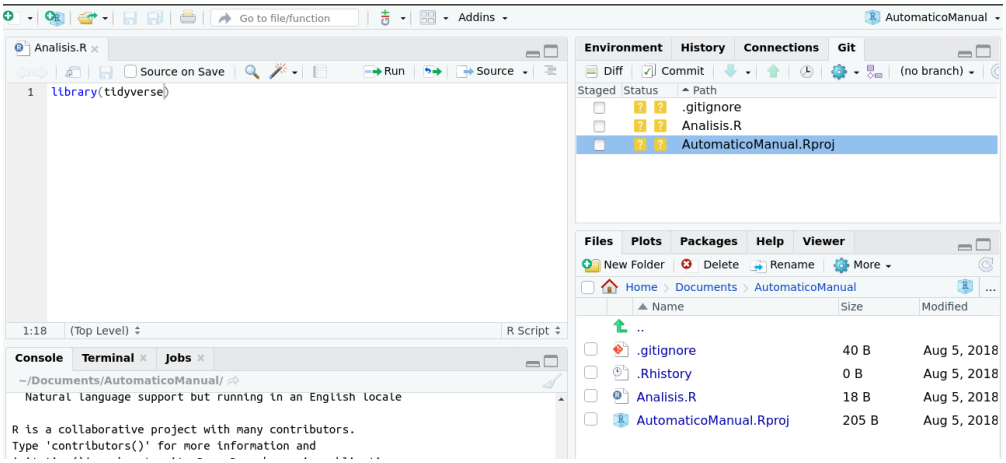

Figure 2.8: Al incluir tu repositorio en tu sesión de Rstudio, aparecera la pestaña git en la ventana superior derecha

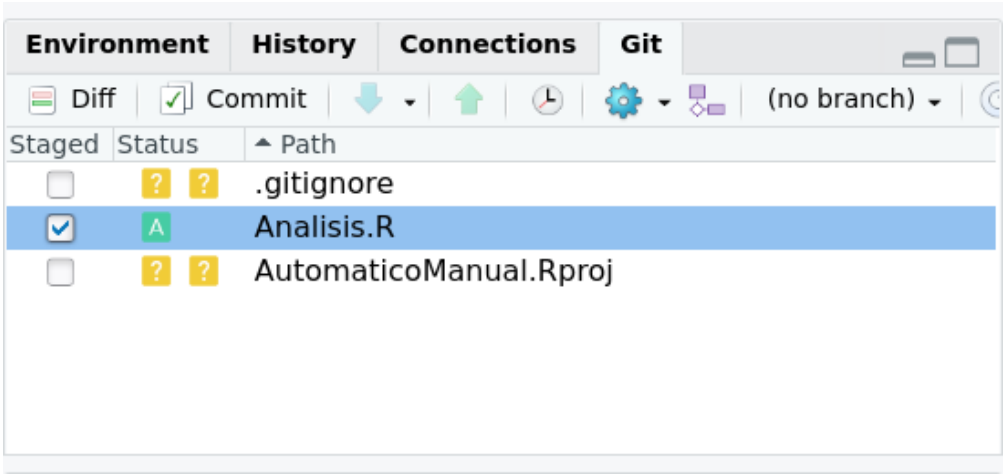

Figure 2.9: Al incluir tu repositorio en tu sesión de Rstudio, aparecera la pestaña git en la ventana superior derecha

significan cada uno de estos pasos tenemos que entender que existen dos repositorios en todo momento. Uno local (en tu computador) u otro remoto (en github.com). Los dos primeros pasos *Add* y commit, solo generan cambios en tu repositorio local. Mientras que *push*, salva los cambios al repositorio remoto.

2.3.3.1 git add

Esta función, es la que agrega archivos a tu repositorio local. Solo estos archivos serán guardados en github. Cuando tienes bases de datos mayores a 1 GB no es convenientes guardarlos en github, ya que si bién te dan repositorios ilimitados, el espacio de cada uno no lo es, en particular en cuanto a bases de datos. Para adicionar un archivo a tu repositorio tan solo debes selecionar los archivos en la pestaña git. Al hacer eso una letra A verde aparecera en vez de los dos signos de interrogación amarillos, como vemos en la figura 2.9. En este caso solo adicionamos al repositorio el archivo *Analisis.r* pero no el resto.

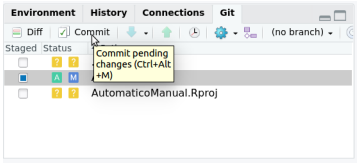

Figure 2.10: Para guardar los cambios en tu repositorio apretar commit en la pestaña git de la ventana superior derecha

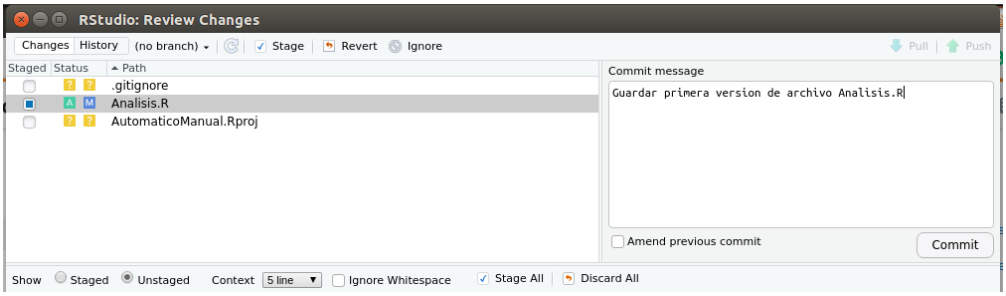

Figure 2.11: Escribir un mensaje que recuerde los cambios que hiciste en la ventana emergente

2.3.3.2 git commit

Cuando ocupas el comando *commit* estas guardando los cambios de los archivos que adicionaste, en tu repositorio local. Para hacer esto en Rstudio, en la misma pestaña de git, debes presionar el botón commit como vemos en la figura 2.10.

Al presionar Commit, se abrira una ventana emergente, donde deberás escribir un mensaje que describa lo que guardaras. Una vez echo eso presiona commit nuevamente en la ventana emergente como aparece en la figura 2.11.

2.3.3.3 git push

Finalmente *push* te permitirá guardar los cambios en tu repositorio remoto, lo cual asegura tus datos en la nube, y además lo hace disponible a otros investigadores. Luego de apretar commit en la ventana emergente (figura 2.11), podemos presionar *push* en la flecha verde de la ventana emergente como se ve el a figura 2.12. Luego se nos pedira nuestro nombre de usuario y contraseña, y ya podemos revisar que nuestro repositorio esta online entrando a nuestra sesión de github.

2.4 Reproducibilidad en R

Existen varios paquetes que permiten que hagamos investigación reproducible en `R`, pero sin duda los más relevantes son *rmarkdown* y *knitr*. Ambos paquetes funcionana en conjunto cuando generamos un archivo *Rmd* (Rmarkdown), en el cual ocupamos al mismo tiempo texto, código de R y otros elementos para generar un documento word, pdf, página web, presentación y/o aplicación web (fig 2.13).

2.4.1 Creando un Rmarkdown

Para crear un archivo Rmarkdown, simplemente ve a el menu *File* > *New file* > *Rmarkdown* y con eso habrás creado un nuevo archivo *Rmd*, veremos en momentos algunos de los elementos más típicos de un arcchivo Rmarkdown.

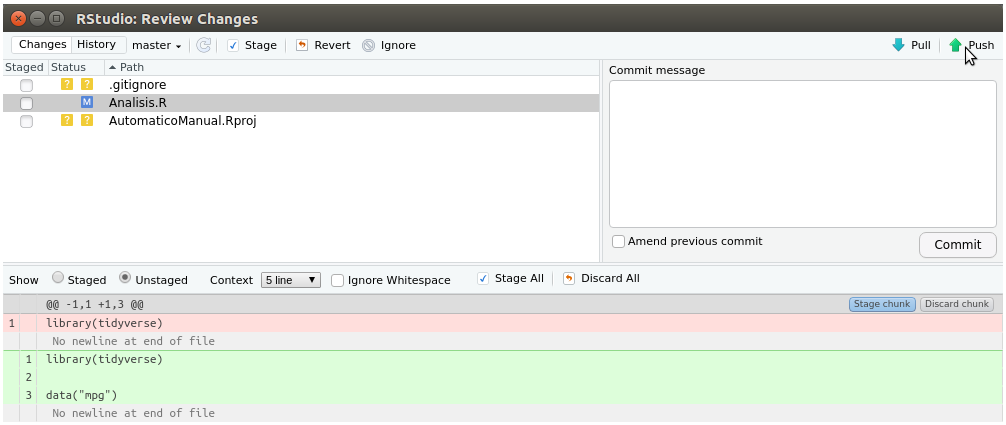

Figure 2.12: Para guardar en el repositorio remoto apretar push en la ventana emergente

Figure 2.13: El objetivo de Rmarkdown es el unir código de r con texto y datos para generar un documento reproducible

2.4.1.1 Markdown

El markdown es la parte del archivo en que simplemente escribimos texto, aunque tiene algunos detalles para el formato como generar texto en negrita, cursiva, títulos y subtitulos.

Para hacer que un texto este en **negrita**, deben ponerlo entre dos asteriscos `**negrita**`, para que un texto aparezca en *cursiva* debe estar entre asteriscos `*cursiva*`, otros ejemplos son los titulos de distintos niveles, los cuales se denotan con distintos números de `#`, así los siguientes 4 títulos o subtitulos:

subtitulo 1

subtítulo 2

subtítulo 3

2.4.1.1.1 subtítulo 4

se vería de la siguiente manera en el código

```
## subtitulo 1

### subtítulo 2

#### subtítulo 3

##### subtítulo 4
```

2.4.1.2 Chunks

Los *chunks* son una de las partes más importantes del un Rmarkdown. En estos es donde se agrega el código de R (u otros lenguajes de programación). Lo cual permíte que el producto de nuestro código no sea solo un escrito con resultados pegados, sino que efectivamente generados en el mismo documento que nuestro escrito, la forma más fácil de agregar un chunk es apretando el botón de insert chunk en Rstudio, este boton se encuentra en la ventana superior izquierda de nuestra sesión de RStudio, tal como se muestra en la figura 2.14

Al apretar este código aparecera un espacio, uno podría agregar un código como el que aparece a continuación, y vemos a continuación los resultados.

````
```{r}
library(tidyverse)
iris %>% group_by(Species) %>% summarize(Petal.Length = mean(Petal.length))
```
````

```
## # A tibble: 3 x 2
##   Species    Petal.Length
##   <fct>             <dbl>
## 1 setosa             1.46
## 2 versicolor         4.26
## 3 virginica          5.55
```

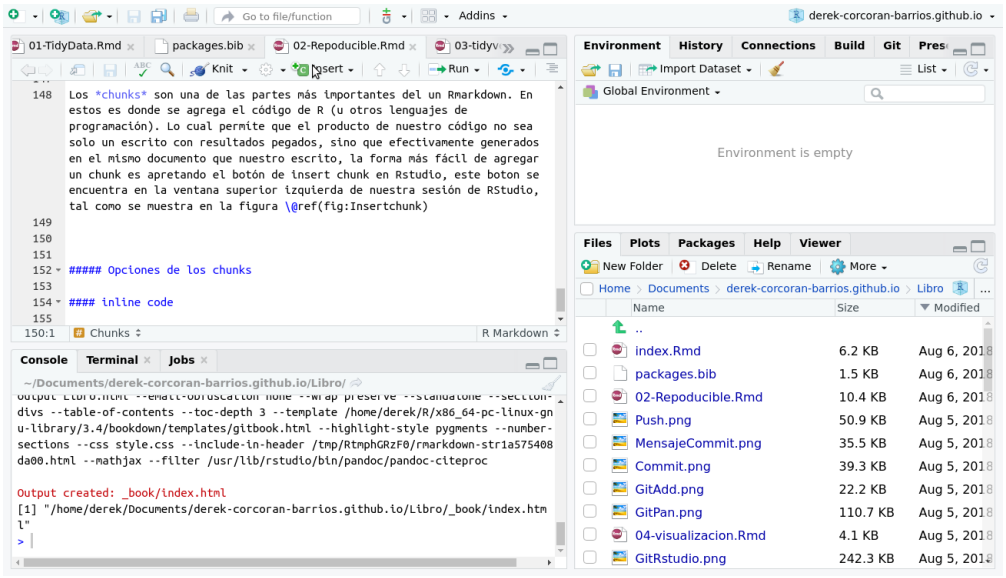

Figure 2.14: Al apretar el botón insert chunk, aparecera un espacio en el cuál insertar código

2.4.1.2.1 Opciones de los chunks

Existen muchas opciones para los chunks, una documentación completa podemos encontrarle en el siguiente link, pero acá mostraremos los más comunes:

- *echo* = T o F muestro o no codigo
- *message* = T o F muestra mensajes de paquetes
- *warning* = T o F muestra advertencias
- *eval* = T o F evaluar o no el código
- *cache* = T o F guarda o no el resultado

2.4.1.3 inline code

syntax for the latter is `` `r R_CODE` ``, and it can be embedded inline with other document elements. R code chunks look like plain code blocks, but have `{r}` after the three backticks and (optionally) chunk options inside `{}`, e.g.,

2.4.1.4 Tablas en Rmarkdown

Chapter 3

El Tidyverso

We describe our methods in this chapter.

Chapter 4

Visualización de datos

En este capítulo aprenderemos a usar el paquete *ggplot2* (Wickham, 2016), parte del paquete *tidyverse* (Wickham, 2017).

4.1 El esqueleto

El esqueleto de una visualización usando *ggplot2* es la siguiente

```
ggplot(data.frame, aes(nombres_de_columna)) + geom_algo(argumentos,
    aes(columnas)) + theme_algo()
```

Como ejemplo para discutir usaremos el siguiente código que genera la figura 4.1:

```
library(tidyverse)
data("diamonds")
ggplot(diamonds, aes(x = carat, y=price)) + geom_point(aes(color = cut)) + theme_classic()
```

En este caso general, lo primero que ponemos después de ggplot es el data.frame desde el cuál graficaremos algo, en el ejemplo de la figura 4.1 usamos la base de datos *diamonds* del paquete *ggplot2* (Wickham, 2016). Luego dentro de `aes` ponemos las columnas que graficaremos como x y/o y, en nuestro ejemplo dentro de aes ponemos como eje x los kilates de los diamantes (caret) y como y el precio de los mismos (price). La necesidad de poner `aes` en ggplot2 (algo que no había sido necesario cuando usamos *dplyr* o *tidyr*) es que ggplot2 es el paquete mas antiguo del *tidyverse*.

4.2 geom_algo

Luego de especificar una base de datos, esto viene seguido de un `geom_algo`, esto nos indicará que tipo de gráfico usaremos, estos pueden ser combinados como veremos en ejemplos futuros

4.2.1 Una variable categórica una continua

Primero veremos algunos de los *geom* que podemos utilizar con una variable categórica y una continua

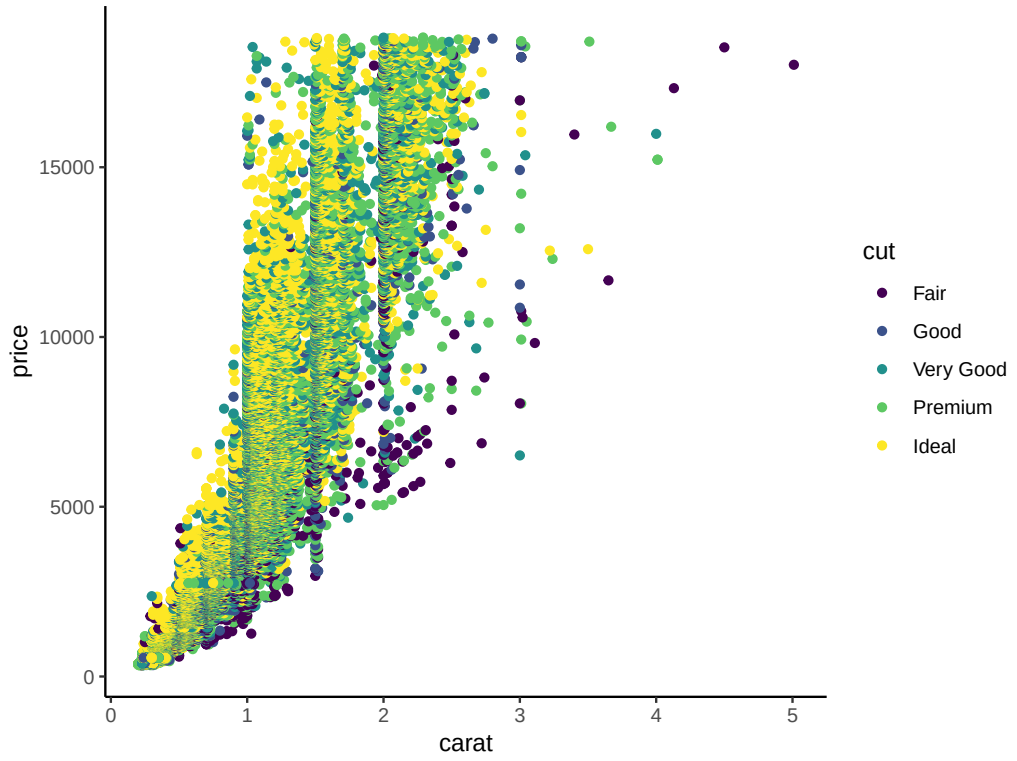

Figure 4.1: Gráfico en el cual gráficamos los quilates de diamantes versus su precio, con el corte del diamante representado por el color

4.2.1.1 geom_boxplot

En la figura 4.2, generado a partir del código a continuacón con la base de datos iris presente en `R` (Anderson, 1935).

```
data("iris")
ggplot(iris, aes(x = Species, y = Sepal.Length)) + geom_boxplot()
```

Los boxplots muestran una linea gruesa central (la mediana), una caja, que delimita el primer y tercer cuartil, y los bigotes, los cuales se extienden hasta los valores extremos. A menos que estos esten por sobre 1.5 veces la distance entre el primer y tercer cuartil, en cuyo caso se consideran outlyers, y estos son representados por puntos. En la figura 4.2, solo *Iris virginica* presenta un outlayer en cuanto a las medidas del largo del sepalo.

Los boxplots, como todos los gráficos pueden ser personalizados usando otros argumentos, los cuales son detallados en la sección 4.3, pero en los ejemplos que mostraremos en esta sección los iremos introduciendo de a poco. Si quisieramos por ejemplo que el color de las cajas del *boxplot* fuera deacuerdo a la especie, cambiamos el llenado (**fill**) de la caja, como vemos en el siguiente ejemplo y figura 4.3

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) + geom_boxplot(aes(fill = Species))
```

Dos cosas a notar en este ejemplo, por un lado la leyenda se genera de forma automática, y por otro lado, vemos que es necesario poner *Species* dentro de `aes`, esto es debido a que Species es una columna y como se explicó al principio de este capítulo, todas las columnas deben ser incuidas dentro de la función `aes` para poder ser referenciadas.

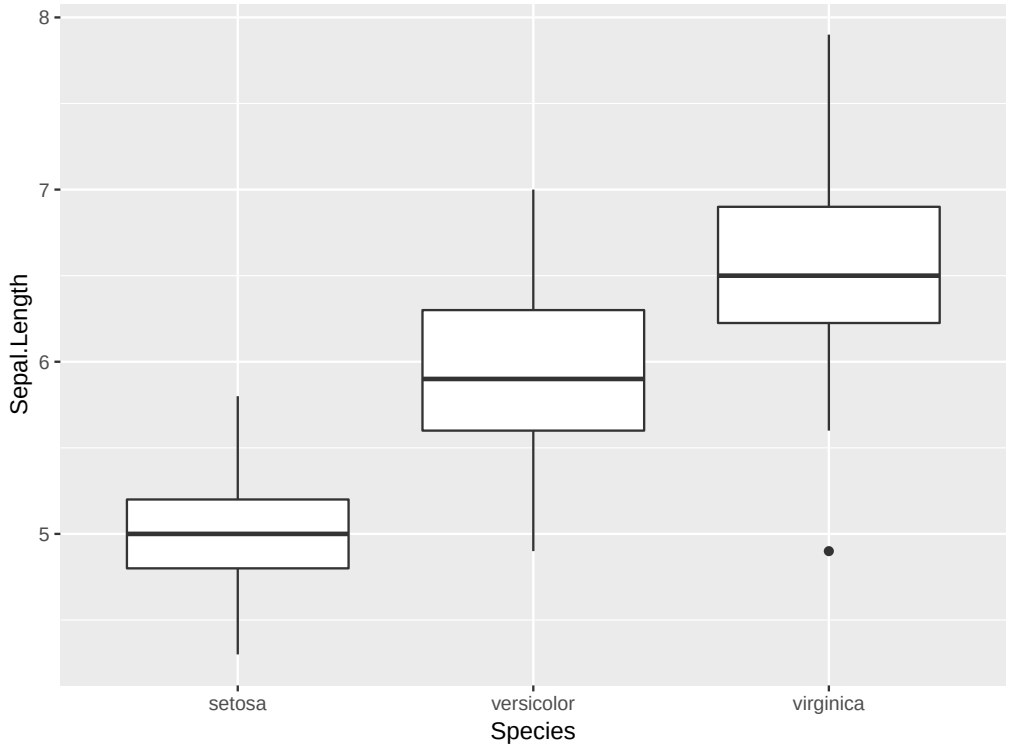

Figure 4.2: Boxplot que representa los largos del sépalo de tres especies del género Iris

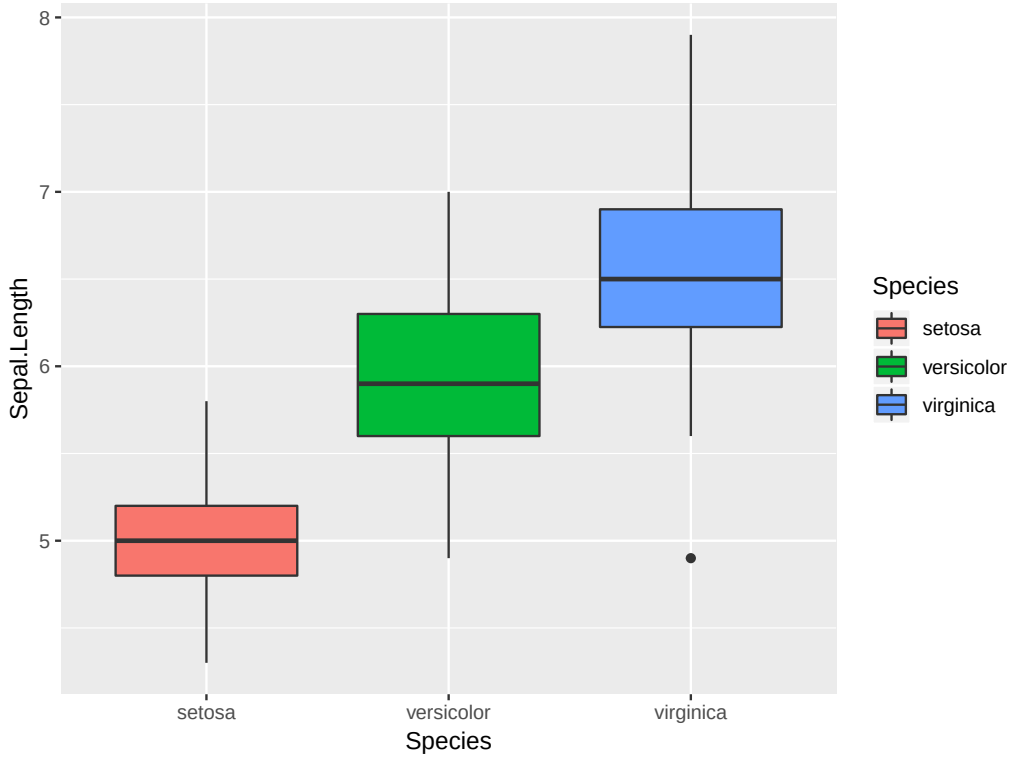

Figure 4.3: Boxplot que representa los largos del sépalo de tres especies del género Iris, en este caso el color de la caja representa la especie

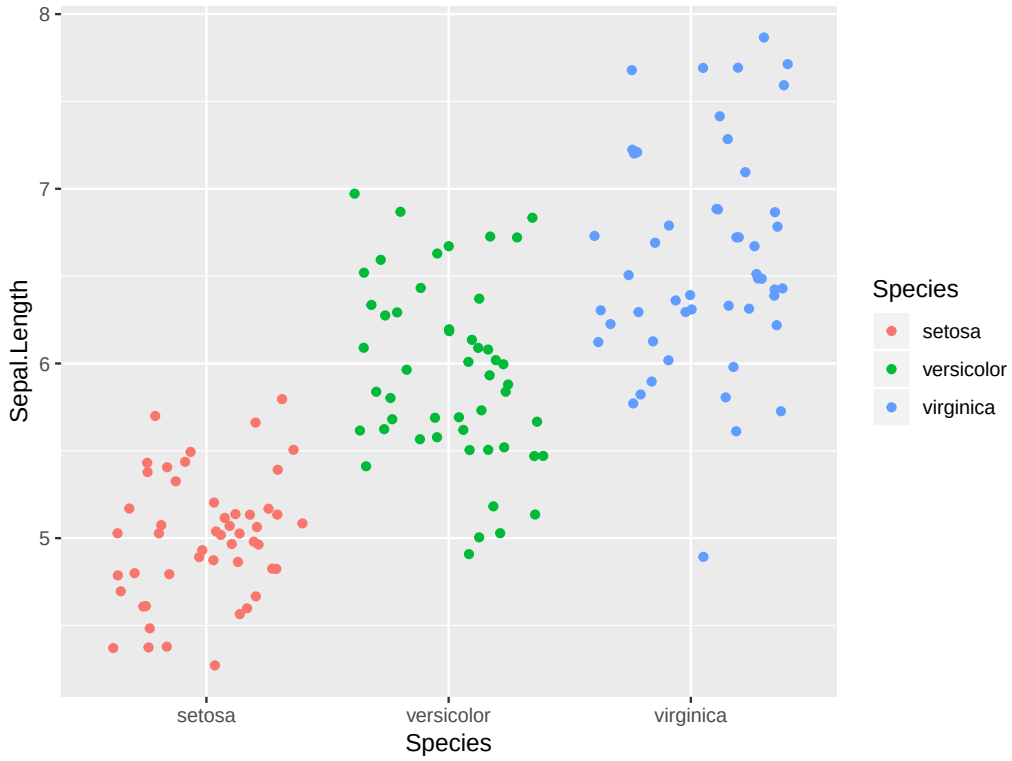

Figure 4.4: Boxplot que representa los largos del sépalo de tres especies del género Iris, en este caso el color de la caja representa la especie

4.2.1.2 geom_jitter

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) + geom_jitter(aes(color = Species))
```

4.3 Argumentos

Chapter 5

Modelos en R

We have finished a nice book.

Chapter 6

Loops (purrr) y bibliografía (rticles)

Chapter 7

Presentaciones en R

Chapter 8

Soluciones a problemas

Todos los problemas en programación tienen más de una forma de llegar a ellos, es por esto que las soluciones acá mostradas deben tomarse solo como una referencia, y revisar si el resultado final de tu código (aunque sea distinto de este), sea igual al que presentamos.

8.1 Capítulo 1

8.1.1 Ejercicio 1

Algunas posibles soluciones:

```
storms %>% filter(status == "hurricane") %>% select(year, wind,
    hu_diameter) %>% group_by(year) %>% summarize_all(mean)
```

```
storms %>% filter(status == "hurricane" & !is.na(hu_diameter)) %>%
    select(year, wind, hu_diameter) %>% group_by(year) %>% summarize_all(mean)
```

```
storms %>% filter(status == "hurricane") %>% select(year, wind,
    hu_diameter) %>% group_by(year) %>% summarize_all(funs(mean),
    na.rm = TRUE)
```

8.1.2 Ejercicio 2

Una de las soluciones posibles:

```
Solution <- mpg %>% filter(year > 2004 & class == "compact") %>%
    mutate(kpl = (cty * 1.609)/3.78541)
```

Bibliography

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., and Chang, W. (2018). *rmarkdown: Dynamic Documents for R*. R package version 1.10.

Anderson, E. (1935). The irises of the gaspe peninsula. *Bulletin of the American Iris society*, 59:2–5.

Henderson, H. V. and Velleman, P. F. (1981). Building multiple regression models interactively. *Biometrics*, pages 391–411.

Hlavac, M. (2018). *stargazer: Well-Formatted Regression and Summary Statistics Tables*. Central European Labour Studies Institute (CELSI), Bratislava, Slovakia. R package version 5.2.2.

Kross, S., Carchedi, N., Bauer, B., and Grdina, G. (2017). *swirl: Learn R, in R*. R package version 2.4.3.

Leek, J. (2015). The elements of data analytic style. *J. Leek.—Amazon Digital Services, Inc.*

Peng, R. D. (2011). Reproducible research in computational science. *Science*, 334(6060):1226–1227.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.

Wickham, H. (2017). *tidyverse: Easily Install and Load the 'Tidyverse'*. R package version 1.2.1.

Wickham, H. et al. (2014). Tidy data. *Journal of Statistical Software*, 59(10):1–23.

Wickham, H., François, R., Henry, L., and Müller, K. (2018). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.6.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.