

Introducción al Análisis de Datos con R

Agustin Alesso

Patricia Acetta

Tabla de contenidos

Bienvenidos!	5
Objetivos del curso	5
Contenidos	5
1 Introducción	7
1.1 La necesidad de analizar datos	7
1.2 ¿Cómo es el flujo de análisis de datos?	8
1.3 ¿Que es un análisis reproducible?	9
2 Comenzando con R	11
2.1 ¿Qué es R y RStudio ?	11
2.2 ¿Cómo instalar R y RStudio ?	15
2.2.1 Instalación de R	15
2.2.2 Instalación de RStudio	16
2.3 Primera sesión	19
2.3.1 La consola	22
2.3.2 El script	22
2.3.3 Directorio de trabajo y proyectos	24
2.3.4 Ayuda!!!	27
3 Aspectos básicos de R	28
3.1 Operadores matemáticos y lógicos	28
3.2 Variables y objetos	30
3.2.1 Vectores	31
3.2.2 Funciones y argumentos	32
3.2.3 Creando funciones	33
3.3 Tipos de datos	35
3.3.1 Numéricos (numeric)	35
3.3.2 Texto (character)	35
3.3.3 Lógicos (logic)	36
3.3.4 Factores (factor y ordered)	36
3.3.5 Otros tipos de datos	37
3.4 Estructura de datos	38
3.4.1 Matriz (matrix)	39
3.4.2 Listas (list)	39

3.4.3	Hoja de datos (<code>data.frame</code>)	40
4	Paquetes de R	43
4.1	¿Qué son los paquetes?	43
5	Importar datos en R	45
5.1	Función nativa para importar datos	45
5.2	Paquetes para importar datos	47
5.2.1	<code>readr</code>	47
5.2.2	<code>rio</code>	48
5.3	Formas de importar datos	49
5.3.1	Desde la consola (recomendado)	49
5.3.2	Desde el importador de datos de RStudio	50
5.3.3	Desde el portapapeles	51
6	Manipulando datos con <code>dplyr</code> y <code>tidyr</code>	52
6.1	¿Cómo instalar <code>dplyr</code> y <code>tidyr</code> ?	53
6.2	Verbos importantes de <code>dplyr</code> para recordar	53
6.3	Seleccionando variables	54
6.4	Seleccionando observaciones	57
6.5	Encadenando operaciones (operador <code>%>%</code>)	59
6.6	Ordenar las filas	61
6.7	Crear o transformar columnas	63
6.7.1	Resumir datos	63
6.7.2	Agrupar (último pero no menos importante)	64
6.8	Tablas pivot	65
7	Visualización de datos con <code>ggplot2</code>	67
7.1	¿Cómo conseguir <code>ggplot2</code> ?	67
7.2	Componentes del gráfico en <code>ggplot2</code>	68
7.2.1	Layers	68
7.2.2	Scales	68
7.2.3	Coordenadas	69
7.2.4	Paneles (<code>facets</code>)	69
7.2.5	Temas	69
7.3	Primer gráfico paso a paso	69
7.4	Gráficos condicionales o por paneles: <code>facets</code>	77
7.5	Temas	82
8	Reportes dinámicos	85
8.1	¿Cómo conseguir <code>ggplot2</code> ?	85
8.2	Componentes del gráfico en <code>ggplot2</code>	86
8.2.1	Layers	86

8.2.2	Scales	86
8.2.3	Coordenadas	87
8.2.4	Paneles (facets)	87
8.2.5	Temas	87
8.3	Primer gráfico paso a paso	87
8.4	Gráficos condicionales o por paneles: facets	95
8.5	Temas	100
References		103

Bienvenidos!

¡Bienvenidos al curso de *Introducción al Análisis de Datos con R*!

Objetivos del curso

- Reconocer al lenguaje R, y su entorno de desarrollo RStudio, como herramienta para el procesamiento y análisis de datos.
- Identificar y explicar los tipos y estructuras de datos representados en R.
- Identificar y emplear funciones y librerías útiles para el procesamiento, visualización y análisis de datos en R.
- Aplicar técnicas estadísticas básicas para analizar datos en R.
- Crear visualizaciones efectivas para comunicar resultados.
- Aplicar los principios de reproducibilidad para procesar y analizar datos y comunicar resultados

Contenidos

- **Unidad 1:** ¿Qué es R y RStudio? Instalación de R y RStudio. Características RStudio: menús, paneles, etc. Sintaxis de R, convenciones y símbolos de R. Sistema de librerías: instalación y carga. Sistema de ayuda. Tipos de datos: numérico, carácter, lógico. Funciones. Estructuras de datos: vectores, listas, hoja de datos (data frame). El flujo de trabajo en un proyecto de análisis de datos: proyectos, scripts y notebooks. Análisis reproducible.
- **Unidad 2:** Manipulación de datos con R. Importación y exportación de datos. Librería tidyverse. Gramática de manipulación de datos. Limpieza, normalización, combinación y resumen de datos numéricos y categóricos.
- **Unidad 3:** Visualizaciones, gráficos. Nociones generales sobre tipos de gráficos y sus usos. Introducción a ggplot2. La gramática de gráficos: estéticas, geometrías, escalas, temas, etc. Gráficos condicionales y multi-paneles.

- **Unidad 4:** Programación literaria. Generación de informes o reportes con Quarto/RMarkdown.

Sin más preámbulos... comencemos!

1 Introducción

1.1 La necesidad de analizar datos

Hoy en día los datos abundan y están en todos lados. Esto desafía nuestra capacidad para analizarlos y extraer significado de los mismos para tomar decisiones.

La ciencia de datos (data science) es una nueva disciplina que emerge de la combinación de disciplinas existentes (diagrama) y permite convertir datos sin procesar en entendimiento, comprensión y conocimiento.

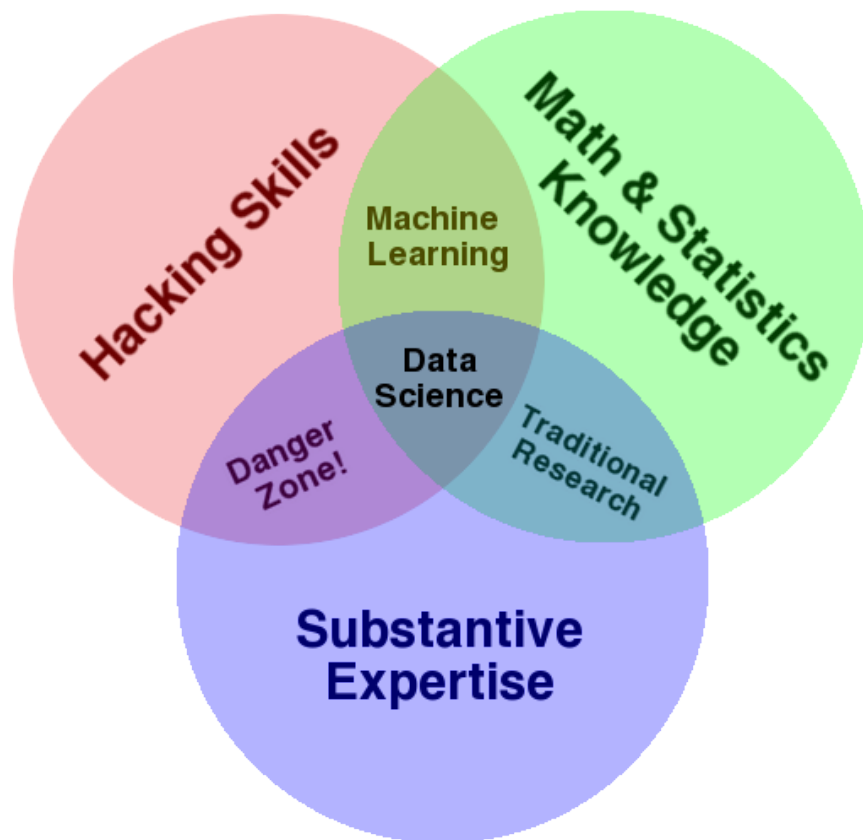


Figura 1.1: Diagrama de Venn de la Ciencia de Datos según Conway

Para poder analizar los datos de manera efectiva, es necesario tener conocimientos de disciplinas como ciencias de la computación (programación y más), matemática y estadística. Pero también tenemos que tener conocimientos para lograr el entendimiento del problema en estudio. La combinación de estas áreas nos lleva al concepto de Ciencia de Datos.

1.2 ¿Cómo es el flujo de análisis de datos?

Si bien el análisis de datos es un proceso no lineal que varía en cada situación, existe consenso en cuanto a las principales actividades que se deben desarrollar. El siguiente gráfico resume el flujo de trabajo o *workflow*.

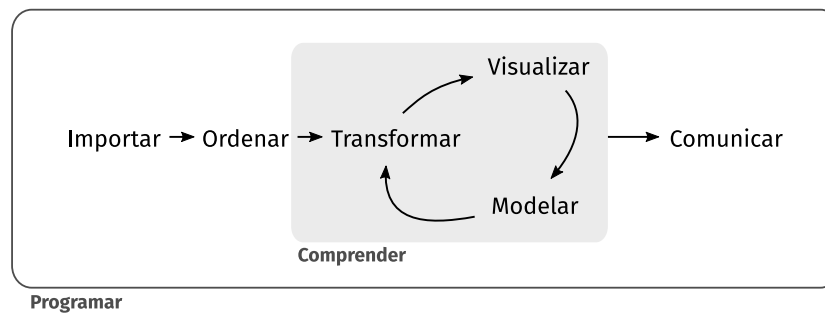


Figura 1.2: Diagrama de Venn de la Ciencia de Datos según Conway

Todo análisis comienza con importar datos a la herramienta que estemos utilizando, **R** en este caso. Los datos pueden estar almacenados de diferentes formas, como archivos, dentro de una base de datos o una API (*application programming interface*). En el curso veremos distintas formas de importar datos dentro de **R** en un formato compatible para su análisis.

Salvo excepciones, una vez importados los datos, hay que ordenarlos de alguna forma que permita realizar el análisis que queremos. En la mayoría de los casos necesitaremos que los datos estén almacenados de manera **rectangular** tal que cada columna represente una variable o atributo de los datos y cada fila una o más observaciones o sujetos (formato *apaisado*).

pais	año	casos	poblacion
Afganistán	1999	745	19987071
Afganistán	2000	2666	20595360
Brasil	1999	37737	172006362
Brasil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

variables

pais	año	casos	poblacion
Afganistán	1999	745	19987071
Afganistán	2000	2666	20595360
Brasil	1999	37737	172006362
Brasil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

observaciones

pais	año	casos	poblacion
Afganistán	1999	745	19987071
Afganistán	2000	2666	20595360
Brasil	1999	37737	172006362
Brasil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

valores

A veces cuando algunas columnas representan valores de una misma variable hay que modificar este formato y *pivotar* a un formato *longitudinal*.

Con los datos ordenados podemos concentrarnos en comprender su estructura y empezar a jugar con ellos. Esta etapa no es para nada lineal y generalmente implica un proceso iterativo donde los datos se visualizan, se transforman y se modelan.

La transformación de los datos implica crear subconjuntos o combinar con otros datos, alterar las variables existentes (e.g. cambiar unidades) o generar nuevos atributos combinando información de otros, resumir los datos agrupando observaciones, etc. El manejo de datos de datos (*data wrangling*) es la combinación de técnicas de ordenamiento y transformación.

La visualización es el arte de convertir los datos de forma tabular en gráficos o diagramas donde los distintos atributos de los datos se relacionan a características gráficas tales como ejes, colores, formas, etc. Una buena visualización permitirá revelar patrones inesperados, confirmar algunas preguntas o sugerir nuevas. Es muy útil para comunicar.

El modelado es una forma de cuantificar lo podemos ver en una visualización. Al igual que las visualizaciones, los modelos son abstracciones de los datos y nos permiten resumir la variación quedándonos con la generalidad. Los modelos son útiles para contestar preguntas sobre los datos. Lo clave es hacer la pregunta correcta! Por otro lado los modelos son tan buenos como los datos de entrada y los supuestos utilizados.

El último paso de un proyecto de ciencia de datos es la comunicación. Esta etapa resume todo nuestro trabajo y determina como podemos transmitir nuestras conclusiones y descubrimientos a otros que no participaron del análisis.

Finalmente, todas y cada una de las etapas de este proceso se desarrollan con la ayuda de la programación. No es necesario ser un hacker, hay que saber pensar como un programador y saber programar ayuda a automatizar tareas.

1.3 ¿Que es un análisis reproducible?

En las Ciencias Experimentales, poder replicar los experimentos es un componente muy importante del método científico ya que le da validez a los descubrimientos. Es por ello que en los trabajos científicos, una sección importante es la de *Materiales y Métodos* donde se describen los pasos que se deben dar para poder *replicar* de los resultados del estudio. Estos pasos incluyen las instrucciones para replicar el experimento físico y de como *reproducir* el análisis de los datos que llevó a las conclusiones.

Así la **replicabilidad** implica que, siguiendo los *materiales y métodos* un experimento independiente puede llegar a los mismos resultados con datos distintos. En cambio la **reproducibilidad** significa que con la misma persona u otra persona con los mismos datos llegue a los mismos resultados, es decir, que pueda reproducir el análisis.

La Ciencia de Datos la **reproducibilidad computacional** es clave ya que en el procesamiento de datos se toman una serie de decisiones que afectan el resultado. ¿Qué significa esto? Dada una serie de datos de entrada, el flujo de trabajo que se aplica para *importar*, *ordenar*, *transformar*, *visualizar* y *modelar* los datos, es decir, convertirlos en información, debe estar correctamente **documentado** para que cualquier persona pueda entender la lógica y eventualmente reproducir los mismos resultados.

R, como cualquier lenguaje de programación, es una herramienta ideal para aplicar el concepto de **reproducibilidad** ya que mediante en el código o *script* quedan plasmados todos los pasos que se aplicaron en el análisis de datos. Más aun, el uso del paradigma de **programación literaria** permite generar documentos donde se narran todos los pasos y los resultados obtenidos.

2 Comenzando con R

En esta sección vamos a ver que es cómo instalar y cómo empezar a usar R y RStudio creando nuestro proyecto de análisis reproducible de datos.

2.1 ¿Qué es R y RStudio?

R es un lenguaje y entorno para el procesamiento, visualización y análisis estadístico de datos. Fue creado en 1993 por R. Gentleman y R. Ihaka, ambos científicos del Departamento de Estadística de la Universidad de Auckland (Nueva Zelanda). Actualmente su desarrollo y mantenimiento está a cargo del R Core Team (2023). El sitio oficial del proyecto es www.r-project.org.

Hoy en día, **R** es la *lingua franca* del procesamiento y análisis de datos, tanto en el ámbito académico como comercial dado que es gratuito, multiplataforma, de código abierto (*open source*, liberado con licencia GNU/GPL). Esto y el ecosistema de paquetes contribuidos por la comunidad de usuarios lo convierte en un software muy potente ya que expresa el estado del arte de los métodos estadísticos.

La flexibilidad y potencia de **R** se basa en su interfaz de comandos (CLI, del inglés *command line interface*) que permite la ejecución de comandos de manera interactiva (en consola) o estructurada mediante scripts.

Existen algunos desarrollos de interfaces gráficas (GUIs, del inglés *graphical user interface*), e.g. RCommander, Deducer, etc., que ofrecen la posibilidad de, mediante menús y botones dedicados, ejecutar algunos análisis relativamente simples minimizando la necesidad de escribir código.

Los entornos de desarrollo integrados (IDE por sus siglas en inglés *integrated development environments*) ofrecen un enfoque intermedio: los menús o funciones asistentes facilitan algunas tareas generales (abrir archivos, carga de datos, exportar gráficos y resultados, etc.) pero dejan la escritura del código y ejecución del análisis estadístico en manos del usuario. Entre estas alternativas se destaca **RStudio** desarrollado por la empresa [posit](https://posit.co/) el cual también es de código abierto (licencia GNU/GPL), multiplataforma y ofrece una versión gratuita.



[\[Home\]](#)

Download

[CRAN](#)

R Project

[About R](#)

[Logo](#)

[Contributors](#)

[What's New?](#)

[Reporting Bugs](#)

[Conferences](#)

[Search](#)

[Get Involved: Mailing](#)

[Lists](#)

[Developer Pages](#)

[R Blog](#)

R Foundation

[Foundation](#)

[Board](#)

[Members](#)

[Donors](#)

[Donate](#)

Help With R

[Getting Help](#)

Documentation

[Manuals](#)

[FAQs](#)

[The R Journal](#)

[Books](#)

[Certification](#)

[Other](#)

Links

[Bioconductor](#)

[R-Forge](#)

[R-Hub](#)

[GSoC](#)

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News

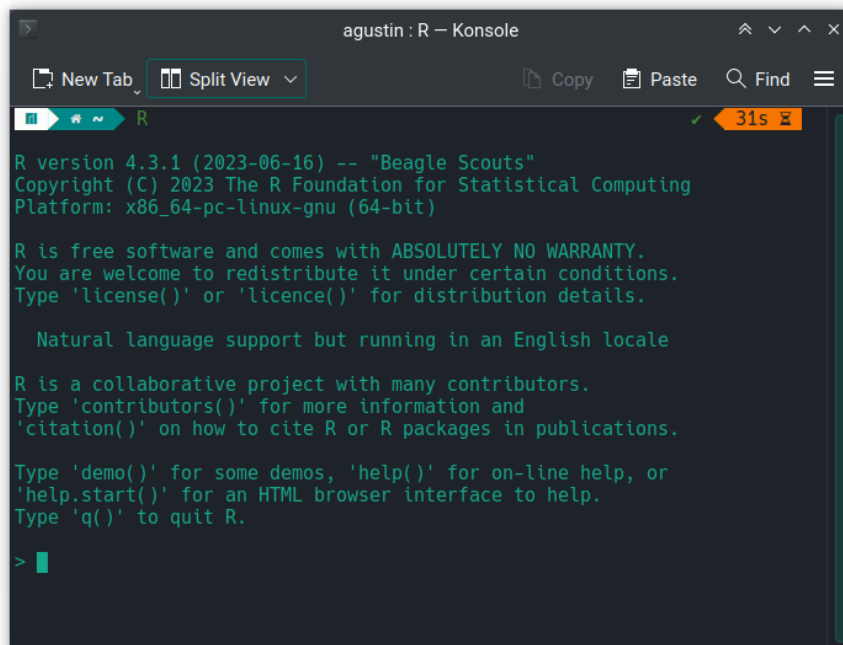
- [R version 4.1.1 \(Kick Things\)](#) has been released on 2021-08-10.
- [R version 4.0.5 \(Shake and Throw\)](#) was released on 2021-03-31.
- Thanks to the organisers of useR! 2020 for a successful online conference. Recorded tutorials and talks from the conference are available on the [R Consortium YouTube channel](#).
- You can support the R Foundation with a renewable subscription as a [supporting member](#)

News via Twitter

[News from the R Foundation](#)

© The R Foundation. For queries about this web site, please contact [the webmaster](#); for queries about R itself, please consult the [Getting Help](#) section.

Figura 2.1: Página oficial de R Project



```
agustin : R - Konsole
New Tab Split View Copy Paste Find
R
R version 4.3.1 (2023-06-16) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Figura 2.2: Ejemplo de consola o terminal de Linux y Windows corriendo la última versión estable de **R**

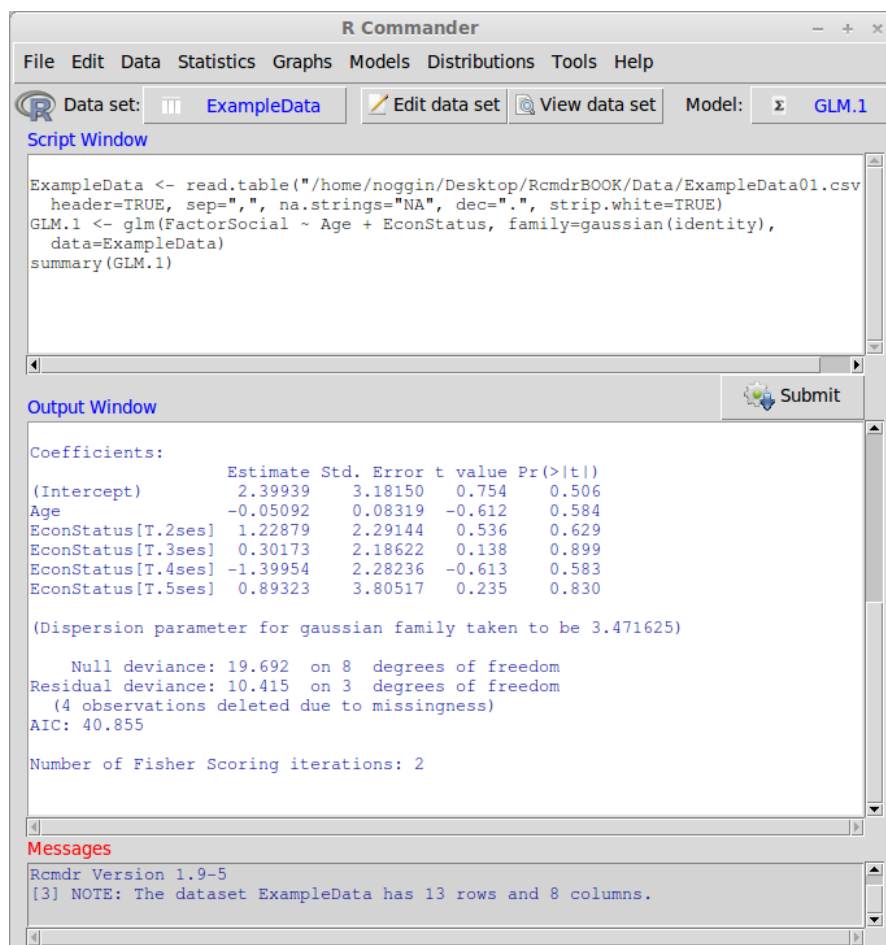


Figura 2.3: Interfase de R Commander

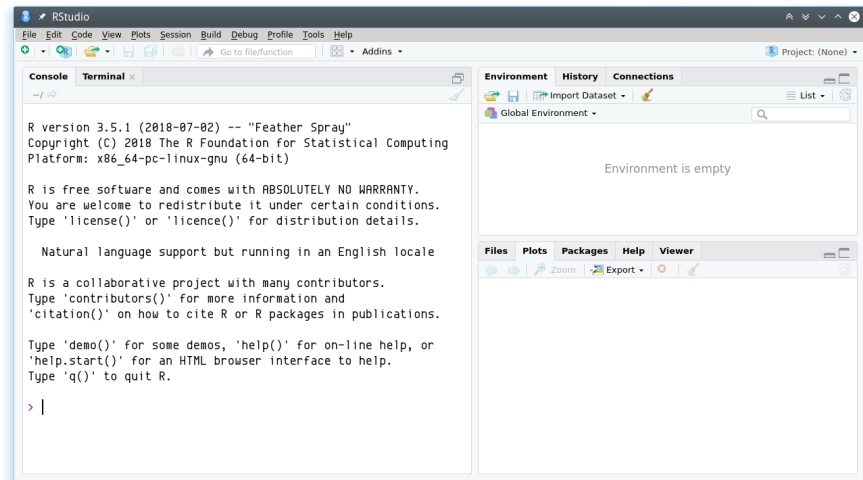


Figura 2.4: Interfase de **RStudio**

2.2 ¿Cómo instalar R y RStudio?

R y **RStudio** se instalan por separado. Ambos softwares son multiplataforma y pueden ser ejecutados en sistemas operativos Windows, Mac OS X y Linux.

R puede funcionar sin **RStudio**, en cambio **RStudio** necesita que al menos una versión de **R** esté instalada en el sistema.

La página de descarga de **posit** <https://posit.co/download/rstudio-desktop/> ofrece un excelente punto de partida para instalar ambos programas.

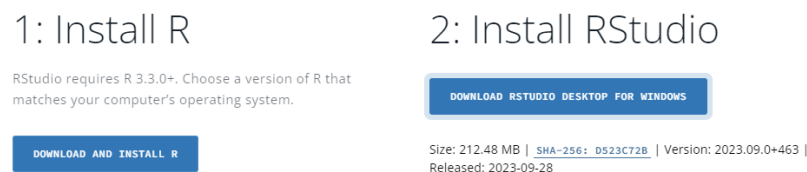


Figura 2.5: Página de descarga de **R**

A continuación se describe el procedimiento para instalar **R** y **RStudio** bajo Windows.

2.2.1 Instalación de **R**

- 1) Click en el botón **DOWNLOAD AND INSTALL R**:

1: Install R

RStudio requires R 3.3.0+. Choose a version of R that matches your computer's operating system.

DOWNLOAD AND INSTALL R

Figura 2.6: Página de descarga de **R**

Descargar el archivo instalador correspondiente a la última versión estable de **R** desde el CRAN¹ (del inglés, *Comprehensive R Archive Network*) visitando el siguiente [link](#).

- 2) Ejecutar el archivo descargado ² y seguir el asistente de instalación con todas las opciones por defecto.
- 3) Si la instalación ha sido exitosa en el menú *Inicio* podrá encontrarse la carpeta *R* que contendrá dos accesos directos a la interfase de usuario mínima que viene con la versión de **R** para Windows.

2.2.2 Instalación de RStudio

- 1) Click en el botón DOWNLOAD RSTUDIO DESKTOP FOR:

Descargar el archivo de instalación correspondiente a nuestra plataforma o sistema operativo. Al momento de escribir estas instrucciones la última versión estable de **RStudio** era RStudio-2023.09.0-463.exe que se encuentra en este [link](#)

En el caso que haya una nueva versión, ir al sitio web de descarga de **RStudio** <https://posit.co/download/rstudio-desktop/>

- 2) Ejecutar el archivo .exe y seguir el asistente de instalación con todas las opciones por defecto.

¹Si bien <- funciona igual que = en la mayoría de los casos, por convención se usa <- para asignar y = para argumentos dentro de las funciones.

²Al momento de escribir estas instrucciones la última versión estable de **R** era la 4.3.1 *Beagle Scouts*, por lo tanto el link apuntará al archivo R-4.3.1-win.exe.



CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

Documentation
[Manuals](#)
[FAQs](#)
[Contributed](#)

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux \(Debian, Fedora/Redhat, Ubuntu\)](#)
- [Download R for macOS](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2021-08-10, Kick Things) [R-4.1.1.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

What are R and CRAN?

R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the [R project homepage](#) for further information.

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN [mirror](#) nearest to you to minimize network load.

Submitting to CRAN

To "submit" a package to CRAN, check that your submission meets the [CRAN Repository Policy](#) and then use the [web form](#).

If this fails, send an email to CRAN-submissions@R-project.org following the policy. Please do not attach submissions to emails, because this will clutter up the mailboxes of half a dozen people.

Note that we generally do not accept submissions of precompiled binaries due to security reasons. All binary distribution listed above are compiled by selected maintainers, who are in charge for all binaries of their platform, respectively.

For queries about this web site, please contact [the webmaster](#).

Figura 2.7: Página de descaga de R



CRAN
[Mirrors](#)
[What's new?](#)
[Search](#)
[CRAN Team](#)

About R
[R Homepage](#)
[The R Journal](#)

Software

R for Windows

Subdirectories:

[base](#) Binaries for base distribution. This is what you want to [install R for the first time](#).
[contrib](#) Binaries of contributed CRAN packages (for R >= 3.4.x).
[old.contrib](#) Binaries of contributed CRAN packages for outdated versions of R (for R < 3.4.x).
[Rtools](#) Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the [R FAQ](#) and [R for Windows FAQ](#).

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Figura 2.8: Dirigirse a Install R for first time

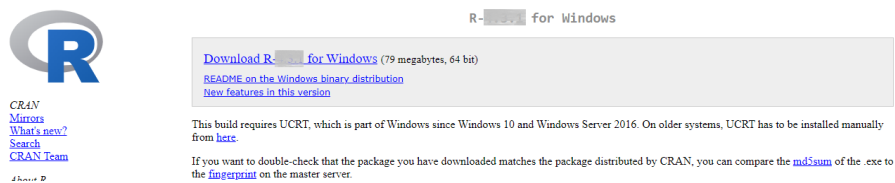


Figura 2.9: Dirigire a “Download R-X.X.X for Windows”

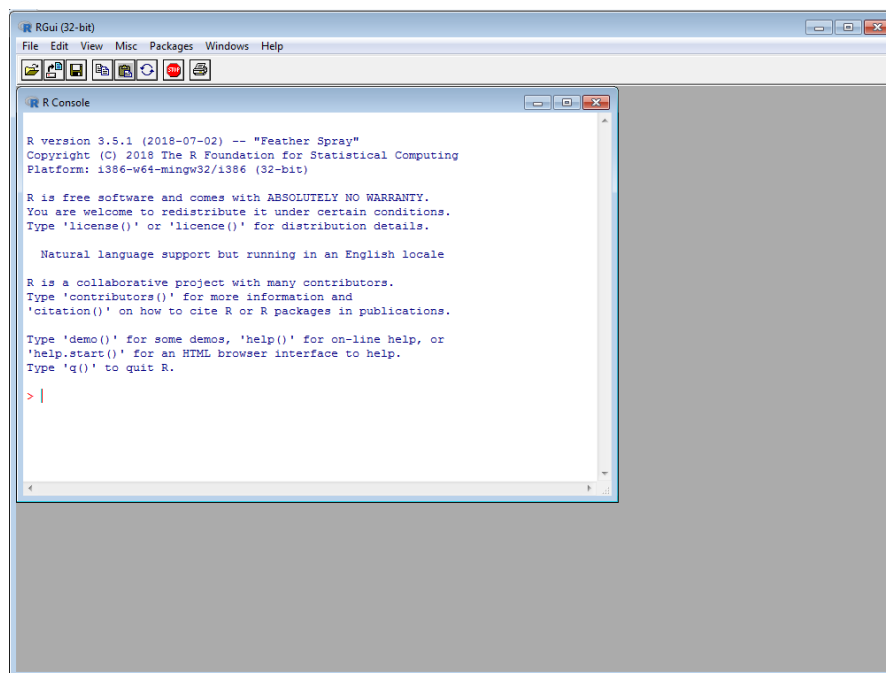


Figura 2.10: R GUI para Windows

2: Install RStudio

DOWNLOAD RSTUDIO DESKTOP FOR WINDOWS

Size: 212.48 MB | [SHA-256: D523C72B](#) | Version:
2023.09.0+463 | Released: 2023-09-28

Figura 2.11: Página de descarga de **R**

- 3) Si la instalación ha sido exitosa en el menú *Inicio* dentro de la carpeta *RStudio* se encontrará el acceso directo a **RStudio** el cual, mediante el menu contextual (botón derecho del ratón) puede enviarse al Escritorio como acceso directo o bien anclar al menu de Inicio o barra de acceso rápido.

Ahora sí, ya tenemos listo **R** y **RStudio** para empezar a trabajar!!

ACTIVIDAD

1. Instalar **R** y **RStudio**

2.3 Primera sesión

El entorno de trabajo de **RStudio** se divide en cuatro paneles. La disposición y contenido de los paneles se puede personalizar yendo a **View > Panes > Panes Layout...** A continuación la descripción de los paneles principales:

1. **Editor**. Es donde se editan los *scripts* que son archivos de texto plano con los comandos para ejecutar en **R**. Este panel no aparece a menos que se cree un nuevo script o se abra uno previamente guardado.
2. **Console** (consola). Es donde vive **R** propiamente dicho. Allí se ejecutan los comandos y se obtienen las salidas de **R**.
3. **Environmnet/History/Connections**. En la primera pestaña se visualizan los objetos (variables, funciones o datos cargados) que están disponibles en el entorno de **R**, i.e. en

RStudio Desktop 1.4.1717 - [Release Notes](#)

1. Install R. RStudio requires [R 3.0.1+](#).
2. Download RStudio Desktop. Find your operating system in the table below.



All Installers

Linux users may need to [import RStudio's public code-signing key](#) prior to installation, depending on the operating system's security policy.

RStudio requires a 64-bit operating system. If you are on a 32 bit system, you can use an [older version of RStudio](#).

OS	Download	Size	SHA-256
Windows 10	RStudio-1.4.1717.exe	156.18 MB	71b36e64
macOS 10.14+	RStudio-1.4.1717.dmg	203.06 MB	2cf2549d
Ubuntu 18/Debian 10	rstudio-1.4.1717-amd64.deb	122.51 MB	e27b2645
Fedora 19/Red Hat 7	rstudio-1.4.1717-x86_64.rpm	138.42 MB	648e2be0
Fedora 28/Red Hat 8	rstudio-1.4.1717-x86_64.rpm	138.39 MB	c76f620a
Debian 9	rstudio-1.4.1717-amd64.deb	123.29 MB	e4ea3a60
OpenSUSE 15	rstudio-1.4.1717-x86_64.rpm	123.15 MB	e69d55db

Zip/Tarballs

OS	Zip/tar	Size	SHA-256
Windows 10	RStudio-1.4.1717.zip	227.77 MB	84b1dc1a
Ubuntu 18/Debian 10	rstudio-1.4.1717-amd64-debian.tar.gz	177.14 MB	ba24900c
Fedora 19/Red Hat 7	rstudio-1.4.1717-x86_64-fedora.tar.gz	177.24 MB	4c05ddca
Debian 9	rstudio-1.4.1717-amd64-debian.tar.gz	177.48 MB	cd6d8462

Source Code

A tarball containing source code for RStudio 1.4.1717 can be downloaded from [here](#).

Figura 2.12: Página de descarga de **RStudio**



Figura 2.13: Icono de **RStudio**

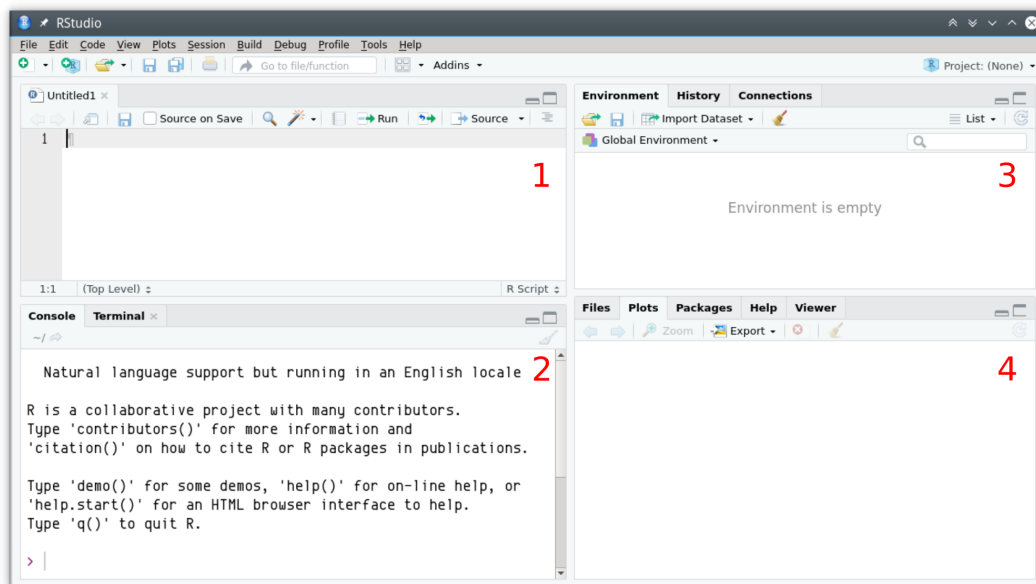


Figura 2.14: Interfase principal de **RStudio**

la memoria. En la segunda se puede ver el historial de comandos ingresados o enviados a la consola. La tercera pestaña visualiza las conexiones establecidas con diferentes base de datos.

4. **Files/Plots/Packages/Help/Viewer.** Allí se puede manejar los archivos del directorio de trabajo, visualizar los gráficos generados en **R** con posibilidad de exportarlos en varios formatos, administrar los paquetes o complementos, buscar o explorar el manual de ayuda y previsualizar archivos HTML.

2.3.1 La consola

La línea de comandos o **consola** es el modo interactivo mediante el cual podemos ejecutar comandos directamente en el intérprete de **R**. El símbolo o *prompt* `>` indica que **R** está disponible esperando una orden. Si la orden no está completa el símbolo se transforma en `+`. Por ejemplo: si tipeamos `2 + 2` y luego **ENTER**:

```
2 + 2
```

```
[1] 4
```

Obtenemos inmediatamente el resultado. Otro ejemplo: el promedio de los números 1, 3 y 4

```
(1 + 3 + 4) / 3
```

```
[1] 2.666667
```

ACTIVIDAD

1. Escribir una operacion matemática, por ejemplo: `3*4`
2. Escribir algo en la consola. ¿Que sucede?
3. Escribir lo anterior entre comillas " ".

2.3.2 El script

El **editor de scripts** (panel #1) es un editor de texto plano que está conectado con la consola (panel #2). Tiene algunas funcionalidades que facilitan la edición del código:

- Resaltado sintaxis: mediante colores resalta las funciones, variables, comandos o palabras claves del lenguaje **R**
- Sangrado automático: agrega espacios en blanco para mantener la sangría de los bloques de código.
- Plegado de código: permite colapsar bloques de código
- Completado automático y ayuda en línea: muestra sugerencias para completar el comando o argumentos usando la tecla **TAB**.

Para crear un nuevo script se puede usar uno de los siguientes métodos:

- Ir a al menu **File > New File > R Script**
- Usar el atajo de teclado **CTRL + SHIFT + N**
- Clickear en el primer ícono de la barra de menu



Figura 2.15: Barra de herramientas de **RStudio**

Una vez abierto el script en blanco, se pueden empezar a escribir los comandos de **R**. Por ejemplo podemos escribir lo siguiente:

```
"Hola Mundo!" # Clásico mensaje "Hola mundo!"

# Calcular el promedio de estos números
(1 + 3 + 4) / 3
```

Estos comandos no se van a ejecutar automáticamente ya que sólo los hemos escrito en el *script*. Para ejecutar los comandos en la consola hay que posicionar el cursor en la línea deseada o bien seleccionar si queremos ejecutar varias a la vez y luego enviarlo a la consola con una de las siguientes opciones:

- Ir al menu **Code > Run Selected Line(s)**
- Usar el atajo de teclado **CTRL + ENTER** o **CTRL + R**
- Usar el ícono **Run** de la barra de herramientas de la pestaña del script



Figura 2.16: Barra de herramientas del panel Editor

El simbolo **#** indica que lo que sigue es un **comentario** y por lo tanto **R** lo ignora cuando es enviado a la consola. Los comentarios pueden ir solos en una línea separada o bien dentro de una línea que tenga algún comando. Si bien no son necesarios para correr el código, los

comentarios son muy útiles para estructurar el script y hacer anotaciones para que otros, o nosotros en un futuro, entiendan lo que hace esa parte del script.

Para guardar el script:

1. Ir al menu **File > Save** o usar el atajo de teclado **CTRL + S** o bien el ícono con el diskette de la barra de herramientas global o de la pestaña del script activo.
2. Elegir la carpeta destino y el nombre de archivo. Automáticamente se agregará la extensión **.R** que corresponde a los scripts.

ACTIVIDAD

1. Abrir un script nuevo
2. Escribir un comentario
3. Escribir un texto o un comando numérico
4. Guardar el script con el nombre `mi_primer_script.R`.

2.3.3 Directorio de trabajo y proyectos

R trabaja con un directorio de trabajo o *working directory* que es la dirección o *path* que figura en el título del panel **Console**. Esto se puede averiguar con `getwd()`

```
getwd()
```

Por defecto es el directorio base del usuario que depende de cada plataforma. En linux es el `/home/usuario` en cambio en Windows es `C:/Users/usuario/Documents`.

A menos que se especifique lo contrario, se asume que los archivos de entrada o salida se ubican en esa. Esto se puede modificar en cualquier momento con la función `setwd()`.

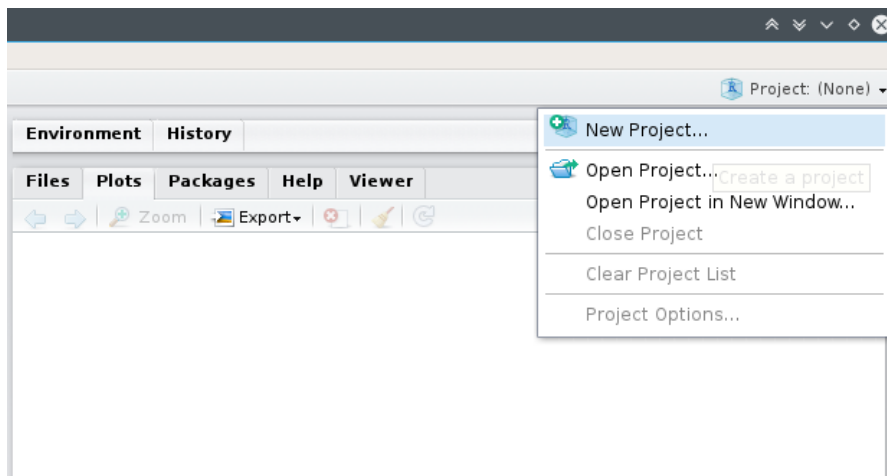
```
setwd("ruta/a/otra/carpeta")
```

RStudio extiende esta característica a través de los proyectos o *projects*. Cada proyecto es una carpeta o *folder* que contienen un archivo `.RProj` con algunas configuraciones específicas.

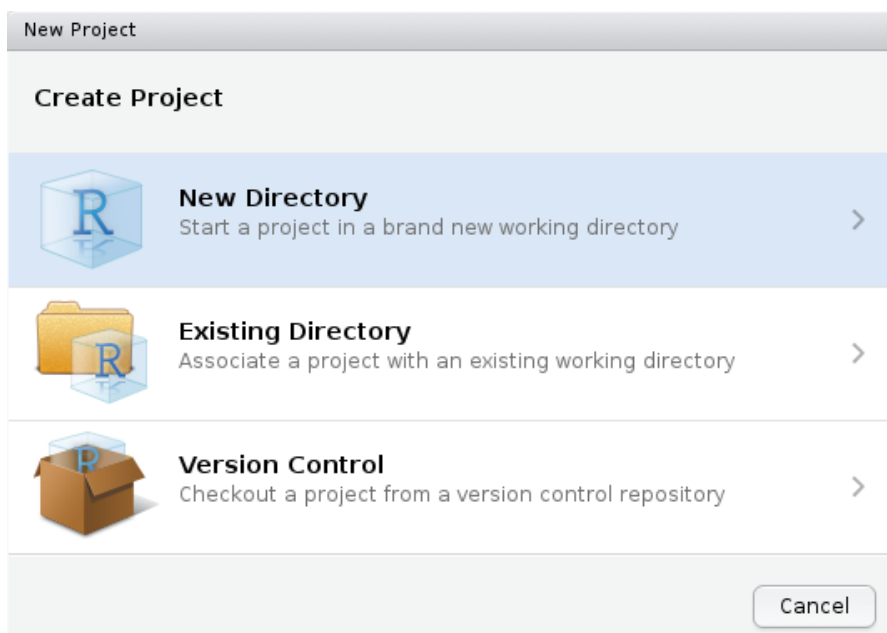
Al abrir un proyecto, automáticamente se cambia el directorio de trabajo a esta carpeta. Esto permite organizar los archivos de datos, las salidas, los scripts, etc., dentro de un directorio de trabajo (*working directory*) y volver a ellos de manera más rápida, eficiente, y portable.

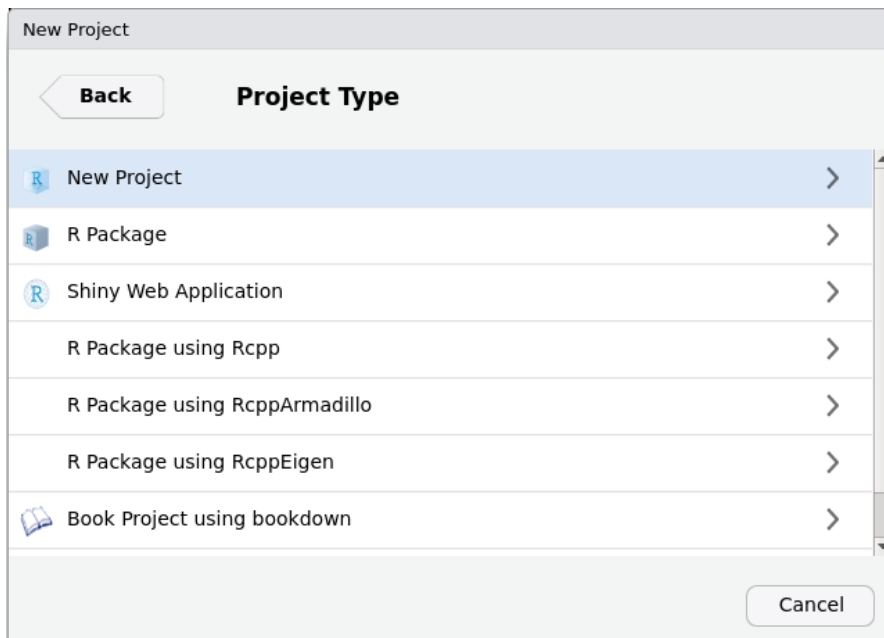
Para crear un proyecto en **RStudio**:

1. Ir a File > New project... o bien el ícono Create project de la barra de herramientas.



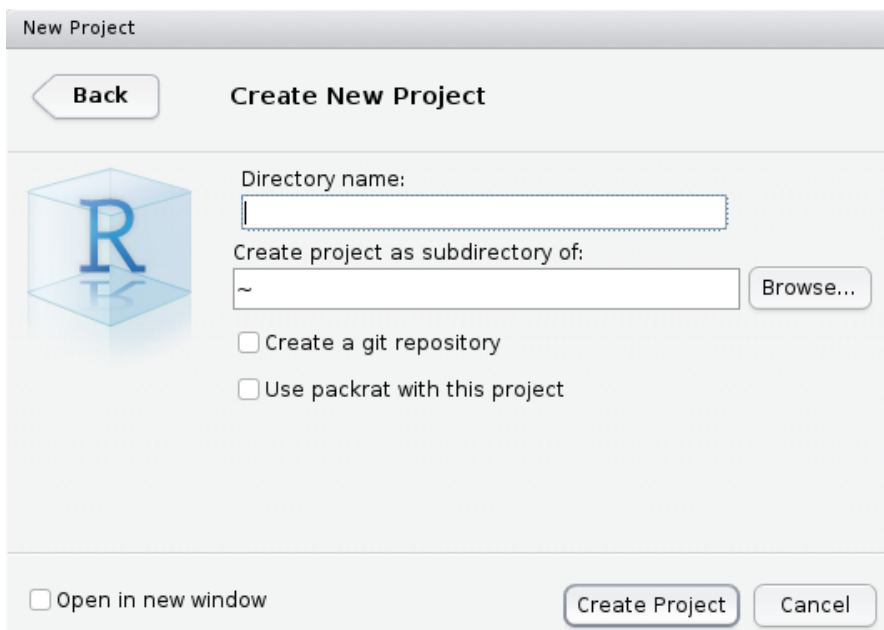
2. Seleccionar New Directory y en Project type seleccionar New project.





- Una vez en el cuadro de diálogo **Create new project** ingresar el nombre del proyecto (e.g. `mi_proyecto`) en **Directory name** que será a su vez el nombre de la carpeta que **RStudio** va a crear por nosotros.

Luego en **Create project as a subdirectory of** indicar *donde* queremos que **RStudio** cree la carpeta.



4. Si todo sale bien, se crea la carpeta con el nombre que indicamos y dentro de ésta un archivo con extensión `.Rproj`. Este archivo solamente se usa para abrir el directorio. No se debe sobrescribir con el script.

ACTIVIDAD

1. Crear un proyecto nuevo con el nombre “Intro_R” en el escritorio o lugar de preferencia.
2. Cerrar **Rstudio**.
3. Abrir **Rstudio** desde el proyecto.
4. Copiar el script creado en la actividad anterior dentro del nuevo proyecto.
5. Abrir el script.

2.3.4 Ayuda!!!

Por último, y no menos importante, **R** y **RStudio** cuentan con un completo sistema de ayuda.

Desde la consola se puede acceder usando la función `?` seguida del nombre de la función o bien `help("nombre")`

```
# Pedir ayuda de la función mean
?mean
help(mean)
```

Una de las ventajas de **RStudio** es que dispone de un panel (Panel #4) dedicado a visualizar las páginas de ayuda. Allí se puede navegar por las páginas utilizando los links, realizar búsquedas, etc. Leer la documentación nunca viene mal y generalmente ahorra dolores de cabeza.

3 Aspectos básicos de R

En esta sección vamos a aprender nociones básicas de la sintaxis de R, tipos de datos y estructuras.

3.1 Operadores matemáticos y lógicos

Como vimos antes, las operaciones matemáticas básicas se realizan usando los símbolos convencionales:

- suma (+)
- resta (-)
- división (/)
- producto (*)
- potencia (^)

Por ejemplo, $1 + \left(3 \times 4 + \frac{5-2}{3}\right)^2$ en **R** es:

```
1 + (3 * 4 + (5 - 2)/3)^2
```

```
[1] 170
```

También se pueden evaluar expresiones lógicas:

- *igual que* (==)
- *distinto que* (!=)
- *mayor que* (>)
- *menor que* (<)
- *mayor o igual que* (>=)
- *menor o igual que* (<=)

El resultado es **TRUE** (verdadero) o **FALSE** (falso). Por ejemplo, podemos evaluar si 3 es igual a 4

```
3 == 4
```

```
[1] FALSE
```

O si 5 es mayor o igual a 3

```
5 >= 3
```

```
[1] TRUE
```

También se pueden combinar con los operadores *intersección* (&), *unión* (|) y *negación* (!).

Por ejemplo, evaluar si se cumplen las dos cosas anteriores a la vez

```
3 == 4 & 5 >= 3
```

```
[1] FALSE
```

Devuelve **FALSE** porque `3 == 4` no es verdadero. Si reemplazamos & por | va a devolver evaluar si una de las dos se cumple:

```
3 == 4 | 5 >= 3
```

```
[1] TRUE
```

También se pueden combinar con operaciones matemáticas...

```
4 * 2 == 8
```

```
[1] TRUE
```

En este caso primero evalúa `4 * 2` y luego compara el resultado con 8

ACTIVIDAD

1. Hallar el resultado de la siguiente expresión $(2 + 4)^2$ y $2^2 + 4^2$.
2. Comparar ambos resultados en una línea de comando.

3.2 Variables y objetos

Un objeto es un espacio de la memoria que almacena un pedazo de información (una cifra, un conjunto de números, el resultado de un análisis, etc). También se denomina *variables* ya que su contenido puede cambiar. En **R** prácticamente todo puede representarse como un *objeto*.

Los objetos o variables se crean *asignándoles* información (números, letras, resultados de operaciones, etc), con el símbolo `<-` (ALT + -) o `=`¹. Esta información se puede recuperar, modificar o utilizar para otros cálculos.

Supongamos que queremos asignar el valor 2 a la variable `x`.

```
x <- 2
```

En la consola vuelve a aparecer el símbolo `>` y nada más. En el ambiente se ve una entrada que dice `x` y el valor. Podemos recuperar el valor en la consola tipeando el nombre del objeto:

```
x
```

```
[1] 2
```

También podemos reusarlo en otro cálculo, por ejemplo obtener 2 veces `x`.

```
2 * x
```

```
[1] 4
```

O bien obtener una nueva variable

```
y <- 2 * x + 1  
y
```

```
[1] 5
```

Nombres

Los nombres de las variables no deben empezar con números ni contener espacios. No pueden usarse operadores (`*+-/%%`) en los nombres pero puede usarse `.` o `_`.

¹Si bien `<-` funciona igual que `=` en la mayoría de los casos, por convención se usa `<-` para asignar y `=` para argumentos dentro de las funciones.

```
# Mal
2x <- 3
mi variable <- 3

# Bien
x_2 <- 3
x.2 <- 3
x2 <- 3
```

Mayúsculas

También **R** es sensible a mayúsculas

```
# Definir 'A' y 'a'
A <- 3
a <- 5

# Verificar si 'A' y 'a' son lo mismo
A == a
```

```
[1] FALSE
```

3.2.1 Vectores

Son los objetos más simples a partir de los cuales se construyen otros tipos de objetos. Se crean utilizando la función `c()` (*combine*) para “combinar” **datos del mismo tipo**

```
x <- c(13, 45, 67, 45)
x
```

```
[1] 13 45 67 45
```

En el caso de mezclar de datos, **R** los va a *convertir* al tipo de datos más simple.

Por ejemplo: si queremos crear un vector con 3 valores: lógico, numérico y texto, **R** va a asumir que todos los elementos son de tipo *texto*

```
y <- c(TRUE, 34, "hola")
y
```

```
[1] "TRUE" "34"  "hola"
```

Los vectores están indexados. Se puede acceder a sus elementos usando el operador `[]` e indicando el número de orden.

Por ejemplo: para recuperar el 3er elemento del vector `x`

```
y[3]
```

```
[1] "hola"
```

Veremos más adelante los distintos tipos.

3.2.2 Funciones y argumentos

Para crear los vectores sin pensarlo utilizamos una función `c()`. Las funciones son series de comandos que hacen alguna acción y producen un resultado.

Generalmente tienen la siguiente la siguiente forma:

```
nombre_funcion(arg1, arg2, ...)
```

donde `arg` son los argumentos (valores de entrada u opciones). Algunos argumentos toman valores por defecto otros hay que declararlos.

Por ejemplo, la función `round()` tiene los argumentos:

- `x`, para pasar el número o vector numérico que queremos redondear
- `digits = 0` para indicar el número de dígitos a usar, por defecto 0.

Supongamos que queremos redondear el número 3.141593 a 3 dígitos. Para eso usamos la función `round()` (buscar en la ayuda `?round`)

```
# Indicando los argumentos
round(x = 3.141593, digits = 3)
```

```
[1] 3.142
```

```
# Sin indicar los argumentos
round(3.141593, 3)
```

```
[1] 3.142
```


En este último caso, el orden de los argumentos es clave ya que **R** asigna los valores en función de la posición.

```
# Sin indicar los argumentos  
round(3, 3.141593)
```

```
[1] 3
```

Devuelve 3 por considera que queremos redondear el número 3

3.2.3 Creando funciones

Así como **R** tiene viene con funciones ya definidas, nosotros podemos crear nuestras propias funciones para poder simplificar nuestro análisis encapsulando tareas repetitivas.

Por ejemplo, supongamos que queremos calcular el área de 3 rectángulos cuyas dimensiones son:

- Rectángulo 1: base = 10, altura = 20
- Rectángulo 1: base = 15, altura = 35
- Rectángulo 1: base = 20, altura = 5

Sabemos que la fórmula del área es: $A = b \times a$, donde A es el area, b es la base y a es la altura.

Podríamos hacer:

```
10 * 20
```

```
[1] 200
```

```
15 * 35
```

```
[1] 525
```

```
20 * 5
```

```
[1] 100
```

Como vimos antes podemos usar vectores y operar con ellos:

```
bases <- c(10, 15, 20)
alturas <- c(20, 35, 5)
bases * alturas
```

```
[1] 200 525 100
```

Yendo un poco mas lejos, en vez de repetir siempre la operación podríamos encapsularla en una *función*. Las funciones se crean con `function()` o `\()` y adentro se declaran los argumentos. Por defecto la función devuelve el ultimo comando o bien lo que se indique en `return()`

```
area_rectangulo <- function(base, altura) {

  # Calculo superficie
  area <- base * altura

  return(area)

}
```

R no devuelve nada porque lo que hizo fue crear la función, a continuación podemos usarla:

```
area_rectangulo(40,50)
```

```
[1] 2000
```

```
area_rectangulo(bases, alturas)
```

```
[1] 200 525 100
```

ACTIVIDAD

1. Crear una función que calcule el area de un círculo. Tip: la constante π en **R** está en el objeto `pi`.
2. Avanzado: crear una función más avanzada llamada *area* que calcule el área de un rectángulo o círculo indicando en el argumento `figura` y suministrando el dato de base y altura o altura, según corresponda.
3. ¿Cómo podríamos expandirla para controlar el número de decimales de la respuesta?

4. ¿Por qué este código no funciona?

```
my_variable <- 10
my_variable
```

3.3 Tipos de datos

3.3.1 Numéricos (numeric)

Números racionales (enteros o con coma).

```
x <- c(3, 4, 5)
class(x)
```

```
[1] "numeric"
```

Los números enteros se tratan como **numeric** a menos que se los convierta con `as.integer()`.

```
y <- as.integer(x)
class(y)
```

```
[1] "integer"
```

Los datos numéricos permiten todas las operaciones algebraicas

```
mean(x)
```

```
[1] 4
```

```
mean(y)
```

```
[1] 4
```

3.3.2 Texto (character)

Cadenas de texto o número delimitadas por comillas (simples o dobles, nom mezclar).

```
x <- c("hola", '3')
class(x)
```

```
[1] "character"
```

Lógicamente no se pueden realizar operaciones numéricas. **R** avisa y devuelve NA

```
mean(x)
```

Warning in mean.default(x): argument is not numeric or logical: returning NA

```
[1] NA
```

3.3.3 Lógicos (logic)

Condición verdadero (TRUE o T) o falso (FALSE o F)

```
logico <- c(T, F, T, TRUE, FALSE, F)
logico
```

```
[1] TRUE FALSE TRUE TRUE FALSE FALSE
```

Otro ejemplo: ¿cuáles de los siguientes números son mayores a 30?

```
x <- c(23, 43, 21, 34, 56, 3, 23, 3)
x > 30
```

```
[1] FALSE TRUE FALSE TRUE TRUE FALSE FALSE FALSE
```

3.3.4 Factores (factor y ordered)

Si los elementos de vector de tipo texto (**character**) y representan niveles nominales (categorías), el objeto puede convertirse a **factor** de modo tal que los valores son reemplazados por un número que se asocia a los niveles del factor (ordenados alfabéticamente, a menos que se indique otra cosa).

Un ejemplo de un vector tipo **character**.

```
x <- c('bajo', 'medio', 'alto', 'alto', 'bajo', 'bajo')
x
```

```
[1] "bajo" "medio" "alto" "alto" "bajo" "bajo"
```

Sólo se muestran los valores (bajo, medio y alto). No hay información de niveles. Ahora si aplicamos `factor(x)`:

```
y <- factor(x)
y
```

```
[1] bajo medio alto alto bajo bajo
Levels: alto bajo medio
```

Los valores pasaron al atributo `levels` y los datos fueron reemplazados por los identificadores 2, 3, y 1 según el orden alfabético de los niveles.

```
as.numeric(y)
```

```
[1] 2 3 1 1 2 2
```

Cuando los niveles tienen una jerarquía u orden, se puede especificar este tipo de relación mediante `as.ordered()` que convierte el `factor` en uno especial `ordered` agregando la relación entre los niveles

```
z <- factor(x, levels = c('bajo', 'medio', 'alto'))
z <- as.ordered(z)
z
```

```
[1] bajo medio alto alto bajo bajo
Levels: bajo < medio < alto
```

Los factores como cualquier vector también se indexan con `[]`.

3.3.5 Otros tipos de datos

Los valores faltantes se simbolizan en **R** con `NA` (*not available*). Indican que debería haber un valor pero que está faltando.

```
x <- c(1, 2, 3, NA, 4)
is.na(x)
```

```
[1] FALSE FALSE FALSE  TRUE FALSE
```

A diferencia del NA, un valor de tipo NULL indica que no hay información y que tampoco se esperaba que la haya.

```
x <- c(1, 2, 3, NULL, 4)
x
```

```
[1] 1 2 3 4
```

Algunas operaciones matemáticas devuelven valores NaN (*not a number*) cuando no están definidas, por ejemplo:

```
0/0
```

```
[1] NaN
```

O bien valores infinitos (Inf):

```
1/0
```

```
[1] Inf
```

ACTIVIDAD

1. Ingrese lo siguiente en un vector con el nombre **color**: púrpura, rojo, amarillo, marrón
2. Mostrar el segundo elemento en el vector (rojo) en la consola.
3. Ingrese lo siguiente en un vector con el nombre **peso**: 23, 21, 18, 26

3.4 Estructura de datos

A partir de los tipos de datos que vimos antes, se pueden construir objetos más complejos.

3.4.1 Matriz (`matrix`)

Colección de vectores de *igual longitud y mismo tipo de datos*. Se crea con la función `matrix()`, o combinando filas o columnas de igual longitud con `rbind()` o `cbind()`.

Por ejemplo la matriz:

$$\mathbf{M} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

en **R** se representa así:

```
M <- matrix(c(1, 2, 3, 4, 5, 6), ncol = 2)
M
```

```
      [,1] [,2]
[1,]     1     4
[2,]     2     5
[3,]     3     6
```

Se puede indexar usando `[n, p]` donde **n** es el numero de fila y **p** numero de columna. Por ejemplo para obtener el elemento m_{12}

```
M[1,2]
```

```
[1] 4
```

O todos los elementos de la columna 2

```
M[, 2]
```

```
[1] 4 5 6
```

3.4.2 Listas (`list`)

Es una generalización de los vectores ya que *los elementos pueden ser de igual o diferente tipo de datos*

```
lst <- list(23, "hola", TRUE)
lst
```

```
[[1]]
[1] 23
```

```
[[2]]
[1] "hola"
```

```
[[3]]
[1] TRUE
```

Se pueden indexar usando `[[]]`

```
# El segundo elemento de l
lst[[2]]
```

```
[1] "hola"
```

Cada elemento a su vez puede ser cualquier objeto de los vistos anteriormente.

3.4.3 Hoja de datos (`data.frame`)

Similares a las matrices pero cada columna puede ser de un tipo de dato diferente. Útil para guardar datos donde cada fila es un caso y cada columna una variable.

Supongamos que tenemos la tabla de datos:

Lote	Variedad	Rendimiento
1	Escorpion	34
2	Escorpion	36
3	Yarara	40
4	Baguette11	28
5	Tijetera	31

En **R** podemos representarla así:


```
trigo <- data.frame(
  lote = 1:5,
  variedad = c('Escorpion', 'Escorpion', 'Yarara', 'Baguette 11', 'Tijetera'),
  rendimiento = c(34, 36, 40, 28, 31)
)
trigo
```

	lote	variedad	rendimiento
1	1	Escorpion	34
2	2	Escorpion	36
3	3	Yarara	40
4	4	Baguette 11	28
5	5	Tijetera	31

Al igual que las matrices, un `data.frame` se puede indexar con `[]`. Por ejemplo, si quisieramos El bombre de la variedad de la fila 2

```
trigo[2, 3]
```

```
[1] 36
```

O todos los nombres de la fila 2

```
trigo[2, ]
```

	lote	variedad	rendimiento
2	2	Escorpion	36

Tambien podemos hacer consultas más específicas: “Lotes con rendimiento mayor a 35 qq/ha”

```
trigo[trigo$rendimiento > 35, ]
```

	lote	variedad	rendimiento
2	2	Escorpion	36
3	3	Yarara	40

Las variables o columnas se pueden acceder individualmente usando o el operador `$` seguido dle nombre de la columna o `[, "nombre"]`, o `[, posicion]`. Ejemplo: extraer la columna `rendimiento` que es la número 3

```
trigo$rendimiento
```

```
[1] 34 36 40 28 31
```

```
trigo[, "rendimiento"]
```

```
[1] 34 36 40 28 31
```

```
trigo[, 3]
```

```
[1] 34 36 40 28 31
```

Otras estructuras

Las estructura listadas arriba son nativas de **R**. Los paquetes o complementos pueden agregar nuevas estructuras o redifinir las existentes.

ACTIVIDAD

1. Unir los 2 vectores de la actividad anterior usando la función `data.frame` para crear un marco de datos llamado `info` con 2 columnas y 4 filas. Llame a la primera columna `'color'` y a la segunda `peso`.
2. Ver la estructura de datos en la consola.
3. Mostrar solo la fila 3 en la consola
4. Mostrar solo la columna 1 en la consola
5. Mostrar el elemento de datos en la fila 4, columna 1

4 Paquetes de R

En esta sección vamos a aprender como extender las funcionalidades de **R** mediante la instalación y utilización de paquetes o *packages*

4.1 ¿Qué son los paquetes?

R viene con un conjunto de librerías mínimo denominado *core* que permite realizar una amplia variedad de análisis y operaciones con los datos. La comunidad que desarrolla **R** provee un repositorio de librerías o paquetes complementarios (*packages*) que expanden notablemente las funcionalidades de **R**.

Los paquetes se deben *instalar* primero usando la función `install.packages()` por única vez y en cada sesión se deben *cargar* con `library()`. La siguiente figura resume el proceso:



Images sourced from <https://www.wikihow.com/Change-a-Light-Bulb>

Suponiendo que queremos instalar el paquete `foo`, se debe ejecutar por única vez:

```
install.packages("foo")
```

Luego, para acceder a todas las funciones que aporta `foo`, en cada sesión de trabajo ejecutar hay que ejecutar:

```
library("foo")
```

Alternativamente, si una vez instalado el paquete `foo` queremos usar la función `bar()` pero sin cargar el resto del paquete, entonces:

```
foo::bar(...)
```

El manejo de paquetes se puede simplificar enormemente con el paquete `pacman`. Entre otras funciones ofrece la función `p_load()` que carga los paquetes y si no están instalados los instala previamente.

Para instalar `pacman` por primera vez correr el siguiente comando:

```
# Instalar por unica vez  
install.packages("pacman")
```

Luego, cuando necesitemos podemos ejecutar `pacman::p_load()`. Por ejemplo, si queremos cargar el paquete `moments`

```
pacman::p_load(moments)
```

ACTIVIDAD

1. Instalar el paquete `pacman` usando `install.packages()`
2. Cargar/Instalar el paquete `tidyverse` usando `pacman::p_load()`

5 Importar datos en R

Un aspecto importante para cualquier análisis de datos es acceder a los **datos**!

Los datos pueden estar almacenados en diversos formatos: archivos de texto (`*.txt`, `*.dat`, etc), texto separado por comas (`*.csv`), planillas de cálculos (`*.xls` o `*.xlsx`), etc.

En este curso vamos a trabajar con archivos que ya se encuentran en planillas de cálculo tipo Excel o archivos de texto plano.

5.1 Función nativa para importar datos

R viene la función `read.table()` y derivados, que permiten leer datos desde formatos tipo **texto plano** (`plain text format`). El más popular entre estos es `*.csv`. Este formato asume que los datos están en formato de tabla o *rectangular* (e.g. variables en columnas y observaciones en filas) y devuelve un `data.frame`. En `?read.table` se detallan todos los argumentos, los más importantes son:

- `file` para indicar el nombre o ruta al archivo
- `header` para indicar si las columnas tienen encabezados que deben ser usados como nombre de las variables.
- `sep` para indicar el separador de columnas
- `dec` para indicar el símbolo decimal

Dependiendo de las combinaciones de estos 3 argumentos hay variantes (`read.csv()`, `read.csv2()`, `read.delim()`, `read.delim2()`) que son atajos de ‘`read.table()`’ (ver ayuda).

Mediante algún editor de textos (puede ser dentro de **RStudio**) conviene abrir el archivo y examinarlo para determinar:

- Tiene encabezados?
- Cómo están separadas las columnas?
- Cuál es el símbolo del decimal?

Supongamos que tenemos el archivo de texto `prueba.csv`, las alternativas podrían ser:

```
# Con encabezados, separado por tabulaciones y el decimal es el punto
prueba <- read.table("prueba.csv", header = T, sep = "\t", dec = ".")

# Con encabezados, separado por tabulaciones y coma como decimal
prueba <- read.table("prueba.csv", header = T, sep = "\t", dec = ",")

# Con encabezados, separado comas y punto como decimal
prueba <- read.table("prueba.csv", header = T, sep = ",", dec = ".")

# Con encabezados, separado punto y coma, y con coma como punto decimal
prueba <- read.table("prueba.csv", header = T, sep = ";", dec = ",")
```

En el caso que el archivo `prueba.csv` esté en otro directorio o ubicación que no sea el proyecto o `getwd()` hay que indicar la ruta completa al archivo.

Una vez importados los datos es conveniente verificar como han sido leídos en el **R**. Una alternativa es *imprimirlo* escribiendo el nombre del objeto directamente en la consola.

```
prueba
```

Otra alternativa es utilizar la función `View()` que muestra la tabla de datos en un formato de planilla interactiva de solo lectura.

```
View(prueba)
```

Si bien podemos inferir que tipo de datos se leyeron, una alternativa mejor es mirar la estructura con la función `str()`.

```
str(prueba)
```

ACTIVIDAD

1. Abrir un documento de texto en **RStudio**
2. Retomando el ejemplo del set de datos de trigo, crear un archivo separado por comas con los siguientes datos

```
Lote,Variedad,Rendimiento
1,Escorpion,34
2,Escorpion,36
3,Yarara,40
```

```
4,Baguette11,28
5,Tijetera,31
```

3. Guardar el archivo como `prueba_trigo.csv`.
4. Leer el set de datos dentro de **R** usando la función `read.table()` o alguna de sus variantes.
5. Verificar la importacion de los datos.

5.2 Paquetes para importar datos

Existen paquetes específicos que permiten leer virtualmente cualquier formato de archivos.

5.2.1 readr

Si bien la función `read.table()` y derivadas permiten leer datos *rectangulares* en formato texto, el paquete `readr` [link](#) provee una implementación más moderna (y rápidas) de estas funciones. Las funciones se llaman igual que las nativas pero se reemplaza `.` por `_` en el nombre. Ejemplo: `read_table()`

Este paquete ya está integrado a `tidyverse`. Para leer el set de datos que creamos en la actividad anterior:

```
pacman::p_load(tidyverse)
prueba_trigo <- read_csv("datasets/prueba_trigo.csv")
```

```
Rows: 5 Columns: 3
```

```
-- Column specification -----
```

```
Delimiter: ","
```

```
chr (1): Variedad
```

```
dbl (2): Lote, Rendimiento
```

```
i Use `spec()` to retrieve the full column specification for this data.
```

```
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
prueba_trigo
```

```
# A tibble: 5 x 3
  Lote Variedad Rendimiento
<dbl> <chr>      <dbl>
1     1 Escorpion      34
2     2 Escorpion      36
3     3 Yarara         40
4     4 Baguette11     28
5     5 Tijetera       31
```

A diferencia de la función nativa, las funciones de **readr** devuelven un objeto llamado **tibble** que es una especie de **data.frame** pero con algunas propiedades extra.

5.2.2 rio

A diferencia de **readr** que es una reimplementación de funciones de **R**, hay un paquete llamado **rio** [link](#) que es una especie de *metapaquete* y permite simplificar la *importación*, *exportación*, y *conversión* de formatos en una sintaxis unificada.

Este paquete trabaja con una mayor variedad de formatos y, basado en la extensión del archivo, busca la función y/o paquete apropiado para leer o guardar los datos. En el caso de ser necesario, se pueden pasar argumentos a las funciones.

Retomando el ejemplo de trigo, podemos leer los datos con la función **import()**

```
pacman::p_load(rio)
prueba_trigo <- import("datasets/prueba_trigo.csv")
prueba_trigo
```

```
  Lote  Variedad Rendimiento
1     1 Escorpion      34
2     2 Escorpion      36
3     3 Yarara         40
4     4 Baguette11     28
5     5 Tijetera       31
```

A diferencia de **readr** siempre devuelve un **data.frame**. Si queremos que devuelva un **tibble** podemos usar el argumento **setclass**

```
prueba_trigo <- import("datasets/prueba_trigo.csv", setclass = "tibble")
prueba_trigo
```



```
# A tibble: 5 x 3
  Lote Variedad Rendimiento
<int> <chr>      <int>
1     1 Escorpion      34
2     2 Escorpion      36
3     3 Yarara         40
4     4 Baguette11     28
5     5 Tijetera       31
```

5.3 Formas de importar datos

A continuación vamos a detallar dos formas de abrir el archivo que contiene datos meteorológicos de la ciudad de Urbana (Illinois).

5.3.1 Desde la consola (recomendado)

Una vez que descargamos el archivo datos en la carpeta **datasets** dentro de nuestro directorio de trabajo o proyecto podemos leerlo en **R** usando la función `import()` del paquete **rio**. Esta función se encargará de llamar la función necesaria para leer el archivo que le suministremos.

```
urbana <- import("datasets/urbana_weather.xlsx", setclass = "tibble")
urbana
```

```
# A tibble: 240 x 4
  YEAR month precip temp
<dbl> <chr>  <dbl> <dbl>
1  2000 Jan    1.54  25.6
2  2001 Jan    1.32  25.4
3  2002 Jan    2.81   34
4  2003 Jan    0.79  21.1
5  2004 Jan    2.18   24
6  2005 Jan    6.2   27.8
7  2006 Jan    1.78  37.8
8  2007 Jan    3.03  29.7
9  2008 Jan    2.31  26.2
10 2009 Jan    0.68  18.8
# i 230 more rows
```

Si sólo estuviéramos interesados en el rango A1:C5 (primeros 4 registros de las 3 primeras columnas), podríamos usar:

```
urbana2 <- import(file = "datasets/urbana_weather.xlsx", range = "A1:C5")
urbana2
```

```
YEAR month precip
1 2000 Jan 1.54
2 2001 Jan 1.32
3 2002 Jan 2.81
4 2003 Jan 0.79
```

5.3.2 Desde el importador de datos de RStudio

RStudio cuenta con un asistente de importación de datos (**File > Import Dataset**) que brinda interfase a varias funciones especializadas en la importación de datos de paquetes específicos como **readr**, **readxl**, etc.

En el menú **File > Import Dataset** o bien el ícono del panel **Environment** despliega una lista con distintas opciones de importación: nos interesa **From Excel (readxl)**...



Figura 5.1: Importador de datos

Este menú tiene cuatro paneles:

1. Una barra de dirección para indicar la ruta al archivo o URL.
2. Una vista previa del contenido del archivo
3. Opciones de importación: aquí se puede especificar el nombre del objeto que se creará dentro de **R** (**Name**), la cantidad de líneas a leer, el número de la hoja, el rango de celdas, líneas a saltar (**skip**) y el identificador de datos **NA**.
4. Vista previa del código. En esta parte se puede visualizar como se construye el comando que se ejecutará al clickear en **Import**.

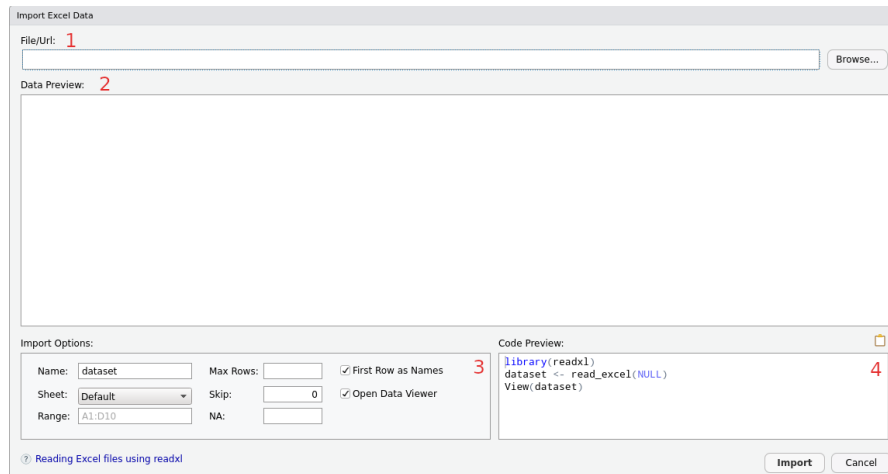


Figura 5.2: Importador de datos

i Aclaración

Si bien esta alternativa es intuitiva y amigable, no es reproducible a menos que el código generado por este asistente sea incluido en el script para futuras sesiones.

5.3.3 Desde el portapapeles

Una alternativa conveniente para acceder rápidamente a los datos es usando el portapapeles. Suponiendo que los datos están en una hoja de cálculos:

1. Seleccionar el rango de celdas **A1:C5** que nos interesa incluyendo los encabezados
2. Copiar en el porta papeles (CTRL + C)
3. Luego en R

```
urbana3 <- read.table("clipboard")
```

i Aclaración

Si bien esta alternativa es rápida, al no ser reproducible (no hay forma de plasmarla en el script para futuras sesiones), **no es recomendable** salvo para una exploración rápida.

6 Manipulando datos con dplyr y tidyr

Luego de importar los datos **R** es necesario explorarlos y, dependiendo de su organización o el análisis que querramos realizar, hay que realizar algún tipo transformación, resumen, o consultas.

dplyr y **tidyr** son dos paquetes de **R** muy potentes para la exploración, transformación y resumen de datos en formato de tabla con filas (observaciones) y columnas (variables). Ambos son componentes de un meta-package **tidyverse** desarrollados por [Hadley Wickham](#)

Ambos paquetes tienen un conjunto de funciones (o *verbos*) que realizan operaciones comunes para el manejo de datos tales como: filtrar filas, seleccionar columnas, re-ordenar filas, agregar o transformar columnas, resumir datos. También permite agrupar los datos facilitando la estrategia *split-apply-combine*, es decir, dividir (*split*) los datos según una variable de grupo, aplicar (*apply*) alguna transformación o resumen y combinar (*combine*) las partes para presentar los resultados.

Si bien **R** base tiene funciones que realizan las mismas tareas (`split()`, `subset()`, `apply()`, `sapply()`, `lapply()`, `tapply()` and `aggregate()`), estos paquetes brindan una interface más consistente que permite trabajar de manera más fácil con `data.frame` (tabla de datos) más que con vectores.

Veamos con un ejemplo como realizar algunas operaciones con estos paquetes. Para esto vamos a usar el set de datos . Suponiendo que ya está descargado en la carpeta de trabajo o proyecto.

```
pacman::p_load(rio)
novillos <- import("datasets/pesada_novillos.xlsx", setclass = "tibble")
novillos
```

```
# A tibble: 1,842 x 13
```

	IDV	Tropa	Procedencia	Fecha_Ingreso	Peso_inicial	Peso_anterior	GPV_anterior
	<chr>	<dbl>	<chr>	<chr>	<dbl>	<dbl>	<dbl>
1	GH73~	2	<NA>	42939	188	306	51
2	IA67~	2	La Rosita	42940	232	290	2
3	PO15~	2	La Alameda	42939	204	290	33
4	GH73~	2	<NA>	42940	204	303	27
5	SZ20~	2	La Alameda	42940	202	300	30
6	IA67~	2	La Rosita	42940	234	333	38

```

7 P015~      2 Los Corral~ 42938      214      287      14
8 P015~      2 Los Corral~ 42939      184      315      37
9 NS50~      2 La Rosita   42938      234      288      0
10 QX67~     2 La Alameda  42940      206      262      2
# i 1,832 more rows
# i 6 more variables: GDM_anterior <dbl>, Fecha <chr>, Hora <chr>, Peso <dbl>,
#   Días <dbl>, Días_total <dbl>

```

6.1 ¿Cómo instalar dplyr y tidyr?

Para acceder a estos paquetes instalarlos/cargarlos individualmente:

```
pacman::p_load(dplyr, tidyr)
```

O bien con otros paquete de la familia tidyverse

```
pacman::p_load(tidyverse)
```

R va a avisarnos en la consola que esta enmascarando (reemplazando) algunas funciones que ya estaban en el entorno, o bien el paquete nos devuelve algun mensaje. A menos que diga `Error ...`, eso está bien.

6.2 Verbos importantes de dplyr para recordar

Toda la estrategia de trabajo con datos de dplyr se basa en 6 *verbos*:

verbo	descripción
<code>select()</code>	selecciona columnas (variables)
<code>filter()</code>	filtra o selecciona las filas (observaciones)
<code>arrange()</code>	re-ordena las filas
<code>mutate()</code>	crea nuevas columnas o modifica las preexistentes
<code>summarise()</code>	resumen los valores de una variable
<code>group_by()</code>	permite aplicar los verbos anteriores en subgrupos (<i>split-apply-combine</i>)

En la mayoría de los casos la sintaxis es:

```
function(que_datos, que_hacer_con_los_datos)
```

donde `que_datos` es el nombre del set de datos (un `data.frame`) y `que_hacer_con_los_datos` indicar que operación, condicion, transformacion, etc. aplicar a las filas y columnas.

Veamos unos ejemplos

6.3 Seleccionando variables

Para seleccionar columnas (variables) de interés usamos `select()`. La selección se puede hacer por nombre de la columna (con o sin comillas) o indicando su posición.

Las columnas del set de datos de ejemplo se denominan:

```
names(novillos)
```

```
[1] "IDV"           "Tropa"          "Procedencia"    "Fecha_Ingreso"
[5] "Peso_inicial"  "Peso_anterior"  "GPV_anterior"   "GDM_anterior"
[9] "Fecha"         "Hora"           "Peso"           "Días"
[13] "Días_total"
```

Si quisiéramos seleccionar las columnas `Procedencia`, `Peso_inicial`, `Peso_anterior` y `Peso`, podríamos ejecutar:

```
select(novillos, Procedencia, Peso_inicial, Peso_anterior, Peso)
```

```
# A tibble: 1,842 x 4
  Procedencia  Peso_inicial Peso_anterior  Peso
  <chr>        <dbl>         <dbl> <dbl>
1 <NA>         188           306   398
2 La Rosita    232           290   387
3 La Alameda   204           290   386
4 <NA>         204           303   382
5 La Alameda   202           300   382
6 La Rosita    234           333   374
7 Los Corralitos 214           287   371
8 Los Corralitos 184           315   369
9 La Rosita    234           288   365
10 La Alameda   206           262   365
# i 1,832 more rows
```

El resultado es un nuevo `data.frame` o `tibble` que sólo contiene las 4 columnas seleccionadas. Si quisiéramos guardar este resultado aparte.

```
pesos <- select(novillos, Procedencia, Peso_inicial, Peso_anterior, Peso)
pesos
```

```
# A tibble: 1,842 x 4
  Procedencia    Peso_inicial Peso_anterior  Peso
  <chr>          <dbl>         <dbl> <dbl>
1 <NA>           188           306   398
2 La Rosita      232           290   387
3 La Alameda     204           290   386
4 <NA>           204           303   382
5 La Alameda     202           300   382
6 La Rosita      234           333   374
7 Los Corralitos 214           287   371
8 Los Corralitos 184           315   369
9 La Rosita      234           288   365
10 La Alameda    206           262   365
# i 1,832 more rows
```

Cuando se imprime en la consola, es lo mismo que usar `print()` la cual por defecto muestra las 10 primeras observaciones y la cantidad de columnas que entran en la pantalla. Aquellas columnas que no entran son indicadas al pie.

Si uno desea ver más registros se puede usar el argumento `n =` de `print()`

```
print(pesos, n = 15)
```

```
# A tibble: 1,842 x 4
  Procedencia    Peso_inicial Peso_anterior  Peso
  <chr>          <dbl>         <dbl> <dbl>
1 <NA>           188           306   398
2 La Rosita      232           290   387
3 La Alameda     204           290   386
4 <NA>           204           303   382
5 La Alameda     202           300   382
6 La Rosita      234           333   374
7 Los Corralitos 214           287   371
8 Los Corralitos 184           315   369
9 La Rosita      234           288   365
10 La Alameda    206           262   365
11 La Alameda    216           308   362
12 La Rosita     240           298   362
```

```

13 La Alameda          164          285    361
14 La Rosita           216          308    361
15 La Alameda          176          289    361
# i 1,827 more rows

```

Con `n = "all"` se imprimen todas (no se muestra por razones obvias)

El orden en que aparecen las variables en el resultado es el orden que se utilizó al seleccionarlás.

```
select(novillos, Procedencia, Peso, Peso_inicial, Peso_anterior)
```

```

# A tibble: 1,842 x 4
  Procedencia      Peso Peso_inicial Peso_anterior
  <chr>          <dbl>         <dbl>         <dbl>
1 <NA>           398           188           306
2 La Rosita      387           232           290
3 La Alameda     386           204           290
4 <NA>           382           204           303
5 La Alameda     382           202           300
6 La Rosita      374           234           333
7 Los Corralitos 371           214           287
8 Los Corralitos 369           184           315
9 La Rosita      365           234           288
10 La Alameda    365           206           262
# i 1,832 more rows

```

También se puede usar los comandos `starts_with()`, `ends_with()`, `contains()`, etc (ver `?select_helpers`) para más opciones). Para elegir varias columnas que tienen un patrón sin tener que tipear todos los nombres.

```
select(novillos, starts_with("P"))
```

```

# A tibble: 1,842 x 4
  Procedencia      Peso_inicial Peso_anterior  Peso
  <chr>          <dbl>         <dbl> <dbl>
1 <NA>           188           306   398
2 La Rosita      232           290   387
3 La Alameda     204           290   386
4 <NA>           204           303   382
5 La Alameda     202           300   382

```



```

6 La Rosita                234          333    374
7 Los Corralitos           214          287    371
8 Los Corralitos           184          315    369
9 La Rosita                234          288    365
10 La Alameda              206          262    365
# i 1,832 more rows

```

Para omitir algunas columnas en la seleccion se puede usar el `-` antes del nombre. Por ejemplo, omitir la columna IDV y todas las que empiezan con P.

```
select(novillos, -IDV, -starts_with("P"))
```

```

# A tibble: 1,842 x 8
  Tropa Fecha_Ingreso GPV_anterior GDM_anterior Fecha Hora      Días Días_total
  <dbl> <chr>          <dbl>         <dbl> <chr> <chr>    <dbl>    <dbl>
1     2  42939             51          1.04 43138 0.80824~    76      293
2     2  42940             2           0.04 43138 0.46511~    76      293
3     2  42939            33          0.67 43138 0.34270~    76      302
4     2  42940            27          0.52 43138 0.34829~    75      293
5     2  42940            30          0.58 43138 0.80134~    75      304
6     2  42940            38          0.73 43138 0.34018~    75      295
7     2  42938            14          0.27 43138 0.32917~    75      303
8     2  42939            37          0.76 43138 0.33152~    76      303
9     2  42938             0           0    43138 0.37302~    76      302
10    2  42940             2          0.08 43143 0.46458~   132      306
# i 1,832 more rows

```

6.4 Seleccionando observaciones

Otra tarea muy frecuente es seleccionar casos o observaciones que cumplan con alguna condición. Esto se lleva a cabo con `filter()`. Se pueden usar los operadores *booleanos* `==`, `>`, `<`, `>=`, `<=`, `!=`, `%in%` para crear pruebas o condiciones lógicas.

Por ejemplo, para seleccionar los novillos de Los Corralitos:

```
filter(novillos, Procedencia == 'Los Corralitos')
```

```

# A tibble: 57 x 13
  IDV   Tropa Procedencia Fecha_Ingreso Peso_inicial Peso_anterior GPV_anterior
  <chr> <dbl> <chr>         <chr>         <dbl>         <dbl>         <dbl>

```

```

1 P015~      2 Los Corral~ 42938      214      287      14
2 P015~      2 Los Corral~ 42939      184      315      37
3 P015~      2 Los Corral~ 42937      168      266      22
4 P015~      2 Los Corral~ 42937      182      281      31
5 P015~      2 Los Corral~ 42938      200      276      20
6 P015~      2 Los Corral~ 42937      162      231      19
7 P015~      2 Los Corral~ 42937      166      240      26
8 P015~      2 Los Corral~ 42942      162      232      28
9 P015~      2 Los Corral~ 42939      200      258      26
10 P015~     2 Los Corral~ 42940      170      252      37
# i 47 more rows
# i 6 more variables: GDM_anterior <dbl>, Fecha <chr>, Hora <chr>, Peso <dbl>,
#   Días <dbl>, Días_total <dbl>

```

Para crear un subconjunto podemos asignarlo y crear un nuevo set de datos.

```
corralitos <- filter(novillos, Procedencia == 'Los Corralitos')
```

La selección se puede hacer por más de una condición. Por ejemplo, seleccionar los de Los Corralitos que pesen más de 300 kg:

```
filter(novillos, Procedencia == 'Los Corralitos', Peso > 300)
```

```

# A tibble: 29 x 13
  IDV   Tropa Procedencia Fecha_Ingreso Peso_inicial Peso_anterior GPV_anterior
  <chr> <dbl> <chr>          <chr>          <dbl>          <dbl>          <dbl>
1 P015~     2 Los Corral~ 42938          214          287          14
2 P015~     2 Los Corral~ 42939          184          315          37
3 P015~     2 Los Corral~ 42937          168          266          22
4 P015~     2 Los Corral~ 42937          182          281          31
5 P015~     2 Los Corral~ 42938          200          276          20
6 P015~     2 Los Corral~ 42937          162          231          19
7 P015~     2 Los Corral~ 42937          166          240          26
8 P015~     2 Los Corral~ 42942          162          232          28
9 P015~     1 Los Corral~ 23/7/2017        252          343          31
10 P015~     1 Los Corral~ 22/7/2017        234          338          23
# i 19 more rows
# i 6 more variables: GDM_anterior <dbl>, Fecha <chr>, Hora <chr>, Peso <dbl>,
#   Días <dbl>, Días_total <dbl>

```

`filter()` asume que cada condicion se debe cumplir en simultaneo para que la observación sea seleccionada. Esto equivale a utilizar el operador `&` (`Y`). En caso de querer seleccionar aquellos

registros que cumple una u otra condicion se usa el operador `|` (*O*). Poniendo `!` delante de la condicion se invierte la selección.

Por ejemplo, para obtener novillos que son de Los Corralitos **O** pesan más de 300 kg.

```
filter(novillos, Procedencia == 'Los Corralitos' | Peso > 300)
```

```
# A tibble: 841 x 13
```

	IDV	Tropa	Procedencia	Fecha_Ingreso	Peso_inicial	Peso_anterior	GPV_anterior
	<chr>	<dbl>	<chr>	<chr>	<dbl>	<dbl>	<dbl>
1	GH73~	2	<NA>	42939	188	306	51
2	IA67~	2	La Rosita	42940	232	290	2
3	P015~	2	La Alameda	42939	204	290	33
4	GH73~	2	<NA>	42940	204	303	27
5	SZ20~	2	La Alameda	42940	202	300	30
6	IA67~	2	La Rosita	42940	234	333	38
7	P015~	2	Los Corral~	42938	214	287	14
8	P015~	2	Los Corral~	42939	184	315	37
9	NS50~	2	La Rosita	42938	234	288	0
10	QX67~	2	La Alameda	42940	206	262	2

```
# i 831 more rows
```

```
# i 6 more variables: GDM_anterior <dbl>, Fecha <chr>, Hora <chr>, Peso <dbl>,
```

```
# Días <dbl>, Días_total <dbl>
```

6.5 Encadenando operaciones (operador `%>%`)

El operador `%>%` permite encadenar operaciones sucesivas para evitar tener que crear tablas intermedias o anidar funciones. El operador `y` se le de izquierda a derecha y se puede traducir como *luego*.

Por ejemplo, reportar los IDV y Peso de los novillos con más de 250 kg. Esto implicaría seleccionar las columnas de interés creando tablas intermedias y *luego* filtrar la tabla o vice versa.

Creando tablas intermedias

```
novillos2 <- select(novillos, IDV, Peso)
novillos2
```

```
# A tibble: 1,842 x 2
```

IDV	Peso
-----	------

```

      <chr>      <dbl>
1 GH738B371    398
2 IA671B298    387
3 P0150A818    386
4 GH738B340    382
5 SZ208H766    382
6 IA671B133    374
7 P0150A532    371
8 P0150A563    369
9 NS509H071    365
10 QX678K546    365
# i 1,832 more rows

```

```

novillos2 <- filter(novillos2, Peso > 250)
novillos2

```

```

# A tibble: 1,682 x 2
  IDV      Peso
  <chr>    <dbl>
1 GH738B371 398
2 IA671B298 387
3 P0150A818 386
4 GH738B340 382
5 SZ208H766 382
6 IA671B133 374
7 P0150A532 371
8 P0150A563 369
9 NS509H071 365
10 QX678K546 365
# i 1,672 more rows

```

Anidando las funciones:

```

filter(select(novillos, IDV, Peso), Peso > 250)

```

```

# A tibble: 1,682 x 2
  IDV      Peso
  <chr>    <dbl>
1 GH738B371 398
2 IA671B298 387

```

```

3 P0150A818 386
4 GH738B340 382
5 SZ208H766 382
6 IA671B133 374
7 P0150A532 371
8 P0150A563 369
9 NS509H071 365
10 QX678K546 365
# i 1,672 more rows

```

Una opción más clara para leer y eficiente es usando %>%

```

novillos %>%
  select(IDV, Peso) %>%
  filter(Peso > 250)

```

```

# A tibble: 1,682 x 2
  IDV      Peso
  <chr>   <dbl>
1 GH738B371 398
2 IA671B298 387
3 P0150A818 386
4 GH738B340 382
5 SZ208H766 382
6 IA671B133 374
7 P0150A532 371
8 P0150A563 369
9 NS509H071 365
10 QX678K546 365
# i 1,672 more rows

```

Con %>% se puede omitir el nombre de la tabla sobre la que se está trabajando (bonus: menos tipeo).

La última opción se lee: __tomar la tabla **novillos**, **luego** seleccionar las columnas **IDV** y **Peso**, **luego** filtrar los novillos con pesos mayores a 250 kg.

6.6 Ordenar las filas

Para ordenar según algún criterio aplicado a las columnas se usa **arrange()**. Por ejemplo, continuar con lo anterior pero mostrar ordenados por peso.

```
novillos %>%
  select(IDV, Peso) %>%
  filter(Peso > 250) %>%
  arrange(Peso)
```

```
# A tibble: 1,682 x 2
```

	IDV	Peso
	<chr>	<dbl>
1	ME367A534	251
2	SZ208H833	251
3	NS545A388	251
4	ME367A545	251
5	NS509H010	251
6	P0150A777	251
7	P0150A752	251
8	SZ208H765	252
9	NS545A455	252
10	SZ208I511	252

```
# i 1,672 more rows
```

Con `decs(variable)` se ordena de mayor a menor

```
novillos %>%
  select(IDV, Peso) %>%
  filter(Peso > 250) %>%
  arrange(desc(Peso))
```

```
# A tibble: 1,682 x 2
```

	IDV	Peso
	<chr>	<dbl>
1	GH738B371	398
2	IA671B276	397
3	IA671B298	387
4	P0150A818	386
5	GH738B340	382
6	SZ208H766	382
7	IA671B133	374
8	P0150A532	371
9	P0150A866	371
10	P0150A563	369

```
# i 1,672 more rows
```

6.7 Crear o transformar columnas

Para crear nuevas columnas aplicando funciones a otras, o bien para transformar columnas se usa `mutate()`. Se pueden modificar más de una columna a la vez. Por ejemplo, suponiendo que interesa obtener la diferencia de peso desde el inicio del ciclo:

```
dif_peso = Peso - Peso_inicial
```

```
novillos %>%  
  mutate(dif_peso = Peso - Peso_inicial) %>%  
  select(Peso, Peso_inicial, dif_peso) # para que se vea mejor el resultado
```

```
# A tibble: 1,842 x 3  
  Peso Peso_inicial dif_peso  
  <dbl>      <dbl>    <dbl>  
1   398         188      210  
2   387         232      155  
3   386         204      182  
4   382         204      178  
5   382         202      180  
6   374         234      140  
7   371         214      157  
8   369         184      185  
9   365         234      131  
10  365         206      159  
# i 1,832 more rows
```

Esto no cambia el set de datos `novillos` ya que no se lo asignó a ningún objeto. Para sobrescribir o actualizar el set de datos `novillos` hay que asignarlo al mismo objeto.

```
novillos <- novillos %>%  
  mutate(dif_peso = Peso - Peso_inicial)
```

Aclaración: Si se hubiese usado `select()` el set de datos `novillos` solamente contendría las columnas seleccionadas.

6.7.1 Resumir datos

Mediante `summarise()` se pueden aplicar funciones para resumir en un solo valor los valores de las columnas. Las funciones a aplicar deben devolver un único valor, por ejemplo `mean()`. Si usamos `summary()` esto devolverá 6 valores y dará error.

```
novillos %>%
  summarise(
    media = mean(Peso),
    sd = sd(Peso),
    n = n(),
    suma = sum(Peso),
    procedencias = n_distinct(Procedencia)
  )
```

```
# A tibble: 1 x 5
  media    sd      n  suma procedencias
  <dbl> <dbl> <int> <dbl>         <int>
1  294.  33.1  1842 541693           4
```

Nuevamente estos resultados pueden asignarse a otro objeto o bien encadenarse con otras operaciones.

6.7.2 Agrupar (último pero no menos importante)

El verbo `group_by()` es muy útil para aplicar operaciones en subgrupos y presentar todo junto (*split-apply-combine*). Lo que hace es indicar que en el `data.frame` hay una o más variables que conforman los grupos. Luego cada operación se aplica a esos subgrupos.

Ejemplo: calcular media, desvío, n y suma para cada procedencia, descartando los que tienen NA en Procedencia

```
novillos %>%
  filter(!is.na(Procedencia)) %>%
  group_by(Procedencia) %>%
  summarise(
    media = mean(Peso),
    sd = sd(Peso),
    n = n(),
    suma = sum(Peso)
  )
```

```
# A tibble: 3 x 5
  Procedencia    media    sd      n  suma
  <chr>         <dbl> <dbl> <int> <dbl>
1 La Alameda    297.  28.3   638 189236
```


2	La Rosita	296.	31.2	849	250951
3	Los Corralitos	300.	33.3	57	17085

6.8 Tablas pivot

Al igual que `dplyr`, las funciones de `tidyr` son compatibles con el operador `%>%`.

Las funciones más importantes de este paquete son `pivot_longer()` y `pivot_wider()`, las cuales permiten de manera fácil convertir tablas apaisadas en tablas longitudinales los nombres de algunas variables pasan a ser los valores de una nueva variable y los valores se combinan en una nueva variable.

Formato *wide*

x	y
1	a
2	b
3	c

Formato *long*

Variable	Valor
x	1
x	2
x	3
y	a
y	b
y	c

Veamos un ejemplo: las variables `Peso`, `Peso_inicial` y `Peso_anterior` pueden verse como la variable `Peso` medida en 3 momentos. Podríamos generar una tabla longitudinal combinando estas 3 variables en una sola y creando otra que indique el momento de medición.

```
pesos_long <- novillos %>%
  pivot_longer(cols = contains("Peso"), names_to = "momento", values_to = "peso") %>%
  select(IDV, Procedencia, momento, peso)
pesos_long
```

```
# A tibble: 7,368 x 4
  IDV      Procedencia momento      peso
  <chr>    <chr>        <chr>    <dbl>
1 GH738B371 <NA>      Peso_inicial  188
2 GH738B371 <NA>      Peso_anterior  306
3 GH738B371 <NA>      Peso          398
4 GH738B371 <NA>      dif_peso      210
5 IA671B298 La Rosita  Peso_inicial  232
6 IA671B298 La Rosita  Peso_anterior  290
7 IA671B298 La Rosita  Peso          387
8 IA671B298 La Rosita  dif_peso      155
9 P0150A818 La Alameda  Peso_inicial  204
10 P0150A818 La Alameda  Peso_anterior  290
# i 7,358 more rows
```

Como vemos, ahora tenemos una tabla que para cada novillo IDV, tiene los pesos en los distintos momentos no como columnas sino como valores de la variable **peso** y la columna **momento** identifica el asocia el momento en que se pesó el animal.

7 Visualización de datos con ggplot2

R tiene por defecto el paquete `graphics`, también conocido como *base plot system*, que provee la función genérica `plot()` para hacer gráficos simples, y otras funciones para gráficos específicos (`hist()`, `barplot()`, `boxplot()`, etc).

Usa un enfoque de *papel y lápiz* por capas donde el gráfico final es una sumatoria de capas que se agregan una a la vez sin posibilidad de modificarse luego. Generalmente es OK para gráficos simples o exploratorios. Para gráficos más complejos (con subgrupos o multipanel) requiere programar más. Una desventaja es la sintaxis poco consistente.

El paquete `ggplot2`, desarrollado por [Hadley Wickham](#), está basado en la filosofía *Gramática de gráficos* (*grammar of graphics* , por eso `gg`). Combina los dos enfoques: *por capas* y *función*.

Uno provee los datos, indica que variables asignar a las estéticas (ejes, escalas, colores, símbolos) y las geometrías o formas que se quieren graficar y `ggplot2` se encarga del resto. Se puede ir agregando capas. Es muy potente para la exploración y visualización de datos en formato de tabla con filas (observaciones) y columnas (variables).

7.1 ¿Cómo conseguir ggplot2?

Para acceder a estos paquetes instalarlos/cargarlos individualmente:

```
pacman::p_load(ggplot2)
```

O bien con otros paquete de la familia `tidyverse`

```
pacman::p_load(tidyverse)
```

R va a avisarnos en la consola que esta enmascarando (reemplazando) algunas funciones que ya estaban en el entorno, o bien el paquete nos devuelve algun mensaje. A menos que diga **Error** ..., eso está bien.

7.2 Componentes del gráfico en ggplot2

ggplot2 implementa una variante *por capas* de este paradigma gramática de gráficos de [Leland Wilkinson](#) (gg es por *grammar of graphics*). Como resultado, se crean una serie de capas que permiten describir y construir visualizaciones de manera estructurada en cuanto a representación de los elementos pero a su vez flexible para generar combinaciones nuevas.

Un gráfico se define por la combinación de capas (**layers**), escalas (**scales**), coordenadas (**coords**) y facetas (**facets**). Adicionalmente a estos componentes se pueden aplicar temas (**themes**) que permiten controlar los detalles del diseño de la visualización.

7.2.1 Layers

Los **layer** se construyen con las funciones **geom_*** y **stat_*** que veremos más adelante. Constan de 5 elementos:

- **data**, set de datos (**data.frame** o similar) que contiene la información que se desea visualizar.
- **mapping**, elementos de mapeo definidos mediante **aes()** para indicar la forma en que la las variables y observaciones van a ser representadas en la visualización mediante atributos estéticos (ejes, líneas, colores, rellenos, etc).
- **stat**, funciones estadísticas que resumen los datos aplicando funciones estadísticas, e.g. promedio, agrupamiento y conteo de observaciones, o ajuste de un modelo lineal o suavizado, etc.
- **geom**, geometrías o formas que representan lo que realmente se ve en un gráfico: puntos, líneas, polígonos, etc.
- **position**, ajuste de posición de los elementos **geoms** dentro de un layer para evitar su superposición.

Generalmente, sobre todo para gráficos simples, **data** y **mapping** se definen una vez para todo el gráfico dentro de la función **ggplot()**. En otras situaciones se hace a nivel de cada **layer**.

7.2.2 Scales

Asignan los valores del espacio de datos a valores en el espacio de los elementos estéticos (**aesthetics** o **aes**). Por ejemplo, el uso de un color, forma o tamaño de en un **geom** puede ser controlado por un atributo de los datos. Las escalas también definen las leyenda y los ejes.

7.2.3 Coordenadas

Sistema de coordenadas (`coord`) que define que variables definiran el espacio del gráfico y como se representarán, e.g. coordenadas cartesianas, polares, etc.

7.2.4 Paneles (facets)

Es un elemento que permite especificar una o más variables para dividir el gráfico en paneles y así mostrar subgrupos de datos. Esto permite ver visualizar relaciones condicionales entre variables, e.g. $y \sim x \mid z$, es decir, que pasa con la variable x e y cuando cambia z .

7.2.5 Temas

Adicionalmente a estos componentes se pueden aplicar temas (`themes`) que permiten controlar los detalles del diseño de la visualización, tipografía, posición de algunos objetos, paleta de colores, etc. Los valores predeterminados de `ggplot2` son un buen punto de partida pero existen opciones predefinidas que pueden modificarse para generar un tema particular. Otra fuente para consultar es el trabajo de [Tufte](#)

7.3 Primer gráfico paso a paso

```
pacman::p_load(rio)
novillos <- import("datasets/pesada_novillos.xlsx", setclass = "tibble")
novillos
```

```
# A tibble: 1,842 x 13
```

	IDV	Tropa	Procedencia	Fecha_Ingreso	Peso_inicial	Peso_anterior	GPV_anterior
	<chr>	<dbl>	<chr>	<chr>	<dbl>	<dbl>	<dbl>
1	GH73~	2	<NA>	42939	188	306	51
2	IA67~	2	La Rosita	42940	232	290	2
3	P015~	2	La Alameda	42939	204	290	33
4	GH73~	2	<NA>	42940	204	303	27
5	SZ20~	2	La Alameda	42940	202	300	30
6	IA67~	2	La Rosita	42940	234	333	38
7	P015~	2	Los Corral~	42938	214	287	14
8	P015~	2	Los Corral~	42939	184	315	37
9	NS50~	2	La Rosita	42938	234	288	0
10	QX67~	2	La Alameda	42940	206	262	2

```
# i 1,832 more rows
```

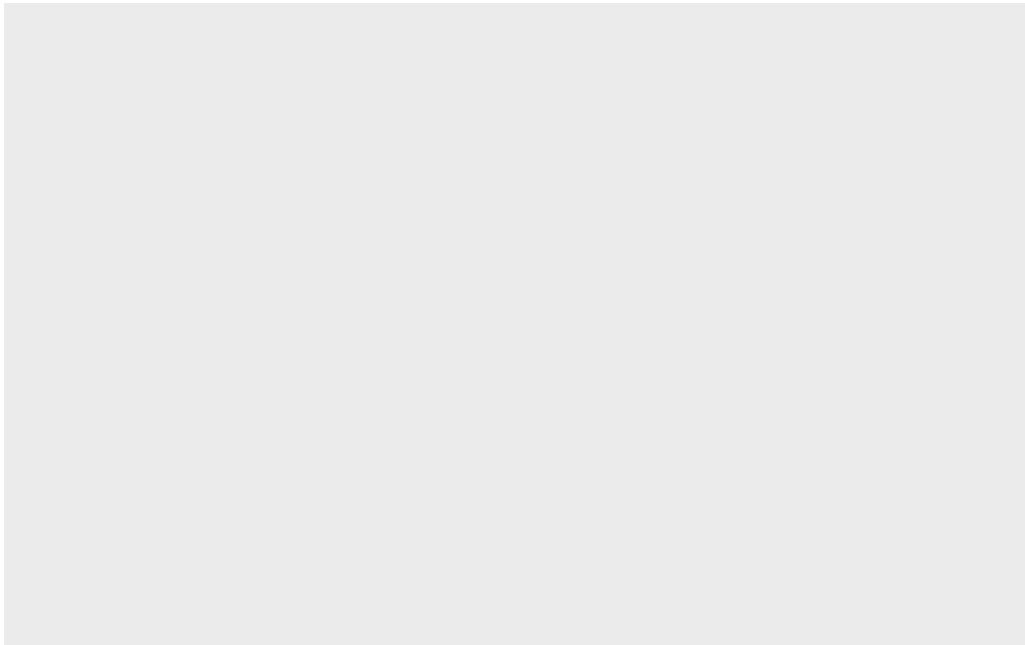
```
# i 6 more variables: GDM_anterior <dbl>, Fecha <chr>, Hora <chr>, Peso <dbl>,  
#   Días <dbl>, Días_total <dbl>
```

Veamos con un ejemplo como se combinan los componentes anteriormente vistos para realizar un gráfico simple. Para esto vamos a usar el set de datos

Nuestro primer gráfico tendrá como objetivo mostrar la relación que existe entre `Peso_anterior` y `Peso` (actual), y potencialmente ver si ésta es similar entre procedencias. Veamos paso por paso como se construye el gráfico.

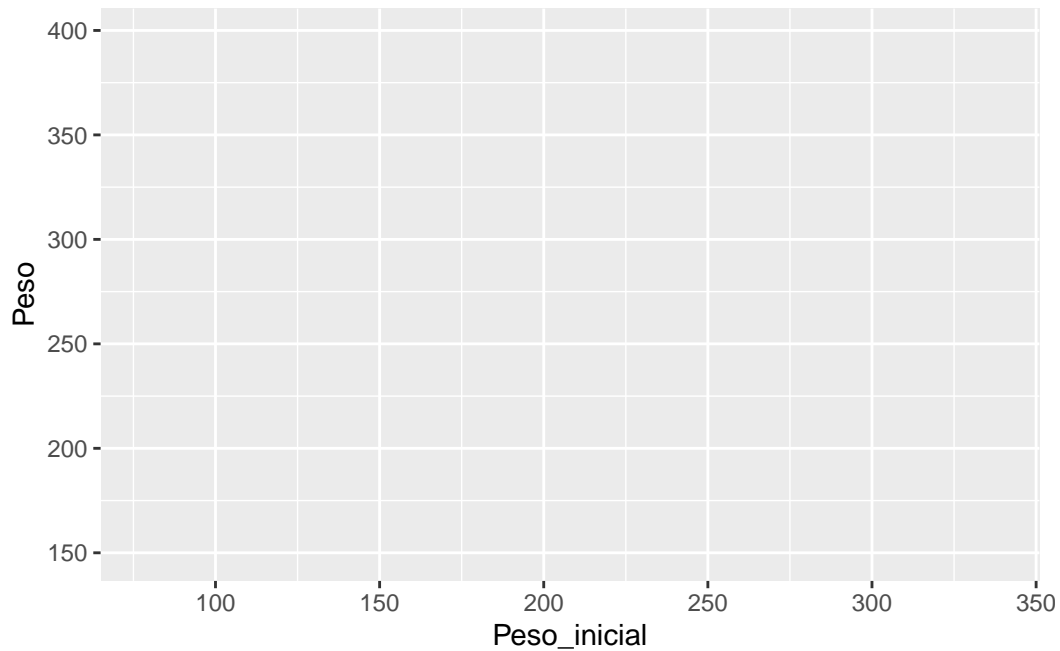
Primero definimos el set de datos que usaremos:

```
ggplot(data = novillos)
```



Como vemos esto no produjo nada ya que no indicamos cuales son las variables que queremos graficar y cómo graficarlas. Nuestro `layer` solo tiene la información de `data`. Agreguemos ahora la información de `mapping` usando `aes()`. Usando el operador `+` podemos concatenarlo al comando anterior.

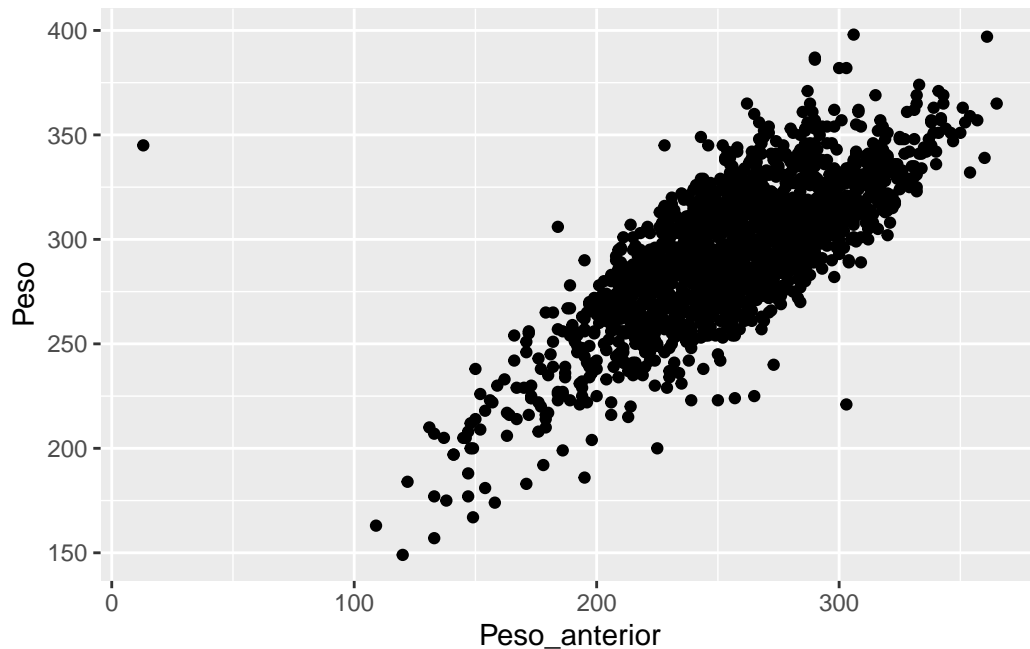
```
ggplot(data = novillos) +  
  aes(x = Peso_inicial, y = Peso)
```



Aquí vemos que, si bien no hemos graficado nada, la información suministrada permite a `ggplot` identificar los ejes, definir el espacio de coordenadas (cartesianas por defecto) y proponer unos límites en función del rango de valores. Agreguemos ahora la geometría: en este caso tiene sentido usar `geom_point()` ya que queremos mostrar un punto por observación

```
ggplot(data = novillos) +  
  aes(x = Peso_anterior, y = Peso) +  
  geom_point()
```

Warning: Removed 32 rows containing missing values (``geom_point()``).



Como vemos ahora el gráfico va tomando forma. Este tipo de gráficos se llama *gráfico de dispersión* y muestra la relación entre ambas variables. Por defecto no se aplica ninguna transformación estadística lo que equivale a (`stat = "identity"`).

A este gráfico vamos a agregarle alguna función que permita resumir la relación entre ambas variables, por ejemplo un modelo de regresión. La mejor forma de representarlo sería una línea. Para eso vamos a agregar otro `layer` con `geom_line()` donde indicaremos una transformación de los datos `stat = smooth`.

```
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm')
```

``geom_smooth()` using formula = 'y ~ x'`

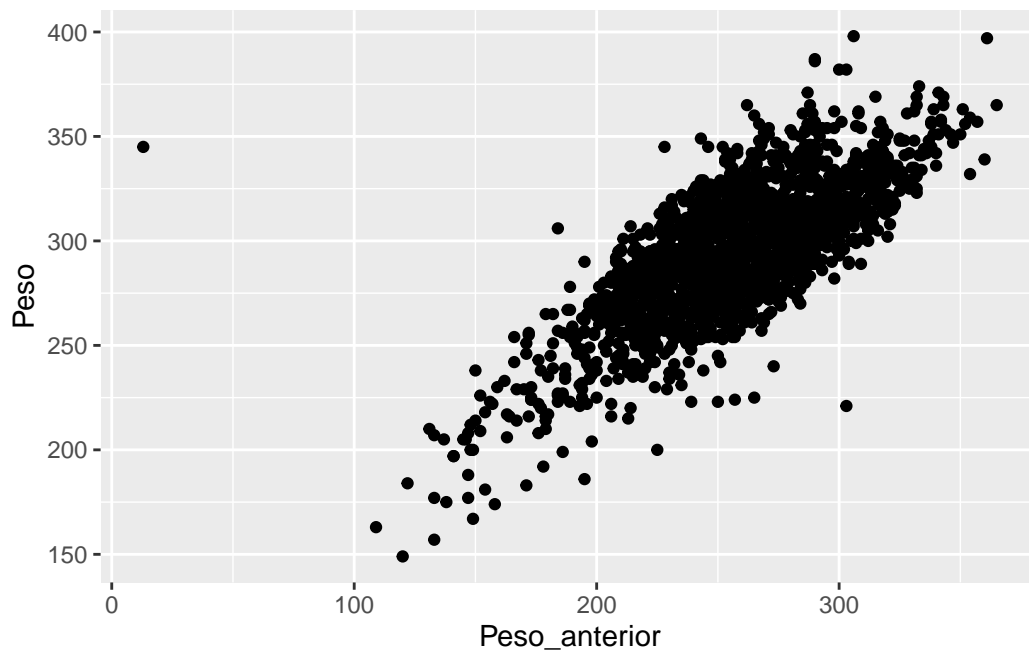
Warning: Removed 32 rows containing non-finite values (``stat_smooth()``).

Warning: Computation failed in ``stat_smooth()``

Caused by error in ``dyn.load()``:

```
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
  libopenblas.so.3: cannot open shared object file: No such file or directory
```


Warning: Removed 32 rows containing missing values (`geom_point()`).



Hay una relación positiva para todo el set de datos pero puede enmascarar algún patrón por Procedencia. Esta información la podemos agregar con otros atributos estéticos como por ejemplo color:

```
ggplot(data = novillos) +  
  aes(x = Peso_anterior, y = Peso, color = Procedencia) +  
  geom_point() +  
  geom_line(stat = 'smooth', method = 'lm')
```

`geom_smooth()` using formula = 'y ~ x'

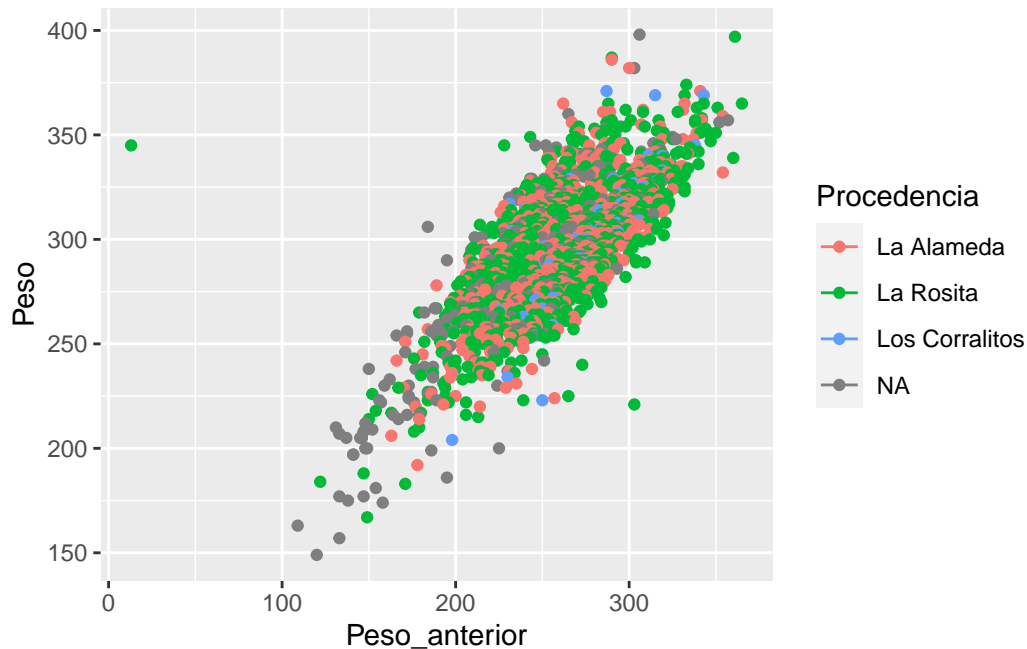
Warning: Removed 32 rows containing non-finite values (`stat_smooth()`).

Warning: Computation failed in `stat_smooth()`

Caused by error in `dyn.load()`:

! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
 libopenblas.so.3: cannot open shared object file: No such file or directory

Warning: Removed 32 rows containing missing values (`geom_point()`).



De este gráfico surge que la relación en todos los casos es positiva pero varía un poco según procedencia.

Dependiendo el tipo de `geom` tenemos distintos atributos estéticos para explorar: `color` y `alpha` (transparencia) para todos, `shape` y `size` para puntos, `linewidth` y `linetype` para líneas, y `fill` para barras, etc. Que tipo de atributo estético depende también de la naturaleza de la variable: continua o discreta. Veamos como queda mapear los valores de `Procedencia` al atributo `shape` (forma):

```
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso, shape = Procedencia) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm')
```

``geom_smooth()` using formula = 'y ~ x'`

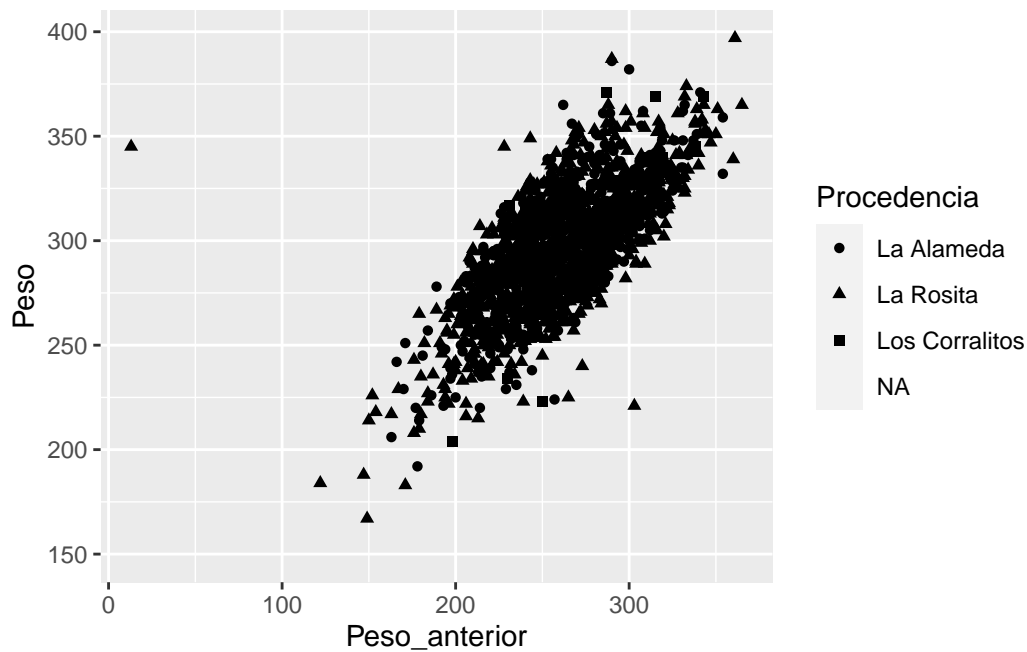
Warning: Removed 32 rows containing non-finite values (``stat_smooth()``).

Warning: Computation failed in ``stat_smooth()``

Caused by error in ``dyn.load()``:

! unable to load shared object `'/usr/lib/R/library/Matrix/libs/Matrix.so'`:
libopenblas.so.3: cannot open shared object file: No such file or directory

Warning: Removed 300 rows containing missing values (``geom_point()``).



Las estéticas se pueden combinar para mostrar mas relaciones entre variables. Por ejemplo, además de `shape = Procedencia` podríamos agregar la información de los pesos iniciales como `color`:

```
ggplot(data = novillos) +  
  aes(x = Peso_anterior, y = Peso, shape = Procedencia,  
      color = Peso_inicial) +  
  geom_point() +  
  geom_line(aes(group = Procedencia), stat = "smooth", method = "lm")
```

``geom_smooth()`` using formula = 'y ~ x'

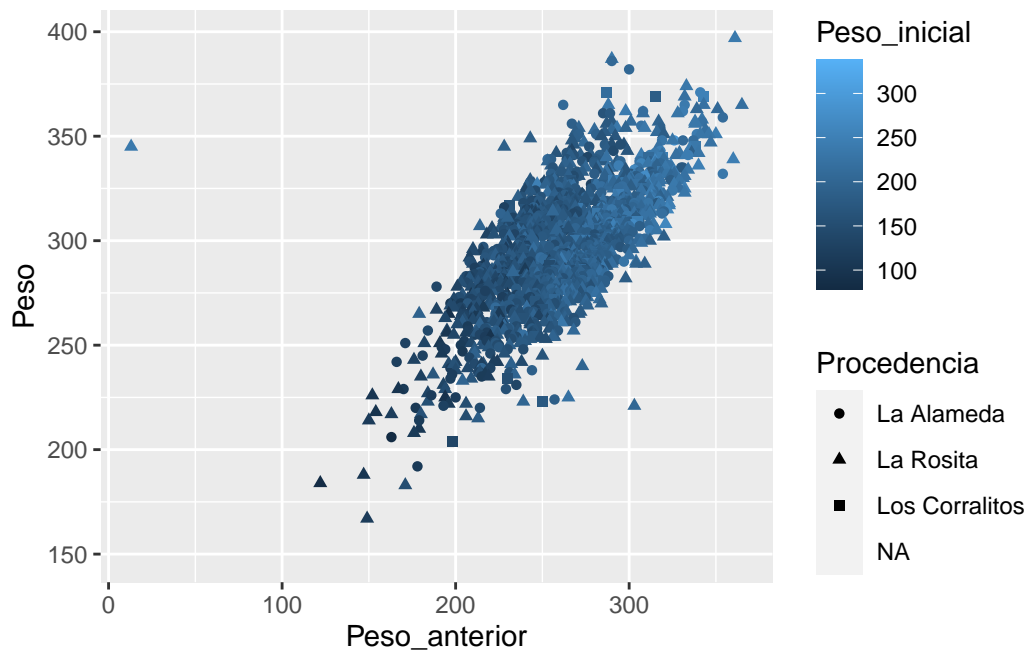
Warning: Removed 32 rows containing non-finite values (``stat_smooth()``).

Warning: Computation failed in ``stat_smooth()``

Caused by error in ``dyn.load()``:

! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
 libopenblas.so.3: cannot open shared object file: No such file or directory

Warning: Removed 300 rows containing missing values (``geom_point()``).



Claramente esto es una exageración pero muestra la potencialidad de `ggplot2`. Siempre tener en cuenta balance entre simplicidad del gráfico y la cantidad de información que queremos comunicar.

Finalmente vamos a ver como mejorar los nombres de los ejes, leyendas y agregar un título. Esto lo hacemos con `labs()`. También agregamos algún tema predefinido como `theme_bw()`.

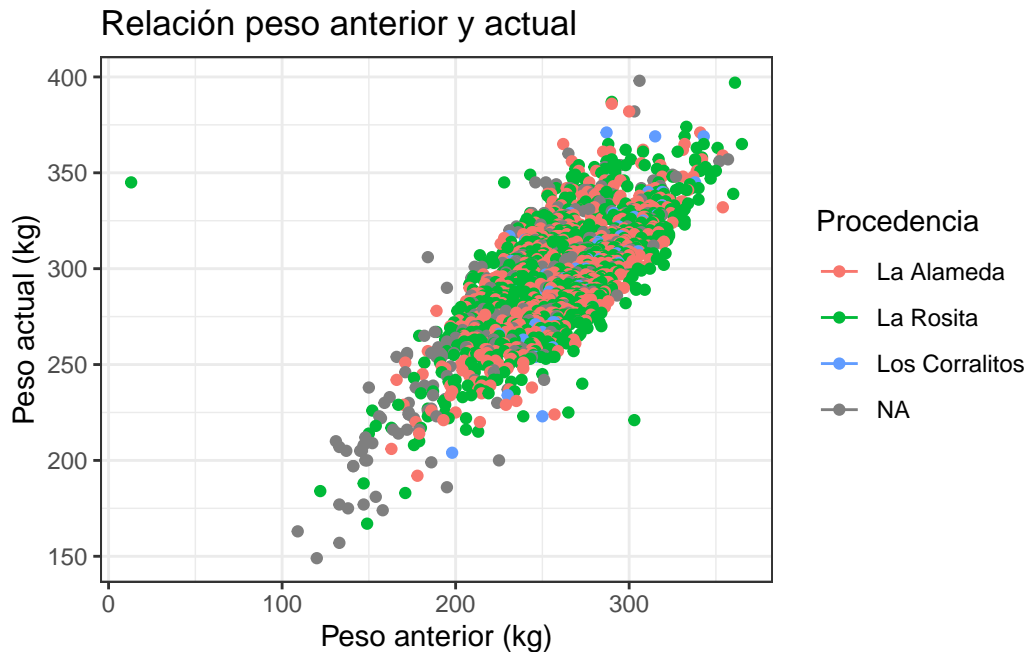
```
ggplot(data = novillos) +  
  aes(x = Peso_anterior, y = Peso, color = Procedencia) +  
  geom_point() +  
  geom_line(stat = "smooth", method = "lm") +  
  labs(x = "Peso anterior (kg)", y = "Peso actual (kg)",  
       color = "Procedencia", title = "Relación peso anterior y actual") +  
  theme_bw()
```

``geom_smooth()`` using formula = 'y ~ x'

Warning: Removed 32 rows containing non-finite values (``stat_smooth()``).

```
Warning: Computation failed in `stat_smooth()`
Caused by error in `dyn.load()`:
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

```
Warning: Removed 32 rows containing missing values (`geom_point()`).
```



7.4 Gráficos condicionales o por paneles: facets

A veces es útil mostrar los subgrupos en gráficos individuales o paneles. Esto se hace fácilmente usando una o más variables condicionales con **facets**. Este tipo de gráficos también se denominan gráficos condicionales ya que muestran la relación de al menos dos variables de interés a través de los niveles de una tercera variable: $y \sim x \mid z$.

Por ejemplo, en el primer gráfico vimos cómo mostrar la relación entre los pesos y las procedencias como color. Eventualmente, ese gráfico podría dividirse en 4 paneles (uno por procedencia) y mostrar en cada uno el subconjunto de puntos.

```
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso) +
  geom_point() +
```

```
geom_line(stat = 'smooth', method = 'lm') +
facet_wrap(~ Procedencia)
```

`geom_smooth()` using formula = 'y ~ x'

Warning: Removed 32 rows containing non-finite values (`stat_smooth()`).

Warning: Computation failed in `stat_smooth()`

Computation failed in `stat_smooth()`

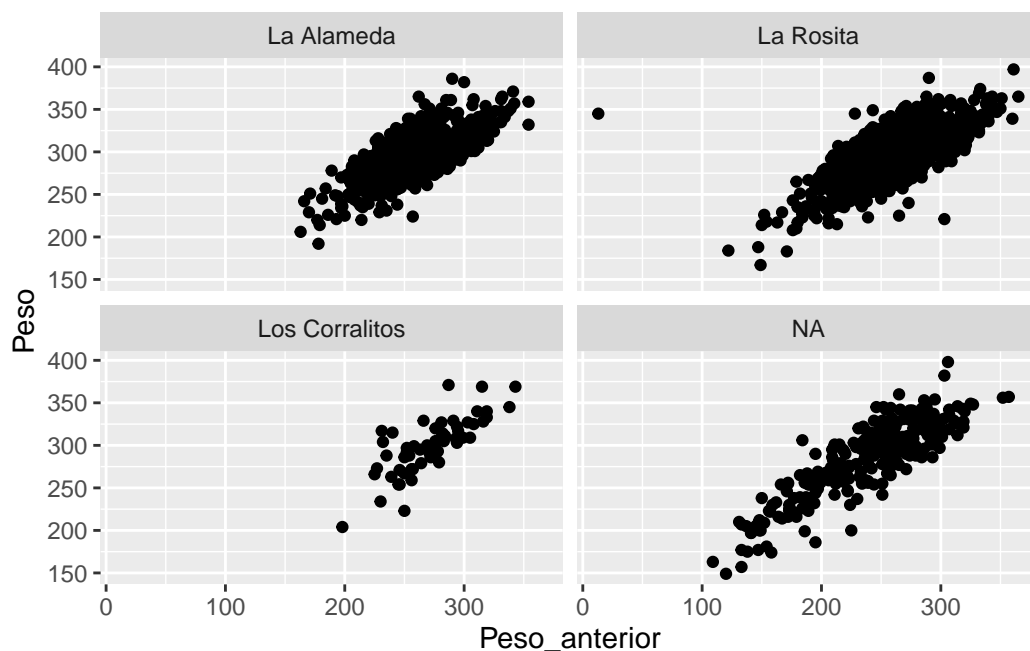
Computation failed in `stat_smooth()`

Computation failed in `stat_smooth()`

Caused by error in `dyn.load()`:

! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
libopenblas.so.3: cannot open shared object file: No such file or directory

Warning: Removed 32 rows containing missing values (`geom_point()`).



Hay dos tipos de facetado: `facet_wrap()` y `facet_grid()`. El primero permite agregar una o mas variables condicionales pero cada subpanel se muestra secuencialmente. Funciona bien

cuando tenemos una sola variable para dividir los subplots o pocos niveles en la combinación. La forma de indicar la variable es `~ variable`.

En cambio, `facet_grid()` permite organizar los subplots en filas y columnas. Las variables se indican en este orden `fila ~ columna`. Para este ejemplo vamos a usar la variable `Tropa`:

```
ggplot(data = novillos) +  
  aes(x = Peso_anterior, y = Peso) +  
  geom_point() +  
  geom_line(stat = 'smooth', method = 'lm') +  
  facet_grid(Tropa ~ Procedencia)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
Warning: Removed 32 rows containing non-finite values (`stat_smooth()`).
```

```
Warning: Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

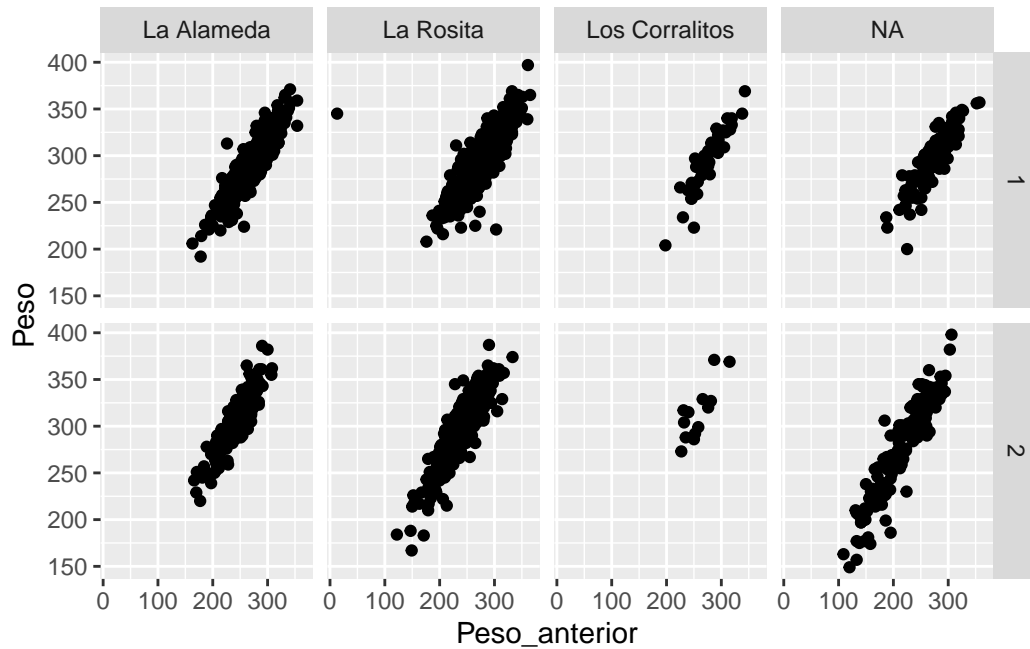
```
Computation failed in `stat_smooth()`
```

```
Caused by error in `dyn.load()`:
```

```
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
```

```
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

```
Warning: Removed 32 rows containing missing values (`geom_point()`).
```



Por defecto los subplots o **facets** tienen escalas iguales en ambos ejes para comparar. A veces conviene dejar una o las dos escalas variar libremente, esto se hace con el argumento **scales** y las palabras clave 'free_y', 'free_x' o 'free' (ambas a la vez).

```
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm') +
  facet_grid(Tropa ~ Procedencia, scales = "free")
```

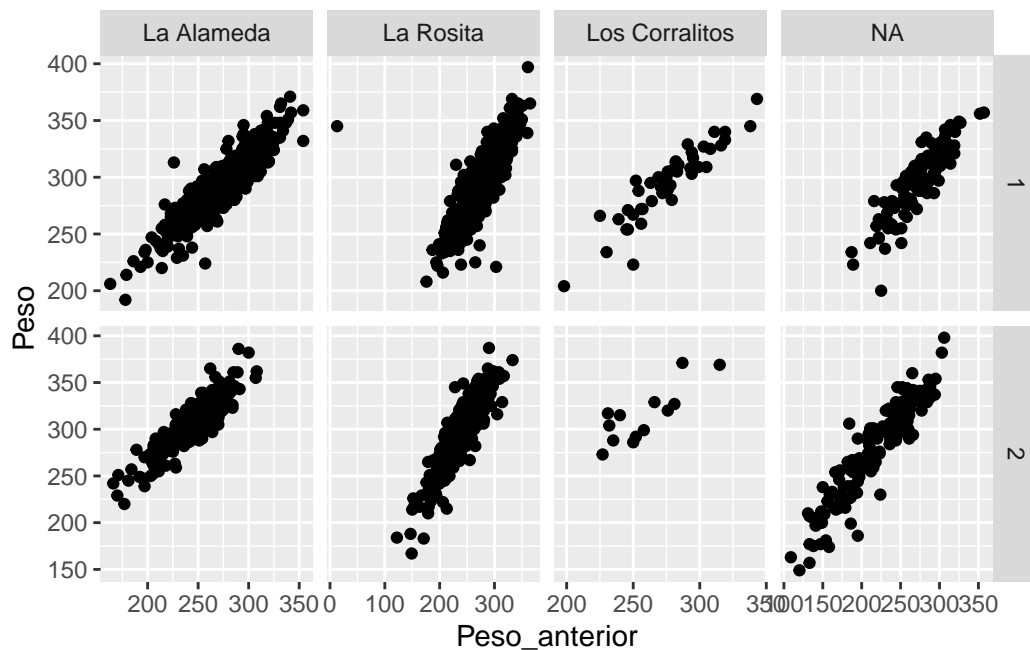
``geom_smooth()` using formula = 'y ~ x'`

Warning: Removed 32 rows containing non-finite values (``stat_smooth()``).

Warning: Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``


```
Computation failed in `stat_smooth()`
Caused by error in `dyn.load()`:
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

Warning: Removed 32 rows containing missing values (`geom_point()`).



Otra aspecto importante en la visualización usando facets es el texto que identifica cada panales. Esto depende de como estan configurados los datos y se controla con el argument `labeller`. Por defecto se toma el valor del factor que se usa para definir los grupos. En algunos casos conviene incluir el nombres de la variable.

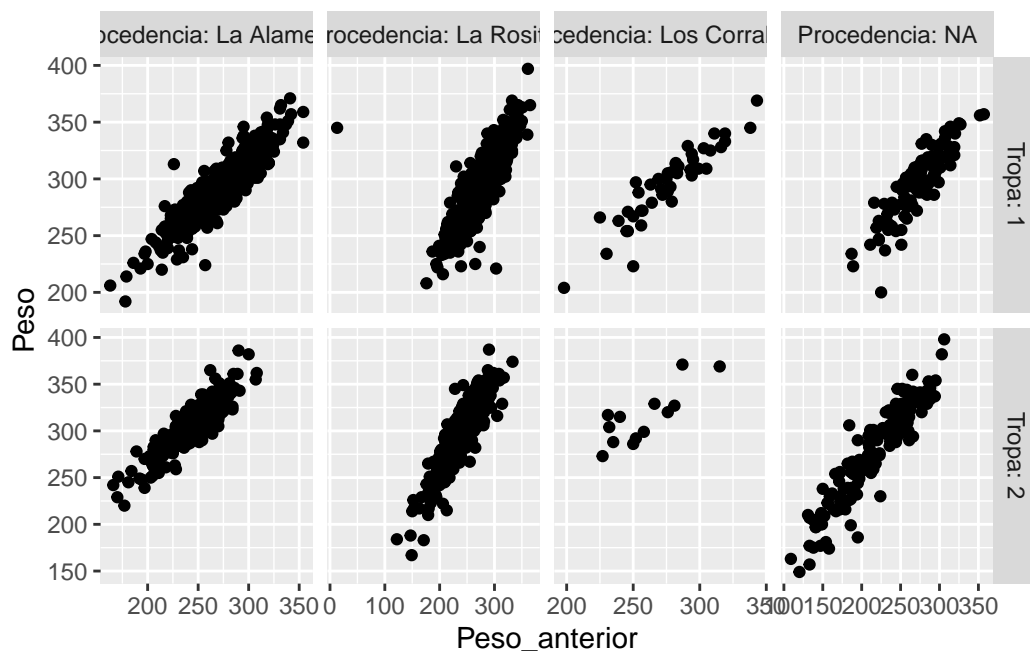
```
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm') +
  facet_grid(Tropa ~ Procedencia, scales = "free", labeller = label_both)
```

``geom_smooth()`` using formula = 'y ~ x'

Warning: Removed 32 rows containing non-finite values (`stat_smooth()`).

```
Warning: Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Caused by error in `dyn.load()`:
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

```
Warning: Removed 32 rows containing missing values (`geom_point()`).
```



7.5 Temas

Los temas en `ggplot` hacen referencia al control fino de la posición, el aspecto, y las formas de los distintos componentes del gráfico. El listado de componentes que se pueden modificar en un tema se incluyen en `?theme()`. Como vemos la lista es larga ya que cada aspecto del gráfico puede controlarse permitiendo crear nuestros propios temas.

ggplot2 algunos temas predefinidos. Por defecto los gráficos utilizan un tema llamado `theme_gray()` que tiene una seleccion de parámetros elegante y que sirve para la mayoría de los casos. Existen otros temas específicos que pueden ser un punto de partida para hacer modificaciones extra.

Por ejemplo, el tema `theme_bw()` remueve el fondo gris pero si queremos quitar la grilla podemos hacer:

```
# Modificar el tema
mi_tema <- theme_bw() + theme(panel.grid = element_blank())

# Aplicar nuestro nuevo tema.
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso, color = Procedencia) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm') +
  mi_tema
```

```
`geom_smooth()` using formula = 'y ~ x'
```

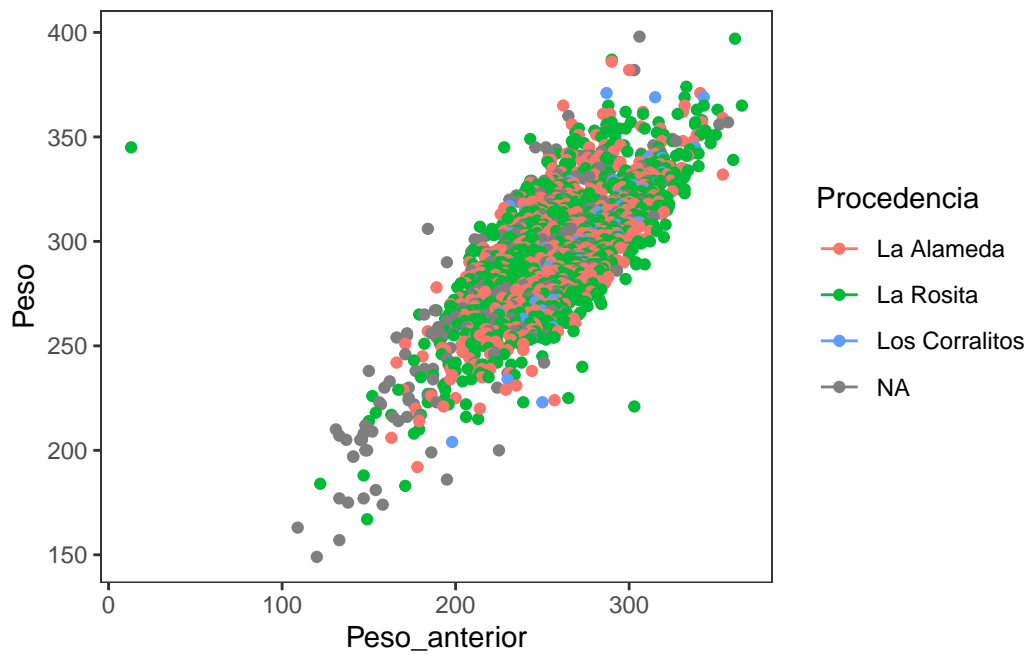
```
Warning: Removed 32 rows containing non-finite values (`stat_smooth()`).
```

```
Warning: Computation failed in `stat_smooth()`
```

```
Caused by error in `dyn.load()`:
```

```
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

```
Warning: Removed 32 rows containing missing values (`geom_point()`).
```



8 Reportes dinámicos

R tiene por defecto el paquete `graphics`, también conocido como *base plot system*, que provee la función genérica `plot()` para hacer gráficos simples, y otras funciones para gráficos específicos (`hist()`, `barplot()`, `boxplot()`, etc).

Usa un enfoque de *papel y lápiz* por capas donde el gráfico final es una sumatoria de capas que se agregan una a la vez sin posibilidad de modificarse luego. Generalmente es OK para gráficos simples o exploratorios. Para gráficos más complejos (con subgrupos o multipanel) requiere programar más. Una desventaja es la sintaxis poco consistente.

El paquete `ggplot2`, desarrollado por [Hadley Wickham](#), está basado en la filosofía *Gramática de gráficos* (*grammar of graphics* , por eso `gg`). Combina los dos enfoques: *por capas* y *función*.

Uno provee los datos, indica que variables asignar a las estéticas (ejes, escalas, colores, símbolos) y las geometrías o formas que se quieren graficar y `ggplot2` se encarga del resto. Se puede ir agregando capas. Es muy potente para la exploración y visualización de datos en formato de tabla con filas (observaciones) y columnas (variables).

8.1 ¿Cómo conseguir ggplot2?

Para acceder a estos paquetes instalarlos/cargarlos individualmente:

```
pacman::p_load(ggplot2)
```

O bien con otros paquete de la familia `tidyverse`

```
pacman::p_load(tidyverse)
```

R va a avisarnos en la consola que esta enmascarando (reemplazando) algunas funciones que ya estaban en el entorno, o bien el paquete nos devuelve algun mensaje. A menos que diga **Error** ..., eso está bien.

8.2 Componentes del gráfico en ggplot2

ggplot2 implementa una variante *por capas* de este paradigma gramática de gráficos de [Leland Wilkinson](#) (gg es por *grammar of graphics*). Como resultado, se crean una serie de capas que permiten describir y construir visualizaciones de manera estructurada en cuanto a representación de los elementos pero a su vez flexible para generar combinaciones nuevas.

Un gráfico se define por la combinación de capas (**layers**), escalas (**scales**), coordenadas (**coords**) y facetas (**facets**). Adicionalmente a estos componentes se pueden aplicar temas (**themes**) que permiten controlar los detalles del diseño de la visualización.

8.2.1 Layers

Los **layer** se construyen con las funciones **geom_*** y **stat_*** que veremos más adelante. Constan de 5 elementos:

- **data**, set de datos (**data.frame** o similar) que contiene la información que se desea visualizar.
- **mapping**, elementos de mapeo definidos mediante **aes()** para indicar la forma en que la las variables y observaciones van a ser representadas en la visualización mediante atributos estéticos (ejes, líneas, colores, rellenos, etc).
- **stat**, funciones estadísticas que resumen los datos aplicando funciones estadísticas, e.g. promedio, agrupamiento y conteo de observaciones, o ajuste de un modelo lineal o suavizado, etc.
- **geom**, geometrías o formas que representan lo que realmente se ve en un gráfico: puntos, líneas, polígonos, etc.
- **position**, ajuste de posición de los elementos **geoms** dentro de un layer para evitar su superposición.

Generalmente, sobre todo para gráficos simples, **data** y **mapping** se definen una vez para todo el gráfico dentro de la función **ggplot()**. En otras situaciones se hace a nivel de cada **layer**.

8.2.2 Scales

Asignan los valores del espacio de datos a valores en el espacio de los elementos estéticos (**aesthetics** o **aes**). Por ejemplo, el uso de un color, forma o tamaño de en un **geom** puede ser controlado por un atributo de los datos. Las escalas también definen las leyenda y los ejes.

8.2.3 Coordenadas

Sistema de coordenadas (`coord`) que define que variables definiran el espacio del gráfico y como se representarán, e.g. coordenadas cartesianas, polares, etc.

8.2.4 Paneles (facets)

Es un elemento que permite especificar una o más variables para dividir el gráfico en paneles y así mostrar subgrupos de datos. Esto permite ver visualizar relaciones condicionales entre variables, e.g. $y \sim x \mid z$, es decir, que pasa con la variable x e y cuando cambia z .

8.2.5 Temas

Adicionalmente a estos componentes se pueden aplicar temas (`themes`) que permiten controlar los detalles del diseño de la visualización, tipografía, posición de algunos objetos, paleta de colores, etc. Los valores predeterminados de `ggplot2` son un buen punto de partida pero existen opciones predefinidas que pueden modificarse para generar un tema particular. Otra fuente para consultar es el trabajo de [Tufte](#)

8.3 Primer gráfico paso a paso

```
pacman::p_load(rio)
novillos <- import("datasets/pesada_novillos.xlsx", setclass = "tibble")
novillos
```

```
# A tibble: 1,842 x 13
```

	IDV	Tropa	Procedencia	Fecha_Ingreso	Peso_inicial	Peso_anterior	GPV_anterior
	<chr>	<dbl>	<chr>	<chr>	<dbl>	<dbl>	<dbl>
1	GH73~	2	<NA>	42939	188	306	51
2	IA67~	2	La Rosita	42940	232	290	2
3	P015~	2	La Alameda	42939	204	290	33
4	GH73~	2	<NA>	42940	204	303	27
5	SZ20~	2	La Alameda	42940	202	300	30
6	IA67~	2	La Rosita	42940	234	333	38
7	P015~	2	Los Corral~	42938	214	287	14
8	P015~	2	Los Corral~	42939	184	315	37
9	NS50~	2	La Rosita	42938	234	288	0
10	QX67~	2	La Alameda	42940	206	262	2

```
# i 1,832 more rows
```

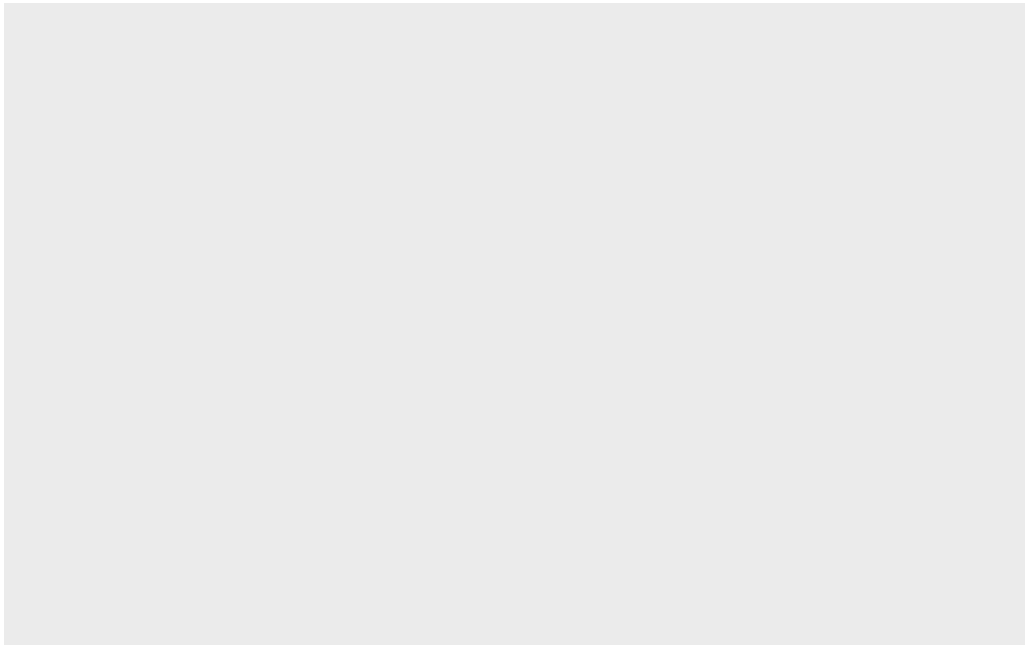
```
# i 6 more variables: GDM_anterior <dbl>, Fecha <chr>, Hora <chr>, Peso <dbl>,  
#   Días <dbl>, Días_total <dbl>
```

Veamos con un ejemplo como se combinan los componentes anteriormente vistos para realizar un gráfico simple. Para esto vamos a usar el set de datos

Nuestro primer gráfico tendrá como objetivo mostrar la relación que existe entre `Peso_anterior` y `Peso` (actual), y potencialmente ver si ésta es similar entre procedencias. Veamos paso por paso como se construye el gráfico.

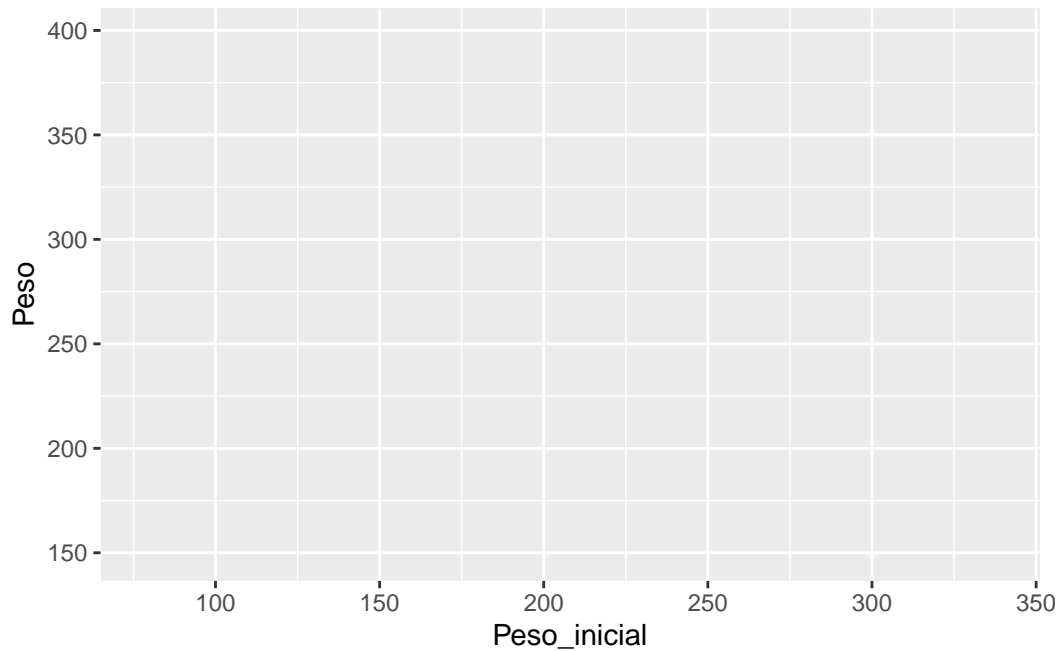
Primero definimos el set de datos que usaremos:

```
ggplot(data = novillos)
```



Como vemos esto no produjo nada ya que no indicamos cuales son las variables que queremos graficar y cómo graficarlas. Nuestro `layer` solo tiene la información de `data`. Agreguemos ahora la información de `mapping` usando `aes()`. Usando el operador `+` podemos concatenarlo al comando anterior.

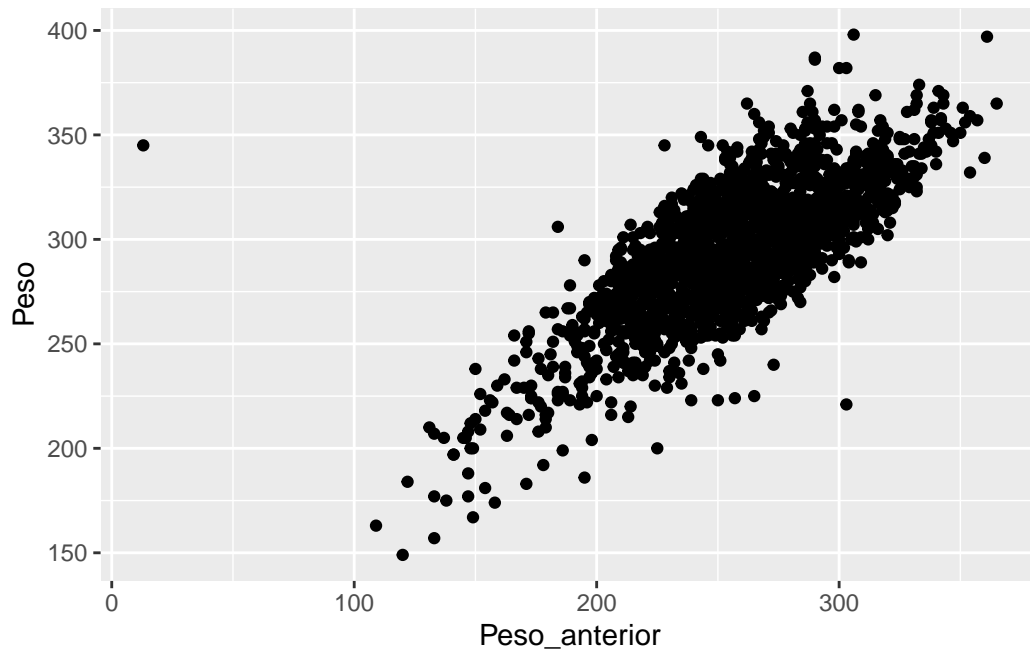
```
ggplot(data = novillos) +  
  aes(x = Peso_inicial, y = Peso)
```

Aquí vemos que, si bien no hemos graficado nada, la información suministrada permite a `ggplot` identificar los ejes, definir el espacio de coordenadas (cartesianas por defecto) y proponer unos límites en función del rango de valores. Agreguemos ahora la geometría: en este caso tiene sentido usar `geom_point()` ya que queremos mostrar un punto por observación

```
ggplot(data = novillos) +  
  aes(x = Peso_anterior, y = Peso) +  
  geom_point()
```

Warning: Removed 32 rows containing missing values (``geom_point()``).



Como vemos ahora el gráfico va tomando forma. Este tipo de gráficos se llama *gráfico de dispersión* y muestra la relación entre ambas variables. Por defecto no se aplica ninguna transformación estadística lo que equivale a (`stat = "identity"`).

A este gráfico vamos a agregarle alguna función que permita resumir la relación entre ambas variables, por ejemplo un modelo de regresión. La mejor forma de representarlo sería una línea. Para eso vamos a agregar otro `layer` con `geom_line()` donde indicaremos una transformación de los datos `stat = smooth`.

```
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm')
```

``geom_smooth()` using formula = 'y ~ x'`

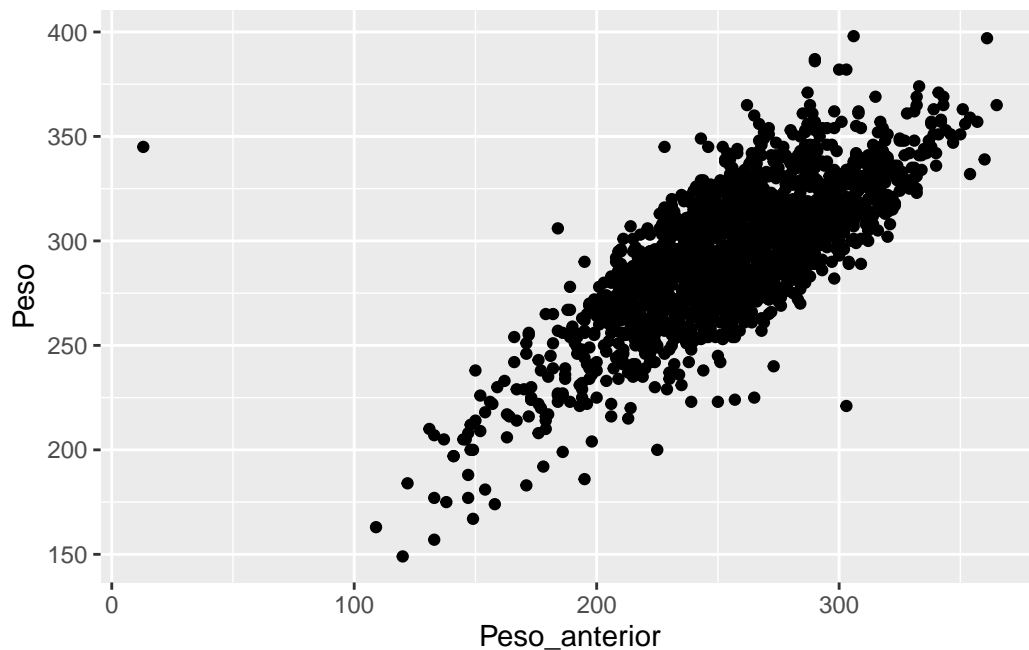
Warning: Removed 32 rows containing non-finite values (``stat_smooth()``).

Warning: Computation failed in ``stat_smooth()``

Caused by error in ``dyn.load()``:

```
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

Warning: Removed 32 rows containing missing values (`geom_point()`).



Hay una relación positiva para todo el set de datos pero puede enmascarar algún patrón por Procedencia. Esta información la podemos agregar con otros atributos estéticos como por ejemplo color:

```
ggplot(data = novillos) +  
  aes(x = Peso_anterior, y = Peso, color = Procedencia) +  
  geom_point() +  
  geom_line(stat = 'smooth', method = 'lm')
```

`geom_smooth()` using formula = 'y ~ x'

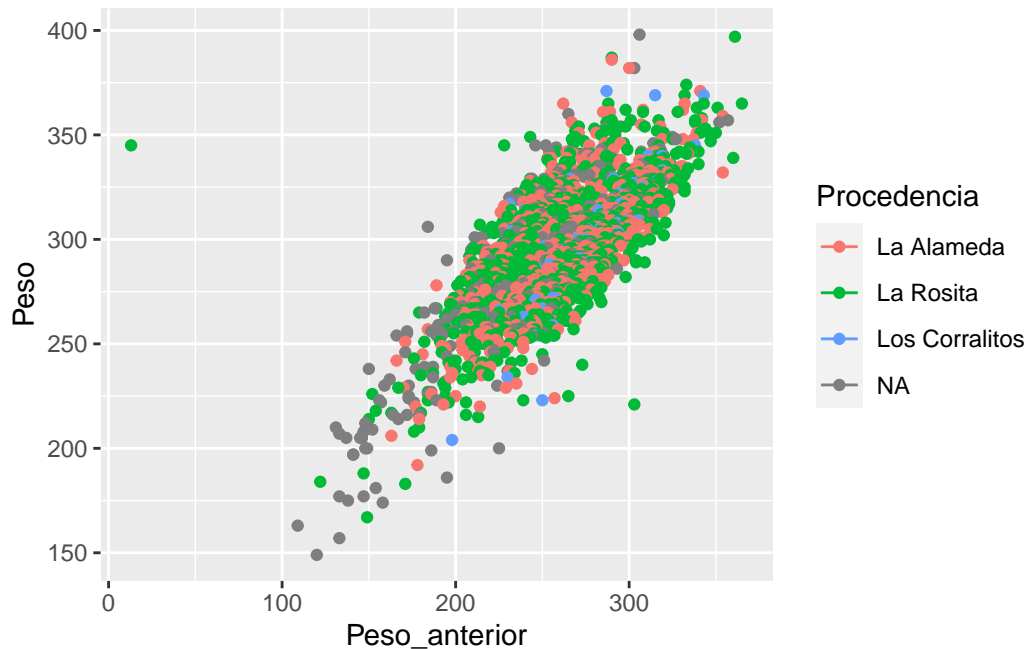
Warning: Removed 32 rows containing non-finite values (`stat_smooth()`).

Warning: Computation failed in `stat_smooth()`

Caused by error in `dyn.load()`:

! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
 libopenblas.so.3: cannot open shared object file: No such file or directory

Warning: Removed 32 rows containing missing values (`geom_point()`).



De este gráfico surge que la relación en todos los casos es positiva pero varia un poco según procedencia.

Dependiendo el tipo de `geom` tenemos distintos atributos estéticos para explorar: `color` y `alpha` (transparencia) para todos, `shape` y `size` para puntos, `linewidth` y `linetype` para líneas, y `fill` para barras, etc. Que tipo de atributo estético depende también de la naturaleza de la variable: continua o discreta. Veamos como queda mapear los valores de `Procedencia` al atributo `shape` (forma):

```
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso, shape = Procedencia) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm')
```

``geom_smooth()` using formula = 'y ~ x'`

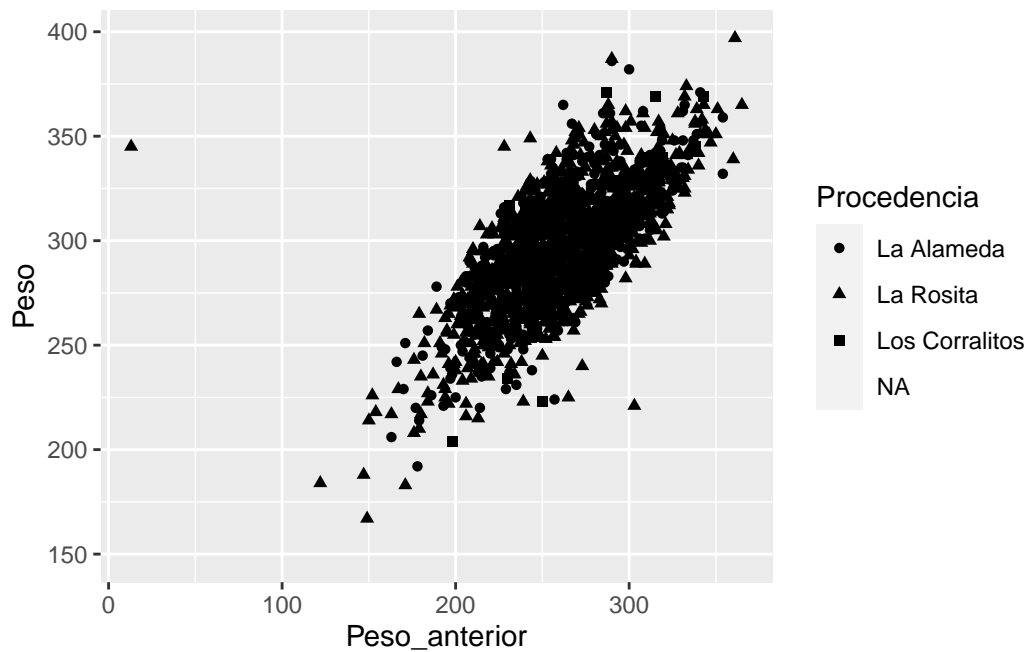
Warning: Removed 32 rows containing non-finite values (``stat_smooth()``).

Warning: Computation failed in ``stat_smooth()``

Caused by error in ``dyn.load()``:

! unable to load shared object `'/usr/lib/R/library/Matrix/libs/Matrix.so'`:
libopenblas.so.3: cannot open shared object file: No such file or directory

Warning: Removed 300 rows containing missing values (``geom_point()``).



Las estéticas se pueden combinar para mostrar mas relaciones entre variables. Por ejemplo, además de `shape = Procedencia` podríamos agregar la información de los pesos iniciales como `color`:

```
ggplot(data = novillos) +  
  aes(x = Peso_anterior, y = Peso, shape = Procedencia,  
      color = Peso_inicial) +  
  geom_point() +  
  geom_line(aes(group = Procedencia), stat = "smooth", method = "lm")
```

``geom_smooth()`` using formula = 'y ~ x'

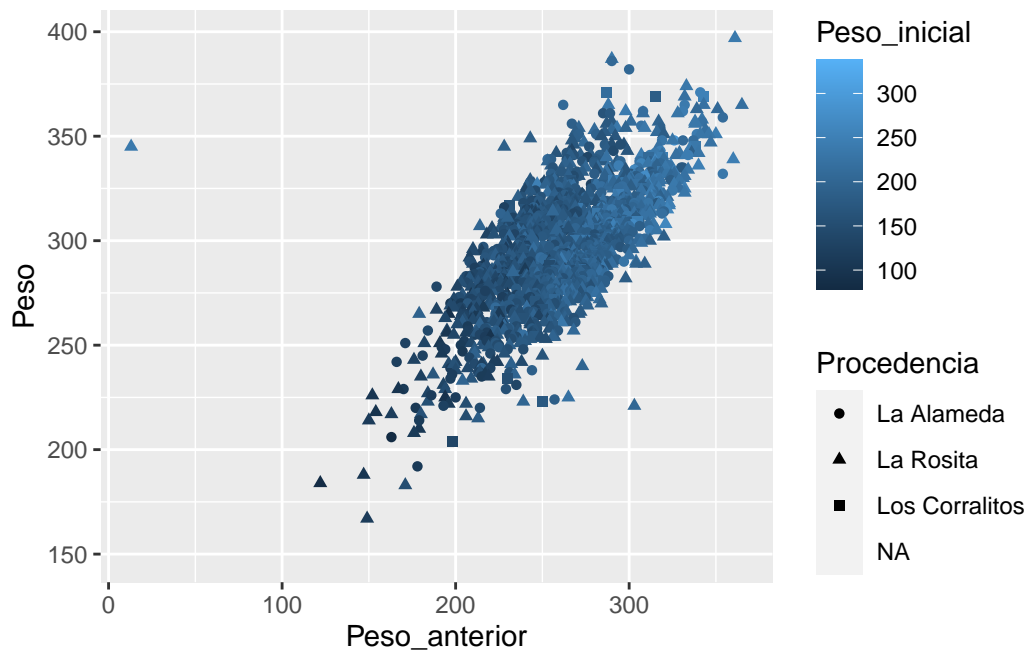
Warning: Removed 32 rows containing non-finite values (``stat_smooth()``).

Warning: Computation failed in ``stat_smooth()``

Caused by error in ``dyn.load()``:

```
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':  
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

Warning: Removed 300 rows containing missing values (``geom_point()``).



Claramente esto es una exageración pero muestra la potencialidad de `ggplot2`. Siempre tener en cuenta balance entre simplicidad del gráfico y la cantidad de información que queremos comunicar.

Finalmente vamos a ver como mejorar los nombres de los ejes, leyendas y agregar un título. Esto lo hacemos con `labs()`. También agregamos algún tema predefinido o como `theme_bw()`.

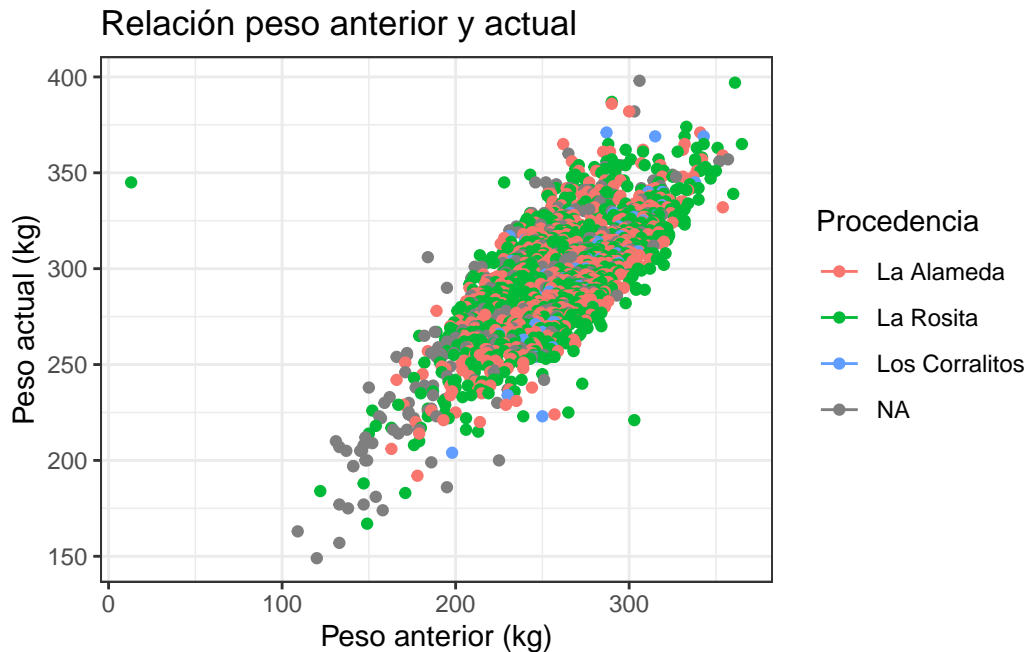
```
ggplot(data = novillos) +  
  aes(x = Peso_anterior, y = Peso, color = Procedencia) +  
  geom_point() +  
  geom_line(stat = "smooth", method = "lm") +  
  labs(x = "Peso anterior (kg)", y = "Peso actual (kg)",  
       color = "Procedencia", title = "Relación peso anterior y actual") +  
  theme_bw()
```

``geom_smooth()`` using formula = 'y ~ x'

Warning: Removed 32 rows containing non-finite values (``stat_smooth()``).

```
Warning: Computation failed in `stat_smooth()`
Caused by error in `dyn.load()`:
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

```
Warning: Removed 32 rows containing missing values (`geom_point()`).
```



8.4 Gráficos condicionales o por paneles: facets

A veces es útil mostrar los subgrupos en gráficos individuales o paneles. Esto se hace fácilmente usando una o más variables condicionales con **facets**. Este tipo de gráficos también se denominan gráficos condicionales ya que muestran la relación de al menos dos variables de interés a través de los niveles de una tercera variable: $y \sim x \mid z$.

Por ejemplo, en el primer gráfico vimos cómo mostrar la relación entre los pesos y las procedencias como color. Eventualmente, ese gráfico podría dividirse en 4 paneles (uno por procedencia) y mostrar en cada uno el subconjunto de puntos.

```
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso) +
  geom_point() +
```

```
geom_line(stat = 'smooth', method = 'lm') +  
facet_wrap(~ Procedencia)
```

`geom_smooth()` using formula = 'y ~ x'

Warning: Removed 32 rows containing non-finite values (`stat_smooth()`).

Warning: Computation failed in `stat_smooth()`

Computation failed in `stat_smooth()`

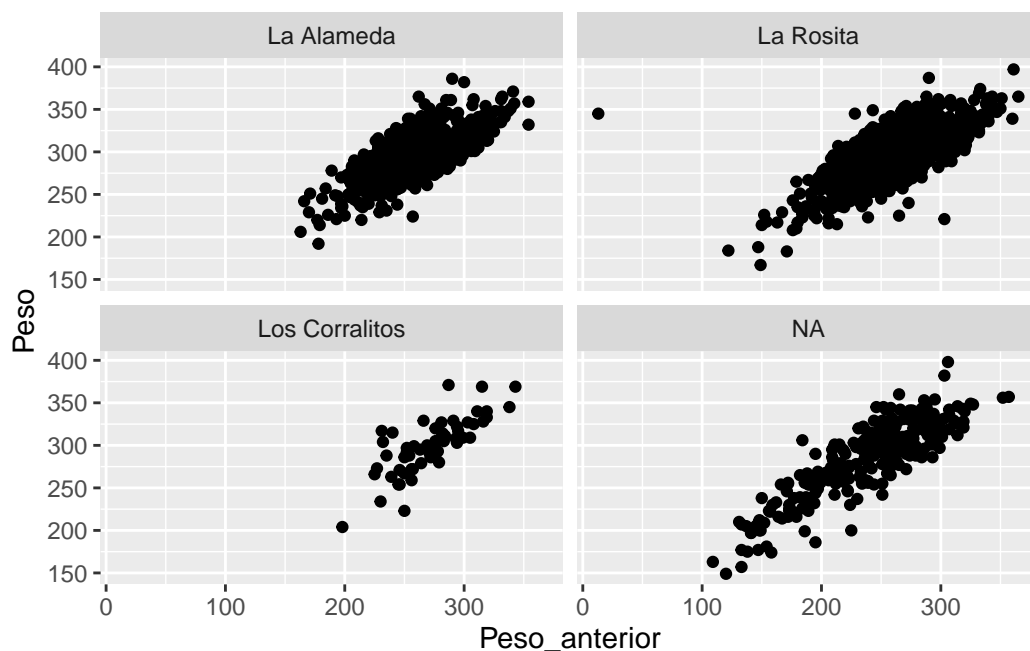
Computation failed in `stat_smooth()`

Computation failed in `stat_smooth()`

Caused by error in `dyn.load()`:

! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
libopenblas.so.3: cannot open shared object file: No such file or directory

Warning: Removed 32 rows containing missing values (`geom_point()`).



Hay dos tipos de facetado: `facet_wrap()` y `facet_grid()`. El primero permite agregar una o mas variables condicionales pero cada subpanel se muestra secuencialmente. Funciona bien

cuando tenemos una sola variable para dividir los subplots o pocos niveles en la combinación. La forma de indicar la variable es `~ variable`.

En cambio, `facet_grid()` permite organizar los subplots en filas y columnas. Las variables se indican en este orden `fila ~ columna`. Para este ejemplo vamos a usar la variable `Tropa`:

```
ggplot(data = novillos) +  
  aes(x = Peso_anterior, y = Peso) +  
  geom_point() +  
  geom_line(stat = 'smooth', method = 'lm') +  
  facet_grid(Tropa ~ Procedencia)
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
Warning: Removed 32 rows containing non-finite values (`stat_smooth()`).
```

```
Warning: Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

```
Computation failed in `stat_smooth()`
```

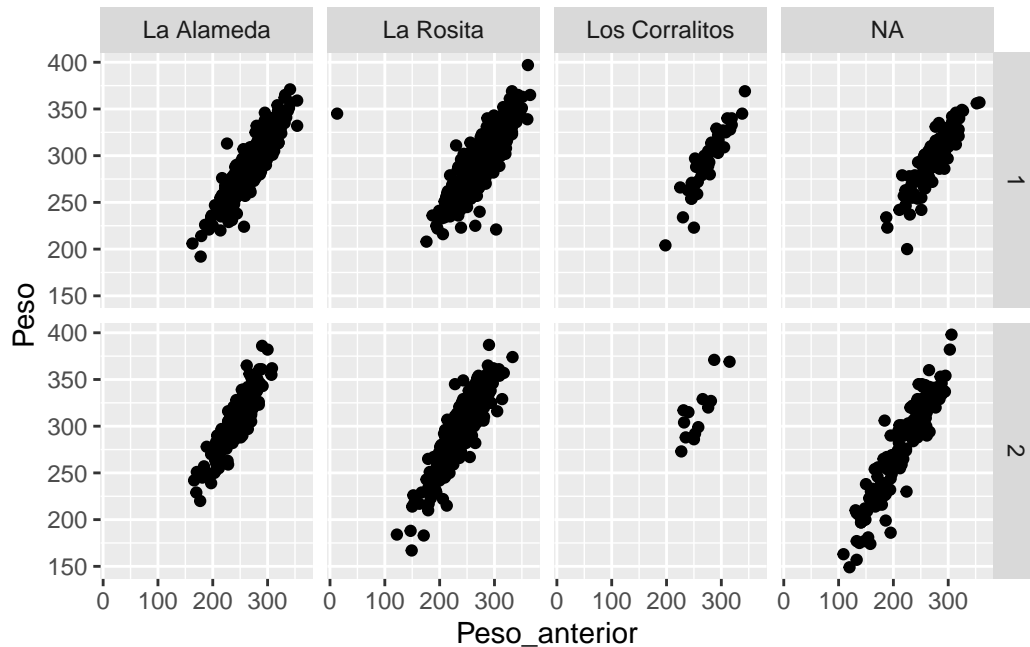
```
Computation failed in `stat_smooth()`
```

```
Caused by error in `dyn.load()`:
```

```
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
```

```
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

```
Warning: Removed 32 rows containing missing values (`geom_point()`).
```



Por defecto los subplots o **facets** tienen escalas iguales en ambos ejes para comparar. A veces conviene dejar una o las dos escalas variar libremente, esto se hace con el argumento **scales** y las palabras clave 'free_y', 'free_x' o 'free' (ambas a la vez).

```
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm') +
  facet_grid(Tropa ~ Procedencia, scales = "free")
```

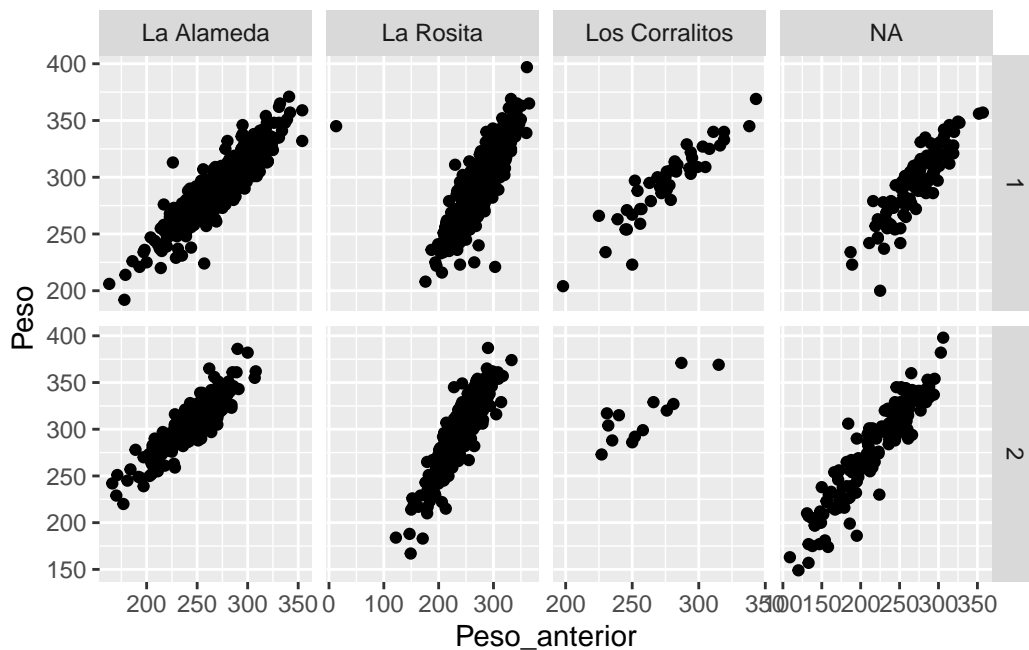
``geom_smooth()`` using formula = 'y ~ x'

Warning: Removed 32 rows containing non-finite values (``stat_smooth()``).

Warning: Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``
 Computation failed in ``stat_smooth()``

```
Computation failed in `stat_smooth()`
Caused by error in `dyn.load()`:
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

Warning: Removed 32 rows containing missing values (`geom_point()`).



Otra aspecto importante en la visualización usando facets es el texto que identifica cada panales. Esto depende de como estan configurados los datos y se controla con el argument `labeller`. Por defecto se toma el valor del factor que se usa para definir los grupos. En algunos casos conviene incluir el nombres de la variable.

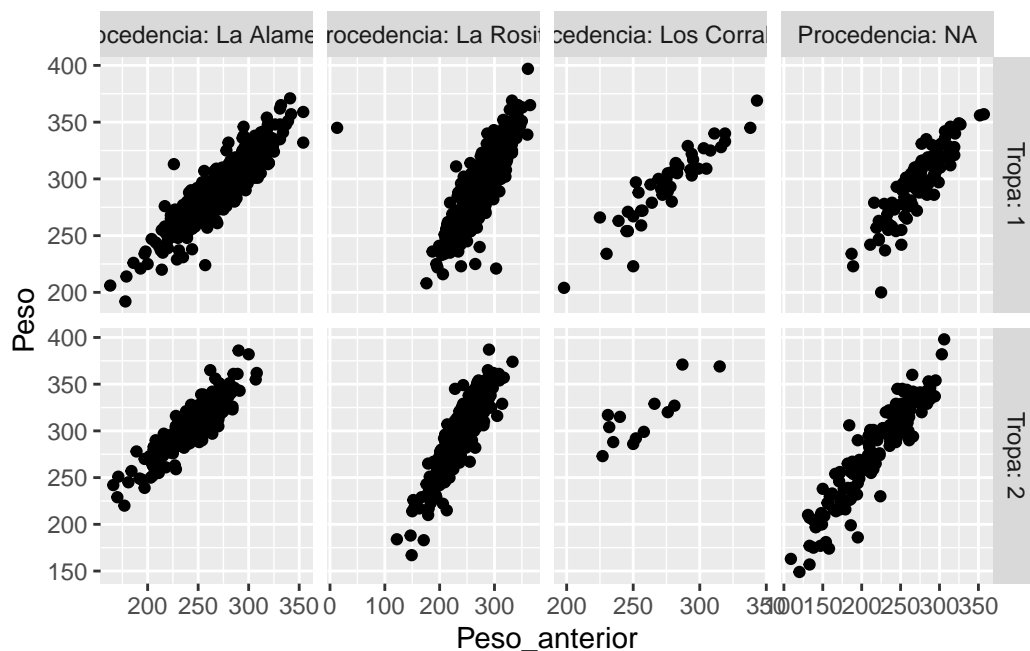
```
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm') +
  facet_grid(Tropa ~ Procedencia, scales = "free", labeller = label_both)
```

``geom_smooth()`` using formula = 'y ~ x'

Warning: Removed 32 rows containing non-finite values (`stat_smooth()`).

```
Warning: Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Computation failed in `stat_smooth()`
Caused by error in `dyn.load()`:
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

Warning: Removed 32 rows containing missing values (`geom_point()`).



8.5 Temas

Los temas en `ggplot` hacen referencia al control fino de la posición, el aspecto, y las formas de los distintos componentes del gráfico. El listado de componentes que se pueden modificar en un tema se incluyen en `?theme()`. Como vemos la lista es larga ya que cada aspecto del gráfico puede controlarse permitiendo crear nuestros propios temas.

ggplot2 algunos temas predefinidos. Por defecto los gráficos utilizan un tema llamado `theme_gray()` que tiene una seleccion de parámetros elegante y que sirve para la mayoría de los casos. Existen otros temas específicos que pueden ser un punto de partida para hacer modificaciones extra.

Por ejemplo, el tema `theme_bw()` remueve el fondo gris pero si queremos quitar la grilla podemos hacer:

```
# Modificar el tema
mi_tema <- theme_bw() + theme(panel.grid = element_blank())

# Aplicar nuestro nuevo tema.
ggplot(data = novillos) +
  aes(x = Peso_anterior, y = Peso, color = Procedencia) +
  geom_point() +
  geom_line(stat = 'smooth', method = 'lm') +
  mi_tema
```

```
`geom_smooth()` using formula = 'y ~ x'
```

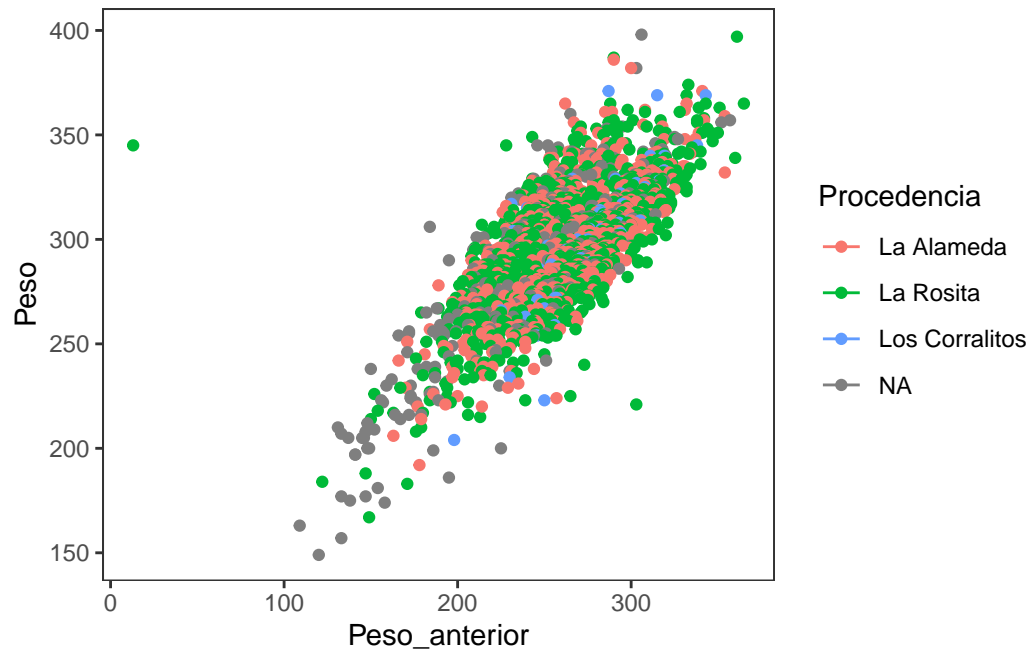
```
Warning: Removed 32 rows containing non-finite values (`stat_smooth()`).
```

```
Warning: Computation failed in `stat_smooth()`
```

```
Caused by error in `dyn.load()`:
```

```
! unable to load shared object '/usr/lib/R/library/Matrix/libs/Matrix.so':
  libopenblas.so.3: cannot open shared object file: No such file or directory
```

```
Warning: Removed 32 rows containing missing values (`geom_point()`).
```



References

R Core Team. 2023. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.