

Visualización de datos en R

Carlos de Oro Aguado

2025-10-14

Contents

Asignatura	5
Justificación	5
Objetivo general de la asignatura	6
Resultados de aprendizaje (Objetivos específicos)	6
Temas de la asignatura	6
1 Introducción	9
1.1 Introducción a R y RStudio	9
1.2 Introducción a Git y GitHub	13
1.3 Pasos para publicar un libro bookdown en GitHub Pages	20
2 ETL con Tidyverse	23
2.1 Data Frames	23
2.2 Importación de datos	23
2.3 Transformación de datos	33
2.4 Ordenación de datos	42
3 Exploración y visualización estática de datos	53
3.1 Visualización de variables categóricas con gráficos de barras	54
3.2 Visualización de variable continua con un histograma	58
3.3 Visualización de la covariación con gráficos de caja	60
3.4 Visualización de la covariación con el gráfico de burbuja	60
3.5 Visualización 2D de contenedores y hexágonos	63
3.6 Visualización de estadísticas (resumen de los datos)	64

3.7	Visualización de un gráfico de correlación	65
3.8	Visualización de un gráfico de dispersión	67
3.9	Visualización del gráfico de dispersión con linea de tendencia . .	69
3.10	Visualización de trazado de categorías	72
3.11	Visualizacion etiquetando nombres	73
3.12	Visualización de leyendas	80
3.13	Visualiazción con facetas	82
3.14	Visualización de tramas de violín	83
3.15	Visualización de gráficos de densidad	87
4	Visualización de datos geográficos con ggmap	93
4.1	Paquete <code>ggmap</code>	93
4.2	Construcción de un mapa base	95
4.3	Integración de capas de información	103
4.4	Integración de capas vectoriales desde Shapefiles	111
5	Análisis con datos faltantes y detección de atípicos	125
5.1	Manejo de datos faltantes	125
5.2	Manejo de datos atípicos	126
5.3	Visualización de los datos faltantes	127
5.4	Eliminación de datos faltantes	136
5.5	Imputación de datos con el promedio o mediana	144
5.6	Imputación de datos con el paquete <code>mice</code>	150
5.7	Imputación de datos con modelos de regresión	156
5.8	Detección de datos atípicos	159
5.9	Tratamiento de datos atípicos	175

Asignatura

En esta asignatura se presentan los principales métodos y procedimientos para la lectura, limpieza y procesamiento de datos utilizando los programas **R** y **Python**. Como aplicación práctica, se abordará la construcción de un modelo de series de tiempo orientado a la visualización de predicciones para la toma de decisiones.

Adicionalmente, se explorarán las herramientas más recientes para la visualización de información procedente de bases de datos, con ejemplos aplicados a escenarios reales. Finalmente, se revisarán alternativas para la automatización de reportes, la generación de informes estadísticos y el diseño de Dashboards, con el fin de ofrecer soluciones efectivas en entornos de análisis de datos.

Justificación

El incremento en la recolección de datos relevantes en diferentes contextos sociales, empresariales y gubernamentales ha generado la necesidad de implementar procesos y metodologías estadísticas que permitan transformar estos datos en información útil para la toma de decisiones estratégicas.

El desarrollo de herramientas computacionales como R y Python, de libre acceso y con gran acogida en la comunidad científica y empresarial, permite abordar de manera eficiente el procesamiento y análisis de bases de datos de diversos volúmenes y estructuras.

Complementariamente, el uso de SQL y gestores de bases de datos como PostgreSQL resulta fundamental para el manejo de grandes volúmenes de información. La combinación de estas herramientas con entornos de visualización interactiva como Dash y Shiny, junto con el despliegue de aplicaciones mediante tecnologías basadas en Dockers, garantiza la construcción de informes estadísticos y predictivos más intuitivos, dinámicos y de fácil interpretación.

Objetivo general de la asignatura

Conocer y aplicar las herramientas disponibles en R y Python para el procesamiento, limpieza, análisis y visualización de datos, así como para el diseño de Dashboards estadísticos y predictivos. Además, implementar el manejo de bases de datos con SQL y PostgreSQL, y desplegar aplicaciones interactivas mediante herramientas de libre acceso basadas en Dockers.

Resultados de aprendizaje (Objetivos específicos)

Al finalizar la asignatura, el estudiante estará en capacidad de:

- Utilizar adecuadamente las herramientas básicas de R y Python para la lectura, procesamiento y limpieza de bases de datos.
- Comprender y aplicar diferentes tipos de gráficos disponibles en ambos programas.
- Implementar procedimientos eficientes para el manejo de bases de datos con R y Python.
- Automatizar la elaboración de reportes e informes estadísticos en R.
- Comprender la teoría estadística detrás de los modelos predictivos e implementarlos de manera eficiente en problemas de toma de decisiones.
- Diseñar y desplegar Dashboards utilizando Dash y Shiny, junto con bases de datos PostgreSQL.

Temas de la asignatura

1. Introducción a la visualización de datos con R y Python
 - Lectura de diferentes tipos de bases de datos
 - Técnicas exploratorias de datos
 - Exportación de bases de datos en distintos formatos
2. Visualización estática y dinámica con R y Python
 - Creación de gráficos descriptivos
 - Gráficos de dispersión y mapas de calor
 - Herramientas interactivas (hover, tooltip, zoom)
3. Visualización interactiva de datos geográficos
 - Mapas coropléticos

- Interacciones estáticas y dinámicas
 - Animaciones en mapas
 - Gráficos de dispersión y líneas geográficas
4. Solución de errores comunes
 - Formato e interpretación de datos
 - Visualización y tratamiento de datos atípicos y faltantes
 5. Importación y manejo de bases de datos SQL
 - Introducción a PostgreSQL
 - Creación de bases de datos
 - Consultas SQL con Python API
 6. Introducción a Dash y Shiny
 - Configuración del ambiente
 - Diseño con componentes HTML y Markdown
 - Uso de componentes reutilizables
 7. Callbacks en Dash y Shiny
 - App layout con figuras y sliders
 - Múltiples entradas y salidas
 - Callbacks encadenados y dinámicos
 8. Ejemplos de Dashboards
 - Dashboard para mapas
 - Dashboard financiero
 - Despliegue de aplicaciones

Chapter 1

Introducción

1.1 Introducción a R y RStudio

El primer paso para comenzar a trabajar con **R**, un lenguaje de programación especializado en estadística, ciencia de datos y visualización, es instalarlo en tu computadora. **R** es compatible con los principales sistemas operativos, incluyendo **Windows**, **macOS** y **Linux**.

1.1.1 ¿Qué es R?

R es un lenguaje de programación y un entorno de software libre dedicado al análisis estadístico y la generación de gráficos. Su potencia radica en una gran variedad de paquetes estadísticos, su comunidad activa y su capacidad de integración con otras herramientas como Python, SQL, y plataformas de visualización como Power BI o Tableau.

A continuación, se indican los enlaces oficiales para descargar (paso a paso) **R**:

Descargar R

- Página oficial del proyecto R (<https://cran.r-project.org>):

En esta página puedes seleccionar tu sistema operativo:

- Para Windows: <https://cran.r-project.org/bin/windows/base/>
- Para macOS: <https://cran.r-project.org/bin/macosx/>
- Para Linux: <https://cran.r-project.org/bin/linux/>

Se realizará el proceso para la instalación de R para el sistema operativo Windows



Figure 1.1: Software R

1.1.2 ¿Qué es RStudio?

RStudio es un **entorno de desarrollo integrado (IDE)** que proporciona una interfaz gráfica intuitiva y muy funcional para trabajar con R. Entre sus características destacan:

- Editor de scripts con **resaltado de sintaxis**.
- Consola interactiva para ejecutar comandos.
- Panel de visualización de datos y objetos en memoria.
- Gráficos integrados.
- Soporte para proyectos y versiones de R.
- Integración con Git, Markdown, Quarto y Shiny.

Aunque se puede usar R sin RStudio, la mayoría de los usuarios prefieren trabajar dentro de este entorno por su productividad, organización y facilidad de uso.

A continuación, se indican los enlaces oficiales para descargar (paso a paso) **RStudio**:

Descargar RStudio

- Página oficial de RStudio (ahora llamado **Posit**, <https://posit.co/download/rstudio-desktop/>):

Selecciona la versión gratuita de **RStudio Desktop** y descarga el instalador correspondiente a tu sistema operativo.

Se realizará el proceso para la instalación de RStudio para el sistema operativo Windows:

Una vez finalizada la instalación, puedes iniciar RStudio desde el acceso directo en tu escritorio o buscándolo en el menú de inicio.

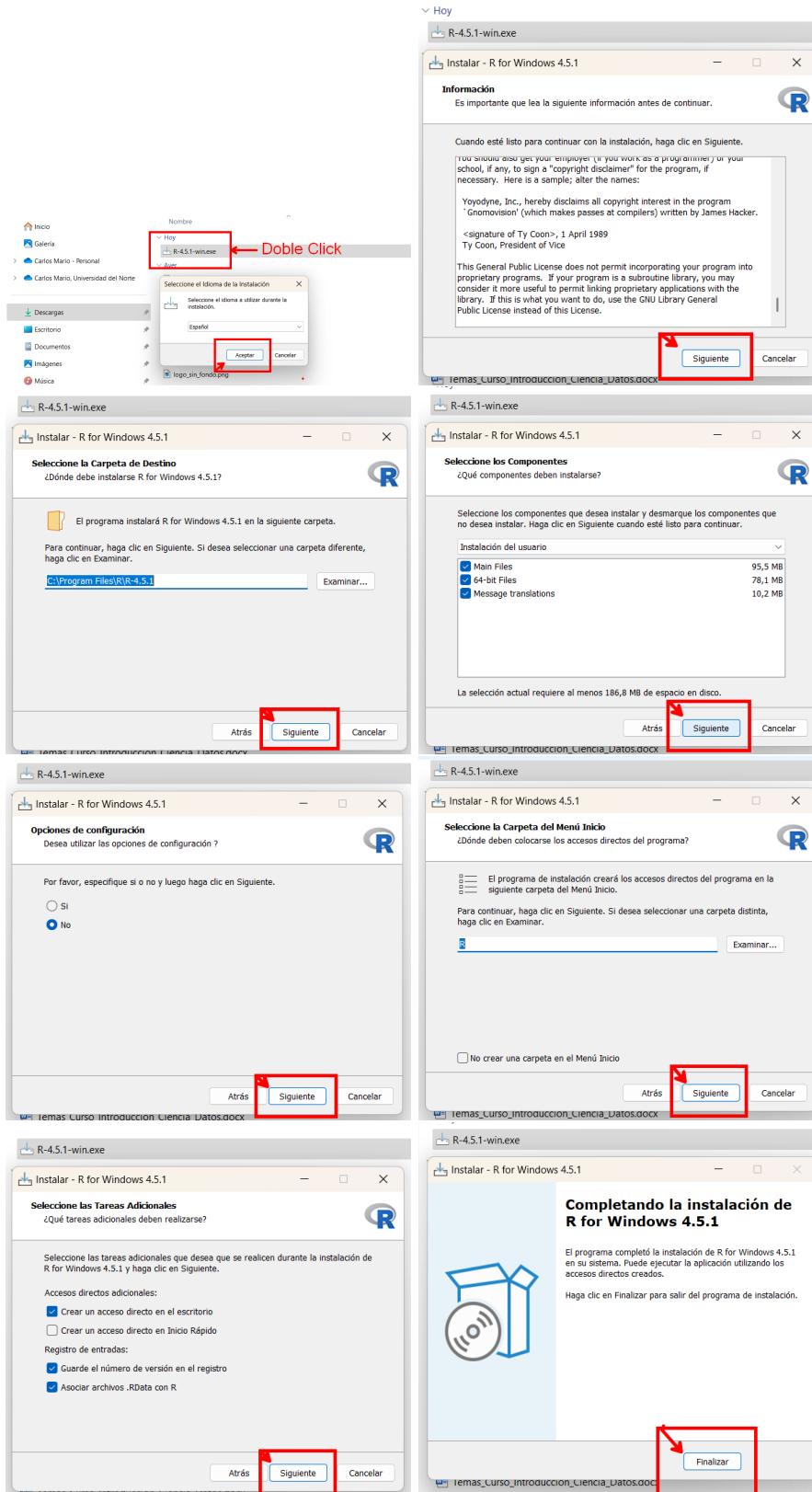
Figure 1.2: Paso a paso de la instalaci^{on} de R (izquierda a derecha)



Figure 1.3: Desarrollo de entorno integrado (IDE) RStudio

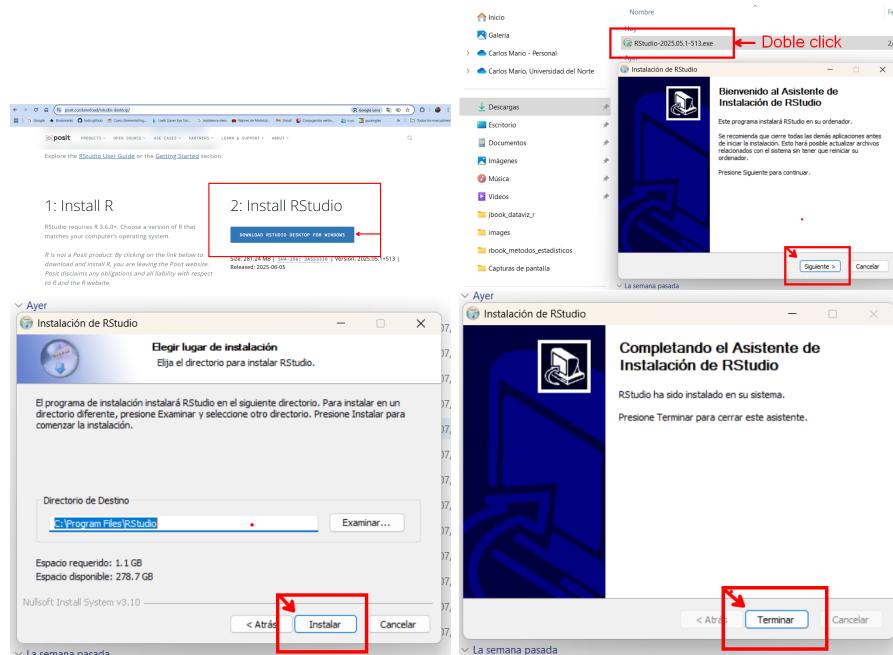
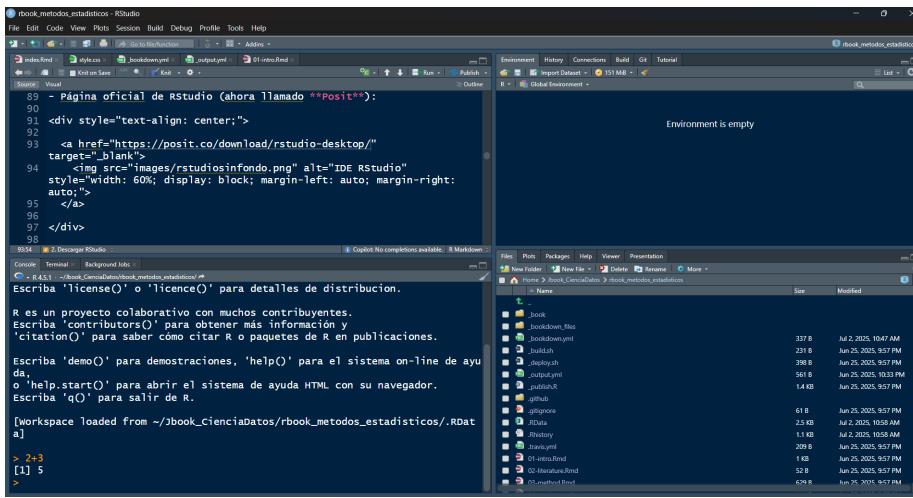


Figure 1.4: Paso a paso de la instalaci $\langle U+00F3 \rangle$ n de RStudio

Figure 1.5: Visualizaci $\langle U+00F3 \rangle$ n de RStudio

Al abrir RStudio por primera vez, se presenta un entorno dividido en cuatro paneles:

- **Script o editor de código** (arriba a la izquierda): donde se escriben los scripts `.R` o `.Rmd`.
- **Consola** (abajo a la izquierda): donde se ejecutan los comandos directamente.
- **Entorno / Historial** (arriba a la derecha): muestra los objetos cargados y el historial de comandos.
- **Archivos, gráficos, paquetes, ayuda y visor** (abajo a la derecha): herramientas auxiliares para explorar y trabajar eficientemente.

Puedes verificar que R y RStudio están funcionando correctamente ejecutando una operación simple en la consola, como:

2 + 3

1.2 Introducción a Git y GitHub

1.2.1 ¿Qué es Git?

Git es un sistema de control de versiones distribuido que permite gestionar y registrar los cambios realizados en archivos de un proyecto a lo largo del tiempo. Fue creado por Linus Torvalds y se ha convertido en el estándar para el desarrollo de software y proyectos colaborativos. El enlace es <https://git-scm.com/>.



Figure 1.6: Software Git

Ventajas principales de Git

- Permite llevar un **historial detallado** de versiones.
- Facilita la **colaboración** en proyectos con múltiples personas.
- Permite trabajar en **ramas** (branches) para desarrollar funcionalidades de forma aislada.
- No depende de internet para el trabajo local.

1.2.1.1 Pasos para instalar Git

Daremos una guía para la instalación de **Git** usando Windows (paso a paso):

1. Instalar **Git**:
<https://git-scm.com/downloads>
2. Ahora, debes realizar lo siguiente:
3. A continuación, comprobemos la instalación de Git.

1.2.2 ¿Qué es GitHub?

GitHub es una plataforma en línea que permite alojar repositorios de Git en la nube. Es ideal para compartir proyectos, colaborar en equipo y automatizar flujos de trabajo. Este es el enlace <https://github.com>.

Funciones principales de GitHub

- Crear y administrar **repositorios** públicos o privados.
- Gestionar cambios mediante **pull requests**.
- Seguir errores o tareas usando **issues**.
- Crear documentación, páginas web y wikis para los proyectos.
- Automatizar procesos con **GitHub Actions**.

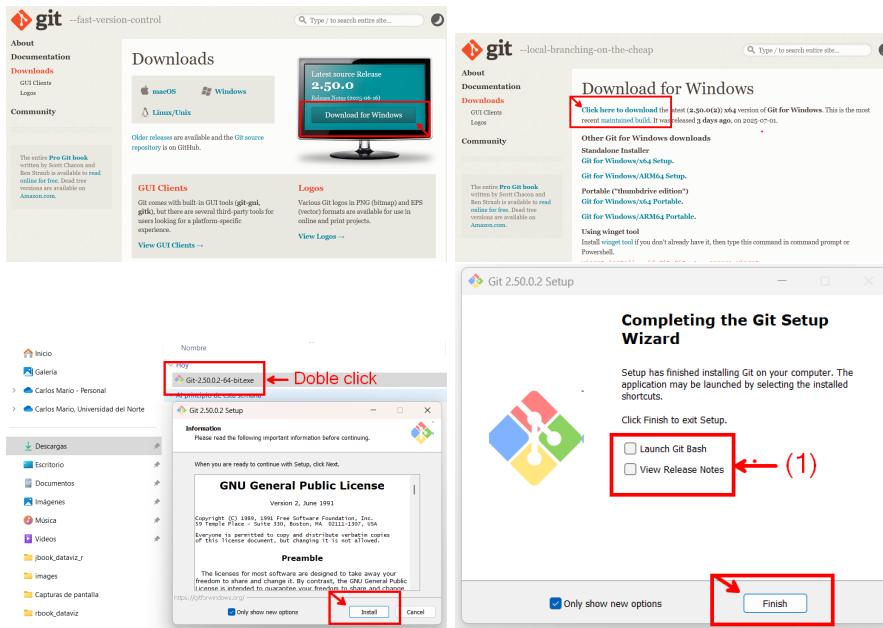
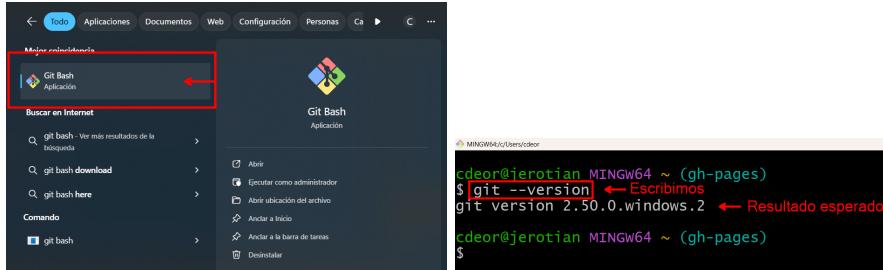
Figure 1.7: Instalaci^{U+00F3}n de GitFigure 1.8: Verificaci^{U+00F3}n de Git

Figure 1.9: Plataforma GitHub

1.2.2.1 Pasos para instalar GitHub

A continuación, se presenta una guía paso a paso para la instalación de **GitHub** en Windows:

1. Crear una cuenta en **GitHub**:
<https://github.com>

2. Ahora, sigue las imágenes:

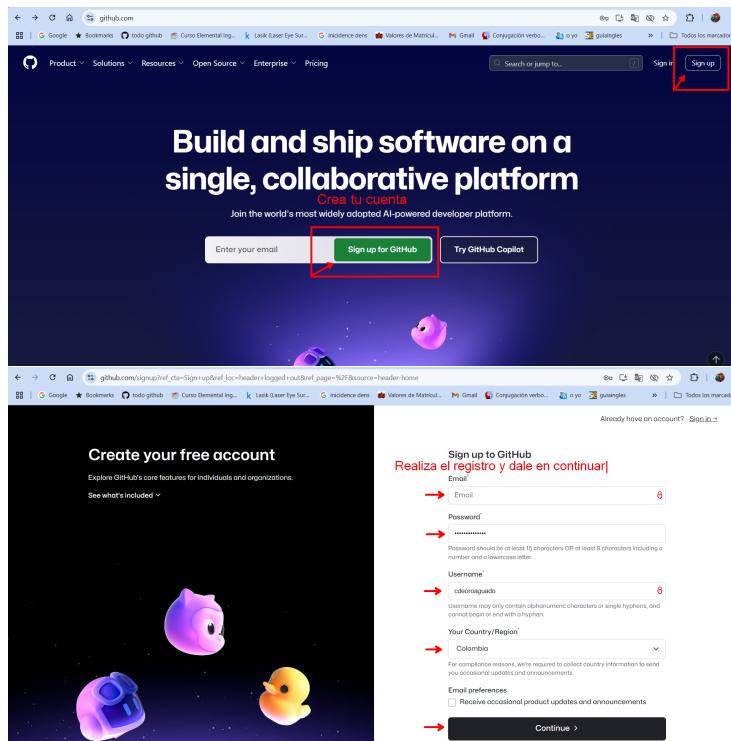


Figure 1.10: Registro en GitHub

Despues haces el proceso de verificación

- Completa el captcha de seguridad.
 - GitHub puede pedirte que verifiques tu correo electrónico. Revisa tu bandeja de entrada y haz clic en el enlace de confirmación.
3. Ingresas al enlace <https://github.com> y despues:

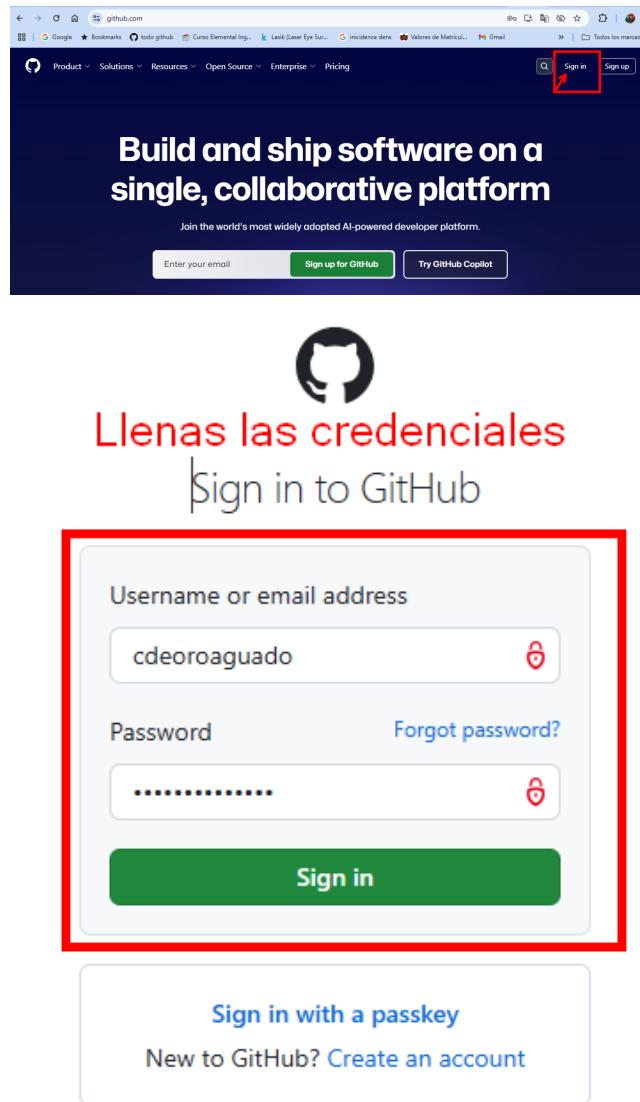


Figure 1.11: Credenciales de verificaci^{U+00F3}n en GitHub

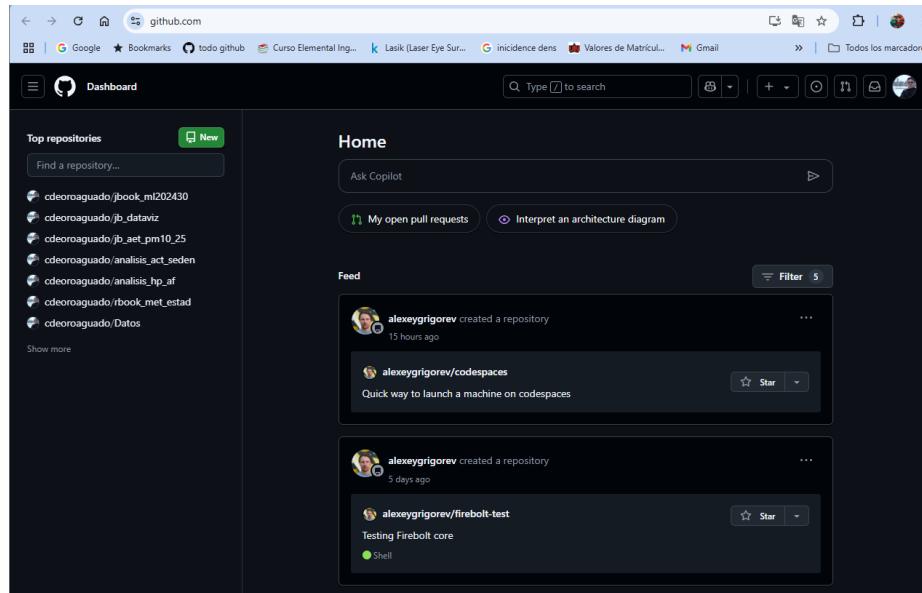


Figure 1.12: Dashboard de GitHub

Al ingresar, este sería el inicio

4. A continuación, vamos a descargar **GitHub Desktop** para Windows; para ello debes usar el enlace <https://desktop.github.com/download/>

Diferencias claves entre Git y GitHub

Git	GitHub
Herramienta local	Plataforma en la nube
Administra versiones	Aloja y comparte repositorios
No requiere internet	Requiere conexión para sincronizar
Se usa desde terminal o IDE	Se accede por navegador o API

1.2.2.2 ¿Cómo se relacionan?

- Git administra tu proyecto **localmente**, guardando versiones y cambios.
- GitHub actúa como **repositorio remoto**, permitiendo subir (push) o descargar (pull) cambios desde y hacia otros colaboradores.
- Juntos permiten trabajar de forma segura, organizada y colaborativa desde distintos lugares.

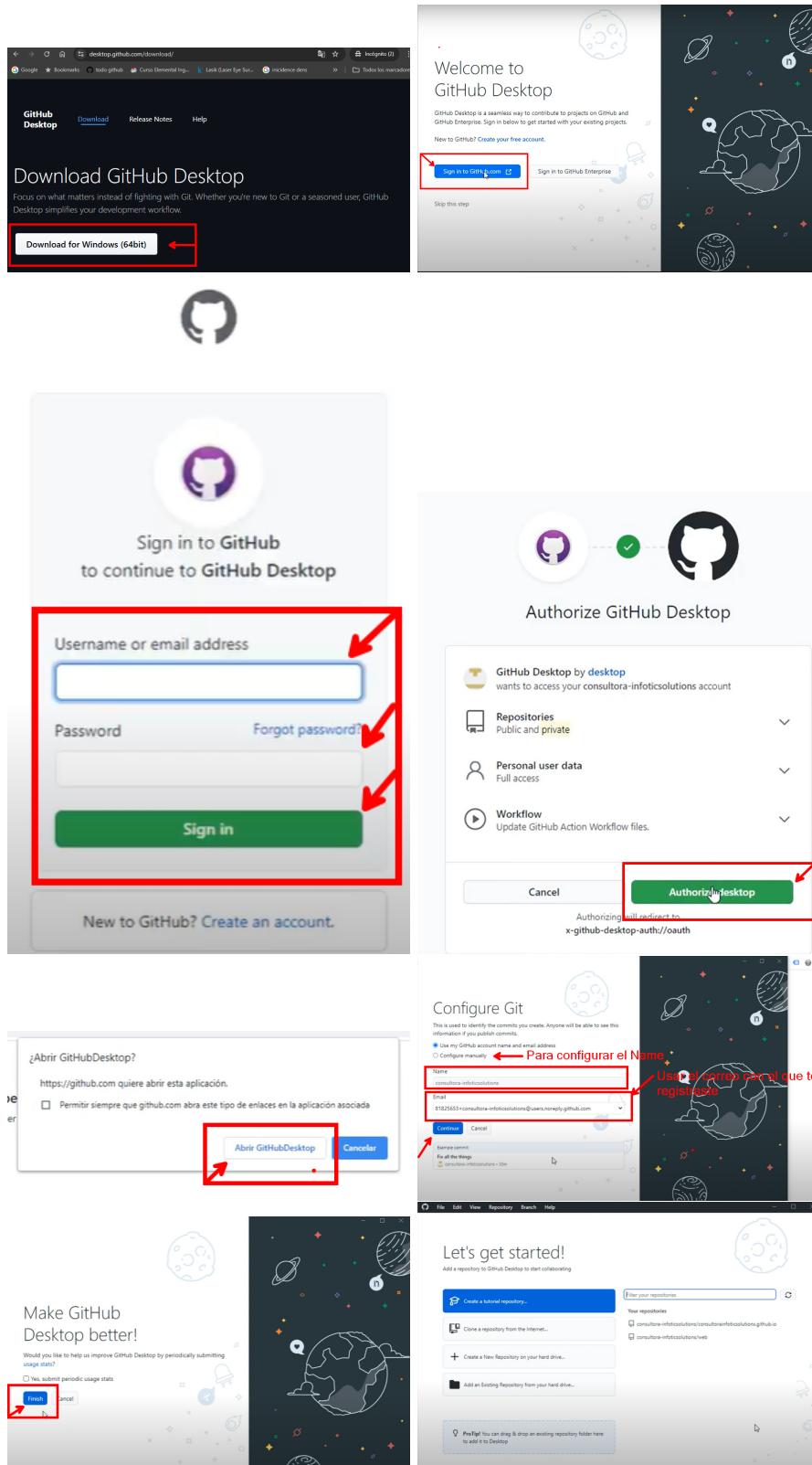


Figure 1.13: Credenciales de verificación en GitHub

1.3 Pasos para publicar un libro bookdown en GitHub Pages

Para publicar un libro bookdown en GitHub Pages debes tener lo siguiente:

1. Pre-requisitos

- Tener una cuenta en GitHub.
- Haber creado un repositorio público (ej. `metodos_estadisticos`).
- Tener Git instalado y configurado.
- Tener R y RStudio instalado y configurado
- Instalar el siguiente paquete

```
install.packages("bookdown")
```

- Tener un proyecto bookdown funcional en tu computadora.

2. Abre el archivo `_bookdown.yml` y asegúrate de incluir:

```
book_filename: "index"
output_dir: "docs"
```

Esto hace que el libro se renderice en la carpeta docs, que es donde GitHub Pages busca el sitio por defecto.

3. En RStudio o consola de R, corre:

```
bookdown::render_book("index.Rmd")
```

4. Abre la terminal en la carpeta del libro y ejecuta:

```
git init
git remote add origin https://github.com/cdeoroaguado/rbook_dataviz.git
git add docs
git commit -m "primer despliegue del libro"
git push -u origin main
```

5. Activa GitHub Pages

1.3. PASOS PARA PUBLICAR UN LIBRO BOOKDOWN EN GITHUB PAGES21

- Ve al repositorio en GitHub.
 - Haz clic en **Settings > Pages**.
 - En “Source”, selecciona:
 - **Branch:** `main` o `master`
 - **Folder:** `/docs`
 - Guarda los cambios.
6. Despu s de unos segundos, tu libro estar  disponible en:

Enlace del texto

Dale click en este enlace:

https://cdeoroaguado.github.io/rbook_dataviz/

1.3.1 Video paso a paso de la publicaci n del libro en GitHub Pages

Uno de los pasos m s importantes al desarrollar un libro con `bookdown` es su publicaci n en l nea, permitiendo el acceso abierto y permanente al contenido. Para ello, **GitHub Pages** se convierte en una herramienta ideal por su facilidad de uso y compatibilidad con proyectos de R. A continuaci n, se presenta un video tutorial donde se explican paso a paso los procedimientos necesarios para publicar correctamente un libro elaborado en `bookdown` a trav s de un repositorio en GitHub:

Cabe resaltar que este libro de muestra est mos utilizando el paquete `bookdown` (Xie, 2025), el cual fue construido sobre R Markdown y knitr (Xie, 2015)

Chapter 2

ETL con Tidyverse

El **Tidyverse** es un conjunto de paquetes integrados para el lenguaje de programación R, diseñados con el objetivo de facilitar el análisis de datos de manera estructurada, legible y eficiente. Su filosofía se basa en el concepto de “**datos ordenados**” (**tidy data**), donde cada variable es una columna, cada observación una fila, y cada tipo de unidad observacional forma una tabla.

Estos paquetes comparten principios de diseño comunes y una gramática coherente, lo que permite a los usuarios aprender un conjunto de reglas aplicables en todo el ecosistema, aumentando así la productividad y la claridad del código.

2.1 Data Frames

Definición

Un data frame es una estructura de datos clave en estadística y en R.

- La estructura básica de un data frame es que *hay una observación por fila y cada columna representa una variable, medida, rasgo o característica de esa observación*.
- **R** tiene una implementación interna de los data frames que probablemente es la que más se utiliza en la práctica.

2.2 Importación de datos

Fuentes de datos en R

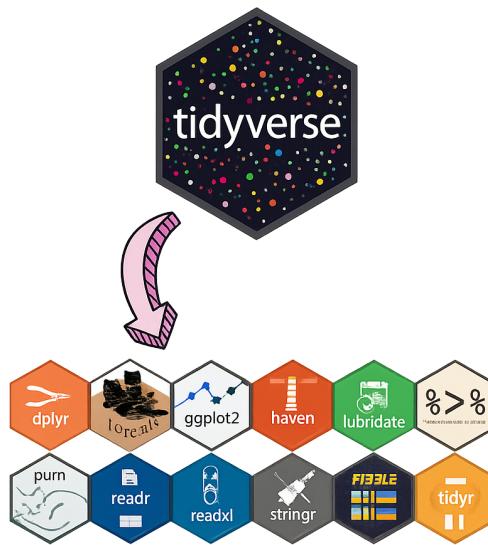


Figure 2.1: Paquetes de tidyverse

Los conjuntos de datos de gran tamaño, generalmente almacenados como data frames en R, suelen provenir de archivos externos.

Existen múltiples tipos de archivos que pueden importarse, entre ellos:

Archivos de texto en formatos como csv, txt, html y json.

Salidas de software estadístico como SAS y SPSS.

Recursos en línea como páginas html y servicios web.

Bases de datos relacionales y no relacionales.

El ecosistema Tidyverse ofrece funciones que permiten importar y gestionar estas diversas fuentes de datos de manera sencilla y eficiente.

2.2.1 Importación de archivos csv con `read.table()`

Definición

La función `read.table()` es una función integrada en R que permite leer archivos de distintos formatos y convertirlos en un data frame.

Es una de las funciones más utilizadas para importar archivos simples en R.

La sintaxis de `read.table()` requiere indicar:

- Un nombre de archivo (ruta de acceso).
- Un valor lógico (`TRUE/FALSE`) para definir si la primera fila contiene los nombres de las columnas.

- Si se establece en `TRUE`, la primera fila se interpreta como encabezado.
- Si se establece en `FALSE`, las columnas se importan sin nombres definidos.

El resultado de la función siempre es un **data frame**.

Una alternativa práctica es la función `file.choose()`, que permite seleccionar el archivo de manera interactiva sin necesidad de escribir la ruta manualmente.

Observación

Pasos para importar un archivo `csv` con `read.table()`:

Abrir RStudio y dirigirse a la consola.

Establecer el directorio de trabajo con código en la consola o desde el menú:
Sesión → Establecer directorio de trabajo → Elegir directorio.

Ejecutar la función `read.table()` indicando el archivo `csv` a importar.

```
setwd("~/Books_CienciaDatos/rbook_dataviz")
```

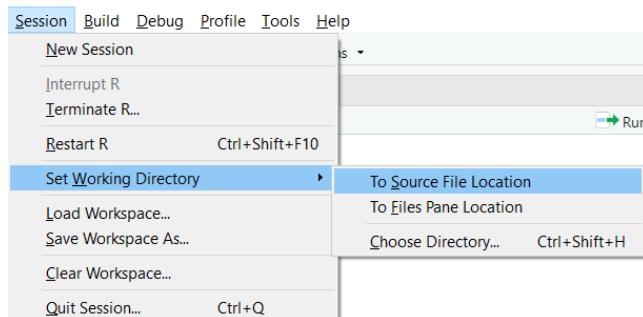


Figure 2.2: setwd

- En la carpeta **RDataSets** se encuentra el archivo **StudyArea.csv**, que corresponde a un archivo separado por comas.
Este archivo contiene información sobre incendios forestales ocurridos entre los años 1980 y 2016 en distintos estados de EE. UU., entre ellos: California, Oregón, Washington, Idaho, Montana, Wyoming, Colorado, Utah, Nevada, Arizona y Nuevo México.

- El archivo incluye más de **439000 registros** distribuidos en **37 columnas**, que describen las características de cada incendio durante ese periodo.
- Para cargar estos datos en un nuevo objeto **data frame**, puede utilizarse la función **read.table()** de la siguiente manera:

```
df <- read.table("data/StudyArea.csv", header = TRUE)
```

- Recibirá un mensaje de error cuando intente ejecutar esta línea de código. El mensaje de error debería aparecer como se ve a continuación



Figure 2.3: error en rmd

- La razón por la que se generó un mensaje de error en este caso es que la función **read.table()** utiliza espacios como delimitador entre registros y nuestro archivo utiliza comas como delimitador

Actualice su llamada a **read.table()** como se ve a continuación para incluir el argumento **sep**, que debería ser una coma

```
df <- read.table("data/StudyArea.csv", sep=",", header = TRUE)
```

- Cuando ejecute esta línea de código verá un nuevo error

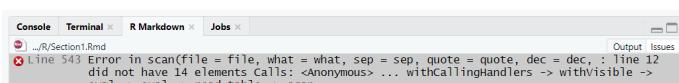


Figure 2.4: continuaci<U+00F3>n del error en rmd

- La función **read.table()** no completa de manera automática las celdas vacías con un valor por defecto como **NA**. Por este motivo, si alguna fila del archivo no contiene el número esperado de columnas (en este caso, 14), se genera un mensaje de error durante la importación.

Para solucionar este inconveniente, puede añadirse el parámetro **fill = TRUE**, lo que permite a **R** llenar los espacios vacíos con **NA** y mantener la estructura correcta del **data.frame**.

```
df <- read.table("data/StudyArea.csv",
                  header = TRUE,
                  fill   = TRUE,
                  sep    = ", ")
nrow(df)

## [1] 153095
```

- Al ejecutar esta instrucción, el contenido del archivo se importa a un objeto de tipo **data frame**. Sin embargo, al revisar la pestaña **Global Environment** en **RStudio**, se observa que solo se han cargado **153095 registros**, a pesar de que el archivo original contiene más de **400000**.
- La causa de esta diferencia suele estar en el manejo de las **comillas (simples o dobles)** presentes dentro del archivo **csv**, las cuales pueden interrumpir la lectura y provocar que algunos registros sean descartados.

Para corregir este problema, se debe añadir el parámetro **quote = ""**, que le indica a **R** que ignore las comillas y procese correctamente todas las filas del archivo.

```
df = read.table("data/StudyArea.csv",
                header=TRUE,
                fill=TRUE,
                quote="",",
                sep=", ")
nrow(df)

## [1] 439362
```

- Al ejecutar esta línea de código, se deberían importar **440476 registros**. Los datos se cargan en un objeto de tipo **R data.frame**, que es una estructura similar a una tabla. Por ahora, puede pensarse en ellos como tablas que contienen columnas y filas.

Observación

La función **read.table()** se utiliza normalmente para cargar archivos de texto delimitados por tabulaciones.

Sin embargo, muchas personas intentan emplearla directamente con archivos en formato csv sin especificar los parámetros adecuados, lo que puede generar errores en la importación.

Una alternativa más práctica es usar la función **read.csv()**, como veremos en el siguiente paso.

2.2.2 Importación de archivos txt delimitados por tabulación con `read.table()`

- La función `read.table()` se utiliza con frecuencia para leer el contenido de archivos delimitados por tabulaciones u otros separadores. En este ejemplo, se trabajará con el archivo `all_genes_pombase.txt`, ubicado en la carpeta `RDataSets`.
- Antes de importarlo en **R**, se recomienda abrir el archivo en **Excel** o en cualquier editor de texto para observar su estructura, campos y delimitadores.
- Una vez identificado el formato del archivo, en la consola de **R** puedes ejecutar el siguiente código para realizar la importación:

```
# Lectura de archivo delimitado por tabulaciones
genes <- read.table("data/all_genes_pombase.txt",
                     header = TRUE,
                     sep = "\t",
                     quote = "\"")
# numero de filas
nrow(genes)

## [1] 7019

# Visualizar las primeras filas
head(genes)

##      ensembl_id      name chromosome
## 1  SPAC1002.01  SPAC1002.01        I
## 2  SPAC1002.02       pom34        I
## 3  SPAC1002.03c      gls2        I
## 4  SPAC1002.04c      taf11        I
## 5  SPAC1002.05c      jmj2        I
## 6  SPAC1002.06c      bqt2        I
##                                         description
## 1                               conserved fungal protein
## 2                               nucleoporin Pom34
## 3                         glucosidase II alpha subunit Gls2
## 4 transcription factor TFIID complex subunit Taf11 (predicted)
## 5                               histone demethylase Jmj2
## 6                         bouquet formation protein Bqt2
##      feature_type strand    start     end
## 1 protein_coding      1 1798347 1799015
## 2 protein_coding      1 1799061 1800053
```

```
## 3 protein_coding      -1 1799915 1803141
## 4 protein_coding      -1 1803624 1804491
## 5 protein_coding      -1 1804548 1806797
## 6 protein_coding      -1 1807270 1807781
```

Esto permitirá cargar **7019 registros** en un objeto `data.frame`. Es importante tener en cuenta que, aunque la función realiza la importación, **varios de sus parámetros deben configurarse adecuadamente al momento de leer el dataset**, lo que hace que el proceso no sea tan directo ni automático como podría suponerse inicialmente.

2.2.3 Importación de archivos csv con `read.csv()`

Definición: `read.csv()`

La función `read.csv()` es una función incorporada en **R** que permite importar archivos delimitados por comas (`csv`) de manera sencilla.

Está diseñada específicamente para este tipo de archivos y establece de forma predeterminada el argumento `header = TRUE` (la primera fila se toma como nombres de columna) y `sep = ","` (coma como delimitador de campos).

Esto hace que `read.csv()` sea una de las formas más rápidas y eficientes de cargar datos en **R** cuando provienen de archivos `csv`.

- La función `read.csv()` es una función incorporada en **R** que resulta casi idéntica a `read.table()`. La principal diferencia es que en `read.csv()` los argumentos de **cabecera** y **relleno** se establecen en `TRUE` por defecto, lo que facilita la carga de archivos delimitados por comas (`csv`). En este punto, se observa que usar `read.csv()` simplifica notablemente el proceso de importación.
- A diferencia de `read.table()`, la función `read.csv()` **gestiona automáticamente la mayoría de las configuraciones necesarias** al leer un archivo `csv`. Esto permite cargar correctamente incluso archivos con más de **400000 registros** sin tener que especificar tantos parámetros manualmente.

```
# Lectura de archivo CSV con read.csv()
df <- read.csv("data/StudyArea.csv")

# numero de fila
nrow(df)
```

```
## [1] 439362
```

2.2.4 Uso de `readr` de tidyverse

Definición: `readr`

`readr` es el paquete del ecosistema tidyverse para lectura y escritura rápida de datos tabulares. Produce objetos tibble, maneja de forma estable codificaciones (UTF-8), tipos de columna y valores perdidos, e incluye herramientas para diagnosticar problemas de importación. Su API es consistente, minimalista y pensada para flujos reproducibles.

Ventajas claves

Velocidad y consistencia en lectura/escritura de archivos delimitados.

Tipos de columnas explícitos con `col_types` y funciones `col_*`.

Diagnóstico con `spec()` y `problems()`.

Salida en tibble (imprime y maneja mejor columnas anchas/fechas).

Soporte de locales (`locale()`) para decimales, fechas, codificación, etc.

Funciones principales de lectura

Función	Propósito	Formato/Delimitador
<code>read_csv()</code>	csv con punto como decimal	,
<code>read_csv2()</code>	csv europeo (coma decimal)	;
<code>read_tsv()</code>	Valores separados por tabulador	\t
<code>read_delim(delim=)</code>	Delimitador personalizado	Cualquiera (p. ej. \!, :)
<code>read_table()</code>	Columnas separadas por espacios	Espacios en blanco
<code>read_fwf()</code>	Formato de ancho fijo	Anchos/posiciones
<code>read_lines()</code>	Leer líneas como vector de caracteres	Texto
<code>read_file()</code>	Leer archivo completo como cadena única	Texto
<code>read_rds()</code>	Leer archivo R serializado	.rds

Funciones de escritura

Función	Propósito
<code>write_csv()</code>	Escribir csv (punto decimal)
<code>write_csv2()</code>	Escribir csv (coma decimal)
<code>write_tsv()</code>	Escribir delimitado por tabulador
<code>write_delim()</code>	Escribir con delimitador personalizado
<code>write_rds()</code>	Guardar objeto R en .rds

Argumentos comunes (lectura)

Argumento	Qué controla	Ejemplo
<code>col_types</code>	Tipos de columnas (explícitos)	<code>col_types =</code> <code>cols(id =</code> <code>col_integer())</code> <code>na = c("", "NA",</code> <code>"NULL")</code>
<code>na</code>	Cadenas que se tratarán como NA	
<code>locale</code>	Configuración regional (decimal, fecha, tz, encoding)	<code>locale(decimal_mark</code> <code>= ",")</code>
<code>skip / n_max</code>	Filas a omitir / máximo de filas a leer	<code>skip = 2, n_max =</code> <code>1e5</code>
<code>comment</code>	Prefijo de comentario para ignorar líneas	<code>comment = "#"</code>
<code>guess_max</code>	Filas usadas para “adivinar” tipos	<code>guess_max =</code> <code>100000</code>
<code>show_col_type</code>	Muestra la conjetura de tipos al leer	<code>show_col_types =</code> <code>FALSE</code>

Tip: cuando los tipos son críticos, **no confíes solo en la inferencia**: pasa `col_types` explícito.

- Veamos el siguiente ejemplo. Carguemos los siguientes datos:

```
library(readr)

# CSV con punto decimal y encabezados
dfReadr <- read_csv("data/StudyArea.csv",
  col_types = cols(.default = "c"), # Columnas (.default) como "character"
  col_names = TRUE)               # la 1ra fila como nombres de las columnas

head(dfReadr)

## # A tibble: 6 x 14
##   FID   ORGANIZATI UNIT  SUBUNIT SUBUNIT2      FIRENAME CAUSE YEAR_
##   <chr> <chr>    <chr> <chr>   <chr>      <chr>   <chr> <chr>
## 1 0     FWS       81682 USCADBR San Diego Bay ~ PUMP HO~ Human 2001
## 2 1     FWS       81682 USCADBR San Diego Bay ~ I5      Human 2002
## 3 2     FWS       81682 USCADBR San Diego Bay ~ SOUTHBAY Human 2002
## 4 3     FWS       81682 USCADBR San Diego Bay ~ MARINA Human 2001
## 5 4     FWS       81682 USCADBR San Diego Bay ~ HILL    Human 1994
## 6 5     FWS       81682 USCADBR San Diego Bay ~ IRRIGAT~ Human 1994
## # i 6 more variables: STARTDATED <chr>, CONTRDATED <chr>,
## #   OUTDATED <chr>, STATE <chr>, STATE_FIPS <chr>, TOTALACRES <chr>
```

```
spec(dfReadr)          # Esquema de tipos
```

```
## cols(
##   .default = col_character(),
##   FID = col_character(),
##   ORGANIZATI = col_character(),
##   UNIT = col_character(),
##   SUBUNIT = col_character(),
##   SUBUNIT2 = col_character(),
##   FIRENAME = col_character(),
##   CAUSE = col_character(),
##   YEAR_ = col_character(),
##   STARTDATED = col_character(),
##   CONTRDATED = col_character(),
##   OUTDATED = col_character(),
##   STATE = col_character(),
##   STATE_FIPS = col_character(),
##   TOTALACRES = col_character()
## )
```

- Al ejecutar nuevamente la función sin el argumento `col_types`, R* intentará detectar automáticamente el tipo de dato de cada columna. En este proceso se mostrará primero un listado con los nombres de las columnas y el tipo asignado a cada una, seguido de un mensaje de advertencia que indica la presencia de errores de análisis durante la importación, lo cual ejemplifica las inconsistencias que pueden surgir al dejar la inferencia de tipos en manos del sistema.
- Actualice el código como se muestra a continuación y ejecútelo nuevamente. En este caso, se especifica que la columna `UNIT` debe ser importada como un dato de tipo carácter (texto):

```
dfReadr = read_csv("data/StudyArea.csv",
                   col_types = list(FID = col_character()),
                   col_names = TRUE)
```

```
head(dfReadr)
```

```
## # A tibble: 6 x 14
##   FID    ORGANIZATI UNIT  SUBUNIT SUBUNIT2      FIRENAME CAUSE YEAR_
##   <chr> <chr>     <chr> <chr>   <chr>     <chr>   <chr> <dbl>
## 1 0     FWS       81682 USCADBR San Diego Bay ~ PUMP HO~ Human  2001
## 2 1     FWS       81682 USCADBR San Diego Bay ~ I5      Human  2002
## 3 2     FWS       81682 USCADBR San Diego Bay ~ SOUTHBAY Human  2002
```

```

## 4 3      FWS          81682 USCADBR San Diego Bay ~ MARINA   Human  2001
## 5 4      FWS          81682 USCADBR San Diego Bay ~ HILL    Human  1994
## 6 5      FWS          81682 USCADBR San Diego Bay ~ IRRIGAT~ Human  1994
## # i 6 more variables: STARTDATED <chr>, CONTRDATED <chr>,
## #   OUTDATED <chr>, STATE <chr>, STATE_FIPS <dbl>, TOTALACRES <dbl>

```

2.3 Transformación de datos

- Antes de realizar un análisis de datos en **R**, con frecuencia es necesario manipular o transformar la información de distintas formas.

Para ello, el paquete **dplyr**, que hace parte del ecosistema **tidyverse**, ofrece un conjunto de funciones que facilitan la transformación y manejo de datos de manera eficiente y estructurada.

- En esta sección abordaremos los siguientes aspectos fundamentales:
 - Filtrar registros para obtener subconjuntos de datos
 - Seleccionar y limitar columnas específicas
 - Ordenar filas en forma ascendente o descendente
 - Incorporar nuevas filas a un conjunto existente
 - Resumir y agrupar información
 - Utilizar canalización (*pipes*) para mejorar la legibilidad y eficiencia del código

2.3.1 El paquete **dplyr**

Definición

El paquete **dplyr** fue desarrollado por Hadley Wickham de RStudio y es una versión optimizada y destilada de su paquete **plyr**.

- Una importante contribución de **dplyr** es que proporciona una **gramática** (en particular, *verbos*) para la *manipulación de datos y operación con data frames*.
- Con esta gramática se puede comunicar de forma comprensible lo que se está haciendo a un data frame, lo cual es muy útil porque proporciona una abstracción para la manipulación de datos que antes no existía.

- Otra ventaja es que las funciones de `dplyr` son muy rápidas, ya que muchas operaciones clave están codificadas en C++.

2.3.2 Operador `%>%`

- El operador `pipeline %>%` resulta muy útil para **encadenar múltiples funciones de dplyr en una secuencia de operaciones**. Antes de usarlo, cuando se necesitaba aplicar varias funciones de manera consecutiva, la expresión debía escribirse como una **secuencia de funciones anidadas**, lo cual resulta poco legible. Por ejemplo:

```
third(second(first(x)))
```

- Este estilo de anidamiento **no refleja la manera natural de pensar en una secuencia de pasos**. En cambio, el operador `%>%` permite expresar la secuencia **de izquierda a derecha**, facilitando la lectura y comprensión del código.

```
first(x) %>%
  second %>%
  third
```

2.3.3 Filtrar datos para crear un subconjunto

Definición: `filter()`

La función `filter()` del paquete `dplyr` se utiliza para extraer subconjuntos de filas de un dataframe en función de una o más condiciones lógicas. Es similar a la función `subset()` de R base, pero resulta más rápida y eficiente.

El primer argumento que recibe `filter()` siempre es un objeto de tipo dataframe, mientras que los argumentos adicionales corresponden a las expresiones condicionales que definen el filtrado.

Ejemplo de incendios forestales

El repositorio **RDataSets** contiene el archivo `StudyArea.csv`, un archivo separado por comas con información de incendios forestales ocurridos entre los años **1980 y 2016** en los estados de **California, Oregón, Washington, Idaho, Montana, Wyoming, Colorado, Utah, Nevada, Arizona y Nuevo México**.

Este archivo cuenta con **aproximadamente 439.000 registros y 37 columnas** que describen las características de cada incendio durante este periodo.

En este **dataframe** de incendios forestales tiene una columna llamada **TOTALACRES**, la cual registra el número de acres quemados por evento.

```
df <- read.csv("data/StudyArea.csv")

head(df, n=3)

##   FID ORGANIZATI UNIT SUBUNIT
## 1    0        FWS 81682 USCADBR
## 2    1        FWS 81682 USCADBR
## 3    2        FWS 81682 USCADBR
##                               SUBUNIT2 FIRENAME CAUSE YEAR_
## 1 San Diego Bay National Wildlife Refuge PUMP HOUSE Human 2001
## 2 San Diego Bay National Wildlife Refuge           I5 Human 2002
## 3 San Diego Bay National Wildlife Refuge SOUTHBAY Human 2002
##   STARTDATED CONTRRDATED OUTDATED      STATE STATE_FIPS TOTALACRES
## 1 1/1/01 0:00 1/1/01 0:00 California       6      0.1
## 2 5/3/02 0:00 5/3/02 0:00 California       6      3.0
## 3 6/1/02 0:00 6/1/02 0:00 California       6      0.5
```

Solución

- Crear un subconjunto de registros que contenga sólo los incendios forestales de más de 25000 acres.

```
library(tidyverse)

df %>%
  filter(TOTALACRES >= 25000)
```

- Crear un subconjunto de registros que contenga sólo los incendios forestales de más de 1000 acres en el año 2016.

```
df %>%
  filter(TOTALACRES >= 1000, YEAR_ == 2016)
```

- Crear un subconjunto de registros que contenga sólo los incendios forestales de más de 1000 acres en el año 2016, pero usa el operador &.

```
df %>%
  filter(TOTALACRES >= 1000 & YEAR_ == 2016)
```

- Crear un subconjunto de registros que contenga sólo los años 2010, 2011 y 2012

```
df %>%
  filter(YEAR_ %in% c(2010, 2011, 2012))
```

2.3.4 Acotar la lista de columnas con `select()`

Definición: `select()`

La función `select()` del paquete `dplyr` se emplea para elegir columnas específicas de un `dataframe`.

Es especialmente útil cuando se trabaja con conjuntos de datos extensos y solo se requiere un subconjunto de variables.

Continuando con el ejemplo de los incendios forestales

- Selecciona las columnas `FIRENAME`, `TOTALACRES`, `YEAR_`

```
df %>%
  select(FIRENAME, TOTALACRES, YEAR_)
```

- Selecciona las columnas `FIRENAME`, `TOTALACRES`, `YEAR_`, cambian el nombre de `TOTALACRES` por `ACRES` y `YEAR_` por `YR`.

```
df %>%
  select("FIRE" = "FIRENAME", "ACRES" = "TOTALACRES", "YR" = "YEAR_")
```

- Selecciona las columnas que contengan la palabra `DATE`

```
df %>%
  select(contains("DATE"))
```

- Selecciona las columnas que contengan la palabra `DATE` y también inician con `TOTAL`

```
df %>%
  select(contains("DATE"), starts_with("TOTAL"))
```

Funciones de ayuda para `select()`

La función `select()` puede complementarse con una serie de funciones auxiliares que permiten filtrar las columnas de manera más flexible y eficiente. Estas funciones son especialmente útiles cuando se trabaja con conjuntos de datos amplios o cuando los nombres de las variables siguen un patrón específico.

`starts_with("texto")`: selecciona las columnas cuyos nombres comienzan con el texto indicado.

`ends_with("texto")`: selecciona las columnas cuyos nombres terminan con el texto indicado.

`contains("texto")`: selecciona las columnas cuyos nombres contienen el texto indicado.

`matches("regex")`: selecciona las columnas cuyos nombres cumplen con una expresión regular.

`num_range("x", 1:5)`: selecciona un rango de columnas numeradas (ejemplo: x1, x2, ..., x5).

2.3.5 Organizar las filas

Definición: `arrange()`

La función `arrange()` del paquete dplyr se utiliza para ordenar las filas de un dataframe en función de una o varias columnas.

De manera predeterminada, organiza en orden ascendente, aunque puede invertirse a orden descendente con la función `desc()`.

Continuando con el ejemplo de los incendios forestales

- Filtrar el conjunto de datos para que contenga solo los incendios de más de 1.000 acres quemados del año 2016. Despues selecciona las columnas FIRENAME, TOTALACRES, YEAR_ y renombralas NAME, ACRES, YR, respectivamente. Finalmente, ordena ACRES de forma ascendente y muestrame las 5 ultimas.

```
df %>%
  filter(TOTALACRES >= 1000, YEAR_ == 2016) %>%
  select("NAME" = "FIRENAME", "ACRES" = "TOTALACRES", "YR" = "YEAR_") %>% arrange(ACRES) %>%
  tail(n=5)
```

```
##           NAME   ACRES    YR
## 148     Cedar  45977 2016
## 149   Erskine  48007 2016
## 150 Range 12 171915 2016
## 151  Junkins 181320 2016
## 152 PIONEER 188404 2016
```

- Filtrar el conjunto de datos para que contenga solo los incendios de más de 2.000 acres quemados del año 2016. Despues selecciona las columnas

`FIRENAME`, `TOTALACRES`, `YEAR_` y renombralas `NAME`, `ACRES`, `YR`, respectivamente. Finalmente, ordena `ACRES` de forma descendente y muestrame las 5 primeras.

```
df %>%
  filter(TOTALACRES >= 2000, YEAR_ == 2016) %>%
  select("NAME" = "FIRENAME", "ACRES" = "TOTALACRES", "YR" = "YEAR_") %>% arrange(desc(ACRES))
head(n=5)

##      NAME    ACRES    YR
## 1 PIONEER 188404 2016
## 2 Junkins 181320 2016
## 3 Range 12 171915 2016
## 4 Erskine 48007 2016
## 5 Cedar   45977 2016
```

2.3.6 Añadir columnas con `mutate()`

Definición: `mutate()`

La función `mutate()` del paquete `dplyr` se utiliza para crear nuevas variables o transformar las existentes dentro de un dataframe.

Es especialmente útil cuando se desea generar columnas derivadas a partir de operaciones aritméticas, funciones estadísticas o transformaciones sobre variables ya presentes en los datos.

Gracias a su sintaxis sencilla y legible, `mutate()` ofrece una forma clara y estructurada de enriquecer un conjunto de datos sin necesidad de sobrescribir la información original.

Continuando con el ejemplo de los incendios forestales

- Seleccione únicamente las columnas `ORGANIZATI`, `STATE`, `YEAR_`, `TOTALACRES`, `CAUSE` y `STARTDATED`, filtre los registros para que solo se incluyan los incendios con más de **1.000 acres quemados** y cuya causa sea **Humana** o **Natural**, cree una nueva columna llamada `DOY` que indique el día del año en que inició cada incendio a partir de la columna `STARTDATED`, y finalmente muestre las primeras filas del resultado.

Carguemos nuevamente los datos

```
df %>%
  select(ORGANIZATI, STATE, YEAR_, TOTALACRES, CAUSE, STARTDATED) %>%
  filter(TOTALACRES >= 1000 & CAUSE %in% c("Human", "Natural")) %>%
```

```
mutate(DOY = yday(as.Date(startdated, format="%m/%d/%y %H:%M")))) -> df_sol

# Vista rápida
knitr::kable(head(df_sol))
```

ORGANIZATI	STATE	YEAR_	TOTALACRES	CAUSE	STARTDATED	DOY
FWS	Arizona	1988	1500	Human	3/26/88 0:00	86
FWS	Arizona	1986	10390	Human	5/15/86 0:00	135
FWS	Montana	1986	1400	Human	6/27/86 0:00	178
FWS	Arizona	2002	1035	Human	2/28/02 0:00	59
FWS	Arizona	2000	5700	Human	4/9/00 0:00	100
FWS	Arizona	2000	2750	Human	5/14/00 0:00	135

Observación sobre lubridate

El paquete lubridate del ecosistema tidyverse está diseñado para simplificar la manipulación de fechas y tiempos en R.

Proporciona funciones intuitivas como ymd(), mdy() o dmy() para convertir cadenas en objetos de tipo fecha, así como ymd_hms() o mdy_hm() para manejar fechas con hora, minutos y segundos.

Además, incluye utilidades como yday() (día del año), wday() (día de la semana), month() o year(), que permiten extraer y operar sobre componentes específicos de una fecha de manera clara y eficiente.

En comparación con las funciones base de R, lubridate ofrece una sintaxis más legible y reduce errores asociados al manejo de distintos formatos de fecha y hora.

Función	Descripción	Ejemplo
ymd()	Convierte cadenas con formato Año-Mes-Día a fecha (YYYY-MM-DD).	ymd("2023-08-21") → 2023-08-21
mdy()	Convierte cadenas con formato Mes-Día-Año a fecha (MM-DD-YYYY).	mdy("08-21-2023") → 2023-08-21
dmy()	Convierte cadenas con formato Día-Mes-Año a fecha (DD-MM-YYYY).	dmy("21-08-2023") → 2023-08-21
ymd_hms()	Convierte a fecha con hora, minutos y segundos.	ymd_hms("2023-08-21 14:30:15") → 2023-08-21 14:30:15
mdy_hm()	Convierte a fecha con hora y minutos.	mdy_hm("08-21-2023 14:30") → 2023-08-21 14:30
year()	Extrae el año de una fecha.	year(ymd("2023-08-21")) → 2023
month()	Extrae el mes de una fecha (numérico o etiqueta si label=TRUE).	month(ymd("2023-08-21"), label=TRUE) → Aug
day()	Extrae el día del mes.	day(ymd("2023-08-21")) → 21

Función	Descripción	Ejemplo
yday()	Devuelve el día del año (1–365/366).	yday(ymd("2023-08-21")) → 233
wday()	Devuelve el día de la semana (numérico o etiqueta si <code>label=TRUE</code>).	wday(ymd("2023-08-21"), <code>label=TRUE</code>) → Mon
hour(), minute(), second()	Extraen hora, minutos o segundos de un objeto fecha-hora.	hour(ymd_hms("2023-08-21 14:30:15")) → 14

2.3.7 Agrupación y resumen de los datos

Definición: `group_by()`

La función `group_by()` del paquete `dplyr` se utiliza para agrupar un dataframe en función de una o varias variables.

Este agrupamiento no modifica los datos en sí, sino que establece una estructura que permite aplicar funciones de resumen o transformación sobre cada grupo de manera independiente.

Generalmente, `group_by()` se usa en combinación con `summarise()`, `mutate()` u otras funciones de `dplyr`.

Continuando con el ejemplo de incendios forestales

- Seleccione únicamente las columnas **ORGANIZATI**, **STATE**, **YEAR_**, **TOTALACRES** y **CAUSE**, filtre los registros para que solo se incluyan los incendios con más de **1.000 acres** quemados, cree una nueva columna llamada **DECADE** que define la década en la que se produjo cada incendio, agrupa por**DECADE** y finalmente un resumen numérico completo del tamaño de los incendios forestales por década.

```
df %>%
  select(ORGANIZATI, STATE, YEAR_, TOTALACRES, CAUSE) %>%
  filter(TOTALACRES >= 1000) %>%
  mutate(DECADE = ifelse(YEAR_ %in% 1980:1989, "1980-1989",
                         ifelse(YEAR_ %in% 1990:1999, "1990-1999",
                         ifelse(YEAR_ %in% 2000:2009, "2000-2009",
                         ifelse(YEAR_ %in% 2010:2016, "2010-2016", "-99"))))) %>%
  group_by(DECADE) %>%
  summarise(media_acres = mean(TOTALACRES),
            ds_acres = sd(TOTALACRES),
            mediana_acres = median(TOTALACRES),
            RIC_acres = IQR(TOTALACRES),
            min_acres = min(TOTALACRES),
            max_acres = max(TOTALACRES),
```

```

q1_acres = quantile(TOTALACRES) [2] ,
q3_acres = quantile(TOTALACRES) [4] ) -> df_sol_2

# Vista rápida
knitr::kable(head(df_sol_2))

```

DECADE	media_acres	ds_acres	mediana_acres	RIC_acres	min_acres	max_acres	q1_acres	q3_acres
1980-1989	8128.645	23681.53	2887.5	5074.50	1000	427680.0	1543.25	66
1990-1999	8333.036	18212.44	2925.0	5641.80	1000	231389.0	1545.00	71
2000-2009	12329.181	30156.35	3653.5	7859.75	1000	590620.0	1803.00	96
2010-2016	14443.197	39272.14	3926.0	8552.00	1000	558198.3	1782.00	103

Ejercicio 1 para entregar

A partir del dataset df, el cual trata de los incendios forestales, realice las siguientes tareas:

Filtrar los registros para incluir únicamente los incendios ocurridos en el estado de Idaho.

Seleccionar únicamente las columnas YEAR_, CAUSE y TOTALACRES.

Renombrar estas columnas con nombres más claros y descriptivos.

Agrupar la información por CAUSE y YEAR_.

Resumir el total de acres quemados para cada combinación de causa y año.

Elaborar una visualización que muestre los resultados de manera clara.

Ejercicio 2 para entregar

Trabajaremos con el conjunto de datos de 120 años de historia olímpica adquirido por Randi Griffin en Randi-Griffin y puesto a disposición en athlete_events.

Su tarea consiste en identificar los cinco deportes más importantes según el mayor número de medallas otorgadas en el año 2016, y luego realizar el siguiente análisis:

Genere una tabla que indique el número de medallas concedidas en cada uno de los cinco principales deportes en 2016.

Elabore una tabla que muestre la distribución de la edad de los ganadores de medallas en los cinco principales deportes en 2016.

Identifique qué equipos nacionales ganaron el mayor número de medallas en los cinco principales deportes en 2016.

Presente un resumen de la tendencia del peso de los atletas masculinos y femeninos ganadores en los cinco principales deportes en 2016.

2.4 Ordenación de datos

La ordenación de datos es una forma coherente de organizar los datos en R y puede facilitarse a través del paquete `tidyverse` que se encuentra en el ecosistema `tidyverse`.

Hay tres reglas que podemos seguir para hacer un conjunto de datos ordenado:

Reglas de datos ordenados

Cada variable debe tener su propia columna.

Cada observación debe tener su propia fila.

Cada valor debe tener su propia celda.

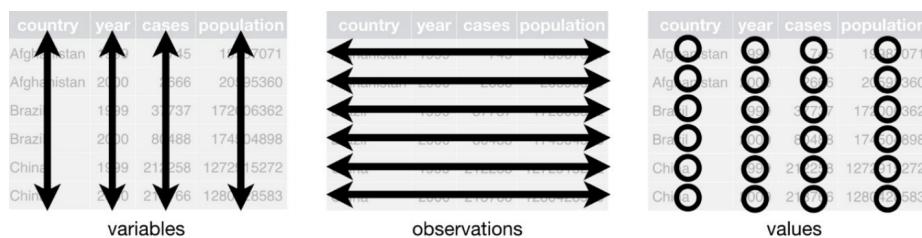


Figure 2.5: Partes de los datos

- En primer lugar, tener una estructura de datos consistente es muy importante.
- Los paquetes que forman parte de `tidyverse` (incluyendo `dplyr` y `ggplot2`) están diseñados para trabajar con datos ordenados.

Observación

Asegurar que tus datos sean uniformes facilita el procesamiento eficiente de tus datos.

- Además, colocar las variables en columnas permite facilitar la vectorización en R.
- Muchos de los conjuntos de datos que encuentre no estarán ordenados y requerirán algo de trabajo por su parte.
Puede haber muchas razones por las que un conjunto de datos no esté ordenado.
A menudo, **las personas que crearon el conjunto de datos no están familiarizadas con los principios de los datos ordenados.**
- Otra razón común por la que los conjuntos de datos no están ordenados es que los datos se organizan a menudo para facilitar algo más que el análisis.

Nota importante

Para que la introducción de datos sea lo más fácil posible, en ocasiones se suelen organizar los datos de forma poco ordenada.

Así, muchos conjuntos de datos requieren algún tipo de ordenación antes de poder empezar el análisis.

- El primer paso es averiguar cuáles son las variables y observaciones del conjunto de datos.
Esto le facilitará la comprensión de lo que deben ser las columnas y las filas.
- Además, también tendrá que resolver uno o dos problemas comunes:
deberá averiguar si una variable está repartida en varias columnas, y si una observación está dispersa en varias filas.
Estos conceptos se conocen como **reunión y dispersión**.

En esta sección veremos:

Recopilación

Distribución

Separación

Unión

2.4.1 Recopilación

- Un problema común en muchos conjuntos de datos es que los **nombres de las columnas no son variables sino valores de una variable**.
- En la figura siguiente, las columnas 1999 y 2000 son en realidad valores de la variable YEAR.
- *Cada fila de la tabla existente representa en realidad dos observaciones.*

El paquete `tidyverse` puede utilizarse para **reunir estas columnas existentes en una nueva variable**.

En este caso, tenemos que crear una nueva columna llamada YEAR y luego reunir en esta los valores existentes en las columnas 1999 y 2000.

Definición de `gather()`

La función `gather()` del paquete `tidyverse` se utiliza para transformar datos de un formato ancho (wide) a un formato largo (long).

Su objetivo principal es reunir varias columnas en una sola, creando una nueva

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

Figure 2.6: Partes de los datos

variable que identifica los nombres de las columnas originales y otra que contiene los valores asociados.

En otras palabras, gather() permite convertir columnas que representan valores en filas ordenadas, lo que facilita el análisis de datos bajo los principios de datos ordenados.

Ejemplo práctico

En este ejercicio aprenderás a utilizar la función gather() para realizar la tarea de ordenación de datos.

Descargue el archivo CountryPopulation.csv localizado en RDataSets.

Solución

- A continuación, tendrás que **nombrar la variable de la nueva columna**. Esto también se llama la clave `key`, y en este caso será la variable del año (`year`). Por último, tendrás que proporcionar el valor `value`, que es el **nombre de la variable cuyos valores se reparten por las celdas**.
- Carguemos los datos

```
library(tidyverse)

df = read_csv("data/CountryPopulation.csv")
head(df,n=5)

## # A tibble: 5 x 10
##   `Country Name` `Country Code`    `2010`  `2011`  `2012`  `2013`  `2014`
```

country	year	cases	country	1999	2000
Afghanistan	1999	745	Afghanistan	745	2666
Afghanistan	2000	2666	Brazil	37737	80488
Brazil	1999	37737	China	212258	213766
Brazil	2000	80488			
China	1999	212258			
China	2000	213766			

Figure 2.7: Ilustración de la función gather

```
##   <chr>      <chr>      <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 Aruba       ABW        101669  1.02e5  1.03e5  1.03e5  1.04e5
## 2 Afghanistan AFG        28803167 2.97e7  3.07e7  3.17e7  3.28e7
## 3 Angola      AGO        23369131 2.42e7  2.51e7  2.60e7  2.69e7
## 4 Albania     ALB        2913021  2.91e6  2.90e6  2.90e6  2.89e6
## 5 Andorra     AND        84449   8.38e4  8.24e4  8.08e4  7.92e4
## # i 3 more variables: `2015` <dbl>, `2016` <dbl>, `2017` <dbl>
```

- Utilice la función `gather` como se ve a continuación

```
df2 = gather(df,
             "2010", "2011", "2012", "2013", "2014", "2015", "2016", "2017",
             key = "YEAR",
             value = "POPULATION")
knitr::kable(head(df2, 5))
```

Country Name	Country Code	YEAR	POPULATION
Aruba	ABW	2010	101669
Afghanistan	AFG	2010	28803167
Angola	AGO	2010	23369131
Albania	ALB	2010	2913021
Andorra	AND	2010	84449

- Otra opción también sería la siguiente, cuando contamos con un gran número de columnas de este tipo, debemos usar la **expresiones regulares**.

```
years <- colnames(df)[grep("^\d{4}$", colnames(df))]

df2 <- df %>%
  gather(key = "YEAR", value = "POPULATION", all_of(years))

knitr::kable(head(df2, 5))
```

Country Name	Country Code	YEAR	POPULATION
Aruba	ABW	2010	101669
Afghanistan	AFG	2010	28803167
Angola	AGO	2010	23369131
Albania	ALB	2010	2913021
Andorra	AND	2010	84449

- Donde, en *expresiones regulares*, " $^{\wedge} \backslash d\{4\}\$$ " tiene el siguiente significado:
 - \wedge : Representa el inicio de una cadena.
 - $\backslash d$: Representa un dígito.
 - $\{4\}$: Indica que el elemento anterior ($\backslash d$, en este caso) debe aparecer exactamente 4 veces.
 - $\$$: Representa el final de una cadena.
- En resumen, " $^{\wedge} \backslash d\{4\}\$$ " coincide con cualquier **cadena que contenga exactamente 4 dígitos y no contenga ningún otro carácter adicional antes o después de los dígitos**.

2.4.2 Distribución

Concepto

La distribución es el proceso opuesto a la reunión y se aplica cuando una sola observación está fragmentada en varias filas.

Su objetivo es reorganizar los datos para que cada observación quede contenida en una única fila.

- En el diagrama siguiente, la tabla debería definir una observación de un país por año. Sin embargo, se observa que está repartida en dos filas: una para cases y otra para population.
- Para solucionar este problema, podemos utilizar la función `spread()` del paquete `tidyverse`.

Definición: Sintaxis de `spread()`

La función `spread()` del paquete `tidyverse` se utiliza para transformar un conjunto de datos de formato largo (long) a formato ancho (wide).

Su propósito es distribuir los valores de una variable en varias columnas.

Toma dos parámetros principales:

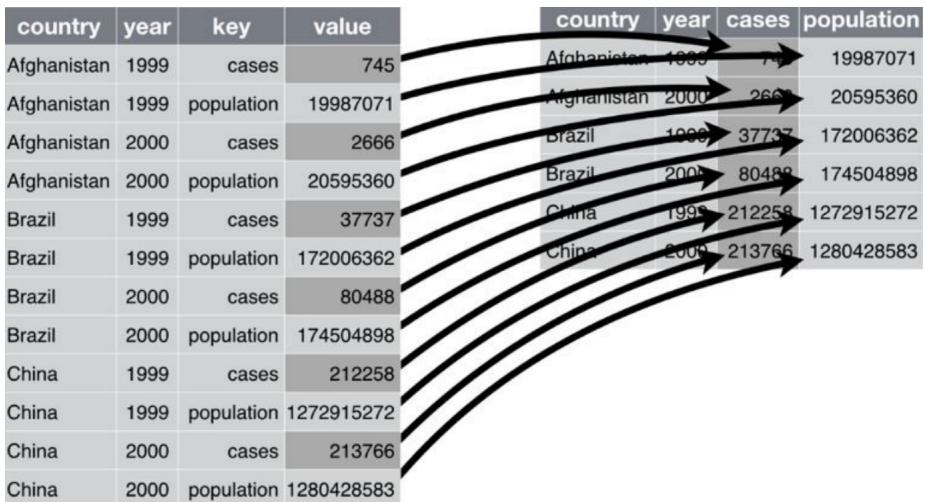


Figure 2.8: Distribuci<U+00F3>n

key: columna que contiene los nombres de las variables que se convertirán en nuevas columnas.

value: columna que contiene los valores que se ubicarán en las nuevas celdas.

- Instale el paquete `devtools` y los conjuntos de datos **DSR** utilizando el código que ve a continuación escribiendo en el panel de la consola. Alternativamente, puede utilizar el panel de paquetes para instalar los librerías

```
# Instalar
# install.packages("devtools")
# devtools::install_github("garrettman/DSR")
library(devtools)
```

- Tomemos los datos de `table2`:

```
knitr::kable(head(table2))
```

country	year	type	count
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362

- Utilice la función `spread()` para corregir este problema.

```
table2b = spread(table2, key = type, value = count)
knitr::kable(head(table2b))
```

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

2.4.3 Separación

Otro caso común es el de dos variables que se colocan en la misma columna. Por ejemplo, la hoja de cálculo siguiente tiene una columna **State-County Name** que en realidad contiene dos variables separadas por una barra.

La función `separate()` puede utilizarse para dividir una columna en varias columnas dividiendo por un separador. Por defecto, la función `separate()` buscará automáticamente cualquier carácter no alfanumérico o se puede definir un carácter específico (ver `separate()`).

- En la carpeta de **RDataSets** hay un archivo llamado **usco2005.csv**. Abra este archivo, por ejemplo con Microsoft Excel, o algún otro tipo de software de hoja de cálculo. El archivo debería tener un aspecto similar al de la captura de pantalla de abajo.
- Cargue el archivo **usco2005.csv** en **RStudio** escribiendo el código que ve a continuación en el panel de la consola

```
df = read_csv("data/usco2005.csv")
knitr::kable(head(df, 5))
```

STATE	STATEFIPS	COUNTYFIPS	FIPS	State-County Name	TP-TotPop
AL	1	1	1001	Alabama-Autauga County	48.612
AL	1	3	1003	Alabama-Baldwin County	162.586
AL	1	5	1005	Alabama-Barbour County	28.414
AL	1	7	1007	Alabama-Bibb County	21.516
AL	1	9	1009	Alabama-Blount County	55.725

- Utilice la función `separate()` para separar el contenido de la columna **State-County Name** en las columnas **StateAbbrev** y **CountyName**

	A	B	C	D	E
1	STATE	STATEFIPS	COUNTYFIPS	FIPS	State-County Name
2	AL	1	1	1001	Alabama-Autauga County
3	AL	1	3	1003	Alabama-Baldwin County
4	AL	1	5	1005	Alabama-Barbour County
5	AL	1	7	1007	Alabama-Bibb County
6	AL	1	9	1009	Alabama-Blount County
7	AL	1	11	1011	Alabama-Bullock County
8	AL	1	13	1013	Alabama-Butler County
9	AL	1	15	1015	Alabama-Calhoun County
10	AL	1	17	1017	Alabama-Chambers County
11	AL	1	19	1019	Alabama-Cherokee County
12	AL	1	21	1021	Alabama-Chilton County
13	AL	1	23	1023	Alabama-Choctaw County
14	AL	1	25	1025	Alabama-Clarke County
15	AL	1	27	1027	Alabama-Clay County
16	AL	1	29	1029	Alabama-Cleburne County
17	AL	1	31	1031	Alabama-Coffee County
18	AL	1	33	1033	Alabama-Colbert County
19	AL	1	35	1035	Alabama-Conecuh County
20	AL	1	37	1037	Alabama-Coosa County
21	AL	1	39	1039	Alabama-Covington County
22	AL	1	41	1041	Alabama-Crenshaw County
23	AL	1	43	1043	Alabama-Cullman County
24	AL	1	45	1045	Alabama-Dale County
25	AL	1	47	1047	Alabama-Dallas County
26	AL	1	49	1049	Alabama-De Kalb County
27	AL	1	51	1051	Alabama-Elmore County
28	AL	1	53	1053	Alabama-Escambia County
29	AL	1	55	1055	Alabama-Etowah County

Figure 2.9: DataSet

	A	B	C	D	E	F
1	STATE	STATEFIPS	COUNTYFIPS	FIPS	State-County Name	TP-TotPop
2	AL		1	1	1001 Alabama-Autauga County	48.612
3	AL		1	3	1003 Alabama-Baldwin County	162.586
4	AL		1	5	1005 Alabama-Barbour County	28.414
5	AL		1	7	1007 Alabama-Bibb County	21.516
6	AL		1	9	1009 Alabama-Blount County	55.725
7	AL		1	11	1011 Alabama-Bullock County	11.055
8	AL		1	13	1013 Alabama-Butler County	20.766
9	AL		1	15	1015 Alabama-Calhoun County	112.141
10	AL		1	17	1017 Alabama-Chambers County	35.46
11	AL		1	19	1019 Alabama-Cherokee County	24.522
12	AL		1	21	1021 Alabama-Chilton County	41.744
13	AL		1	23	1023 Alabama-Choctaw County	14.807
14	AL		1	25	1025 Alabama-Clarke County	27.269
15	AL		1	27	1027 Alabama-Clay County	13.964
16	AL		1	29	1029 Alabama-Cleburne County	14.46
17	AL		1	31	1031 Alabama-Coffee County	45.567
18	AL		1	33	1033 Alabama-Colbert County	54.66
19	AL		1	35	1035 Alabama-Conecuh County	13.257
20	AL		1	37	1037 Alabama-Coosa County	11.162
21	AL		1	39	1039 Alabama-Covington County	37.003
22	AL		1	41	1041 Alabama-Crenshaw County	13.727
23	AL		1	43	1043 Alabama-Cullman County	79.886
24	AL		1	45	1045 Alabama-Dale County	48.748
25	AL		1	47	1047 Alabama-Dallas County	44.366
26	AL		1	49	1049 Alabama-De Kalb County	67.271
27	AL		1	51	1051 Alabama-Elmore County	73.937
28	AL		1	53	1053 Alabama-Escambia County	38.082
29	AL		1	55	1055 Alabama-Etowah County	103.189

Figure 2.10: DataSet de usco2005

```
df2 = separate(df, "State-County Name", into = c("StateAbbrev", "CountyName"))
knitr::kable(head(df2, 5))
```

STATE	STATEFIPS	COUNTYFIPS	FIPS	StateAbbrev	CountyName	TP-TotPop
AL	1	1	1001	Alabama	Autauga	48.612
AL	1	3	1003	Alabama	Baldwin	162.586
AL	1	5	1005	Alabama	Barbour	28.414
AL	1	7	1007	Alabama	Bibb	21.516
AL	1	9	1009	Alabama	Blount	55.725

2.4.4 Unión

Definición

unite() de tidyverse combina varias columnas en una sola, aplicando un separador entre los valores. Es, conceptualmente, la operación inversa de separate().

Parámetros clave

data: marco de datos.

col: nombre de la nueva columna combinada.

...: columnas a unir (en el orden deseado).

sep: separador entre valores (por defecto “_”).

remove: si TRUE, elimina las columnas originales.

na.rm: si TRUE, omite NA al unir; si FALSE, el resultado será NA si alguna parte es NA.

Sintaxis: unite(data, col, ..., sep = “_”, remove = TRUE, na.rm = FALSE)

```
df3 = unite(df2, State_County_Name, StateAbbrev, CountyName)
knitr::kable(head(df3, 5))
```

STATE	STATEFIPS	COUNTYFIPS	FIPS	State_County_Name	TP-TotPop
AL	1	1	1001	Alabama_Autauga	48.612
AL	1	3	1003	Alabama_Baldwin	162.586
AL	1	5	1005	Alabama_Barbour	28.414
AL	1	7	1007	Alabama_Bibb	21.516
AL	1	9	1009	Alabama_Blount	55.725

Ejercicio para entregar

Considere el conjunto de datos us_state_population.tsv ubicado en la carpeta de datos de Python de github. Repita el procedimiento planteado en cada ítem de esta sección para obtener el nuevo dataframe con las nuevas columnas Year y Population. Realice unión y separación utilizando las columnas State y Code.

Chapter 3

Exploración y visualización estática de datos

Definición: Análisis Exploratorio de Datos (EDA)

El Análisis Exploratorio de Datos (EDA) es un flujo de trabajo diseñado para obtener una mejor comprensión de los datos. Se desarrolla en tres pasos:

El primero consiste en generar preguntas sobre los datos. En este paso se quiere ser lo más amplio posible, ya que en este momento no se tiene una buena idea de los datos.

A continuación, se buscan las respuestas a estas preguntas mediante la visualización, la transformación y el modelado de los datos.

Por último, refine sus preguntas o genere nuevas preguntas.

Definición: Tipos de variables en R

Los datos pueden dividirse en dos grandes grupos:

Variables con datos categóricas: Conjunto reducido de valores discretos. Ejemplo: sexo, país, color.

Variables con datos numéricos: Conjunto potencialmente infinito de valores numéricos ordenados. Ejemplo: peso, edad, estatura.

Visualización de variables

Las categóricas suelen representarse con gráficos de barras.

Las numéricas se visualizan con histogramas, boxplots.

Tanto categóricas como continuas también pueden representarse mediante gráficos creados en R.

Definición: Variación y Covariación

Variación: grado en que los valores de una misma variable cambian entre distintas observaciones o mediciones.

Covariación: describe cómo cambian conjuntamente los valores de dos o más variables, es decir, la medida en que la variación de una se asocia con la variación de otra.

Técnicas básicas de exploración gráfica

Medir variación categórica con gráfico de barras.

Medir variación continua con histograma.

Medir covariación con diagramas de caja.

Medir covariación mediante tamaño de símbolos en gráficos de dispersión.

Creación de gráficos 2D y mapas de calor (hexbin).

Generación de estadísticas de resumen.

3.1 Visualización de variables categóricas con gráficos de barras

Un gráfico de barras es una buena manera de visualizar datos categóricos. Se separa cada categoría en una barra separada y la altura de cada barra se define por el número de ocurrencias en esa categoría.

Para los ejercicios de esta sección requieren los siguientes paquetes: `readr`, `dplyr`, `ggplot2`. Pueden cargarse desde el panel de paquetes, el panel de la consola o un script. Estos paquetes están en el paquete de `tidyverse`.

- Utilice la función `read_csv()` para cargar el conjunto de datos de incendios forestales en un marco `dataframe`.

```
library(dplyr)
library(readr)
library(ggplot2)

datos <- read_csv("data/StudyArea.csv")
```

- Visualicemos las 5 primeras observaciones

```
head(datos)
```

3.1. VISUALIZACIÓN DE VARIABLES CATEGÓRICAS CON GRÁFICOS DE BARRAS55

```
## # A tibble: 6 x 14
##   FID ORGANIZATI UNIT SUBUNIT SUBUNIT2      FIRENAME CAUSE YEAR_
##   <dbl> <chr>     <chr> <chr>     <chr>     <chr> <dbl>
## 1     0 FWS       81682 USCADBR San Diego Bay ~ PUMP HO~ Human 2001
## 2     1 FWS       81682 USCADBR San Diego Bay ~ I5      Human 2002
## 3     2 FWS       81682 USCADBR San Diego Bay ~ SOUTHBAY Human 2002
## 4     3 FWS       81682 USCADBR San Diego Bay ~ MARINA  Human 2001
## 5     4 FWS       81682 USCADBR San Diego Bay ~ HILL    Human 1994
## 6     5 FWS       81682 USCADBR San Diego Bay ~ IRRIGAT~ Human 1994
## # i 6 more variables: STARTDATED <chr>, CONTRDATED <chr>,
## #   OUTDATED <chr>, STATE <chr>, STATE_FIPS <dbl>, TOTALACRES <dbl>
```

- Ahora, la estructura de los datos

```
str(datos)
```

```
## spc_tbl_ [439,362 x 14] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
## $ FID      : num [1:439362] 0 1 2 3 4 5 6 18 20 21 ...
## $ ORGANIZATI: chr [1:439362] "FWS" "FWS" "FWS" "FWS" ...
## $ UNIT     : chr [1:439362] "81682" "81682" "81682" "81682" ...
## $ SUBUNIT  : chr [1:439362] "USCADBR" "USCADBR" "USCADBR" "USCADBR" ...
## $ SUBUNIT2 : chr [1:439362] "San Diego Bay National Wildlife Refuge" "San Diego Bay National ...
## $ FIRENAME : chr [1:439362] "PUMP HOUSE" "I5" "SOUTHBAY" "MARINA" ...
## $ CAUSE    : chr [1:439362] "Human" "Human" "Human" "Human" ...
## $ YEAR_    : num [1:439362] 2001 2002 2002 2001 1994 ...
## $ STARTDATED: chr [1:439362] "1/1/01 0:00" "5/3/02 0:00" "6/1/02 0:00" "7/12/01 0:00" ...
## $ CONTRDATED: chr [1:439362] "1/1/01 0:00" "5/3/02 0:00" "6/1/02 0:00" "7/12/01 0:00" ...
## $ OUTDATED : chr [1:439362] NA NA NA NA ...
## $ STATE    : chr [1:439362] "California" "California" "California" "California" ...
## $ STATE_FIPS: num [1:439362] 6 6 6 6 6 6 6 6 6 ...
## $ TOTALACRES: num [1:439362] 0.1 3 0.5 0.1 1 0.1 3 0.1 0.1 0.1 ...
## - attr(*, "spec")=
##   .. cols(
##     .. FID = col_double(),
##     .. ORGANIZATI = col_character(),
##     .. UNIT = col_character(),
##     .. SUBUNIT = col_character(),
##     .. SUBUNIT2 = col_character(),
##     .. FIRENAME = col_character(),
##     .. CAUSE = col_character(),
##     .. YEAR_ = col_double(),
##     .. STARTDATED = col_character(),
##     .. CONTRDATED = col_character(),
##     .. OUTDATED = col_character(),
##     .. STATE = col_character(),
```

56 CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS

```
## .. STATE_FIPS = col_double(),  
## .. TOTALACRES = col_double()  
## ..)  
## - attr(*, "problems")=<externalptr>
```

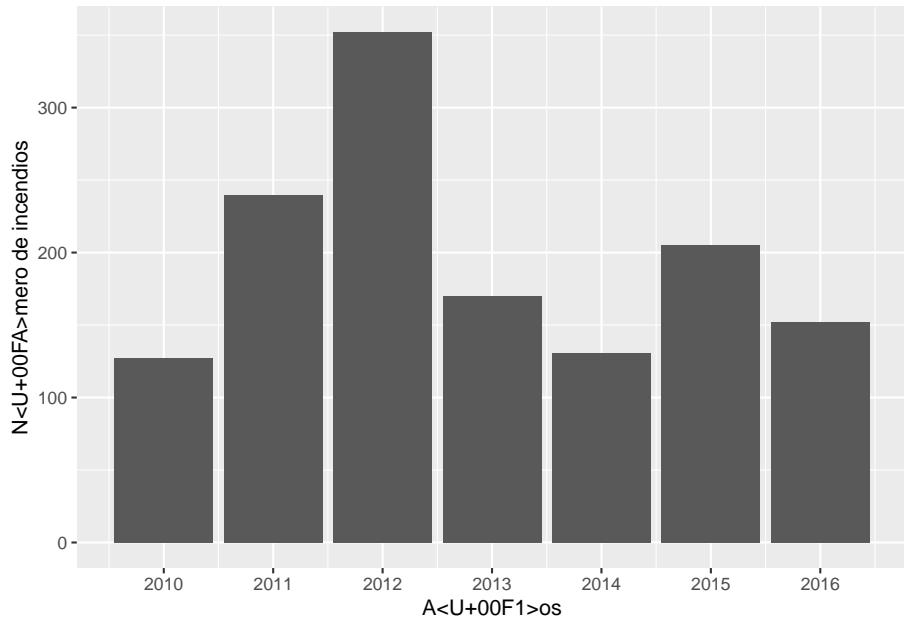
- Para este análisis, filtraremos los datos de modo que solo estén representados los incendios que quemaron más de 1000 acres (4046.86 metros cuadrados o 0.4047 hectáreas) en los años 2010 a 2016.

```
df <- datos %>%  
  filter(TOTALACRES >= 1000,  
         YEAR_ %in% c(2010, 2011, 2012, 2013, 2014, 2015, 2016))
```

- Añada el código que se ve a continuación para filtrar los datos y enviar los resultados a un gráfico de barras.

```
df %>%  
  ggplot(mapping = aes(x = YEAR_)) +  
  geom_bar() +  
  scale_x_continuous(breaks = c(2010, 2011, 2012, 2013, 2014, 2015, 2016)) +  
  labs(x = 'Años',  
       y = 'Número de incendios') -> barras
```

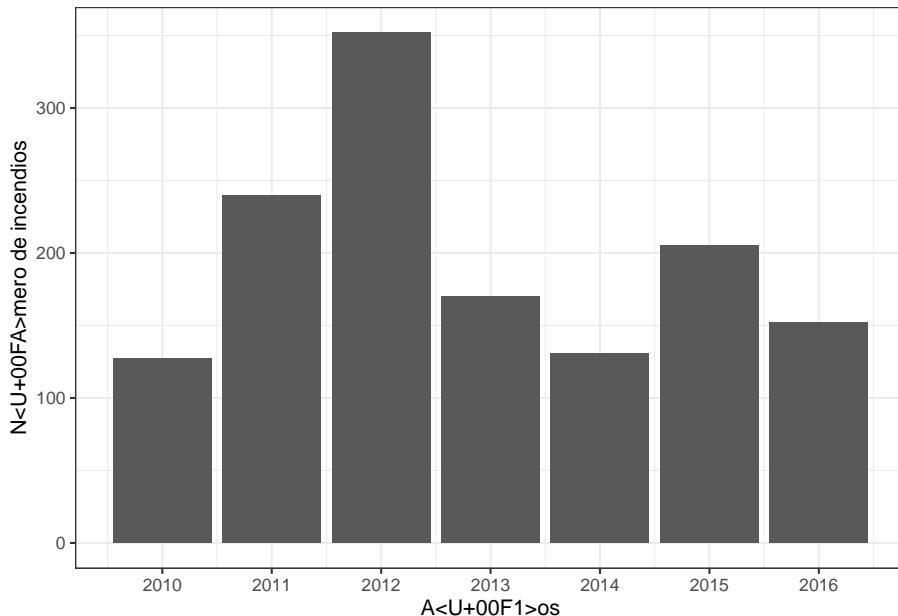
barras



3.1. VISUALIZACIÓN DE VARIABLES CATEGÓRICAS CON GRÁFICOS DE BARRAS57

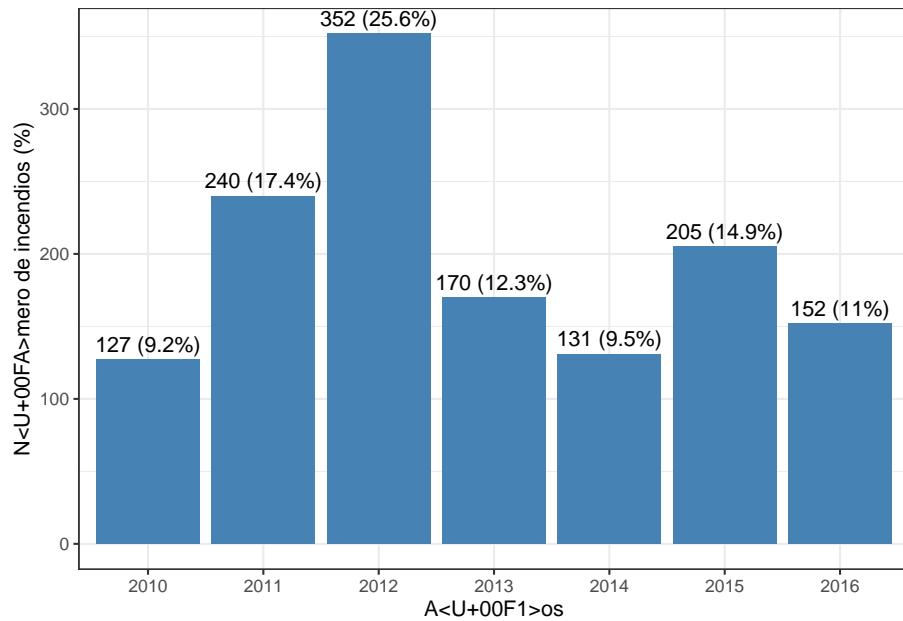
- Si queremos incorporar temáticas para una mejor visualización, tenemos 8 temas que estan incluidos en ggplot2, los cuales son `theme_bw`, `theme_classic`, `theme_dark`, `theme_gray`, `theme_light`, `theme_linedraw`, `theme_minimal`, `theme_void`. Añade el código que ves a continuación para cambiar la faceta a `theme_bw`.

```
barras+  
  theme_bw()
```



- Otra forma de realizar el gráfico de barras es la siguiente

```
df %>%  
  filter(TOTALACRES >= 1000,  
         YEAR_ %in% c(2010, 2011, 2012, 2013, 2014, 2015, 2016)) %>%  
  count(YEAR_) %>%  
  mutate(Porcentaje = (n / sum(n))*100) %>%  
  ggplot(mapping = aes(x = factor(YEAR_), y=n))+  
  geom_col(fill = "steelblue") +  
  geom_text(aes(label = paste0(n, " (", round(Porcentaje,1), "%)")),  
            vjust = -0.5, size = 4) +  
  labs(x = 'Años',  
       y = 'Número de incendios (%)')+  
  theme_bw()
```



3.2 Visualización de variable continua con un histograma

La distribución de una variable continua se puede medir con el uso de un histograma. En este ejercicio crearás un histograma de los acres quemados en incendios forestales.

Siguiendo los datos de incendios forestales del 2010 al 2016:

- Introduzca el marco de datos y utilice la función `select()` para limitar las columnas y filtrar las filas para que sólo se incluyan los incendios superiores a 1.000 acres. Dado que tenemos un gran número de incendios forestales que quemaron sólo un pequeño número de acres, nos centraremos en los incendios que son un poco más grandes en este caso.

```
df %>%
  select(ORGANIZATI, STATE, YEAR_, TOTALACRES, CAUSE) %>%
  filter(TOTALACRES >= 1000)
```

```
## # A tibble: 1,377 x 5
##   ORGANIZATI STATE     YEAR_ TOTALACRES CAUSE
##   <chr>       <chr>     <dbl>      <dbl>   <chr>
```

3.2. VISUALIZACIÓN DE VARIABLE CONTINUA CON UN HISTOGRAMA59

```

##  1 FWS      Washington 2010      2755 Natural
##  2 FWS      Idaho     2010      1801 Undetermined
##  3 FWS      Montana   2010      27898 Natural
##  4 FWS      Montana   2012      9468 Natural
##  5 FWS      Washington 2012      1141 Human
##  6 FWS      Oregon    2010      3083 Human
##  7 FWS      Oregon    2013      51015 Undetermined
##  8 FWS      Washington 2016      171915 Human
##  9 FWS      Arizona   2015      6000 Natural
## 10 FWS     Arizona   2016      1624 Natural
## # i 1,367 more rows

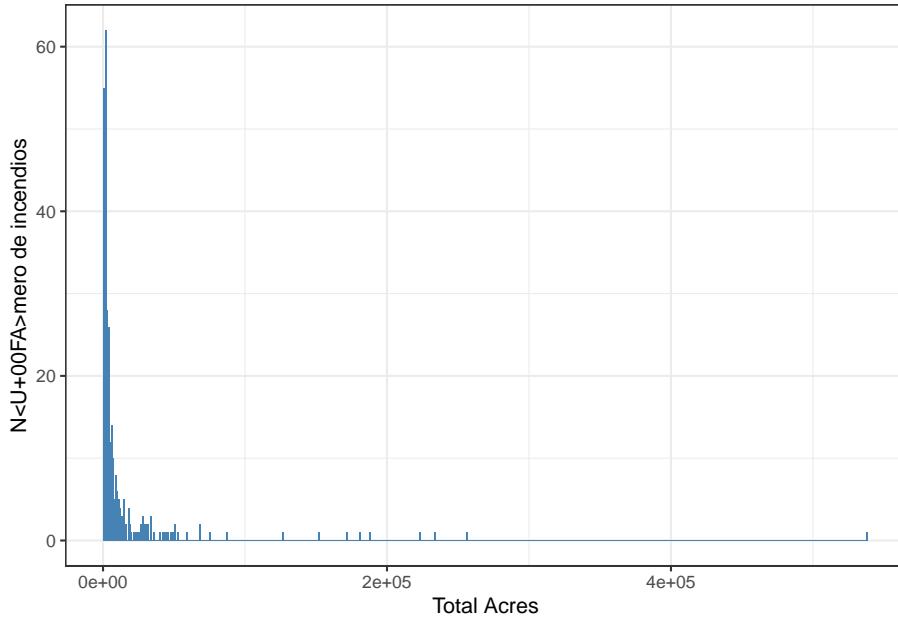
```

- Cree el histograma utilizando `ggplot()` con `geom_hist()` y un tamaño de la celda de 500. Los datos están claramente sesgados hacia el extremo inferior del número de acres quemados.

```

df %>%
  select(ORGANIZATI, STATE, YEAR_, TOTALACRES, CAUSE) %>%
  filter(TOTALACRES >= 1000) %>%
  slice(1:300) %>%
  ggplot(mapping = aes(x=TOTALACRES)) +
  geom_histogram( binwidth=1000, fill = "steelblue") +
  labs(x = "Total Acres", y = "Número de incendios")+
  theme_bw()

```



Ejercicio para entregar

Reconstruya el histograma utilizando un tamaño de celda de 5000. ¿Cuál es el efecto en la salida?

3.3 Visualización de la covariación con gráficos de caja

Los gráficos de caja proporcionan una representación visual de la dispersión de los datos de una variable. Estos gráficos muestran el rango de valores de una variable junto con la mediana y los cuartiles. Siga las instrucciones proporcionadas a continuación para crear un gráfico de caja que mide la covariación entre la organización y la superficie total quemada.

Seguimos con los datos de incendios forestales:

3.4 Visualización de la covariación con el gráfico de burbuja

La función `geom_count()` puede utilizarse con `ggplot()` para medir la covariación entre variables utilizando diferentes tamaños de símbolos. Siga las instrucciones proporcionadas a continuación para medir la covariación entre la organización y la causa del incendio forestal utilizando el tamaño del símbolo.

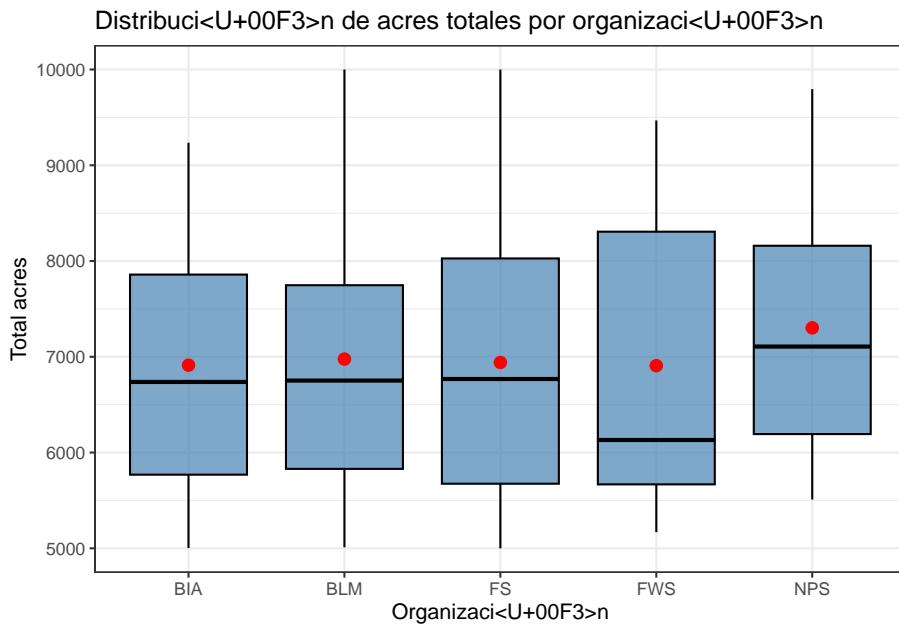
Seguimos con los datos de incendios forestales:

- Extraiga el marco de datos y filtre las filas para que sólo se incluyan los **incendios de entre 5000 y 10000 acres**. A continuación, agrupe los datos por organización. La columna ORGANIZATI del conjunto de datos contiene datos categóricos de los **organismos del gobierno federal de EE.UU.** que han tenido tierras afectadas por incendios forestales. Por último utilice `ggplot()` con `geom_boxplot()` para crear un `boxplot` que muestre la distribución de incendios forestales por organización

```
df %>%
  filter(TOTALACRES >= 5000 & TOTALACRES <= 10000) %>%
  group_by(ORGANIZATI) %>%
  ggplot(aes(x = ORGANIZATI, y = TOTALACRES)) +
  geom_boxplot(fill = "steelblue", alpha = 0.7, color = "black") +
  stat_summary(fun = mean, geom = "point", shape = 20, size = 4, color = "red") +
  labs(
    x = "Organización",
```

3.4. VISUALIZACIÓN DE LA COVARIACIÓN CON EL GRÁFICO DE BURBUJA61

```
y = "Total acres",
title = "Distribución de acres totales por organización"
) +
theme_bw()
```



Ejercicio para entregar

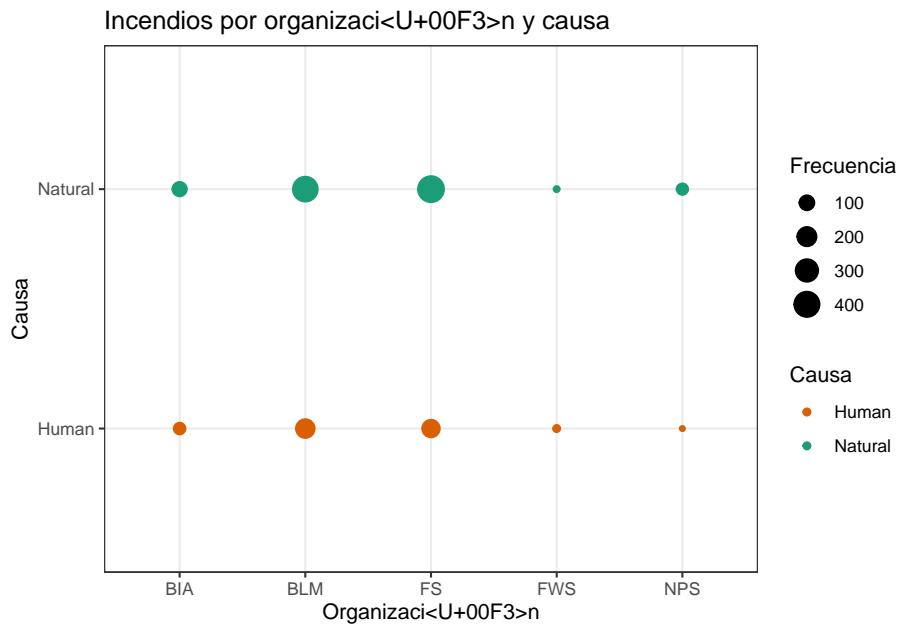
Puede crear también un nuevo boxplot que mapee la covariación de CAUSE y TOTALACRES

- Extraiga el marco de datos y filtre las filas para que sólo se incluyan los **incendios forestales que se originaron por causas naturales o humanas**. Esto eliminará cualquier registro que son desconocidos o tienen valores perdidos. A continuación, utilice `geom_count()` para crear un gráfico de símbolos graduados basado en el número de incendios por organización.

```
df %>%
filter(CAUSE %in% c("Natural", "Human")) %>%
ggplot(aes(x = ORGANIZATI, y = CAUSE)) +
geom_count(aes(color = CAUSE), show.legend = TRUE) +
scale_color_manual(values = c("Natural" = "#1b9e77", "Human" = "#d95f02")) +
labs(x = "Organización",
y = "Causa",
```

62CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS

```
color = "Causa",
size = "Frecuencia",
title = "Incendios por organización y causa") +
theme_bw()
```



- También puede obtener un recuento exacto del número de incendios por organización y causa

```
df %>%
  count(ORGANIZATI, CAUSE)
```

```
## # A tibble: 11 x 3
##   ORGANIZATI CAUSE     n
##   <chr>      <chr>    <int>
## 1 BIA        Human     49
## 2 BIA        Natural    91
## 3 BLM        Human    187
## 4 BLM        Natural   386
## 5 FS         Human    158
## 6 FS         Natural   431
## 7 FWS        Human     10
## 8 FWS        Natural     7
## 9 FWS        Undetermined  6
```

```
## 10 NPS      Human      6
## 11 NPS      Natural    46
```

3.5 Visualización 2D de contenedores y hexágonos

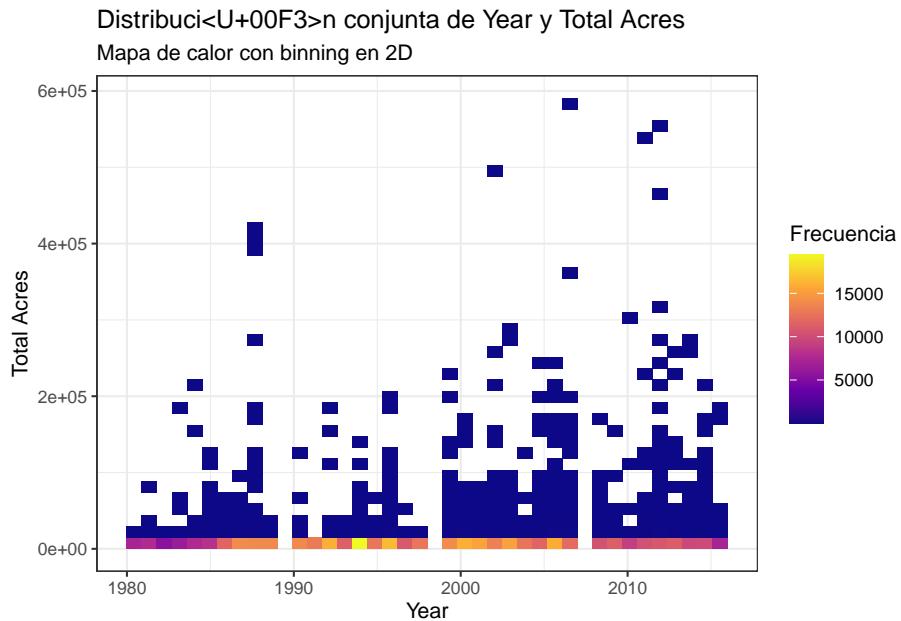
También puede utilizar gráficos 2D de contenedores y hexadecimales como una forma alternativa de ver la distribución de dos variables. Siga las instrucciones que se proporcionan a continuación para crear gráficos 2D de cubos y hexágonos que visualicen la relación entre el año y la superficie total quemada

- Utilice la función `read_csv()` para cargar el conjunto de datos en un marco de datos

```
df <- read_csv("data/StudyArea.csv")
```

- Cree un mapa 2D con `YEAR_` en el eje *x* y `TOTALACRES` en el eje *y*

```
df %>%
  ggplot(aes(x = YEAR_, y = TOTALACRES)) +
  geom_hex(bins = 40) +
  scale_fill_viridis_c(option = "plasma", name = "Frecuencia") +
  labs(
    x = "Year",
    y = "Total Acres",
    title = "Distribución conjunta de Year y Total Acres",
    subtitle = "Mapa de calor con binning en 2D"
  ) +
  theme_bw()
```



El gráfico muestra la distribución conjunta de los años (YEAR_) y el número total de acres (TOTALACRES) mediante un mapa de calor con binning en dos dimensiones. Se observa que la mayor concentración de registros se encuentra en valores bajos de acres (menores a 100.000), lo que indica que la mayoría de las observaciones corresponden a superficies relativamente pequeñas. A su vez, existen pocos casos con valores muy altos (entre 300.000 y más de 600.000 acres), los cuales aparecen como rectángulos aislados y de baja frecuencia, reflejando la presencia de outliers. Finalmente, la acumulación de registros en casi todos los años sugiere un patrón consistente en el tiempo, con algunos picos más notorios después del año 2000.

3.6 Visualización de estadísticas (resumen de los datos)

Otra técnica básica para realizar un análisis exploratorio de datos es generar varias estadísticas de resumen sobre un conjunto de datos. **R** incluye una serie de funciones individuales para generar estadísticas de resumen específicas o puede utilizar la función `summarise()` para generar un conjunto de estadísticas de resumen

- Carguemos nuevamente la data de incendios forestales

```
datos <- read_csv("data/StudyArea.csv")
```

- Restringir la lista de columnas y filtra la lista para incluir sólo los incendios forestales de más de 1000 acres

```
df <- datos %>%
  select(ORGANIZATI, STATE, YEAR_, TOTALACRES, CAUSE) %>%
  filter(TOTALACRES >= 1000)
```

- Hagamos un resumen estadístico de la columna TOTALACRES

```
df %>%
  summarise(n = length(TOTALACRES),
            media = mean(TOTALACRES),
            ds = sd(TOTALACRES),
            mediana = median(TOTALACRES),
            q1 = quantile(TOTALACRES)[2],
            q3 = quantile(TOTALACRES)[4],
            RIC = IQR(TOTALACRES),
            minimo = min(TOTALACRES),
            maximo = max(TOTALACRES)) -> resumen_totalacres
```

El análisis se realizó sobre **7288 registros**, con un valor promedio cercano a **10813** y una gran variabilidad, dado que la desviación estándar es de aproximadamente **28579**. El **50% de los datos** se encuentra en torno a **3240**, lo que indica que la mitad de las observaciones están por debajo de ese valor. El **25% más bajo** de los registros toma valores hasta **1670**, mientras que el **25% más alto** supera los **8283**. El rango intermedio entre el 25% inferior y el 75% superior es de aproximadamente **6613**, reflejando una fuerte dispersión en los datos. Finalmente, los valores observados van desde un mínimo de **1000** hasta un máximo de **590620**, lo que muestra la presencia de valores extremos mucho más altos que la mayoría de las observaciones.

3.7 Visualización de un gráfico de correlación

Es una representación gráfica de una matriz de correlaciones entre varias variables numéricas.

- Se muestran en forma de mapa de calor (heatmap), círculos, o elipses que varían en tamaño y color según la intensidad y signo de la correlación.
- Su objetivo es facilitar la identificación de relaciones lineales fuertes, tanto positivas como negativas, entre todas las variables del conjunto de datos.

66CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS

- Carguemos los datos de la banca

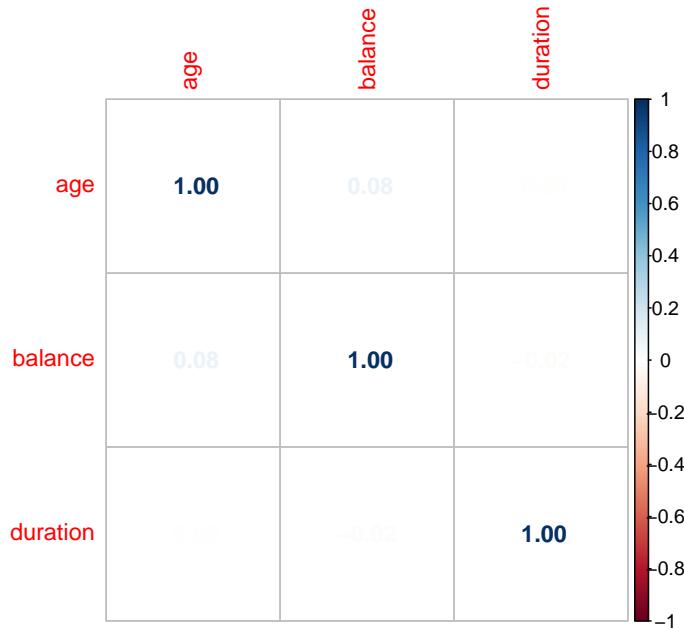
```
bank <- read_delim("https://raw.githubusercontent.com/cdeoroaguado/Datos/refs/heads/main/bank.csv", delim = ";")  
  
head(bank, n=5)  
  
## # A tibble: 5 x 17  
##   age job marital education default balance housing loan contact  
##   <dbl> <chr> <chr> <chr> <dbl> <chr> <chr> <chr>  
## 1    30 unem~ married primary no     1787 no      no cellul~  
## 2    33 serv~ married secondary no    4789 yes     yes cellul~  
## 3    35 mana~ single tertiary no    1350 yes     no cellul~  
## 4    30 mana~ married tertiary no    1476 yes     yes unknown  
## 5    59 blue~ married secondary no      0 yes     no unknown  
## # i 8 more variables: day <dbl>, month <chr>, duration <dbl>,  
## #   campaign <dbl>, pdays <dbl>, previous <dbl>, outcome <chr>,  
## #   y <chr>
```

- Selecciona las variables numéricas

```
# Seleccionar las variables numéricas  
num_vars <- bank %>%  
  select(age, balance, duration)
```

- Hagamos la matriz de correlación y el gráfico heatmap con las variables numéricas

```
# libreria  
library(corrplot)  
  
# Calcular matriz de correlación  
matriz_cor <- cor(num_vars, use = "complete.obs")  
  
print(matriz_cor)  
  
##           age      balance      duration  
## age 1.000000000 0.08382014 -0.002366889  
## balance 0.083820142 1.00000000 -0.015949918  
## duration -0.002366889 -0.01594992 1.000000000  
  
# Heatmap  
corrplot(matriz_cor, method="number")
```



Vemos que no hay correlación entre ellos.

3.8 Visualización de un gráfico de dispersión

Un diagrama de dispersión es un gráfico en el que los valores de dos variables se trazan a lo largo de dos ejes, y el patrón de los puntos resultantes revela cualquier correlación presente

- Cargar el contenido del archivo `StudyArea.csv` en un marco de datos

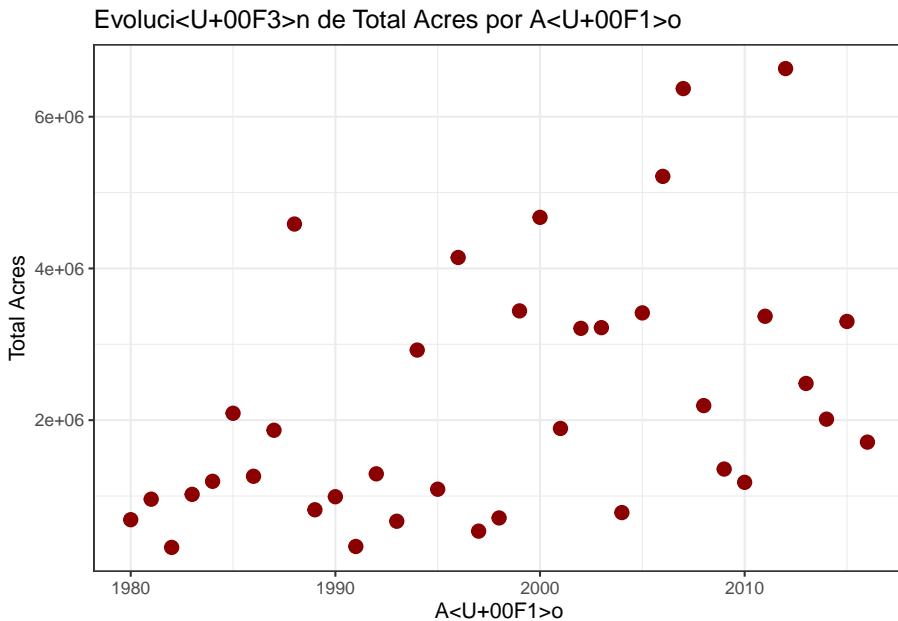
```
df <- read_csv("data/StudyArea.csv")
```

- Utilice `ggplot()` para crear un gráfico de dispersión con el año en el eje *x* y el total de acres quemados en el eje *y*.

```
df %>%
  select(ORGANIZATI, YEAR_, TOTALACRES) %>%
  group_by(YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES, na.rm = TRUE)) %>%
  ggplot(aes(x = YEAR_, y = totalacres)) +
  geom_point(color = "darkred", size = 3) +
  labs(title = "Evolución de Total Acres por Año",
```

68CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS

```
x = "Año",
y = "Total Acres") +
theme_bw()
```

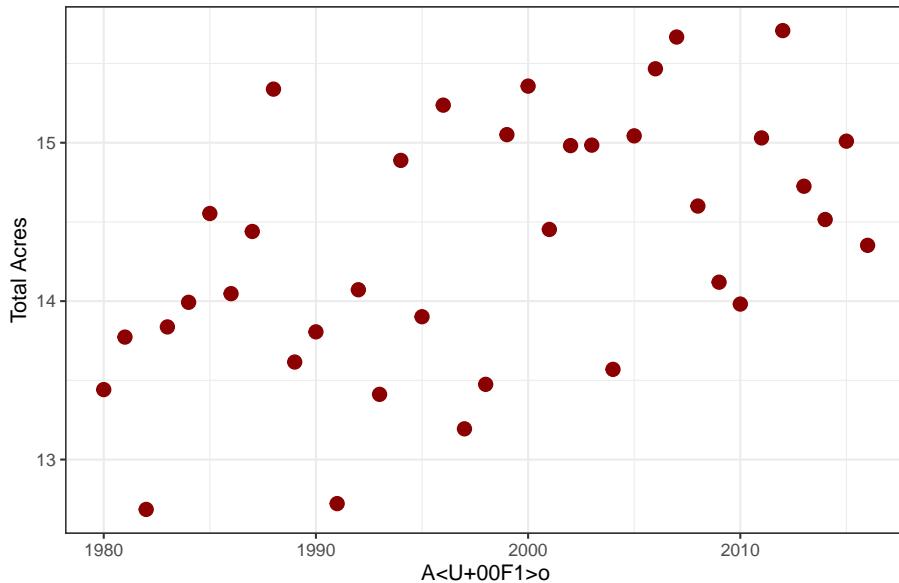


- Hay ocasiones en las que tiene sentido utilizar las escalas logarítmicas en los cuadros y gráficos. Una de las razones es responder a la asimetría hacia los valores grandes, es decir, los casos en los que uno o unos pocos puntos son mucho más grandes que el grueso de los datos. En el gráfico que acabamos de crear hay un par de puntos que entran en esta categoría en el eje Y. Vuelva a crear el gráfico, pero esta vez utilice la función `log()` en la columna `totalacres`

```
df %>%
  select(ORGANIZATI, YEAR_, TOTALACRES) %>%
  group_by(YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES, na.rm = TRUE)) %>%
  ggplot(aes(x = YEAR_, y = log(totalacres))) +
  geom_point(color = "darkred", size = 3) +
  labs(title = "Evolución de Total Acres por Año",
       x = "Año",
       y = "Total Acres") +
  theme_bw()
```

3.9. VISUALIZACIÓN DEL GRÁFICO DE DISPERSIÓN CON LINEA DE TENDENCIA69

Evolucion de Total Acres por Ao



3.9 Visualización del gráfico de dispersión con linea de tendencia

Los gráficos construidos con `ggplot()` pueden tener más de una geometría. Es habitual añadir una línea de predicción (regresión) al gráfico.

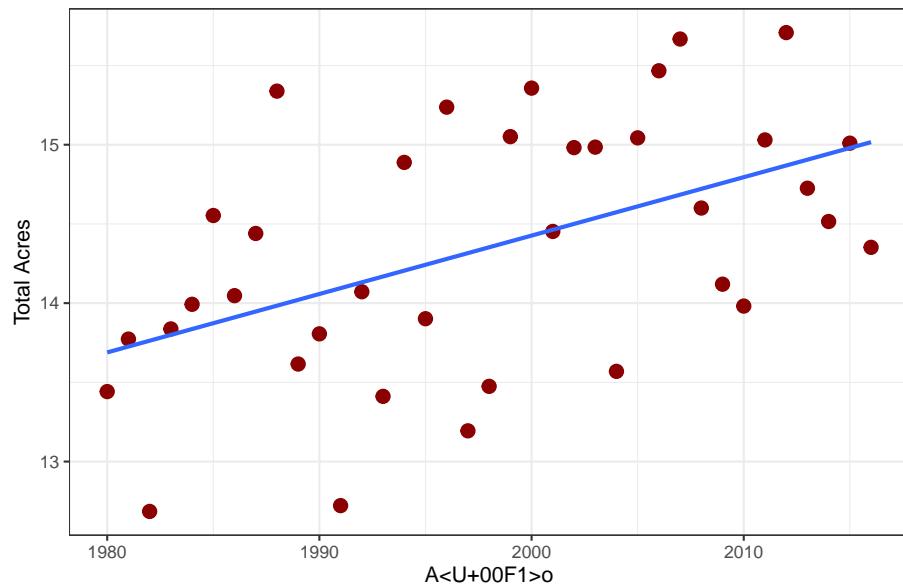
Hay varias maneras de añadir una línea de regresión al gráfico de dispersión, una de ellas es utilizar la función `geom_smooth()`

- Regresion con `method = lm`

```
df %>%
  select(ORGANIZATI, YEAR_, TOTALACRES) %>%
  group_by(YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES, na.rm = TRUE)) %>%
  ggplot(aes(x = YEAR_, y = log(totalacres))) +
  geom_point(color = "darkred", size = 3) +
  geom_smooth(method=lm, se=FALSE) +
  labs(title = "Evolución de Total Acres por Año",
       x = "Año",
       y = "Total Acres") +
  theme_bw()
```

70CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS

Evoluci<U+00F3>n de Total Acres por A<U+00F1>o

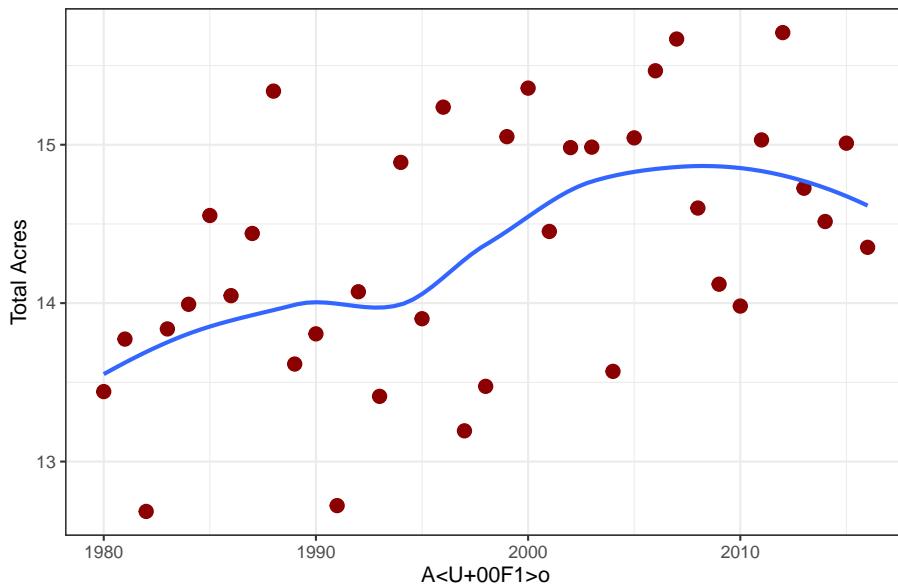


- Regresion con `method = loess`

```
df %>%
  select(ORGANIZATI, YEAR_, TOTALACRES) %>%
  group_by(YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES, na.rm = TRUE)) %>%
  ggplot(aes(x = YEAR_, y = log(totalacres))) +
  geom_point(color = "darkred", size = 3) +
  geom_smooth(method = "loess", se=FALSE) +
  labs(title = "Evolución de Total Acres por Año",
       x = "Año",
       y = "Total Acres") +
  theme_bw()
```

3.9. VISUALIZACIÓN DEL GRÁFICO DE DISPERSIÓN CON LINEA DE TENDENCIA71

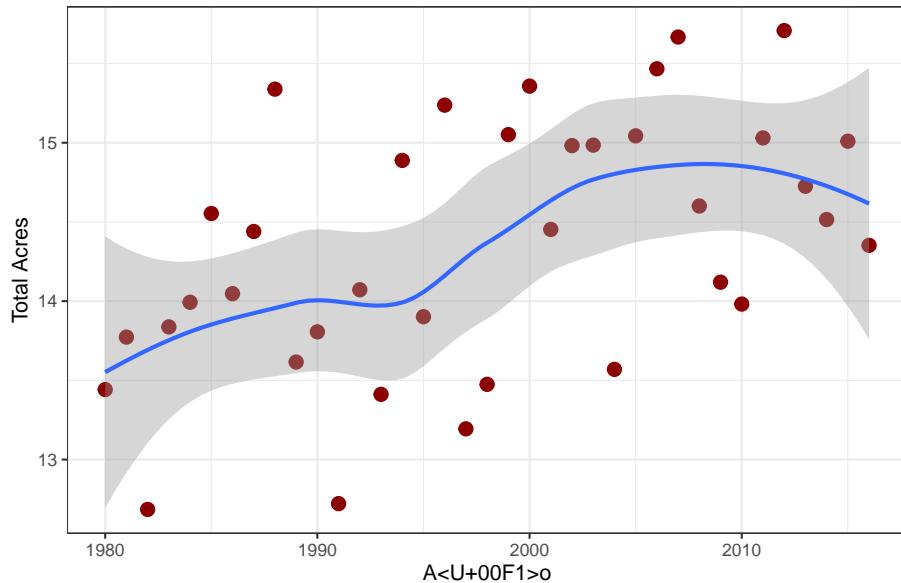
Evolucion de Total Acres por Ao



- Regresion con `method = loess` con intervalo de confianza

```
df %>%
  select(ORGANIZATI, YEAR_, TOTALACRES) %>%
  group_by(YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES, na.rm = TRUE)) %>%
  ggplot(aes(x = YEAR_, y = log(totalacres))) +
  geom_point(color = "darkred", size = 3) +
  geom_smooth(method = "loess", se=TRUE) +
  labs(title = "Evolución de Total Acres por Año",
       x = "Año",
       y = "Total Acres") +
  theme_bw()
```

Evolución de Total Acres por Año

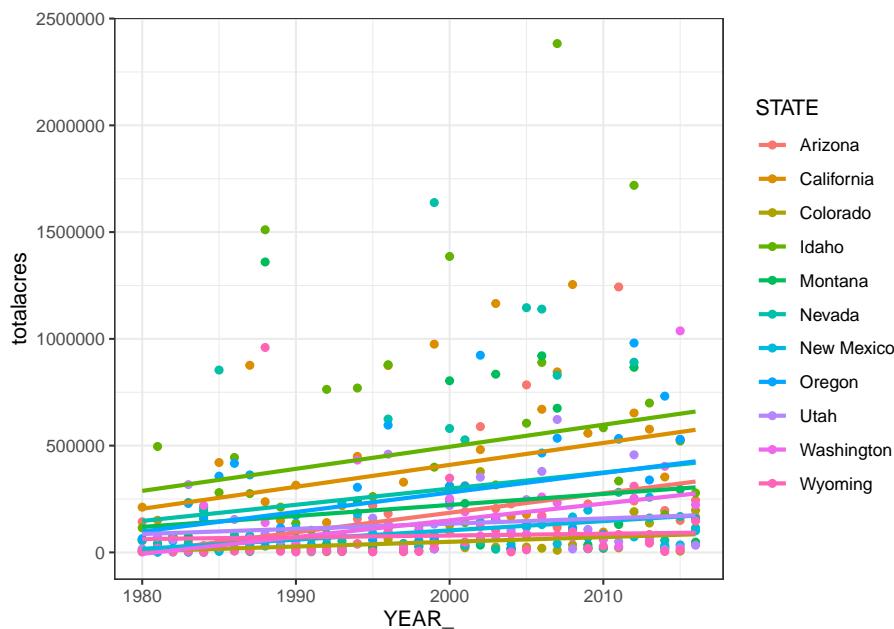


3.10 Visualización de trazado de categorías

En lugar de representar gráficamente todo el conjunto de incendios forestales, es posible que se quiera comprender mejor las tendencias por estado. En este paso creará un nuevo gráfico de dispersión que visualice las tendencias de los incendios forestales a lo largo del tiempo por estado

- Reagrupar el marco de datos de los incendios forestales por estado y año, despues resuma los grupos por el total de hectáreas quemadas y Añade un parámetro de parámetro de color a la función `aes()` para que los puntos y la línea de regresión se mapeen de acuerdo con el estado en el que se produjeron

```
df %>%
  group_by(STATE, YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES)) %>%
  ggplot(aes(x=YEAR_, y=totalacres, colour=STATE)) +
  geom_point(aes(colour = STATE)) +
  stat_smooth(method=lm, se=FALSE) +
  theme_bw()
```

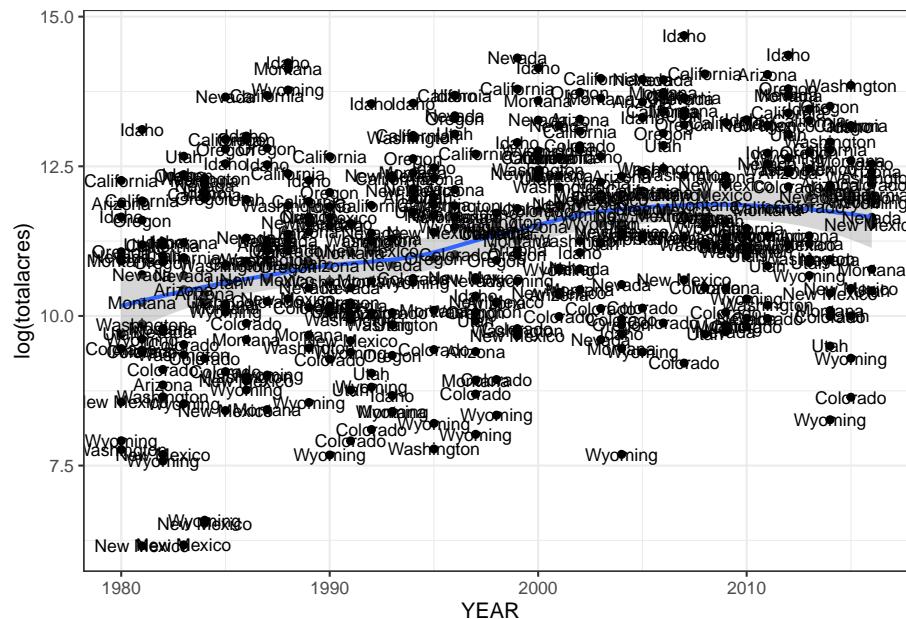


3.11 Visualizacion etiquetando nombres

Puede añadir etiquetas a su gráfico mediante la función `geom_text()` o la función `geom_label()`

```
df %>%
  group_by(STATE, YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES)) %>%
  ggplot(aes(x=YEAR_, y=log(totalacres))) +
  geom_point() +
  geom_smooth(method=loess, se=TRUE) +
  geom_text(aes(label=STATE), size=3) +
  theme_bw()
```

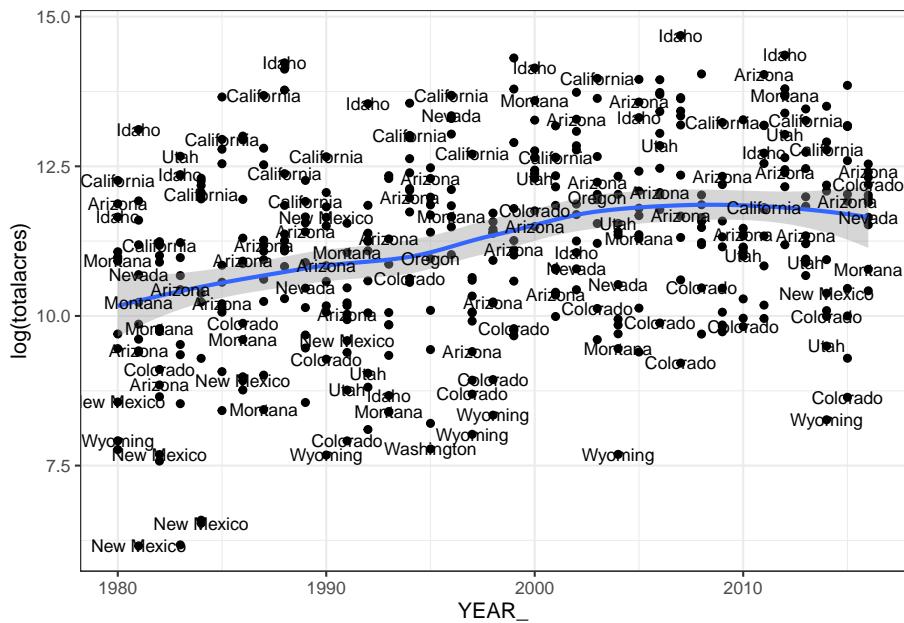
74 CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS



La pantalla está extremadamente desordenada así que vamos a ajustar algunos parámetros para hacer esto más fácil de leer.

- Puede utilizar el parámetro `check_overlap` para eliminar cualquier etiqueta superpuesta. Actualice su código como se ve a continuación

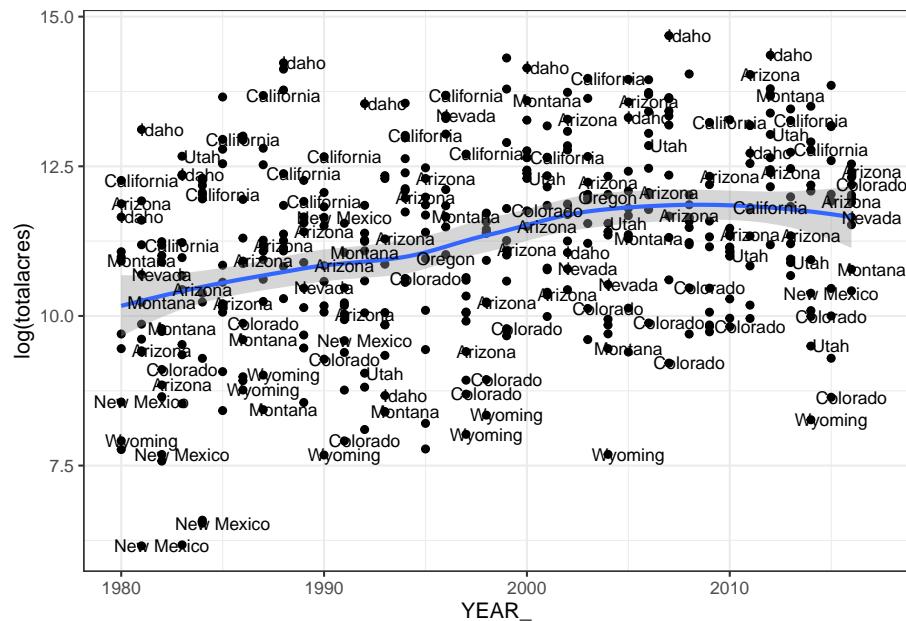
```
df %>%
  group_by(STATE, YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES)) %>%
  ggplot(aes(x=YEAR_, y=log(totalacres))) +
  geom_point() +
  geom_smooth(method=loess, se=TRUE) +
  geom_text(aes(label=STATE), size=3, check_overlap = TRUE) +
  theme_bw()
```



- Esto se ve un poco mejor, pero si se cambia el tamaño de la etiqueta a 2 se reducirá aún más reducir el desorden y la superposición, mientras que esperamos que siga siendo legible. Habrás observado que las etiquetas se sitúan directamente sobre los temas. Puede utilizar los parámetros `nudge_x` y `nudge_y` para mover las etiquetas en relación con el punto. Utilice `nudge_x` como se ve a continuación para ver cómo esto mueve las etiquetas horizontalmente

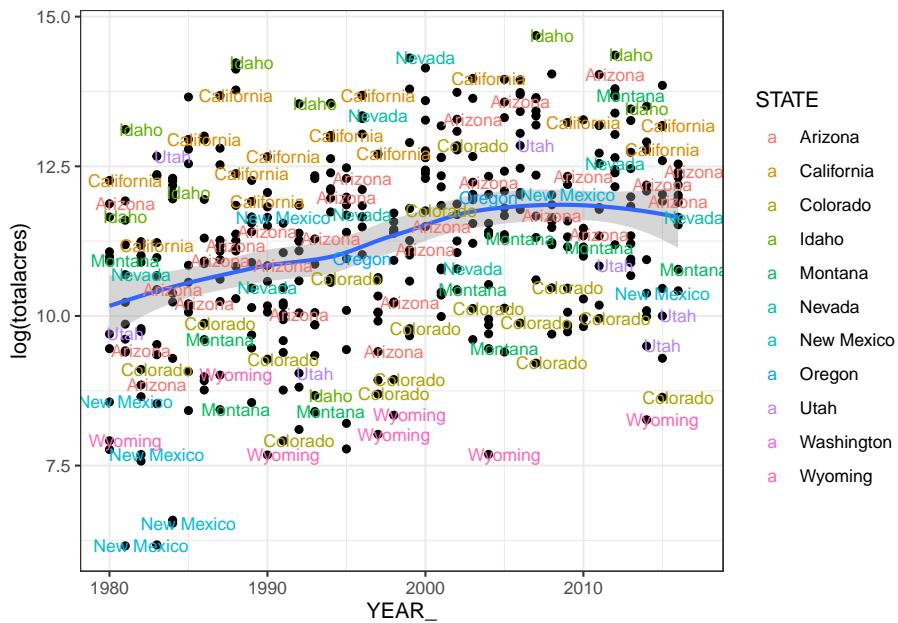
```
df %>%
  group_by(STATE, YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES)) %>%
  ggplot(aes(x=YEAR_, y=log(totalacres))) +
  geom_point() +
  geom_smooth(method=loess, se=TRUE) +
  geom_text(aes(label=STATE), size=3, check_overlap = TRUE, nudge_x = 1.0) +
  theme_bw()
```

76CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS



- También puede colorear las etiquetas por categoría añadiendo el parámetro de color a la función `aes()` para `geom_text()`

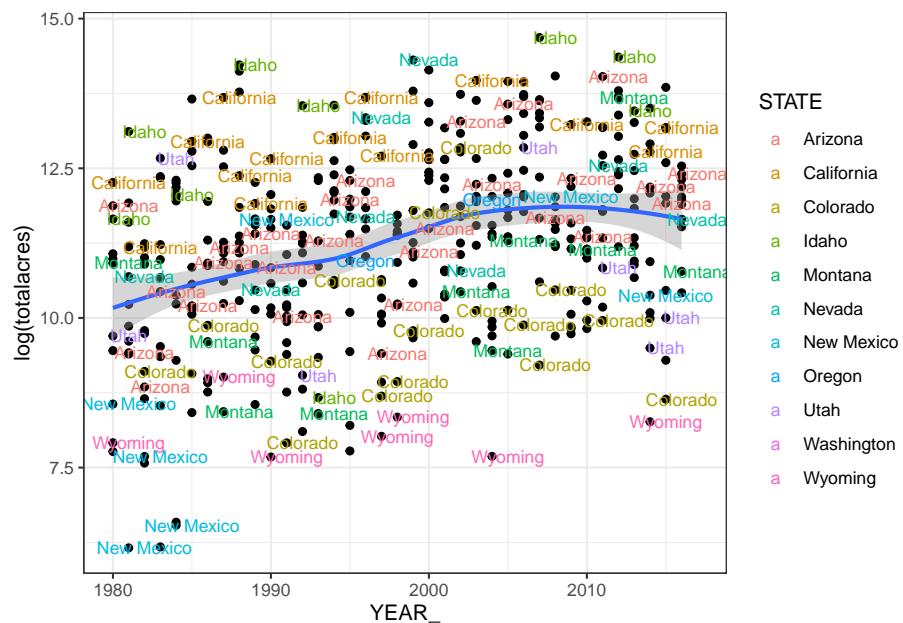
```
df %>%
  group_by(STATE, YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES)) %>%
  ggplot(aes(x=YEAR_, y=log(totalacres))) +
  geom_point() +
  geom_smooth(method=loess, se=TRUE) +
  geom_text(aes(label=STATE,color=STATE), size=3, check_overlap = TRUE,nudge_x = 1.0)+
```



- También puedes añadir un subtítulo y un pie de foto con el código que ves a continuación

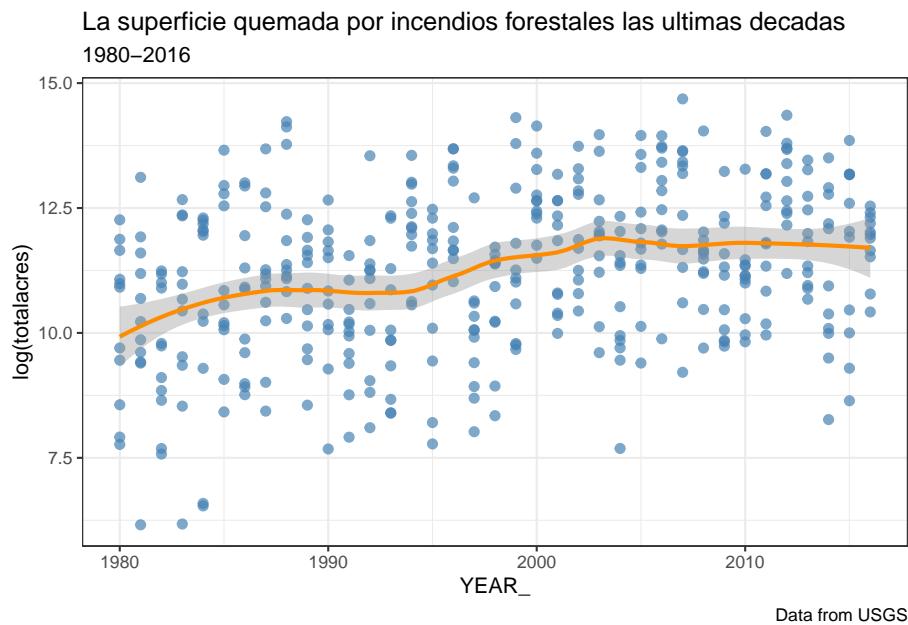
```
df %>%
  group_by(STATE, YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES)) %>%
  ggplot(aes(x=YEAR_, y=log(totalacres))) +
  geom_point() +
  geom_smooth(method=loess, se=TRUE) +
  geom_text(aes(label=STATE,color=STATE), size=3, check_overlap = TRUE,nudge_x = 1.0) +
  theme_bw()
```

78CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS



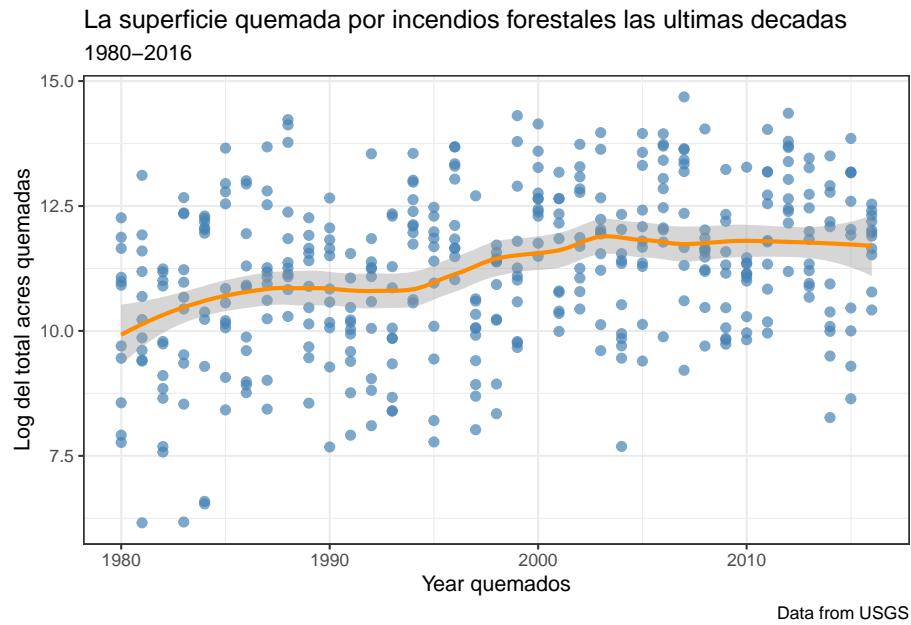
- También puedes añadir un subtítulo y un pie de foto con el código que ves a continuación

```
df %>%
  group_by(STATE, YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES)) %>%
  ggplot(aes(x = YEAR_, y = log(totalacres))) +
  geom_point(color = "steelblue", size = 2, alpha = 0.7) +
  geom_smooth(method = "loess", se = TRUE, span = 0.5, color = "darkorange")+
  labs(title=paste("La superficie quemada por incendios forestales las ultimas decadas"),
       theme_bw())
```



- También puede actualizar las etiquetas X e Y del gráfico. Actualice estas etiquetas en su gráfico usando el código que ve abajo

```
df %>%
  group_by(STATE, YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES)) %>%
  ggplot(aes(x = YEAR_, y = log(totalacres))) +
  geom_point(color = "steelblue", size = 2, alpha = 0.7) +
  geom_smooth(method = "loess", se = TRUE, span = 0.5, color = "darkorange") +
  labs(title=paste("La superficie quemada por incendios forestales las ultimas decadas"), subtitle=""),
  scale_y_continuous(name="Log del total acres quemadas") +
  scale_x_continuous(name="Year quemados") +
  theme_bw()
```

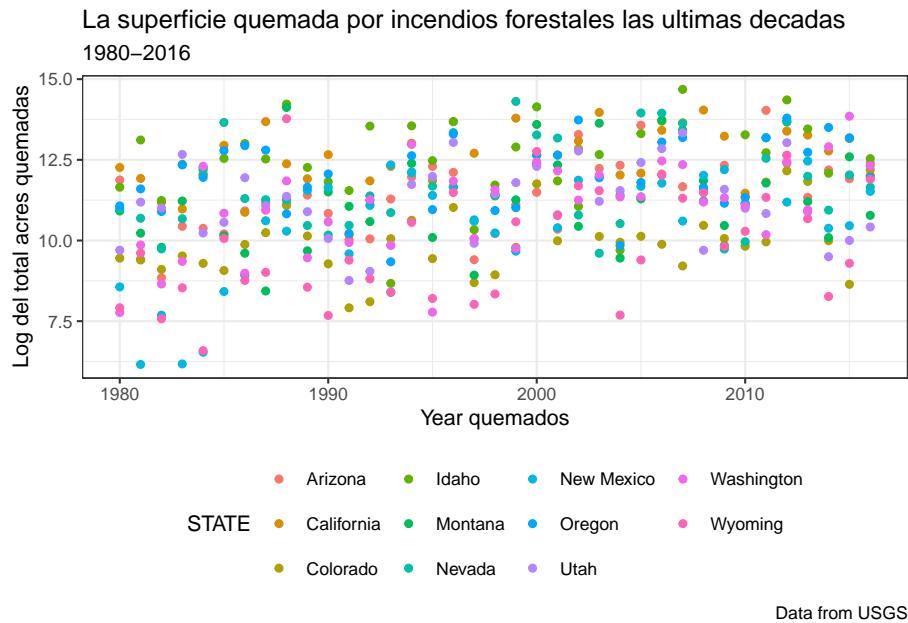


3.12 Visualización de leyendas

La función `theme()` puede utilizarse para controlar la ubicación de la leyenda y la función `guide()` puede utilizarse para proporcionar un control adicional de la leyenda

- La función `theme()` junto con el argumento `legend.position` se utiliza para controlar la ubicación de la leyenda en el gráfico. Por defecto, la leyenda que hemos visto hasta ahora se ha colocado en el lado derecho del gráfico con una orientación vertical. Reposicione la leyenda en la parte inferior con el código siguiente.

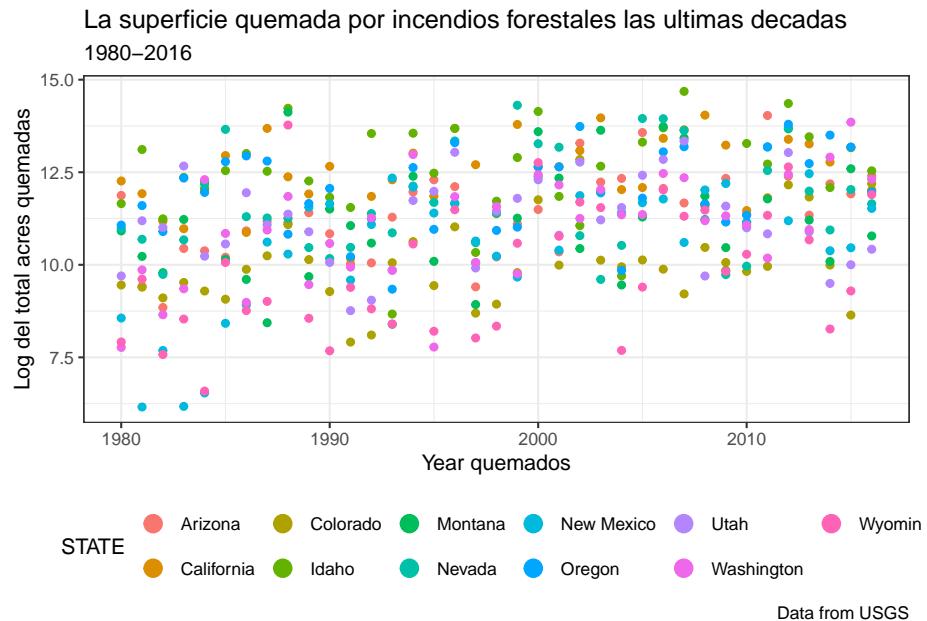
```
df %>%
  group_by(STATE, YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES)) %>%
  ggplot(aes(x = YEAR_, y = log(totalacres), color=STATE)) +
  geom_point() +
  labs(title=paste("La superficie quemada por incendios forestales las ultimas decadas"),
       scale_y_continuous(name="Log del total acres quemadas") +
       scale_x_continuous(name="Year quemados")+
       theme_bw()+
       theme(legend.position="bottom"))
```



También puede eliminar explícitamente una leyenda estableciendo `legend.position = "none"`. Pruebe eso ahora si lo desea

- Otros aspectos de la leyenda, como el número de filas en la leyenda, así como el tamaño del símbolo pueden ser controlados a través de la función `guides()`. Utilice el código que se ve a continuación para actualizar la leyenda a dos filas y con cada símbolo de tamaño 4

```
df %>%
  group_by(STATE, YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES)) %>%
  ggplot(aes(x = YEAR_, y = log(totalacres), color=STATE)) +
  geom_point() +
  labs(title=paste("La superficie quemada por incendios forestales las ultimas decadas"), subtitle=""),
  scale_y_continuous(name="Log del total acres quemadas") +
  scale_x_continuous(name="Year quemados")+
  theme_bw()+
  theme(legend.position="bottom")+
  guides(color=guide_legend(nrow=2,override.aes=list(size=4)))
```

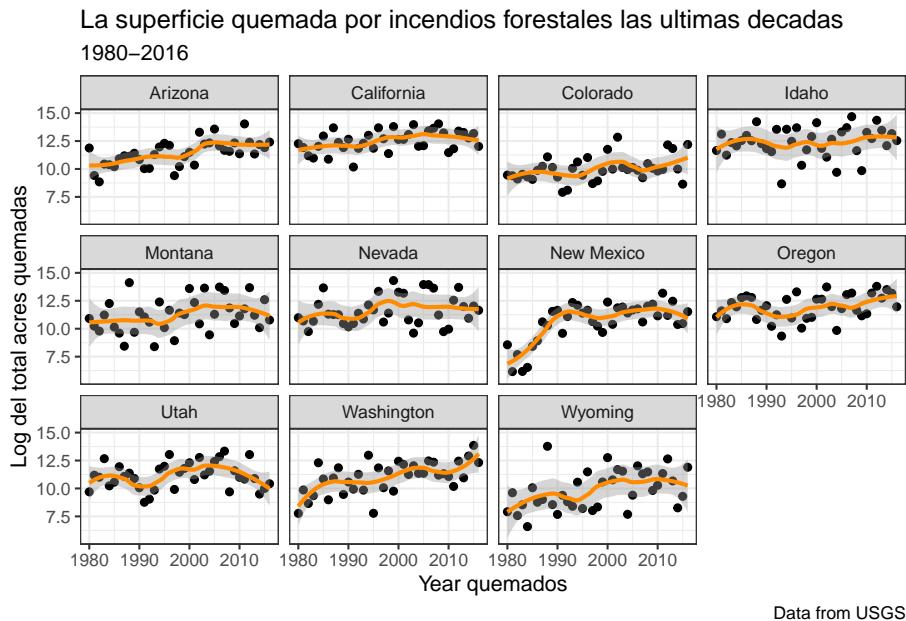


3.13 Visualización con facetas

Una forma particularmente buena de graficar variables categóricas es dividir el gráfico en facetas, que son subparcelas que muestran cada una un subconjunto de los datos. Las funciones `facet_wrap()` y `facet_grid()` pueden utilizarse para crear facetas

- Utilice la función `facet_wrap()` que se muestra en el código siguiente para crear un mapa de facetas que muestre el total de acres quemados por estado

```
df %>%
  group_by(STATE, YEAR_) %>%
  summarise(totalacres = sum(TOTALACRES)) %>%
  ggplot(aes(x = YEAR_, y = log(totalacres))) +
  geom_point() +
  geom_smooth(method = "loess", se = TRUE, span = 0.5, color = "darkorange") +
  labs(title=paste("La superficie quemada por incendios forestales las ultimas decadas"),
       scale_y_continuous(name="Log del total acres quemadas") +
       scale_x_continuous(name="Year quemados") +
       theme_bw() +
       facet_wrap(~STATE)
```



3.14 Visualización de tramas de violín

Los gráficos de violín, que son similares a los gráficos de caja, también muestran la densidad de probabilidad en varios valores. Las áreas más gruesas del gráfico de violín indican una mayor probabilidad en ese valor. Normalmente, los gráficos de violín también incluyen un marcador para la mediana junto con el rango intercuartil (IQR). La función `geom_violin()` se utiliza para crear gráficos violín en `ggplot2`

- Cargue el archivo `StudyArea.csv` y obtenga un subconjunto de columnas

```
datos <- read_csv("data/StudyArea.csv")
```

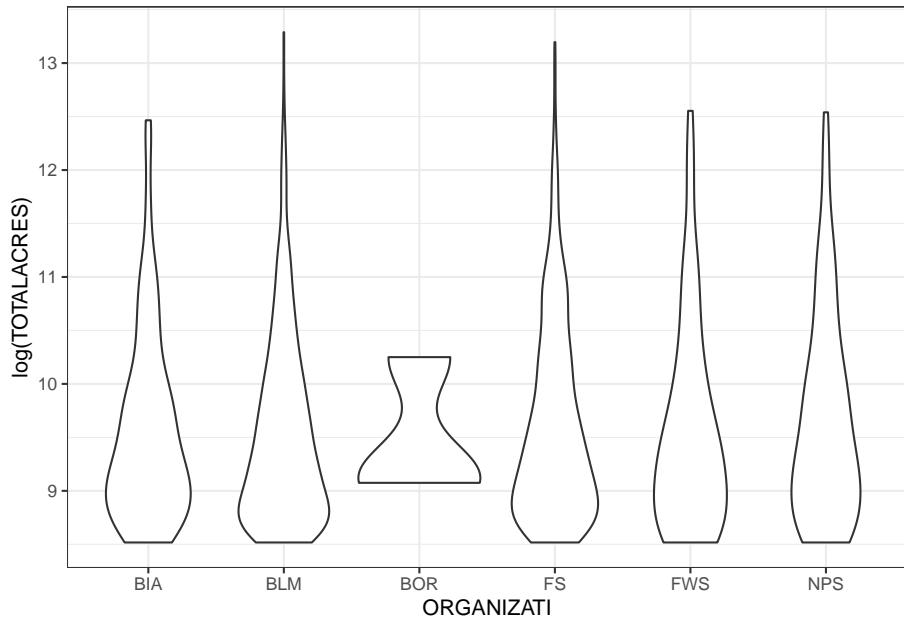
- Obtenga un subconjunto de columnas, filtrar el marco de datos para que sólo se incluyan los incendios forestales de más de 5.000 acres se incluyan. Agrupe los incendios forestales por organización y guardalo en un objeto

```
df <- datos %>%
  select(ORGANIZATI, STATE, YEAR_, TOTALACRES, CAUSE) %>%
  filter(TOTALACRES >= 5000) %>%
  group_by(ORGANIZATI)
```

84 CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS

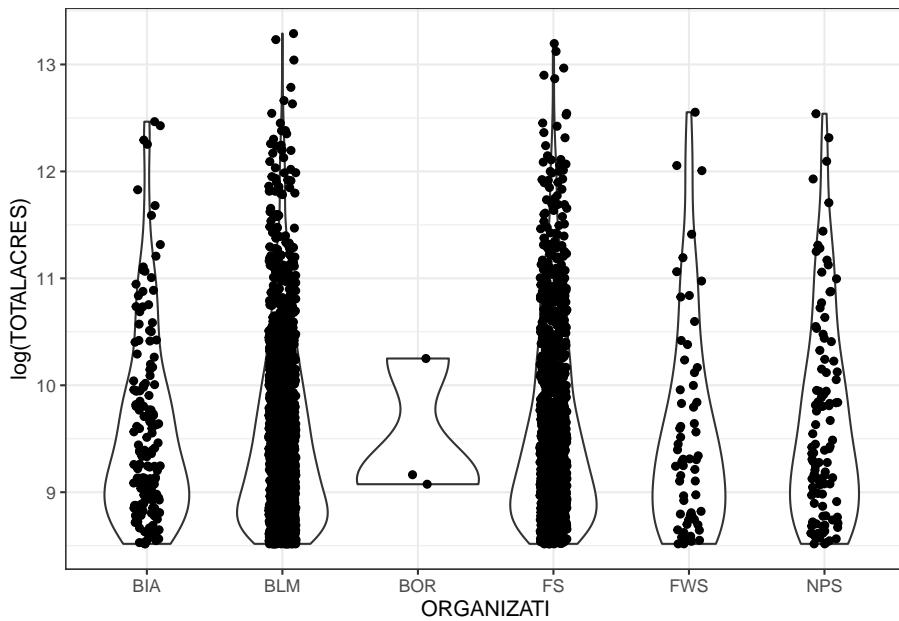
- Crear una gráfica básica de violín.

```
df %>%
  ggplot(mapping = aes(x=ORGANIZATI, y=log(TOTALACRES))) +
  geom_violin()+
  theme_bw()
```



- Puede añadir las observaciones individuales utilizando `geom_jitter()`. Si necesita mantener los puntos y dispersarlos horizontalmente, puede utilizar `geom_jitter(height = 0)`

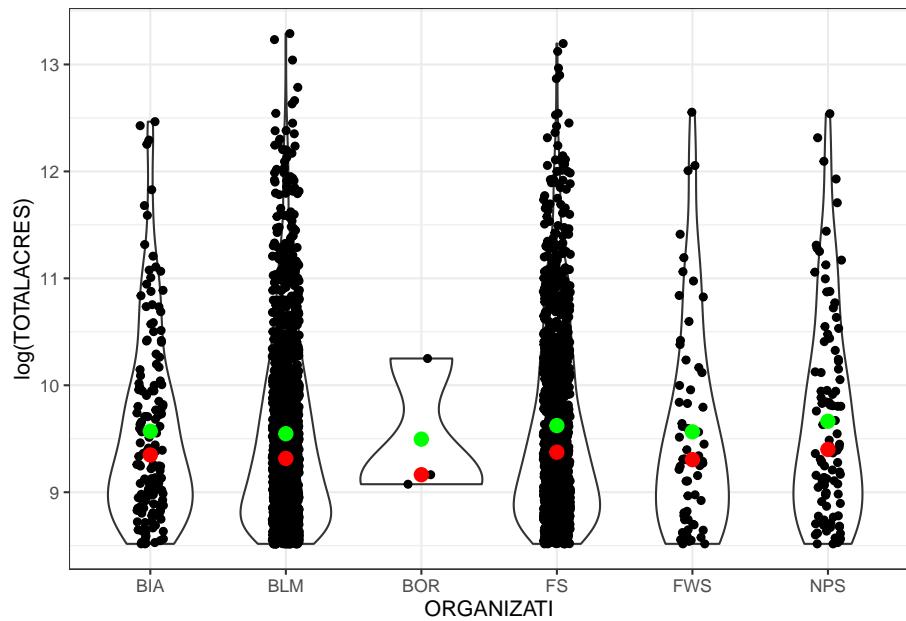
```
df %>%
  ggplot(mapping = aes(x=ORGANIZATI, y=log(TOTALACRES))) +
  geom_violin() +
  geom_jitter(height = 0, width = 0.1)+
  theme_bw()
```



- La media puede añadirse utilizando `stat_summary()` como se ve a continuación

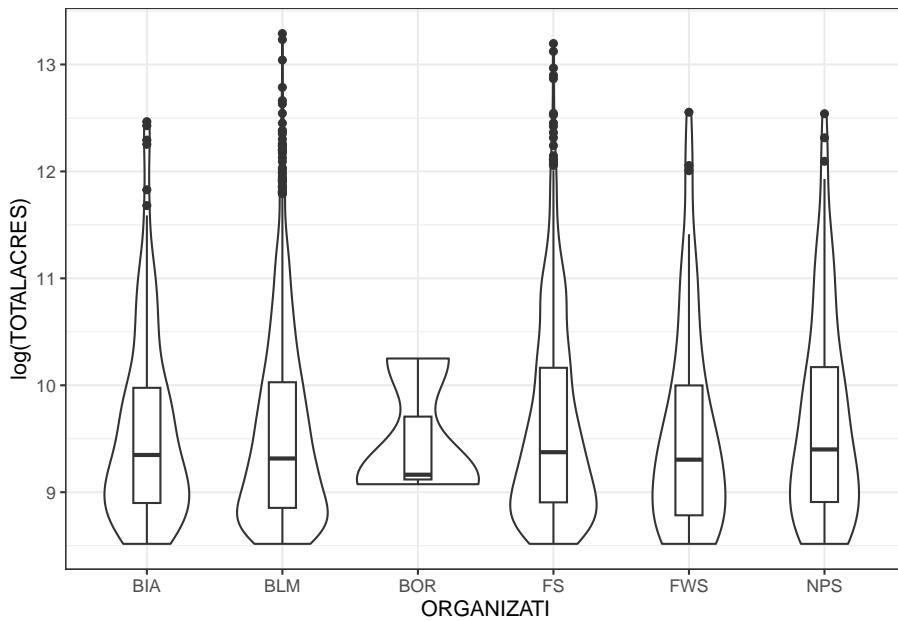
```
df %>%
  ggplot( mapping = aes(x=ORGANIZATI, y=log(TOTALACRES))) +
  geom_violin() + geom_jitter(height = 0, width = 0.1) +
  stat_summary(fun.y=median, geom="point", size=3, color="red") +
  stat_summary(fun.y=mean, geom="point", size=3, color="green") +
  theme_bw()
```

86 CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS



- La función `box_plot()` puede utilizarse para añadir la mediana y el IQR

```
df %>%
  ggplot(mapping = aes(x=ORGANIZATI, y=log(TOTALACRES))) +
  geom_violin() +
  geom_boxplot(width=0.2) +
  theme_bw()
```



3.15 Visualización de gráficos de densidad

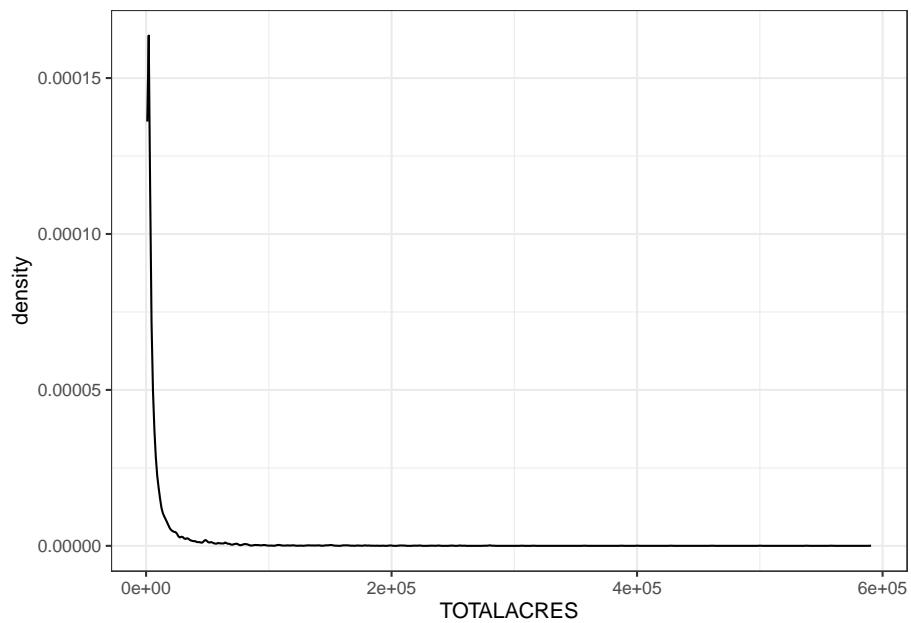
- Los gráficos de densidad, creados con `geom_density()` calculan una estimación de la densidad, que es una versión suavizada de un histograma y se utiliza con datos continuos. `ggplot2` también puede calcular versiones 2D de la densidad, incluyendo contornos y gráficos de densidad con estilo de polígono
- Tomemos los incendios forestales de más de 1.000 acres

```
df2 <- datos %>%
  select(ORGANIZATI, STATE, YEAR_, TOTALACRES, CAUSE) %>%
  filter(TOTALACRES >= 1000)
```

- Cree un gráfico de densidad con la función `geom_density()`

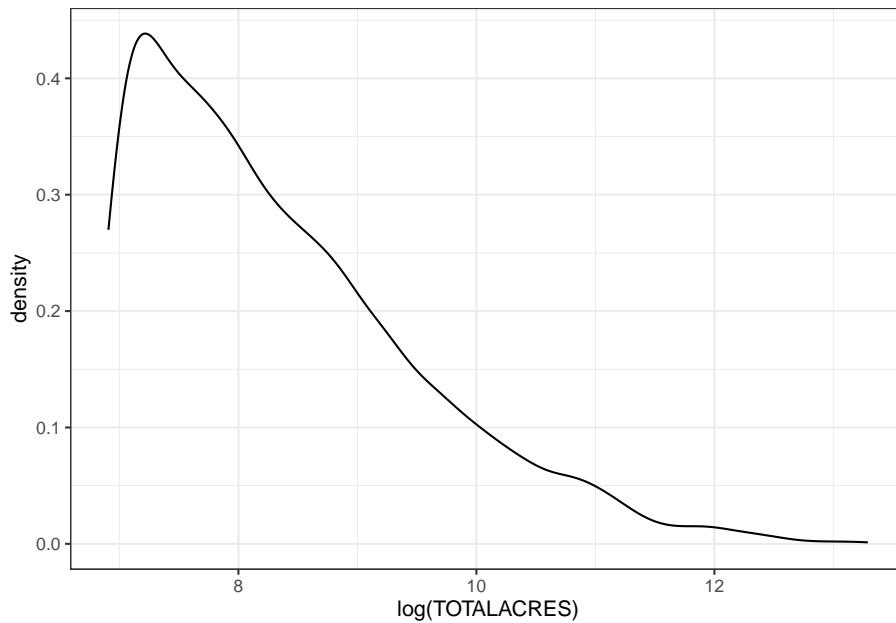
```
df2 %>%
  ggplot(aes(TOTALACRES)) +
  geom_density() +
  theme_bw()
```

88 CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS



- También puede crear el mismo gráfico de densidad con una versión registrada de los datos

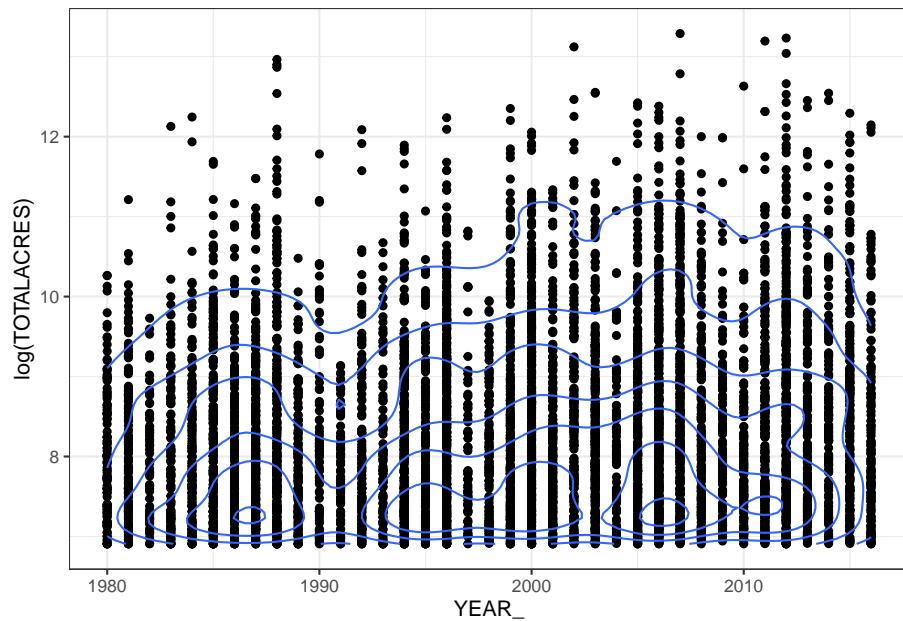
```
df2 %>%
  ggplot(aes(log(TOTALACRES))) +
  geom_density() +
  theme_bw()
```



- A continuación, crearás gráficos 2D de los datos empezando por los contornos. Añade el código que ves a continuación

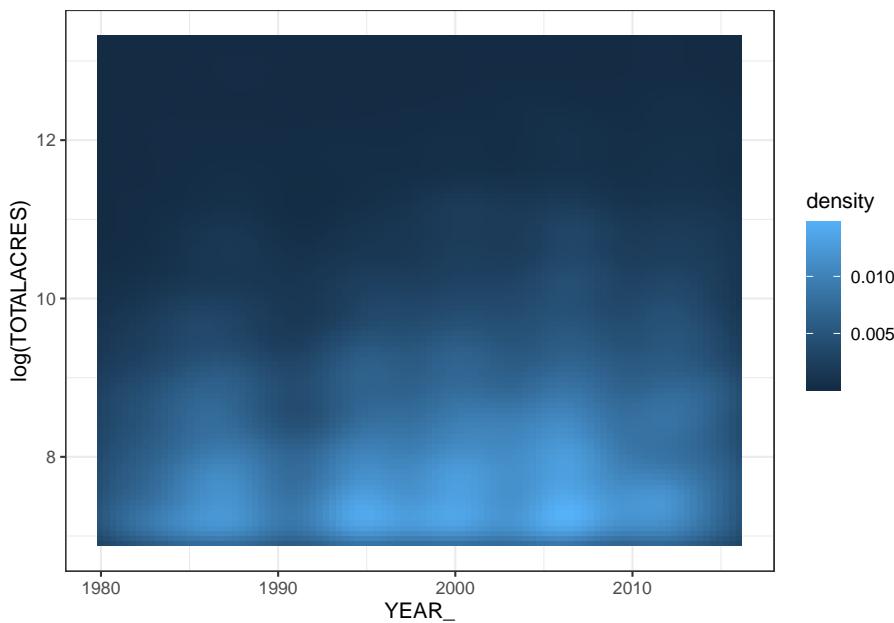
```
df2 %>%
  ggplot(aes(x=YEAR_, y=log(TOTALACRES))) +
  geom_point() +
  geom_density_2d()+
  theme_bw()
```

90CHAPTER 3. EXPLORACIÓN Y VISUALIZACIÓN ESTÁTICA DE DATOS



- Por último, cree una superficie de densidad 2D utilizando `stat_density_2d()`

```
df2 %>%
  ggplot(aes(x=YEAR_, y=log(TOTALACRES))) +
  geom_density_2d() +
  stat_density_2d(geom="raster", aes(fill=..density..), contour=FALSE) +
  theme_bw()
```



Ejercicio para entregar

Considere los datos asociados a precios de casas `house_prices.csv` que aparece en el directorio **RDataSets.zip** en Github. Nótese que el archivo posee su respectiva descripción `house_prices_description.txt` que puede ser de gran utilidad. **Realice un análisis exploratorio de los datos relacionados con los precios de casas**, teniendo en cuenta cada ítem estudiado en esta sección. Debería agregar todas las visualizaciones extras, que sean necesarias y le permitan **generar y refinar preguntas sobre los datos**.

Ejercicio para entregar

Considere los datos asociados a precios de casas `house_prices.csv` que aparece en el directorio **RDataSets.zip** en Github. A manera de aplicación, utilice las técnicas básicas de visualización de datos estudiadas en cada ítem de esta sección, ahora aplicadas al conjunto de datos relacionado con la predicción del precio de casas, el cual puede ser influenciado por los distintos factores, presentados en las columnas del archivo `house_prices.csv`, cuyas descripciones puede encontrar en el archivo `house_prices_description.txt`. Puede agregar visualizaciones extras, que le permitan generar y refinar preguntas sobre los datos.

Chapter 4

Visualización de datos geográficos con `ggmap`

4.1 Paquete `ggmap`

Definición: `ggmap`

El paquete `ggmap` permite visualizar datos espaciales y estadísticas espaciales en formato de mapa, utilizando el enfoque por capas de `ggplot2`.

Incluye mapas base como Google Maps, Open Street Map, Stamen Maps y CloudMade, y además proporciona funciones adicionales para geocodificación, matrices de distancia y direcciones.

4.1.1 Funciones principales de `ggmap`

Nota de `get_map()`

Descarga un mapa base desde un proveedor (Google, Stamen, OSM).

Los parámetros principales son

`location` → Nombre del lugar, coordenadas o bounding box.

`zoom` → Nivel de zoom (1 a 21).

`maptype` → Tipo de mapa (terrain, satellite, roadmap, hybrid, toner, etc.).

`source` → Fuente del mapa (google, stamen, osm).

Nota de `ggmap()`

94 CHAPTER 4. VISUALIZACIÓN DE DATOS GEOGRÁFICOS CON GGMAP

Dibuja un mapa base en un gráfico de ggplot2.
Los parámetros principales son

object → Objeto retornado por get_map().
extent → Ajuste de límites (panel, normal, device).
interpolate → Suavizado de píxeles (TRUE/FALSE).

Nota de `geocode()`

Convierte una dirección en coordenadas (latitud y longitud).
Los parámetros principales son

location → Dirección en texto.
output → Formato de salida (latlon, latlona, more).
Nota de `revgeocode()`

Convierte coordenadas en dirección postal.
Los parámetros principales son

location → Vector con coordenadas (c(lon, lat)).
Nota de `route()`

Obtiene la ruta entre dos ubicaciones.
los parámetros principales son

from → Dirección de inicio.
to → Dirección de destino.
mode → Medio de transporte (driving, walking, bicycling).
Nota de `qmap()`

Versión rápida de get_map() + ggmap().
Los parámetros principales son

location → Lugar en texto o coordenadas.
zoom, maptype, source.

4.1.2 Relación con ggplot2

Observación

El paquete `ggmap` se construye sobre `ggplot2`, por lo que hereda sus capas estéticas y geométricas:

Capas geométricas: `geom_point()`, `geom_polygon()`, `geom_line()`.

Sistema de coordenadas: `coord_map("mercator")`.

Facetas: `facet_wrap()`, `facet_grid()`.

Esto permite superponer capas de datos espaciales sobre mapas base descargados.

Temas que se cubrirán

- Creación de un **mapa base** con `get_map()` y `ggmap()`.
- Añadir **capas operativas** (puntos, polígonos, rutas).
- Incorporar capas desde un **shapefile** usando `sf` o `rgdal`.

4.2 Construcción de un mapa base

Hay dos pasos básicos para crear un mapa con `ggmap`. En general sólo hay que descargar el mapa rasterizado (*basemap*) y luego trazar los datos operativos en el mapa base. El primer paso se consigue con la función `get_map()`, que puede utilizarse para crear un mapa base desde **Google**, **Stamen**, **Open Street Map** o **CloudMade**. Aprenderás cómo hacerlo en este paso. En un próximo paso aprenderás a añadir y estilizar los datos operativos de varias maneras

1. Cargue el paquete `ggmap` yendo al panel de paquetes en **RStudio** y haciendo clic en la casilla junto al nombre del paquete. También puede cargarlo desde la consola escribiendo

```
install.packages("ggmap")
```

2. Crea una variable llamada **myLocation** y defínela como **California**, ubicación que usaremos más adelante

```
myLocation <- "California"
```

3. Llama a la función `get_map()` y pasa la variable de localización junto con un nivel de *Zoom de 6*. Necesitarás primero crear una **API key** en **Google Cloud Platform**. Si descargas la versión de desarrollo de `ggmap` desde **GitHub**, hay una función que guarda tu clave de la **API** de **Google** y la utiliza en las siguientes llamadas a la **API** a través de `ggmap`. Para instalarla desde la consola escribir

```
devtools::install_github("dkahle/ggmap")
```

```
> devtools::install_github("dkahle/ggmap")
These packages have more recent versions available.
It is recommended to update all of them.
Which would you like to update?

1: All
2: CRAN packages only
3: None ←
4: purrr (1.0.4 -> 1.1.0) [CRAN]
5: curl (6.4.0 -> 7.0.0) [CRAN]

Enter one or more numbers, or an empty line to skip updates: 3
```

Figure 4.1: Instalaci^{U+00F3}n de ggmap

4. Utilice las siguientes instrucciones para descargar el mapa. Cargue primero su **API key** y luego use la función `get_googlemap` para obtener el mapa de interés. Para que las siguientes instrucciones funcionen, debe realizar los siguientes pasos en **Google Cloud Platform**
 - Debe registrarse en Google Cloud Platform. Desafortunadamente para el registro deberá usar una **Tarjeta de credito**. Obtendrá \$300 gratis para usarlos por 90 días. Luego de esto se cargarán a su cuenta \$5 mensuales, dependiendo del uso que haga de la API. Recomiendo usar una tarjeta crédito prepago como por ejemplo **Daviplata** o similares.
 - Ahora, creamos nuestro primer proyecto en google cloud platform:
 - Debe habilitar los siguientes servicios en la **Biblioteca**
 - Maps Static API
 - * Cuando habilites tu primera API, puedes crear tu llave. Si no en la parte de arriba dice + **crear credenciales** y allí puedes crear tu llave.
 - Geocoding API
 - Geolocation API
 - Maps Embed API
 - Ahora vamos a restringir la clave a APIs específicas
5. Empecemos a crear mapas. Para obtener mas información sobre el uso de la función `get_googlemap()` ver `ggmap`.

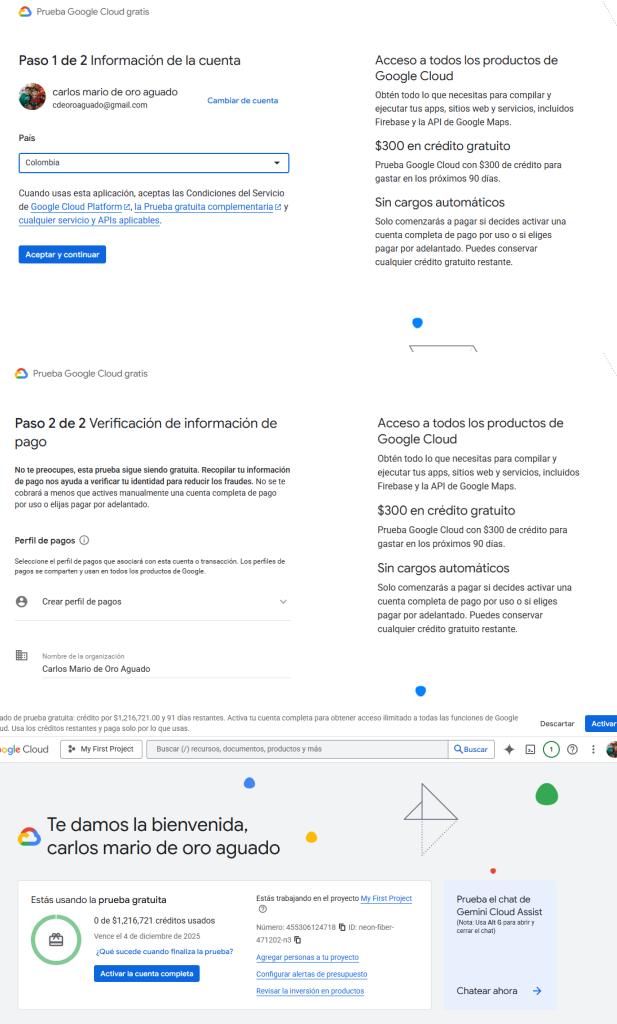


Figure 4.2: Paso a paso para cuenta en google cloud plarform

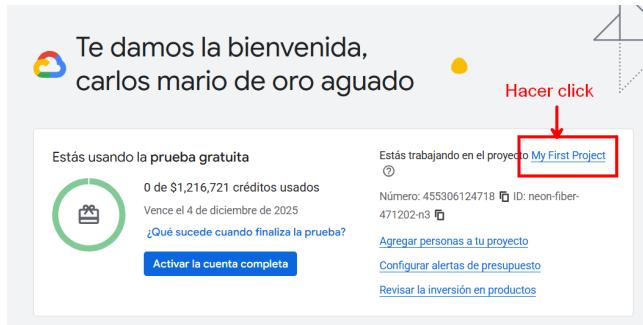
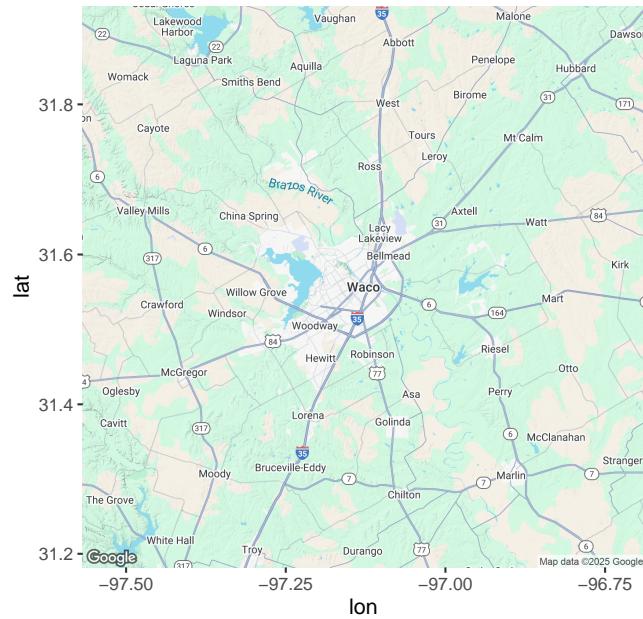


Figure 4.3: Paso a paso para crear proyecto

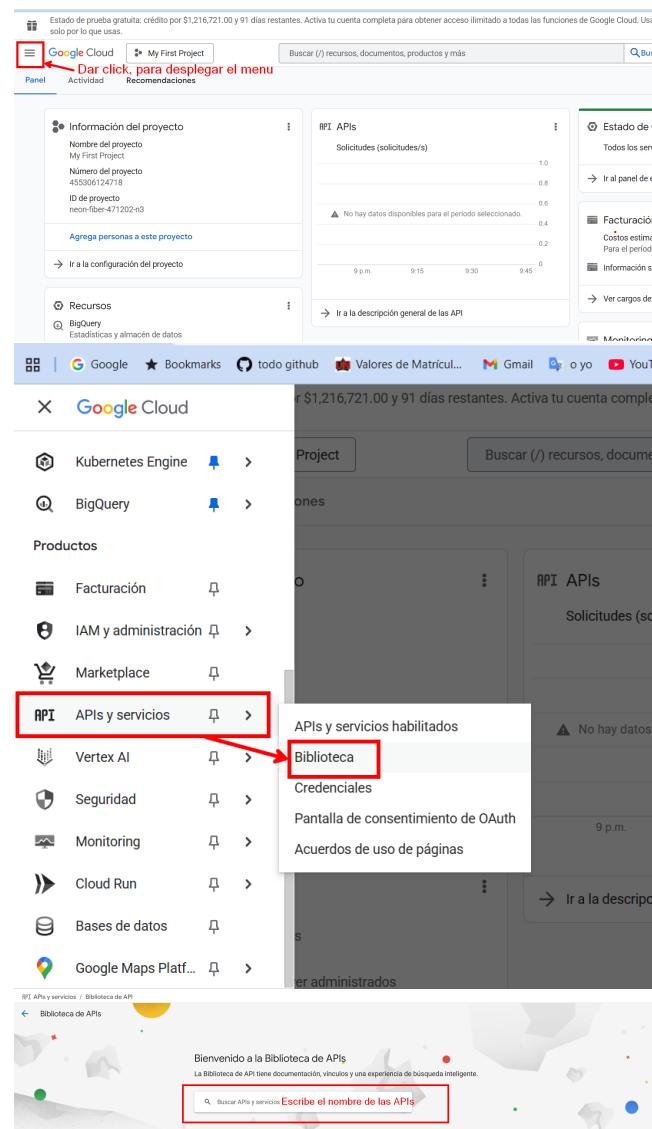
```
library(ggmap)

register_google(key = mykey)

get_googlemap("waco, texas") %>%
  ggmap()
```



6. Lo siguientes tipos de mapas también están disponibles para vista satelital en los mapas. Para ver más opciones visitar la documentación de `get_googlemap`



Maps Static API

[Google Enterprise API](#)

Simple, embeddable map image with minimal code.

Habilitar Dar click|

Figure 4.4: Habilitaci^{U+00F3}n de las 4 APIs

100CHAPTER 4. VISUALIZACIÓN DE DATOS GEOGRÁFICOS CON GGMAP

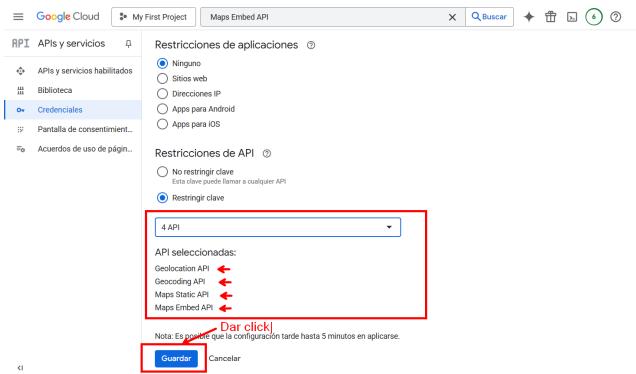
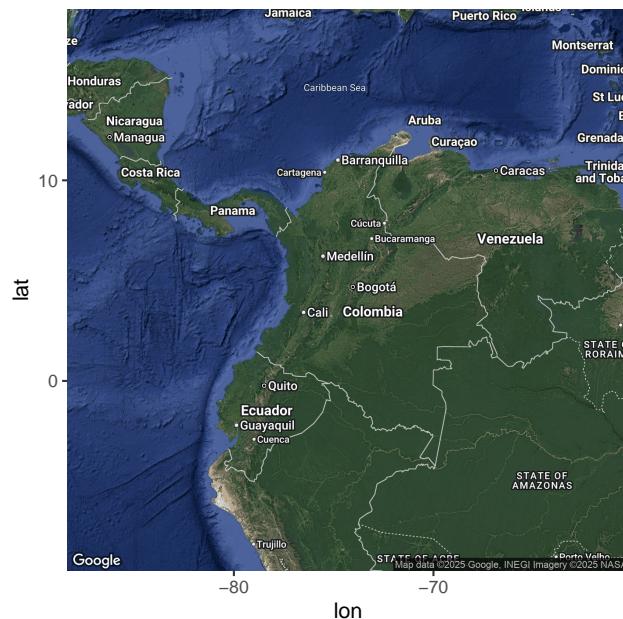


Figure 4.5: Restricciones de la clave

```
# Obtener mapa de Google centrado en Colombia

get_ggmap("Colombia",
           maptype = "hybrid", # tipo de mapa híbrido (satélite + calles)
           scale = 2,           # escala del mapa (resolución)
           zoom = 5)            # nivel de acercamiento
ggmap()                  # mostrar el mapa
```



```
# Obtener mapa de Google centrado en "Universidad del Norte"
```

```
get_googlemap("Universidad del Norte",
              maptype = "roadmap", # tipo de mapa (carreteras)
              scale = 2,           # escala del mapa (resolución)
              zoom = 12) %>%      # nivel de acercamiento
ggmap()                  # mostrar el mapa
```



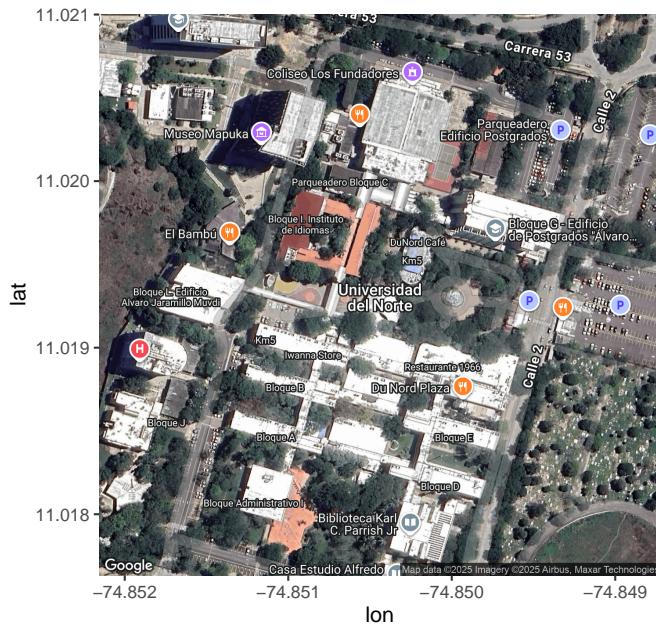
```
# Obtener mapa de Google centrado en "Universidad del Norte"
```

```
get_googlemap("Universidad del Norte",
              maptype = "terrain", # tipo de mapa (relieve/terreno)
              scale = 2,           # escala del mapa (resolución)
              zoom = 16) %>%      # nivel de acercamiento (detalle alto)
ggmap()                  # mostrar el mapa
```

102 CHAPTER 4. VISUALIZACIÓN DE DATOS GEOGRÁFICOS CON GGMAP



```
# Obtener mapa de Google centrado en "Universidad del Norte"
get_ggmap("Universidad del Norte",
           maptype = "hybrid", # híbrido (satélite + nombres de calles)
           scale = 2,           # escala del mapa (resolución)
           zoom = 18) %>%
ggmap()                # mostrar el mapa
```



4.3 Integración de capas de información

`ggmap()` devuelve un objeto `ggplot`, lo que significa que actúa como la capa base en `ggplot2`.

Gracias a esto, se pueden aprovechar todas las capacidades de `ggplot2`, incluyendo:

- Trazar puntos geográficos en el mapa
- Añadir contornos y polígonos
- Generar mapas de calor en 2D
- Superponer capas de datos adicionales

En las siguientes secciones exploraremos algunas de estas funcionalidades aplicadas a datos espaciales.

1. El siguiente código genera un mapa del estado de **California**, en el cual se visualizan los incendios forestales ocurridos entre los años **1980 y 2016** que afectaron superficies superiores a **1.000 hectáreas**.

```
myLocation <- "California"
myMap <- get_map(location = myLocation, zoom = 6)
```

- Carguemos los datos de **incendios forestales**

```
library(tidyverse)

datos <- read_csv("data/StudyArea_SmallFile.csv", col_names = TRUE)
```

- Seleccionemos las siguientes variables STATE, YEAR_, TOTALACRES, DLATITUDE, DLONGITUDE

```
library(tidyverse)

datos %>%
  select(STATE, YEAR_, TOTALACRES, DLATITUDE, DLONGITUDE) -> df

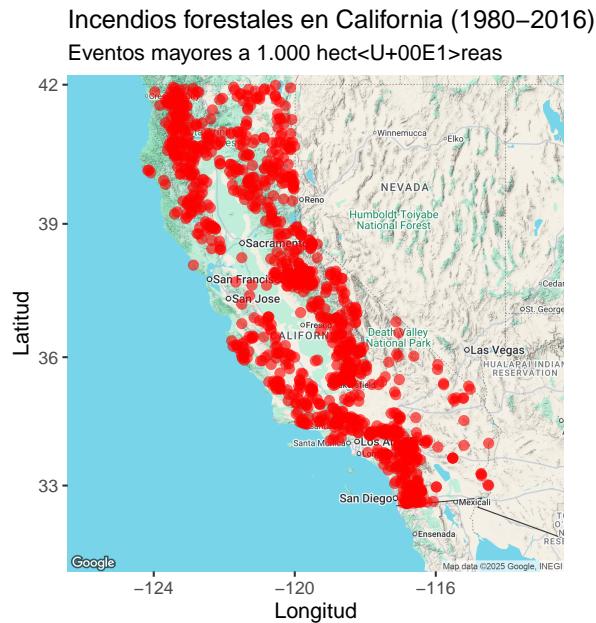
knitr::kable(head(df))
```

STATE	YEAR_	TOTALACRES	DLATITUDE	DLONGITUDE
Arizona	1988	1500	31.58333	-111.5500
Arizona	1986	10390	32.50000	-111.5167
Montana	1986	1400	47.50000	-111.4333
Arizona	2002	1035	31.70000	-111.4830
Arizona	2000	5700	31.51600	-111.5170
Arizona	2000	2750	31.64900	-111.4910

2. Tomemos el estado de California y total de acres mayor o igual a 1000 y representemoslo en un diagrama de dispersión

```
# Filtrar incendios de más de 1000 acres en California
df_a <- df %>%
  filter(TOTALACRES >= 1000 & STATE == "California")

# Graficar mapa base y ubicar los incendios
ggmap(myMap) +
  geom_point(data = df_a, aes(x = DLONGITUDE, y = DLATITUDE),
             color = "red", alpha = 0.6, size = 2) +
  labs(title = "Incendios forestales en California (1980-2016)",
       subtitle = "Eventos mayores a 1.000 hectáreas",
       x = "Longitud", y = "Latitud")
```



3. Ahora vamos a hacer el análisis un poco más interesante.
En primer lugar, utilizaremos la función de `dplyr::mutate()` para agrupar los incendios por década.

```
# Crear variable DECADE agrupando por rangos de años
df_b <- df %>%
  mutate(DECADE = ifelse(YEAR_ %in% 1980:1989, "1980-1989",
                        ifelse(YEAR_ %in% 1990:1999, "1990-1999",
                        ifelse(YEAR_ %in% 2000:2009, "2000-2009",
                        ifelse(YEAR_ %in% 2010:2016, "2010-2016", NA)))))
```

- A continuación, codificaremos por colores los incendios forestales según la década (DECADE) y crearemos un mapa de símbolos graduados en función del tamaño de cada incendio.
 - La propiedad `color` se usará para distinguir las décadas.
 - La propiedad `size`

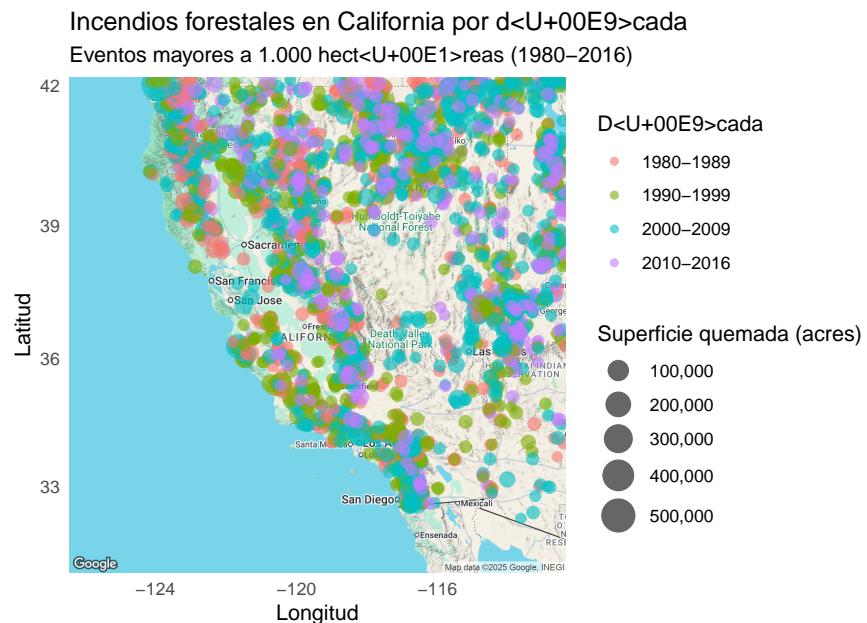
```
library(scales)

# Mapa con incendios codificados por década y tamaño
ggmap(myMap) +
  geom_point(data = df_b,
```

```

aes(x = DLONGITUDE, y = DLATITUDE,
   color = DECADE, size = TOTALACRES),
  alpha = 0.6) +
  scale_size_continuous(name = "Superficie quemada (acres)",
                        labels = label_number(big.mark = ","),
                        range = c(2, 8)) +
  labs(title = "Incendios forestales en California por década",
       subtitle = "Eventos mayores a 1.000 hectáreas (1980–2016)",
       x = "Longitud", y = "Latitud", color = "Década") +
  theme_minimal()

```



- También es posible **hacer dinámico el mapa** usando ggplotly. Para ello se emplea la función `ggplot_build()` para convertir el objeto de `ggmap` en un objeto `ggplot`.

```

# Convertir a objeto ggplot y luego hacerlo interactivo
p <- ggmap(myMap) +
  geom_point(data = df_b,
             aes(x = DLONGITUDE, y = DLATITUDE,
                 color = DECADE, size = TOTALACRES),
             alpha = 0.6) +
  scale_size_continuous(range = c(2, 8))

```

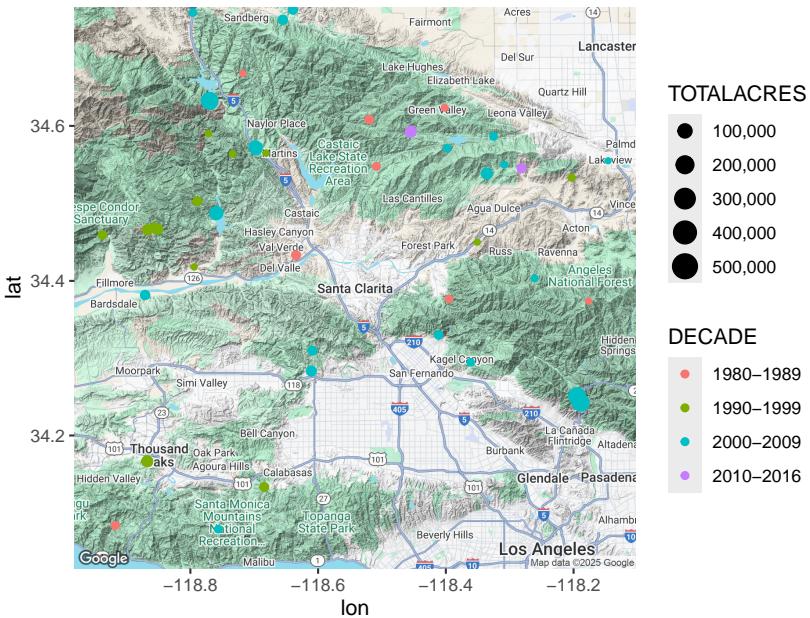
```
#install.packages("plotly")
library(plotly)

gg <- ggplot_build(p)$plot
plotly_plot <- ggplotly(gg)
plotly_plot
```

4. Ahora vamos a modificar la vista del mapa para **enfocarnos en el sur de California**, específicamente en la **zona ubicada al norte de Los Ángeles**. Para ello, utilizaremos coordenadas aproximadas de latitud y longitud que centren el mapa en esta región, lo que nos permitirá observar con mayor detalle la distribución espacial de los incendios forestales en dicha área.

```
myMap <- get_map(location = "Santa Clarita, California", zoom = 10)

ggmap(myMap) +
  geom_point(data=df_b,
             aes(x = DLONGITUDE, y = DLATITUDE, colour= DECADE, size = TOTALACRES))+
  scale_size_continuous(labels = label_number(big.mark = ","))
```



5. A continuación, añadiremos **capas de contorno y de calor** para resaltar las áreas con mayor concentración de incendios.

- La función `geom_density2d()` se utiliza para **dibujar los contornos de densidad**.
- La función `stat_density2d()` permite **crear el mapa de calor**.

Podemos personalizar los colores de los contornos y del mapa de calor con la función `scale_fill_gradient(low, high)`.

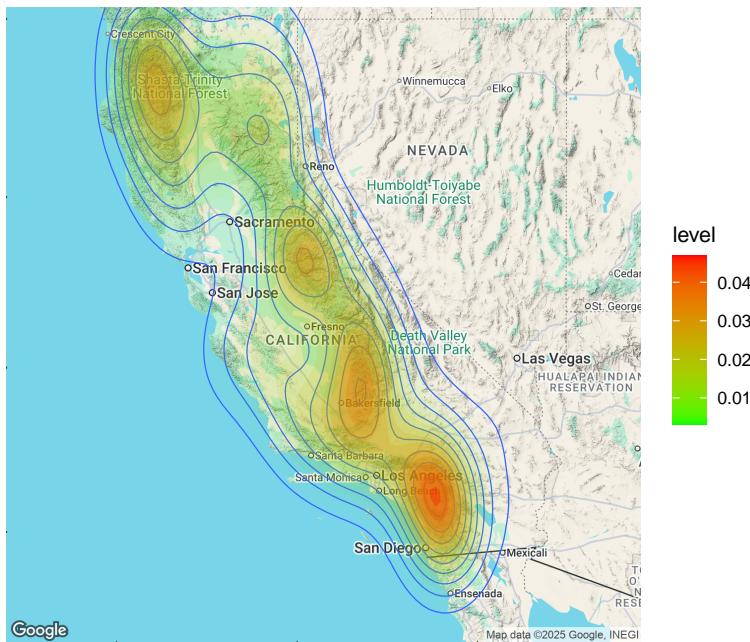
En este ejemplo, hemos definido un gradiente de **verde a rojo**, aunque puede adaptarse según la preferencia del usuario.

Además, la sintaxis `fill = ..level..` se emplea para **rellenar los contornos de acuerdo con los niveles de densidad de los puntos en el mapa**.

- El siguiente código genera un **mapa de California con capas de contorno y de calor**, resaltando las zonas con mayor densidad de incendios.

```
# Obtener mapa base de California
myMap <- get_map(location = "California", zoom = 6)

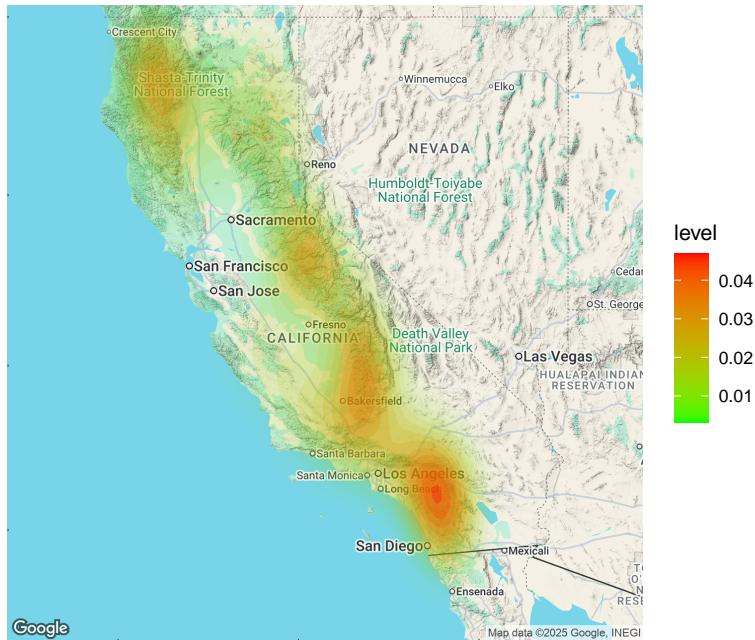
# Añadir contornos y mapa de calor
ggmap(myMap, extent = "device") + # Mapa expandido
  # Contornos de densidad
  geom_density2d(data = df_a, aes(x = DLONGITUDE, y = DLATITUDE), size = 0.3) +
  
  # Mapa de calor (relleno por niveles de densidad)
  stat_density2d(data = df_a,
    aes(x = DLONGITUDE, y = DLATITUDE,
        fill = ..level.., alpha = ..level..),
    size = 0.01, bins = 16, geom = "polygon") +
  
  # Colores del gradiente (verde = baja densidad, rojo = alta densidad)
  scale_fill_gradient(low = "green", high = "red") +
  
  # Transparencia de las capas
  scale_alpha(range = c(0, 0.3), guide = FALSE)
```



6. Si prefiere ver el mapa de calor sin contornos, el código puede simplificarse como sigue

```
# Obtener mapa base de California
myMap <- get_map(location = "California", zoom = 6)

# Añadir solo mapa de calor (sin contornos)
ggmap(myMap, extent = "device") + # Mapa expandido
  stat_density2d(data = df_a,
                 aes(x = DLONGITUDE, y = DLATITUDE,
                     fill = ..level.., alpha = ..level..),
                 size = 0.01, bins = 16, geom = "polygon") +
  scale_fill_gradient(low = "green", high = "red") + # Gradiente de color
  scale_alpha(range = c(0, 0.3), guide = FALSE)      # Transparencia
```



7. Para finalizar, crearemos un **mapa con facetas** que muestre los **puntos críticos de incendios en cada año de la década actual**.

El conjunto de datos disponible llega hasta el año **2016**, por lo que incluiremos únicamente la información correspondiente a ese periodo.

La función `facet_wrap()` de `ggplot2` resulta especialmente útil en este caso, ya que permite **dividir el gráfico en múltiples paneles organizados en cuadrícula**, mostrando de manera independiente cada subconjunto de datos y facilitando la comparación visual entre los diferentes años.

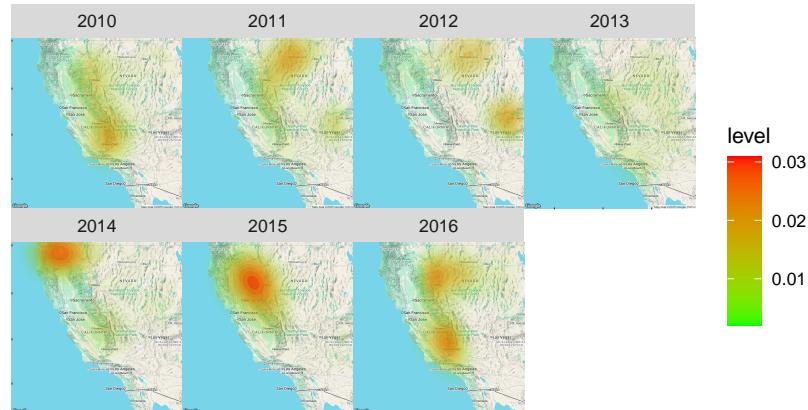
```
# Filtrar incendios de 2010 a 2016
df_c <- filter(df, YEAR_ %in% 2010:2016)

# Crear mapa facetado por año
gg <- ggmap(myMap, extent = "device") +
  # Capa de densidad (mapa de calor)
  stat_density2d(data = df_c,
                 aes(x = DLONGITUDE, y = DLATITUDE,
                     fill = ..level.., alpha = ..level..),
                 size = 0.01, bins = 16, geom = "polygon") +
  # Gradiente de colores (verde = baja densidad, rojo = alta densidad)
  scale_fill_gradient(low = "green", high = "red") +
  # Control de transparencia
  scale_alpha(range = c(0, 0.3), guide = FALSE) +
```

4.4. INTEGRACIÓN DE CAPAS VECTORIALES DESDE SHAPEFILES111

```
# Facetas por año (4 columnas)
facet_wrap(~YEAR_, ncol = 4) +
# Estilo del título
theme(plot.title = element_text(size = 20)) +
# Márgeos del gráfico
theme(plot.margin = margin(1, 1, 1, 1, "cm"))

# Mostrar gráfico
gg
```



4.4 Integración de capas vectoriales desde Shapfiles

Los contenidos y ejemplos presentados en esta sección han sido desarrollados a partir de los materiales de **Paula Moraga** en su libro *Geospatial Health Data: Modeling and Visualization with R-INLA and Shiny* (Moraga, 2019).

Esta obra constituye una referencia fundamental para el aprendizaje y la aplicación de técnicas de análisis y visualización de datos espaciales en salud, utilizando el ecosistema de R.

En particular, se han retomado y adaptado conceptos sobre la creación de mapas temáticos con **ggplot2**, **leaflet**, **mapview** y **tmap**, así como estrategias para

la construcción de mapas estáticos e interactivos, la sincronización de capas y la personalización de estilos gráficos.

Estas herramientas permiten una exploración más eficiente y comprensible de la información geoespacial, contribuyendo a la toma de decisiones basadas en evidencia en contextos de salud pública y ambiental.

4.4.1 ¿Qué es un Shapefile?

¿Qué es un Shapefile?

Los datos geográficos pueden representarse utilizando un formato de almacenamiento denominado **Shapefile**, el cual guarda la ubicación, forma y atributos de entidades espaciales como puntos, líneas y polígonos.

Un Shapefile no es un único archivo, sino un **conjunto de archivos relacionados** que comparten el mismo nombre y se almacenan en el mismo directorio.

4.4.1.1 Archivos obligatorios de un Shapefile

Un Shapefile tiene tres archivos esenciales:

- **.shp**: contiene los datos geométricos.
- **.shx**: índice posicional de la geometría, que permite buscar hacia adelante y hacia atrás en el archivo **.shp**.
- **.dbf**: almacena los atributos asociados a cada entidad.

4.4.1.2 Archivos adicionales (opcionales)

Además de los anteriores, un Shapefile puede incluir otros archivos de soporte:

- **.prj**: archivo de texto que describe el sistema de proyección.
- **.sbn** y **.sbx**: índices espaciales de los datos geométricos.
- **.shp.xml**: metadatos geoespaciales en formato XML.

Nota importante

Al trabajar con Shapefiles, **no es suficiente con disponer del archivo .shp** que contiene la geometría; también se requieren los demás archivos complementarios para garantizar la correcta lectura y análisis de los datos.

4.4.2 Lectura de shapefiles en R con sf

En **R** podemos leer shapefiles usando la función `st_read()` del paquete **sf**. Como ejemplo, usaremos el shapefile de **North Carolina** que viene incluido en el paquete.

- El paquete **sf** incluye un shapefile de ejemplo (`nc.shp`) que contiene información sobre los **condados de Carolina del Norte (EE.UU.)**. Cada polígono representa un condado e incluye atributos socioeconómicos y de salud para los años 1974 y 1979.
- Algunas variables importantes en este conjunto son:
 - `NAME` → Nombre del condado.
 - `AREA` → Área del condado (en grados²).
 - `PERIMETER` → Perímetro del condado.
 - `BIR74` y `BIR79` → Número de nacimientos en 1974 y 1979.
 - `SID74` y `SID79` → Número de muertes por síndrome de inmunodeficiencia en 1974 y 1979.
 - `NWBIR74` y `NWBIR79` → Número de nacimientos de madres no blancas en 1974 y 1979.

```
# install.packages("sf")
library(sf)

# Nombre del shapefile de North Carolina en el paquete sf
nameshp <- system.file("shape/nc.shp", package = "sf")
```

```
# Leer shapefile con st_read()
map <- st_read(nameshp, quiet = TRUE)
```

```
# Ver la clase del objeto
class(map)
```

```
## [1] "sf"           "data.frame"
```

```
# Mostrar primeras filas
head(map)
```

```

## Simple feature collection with 6 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension: XY
## Bounding box: xmin: -81.74107 ymin: 36.07282 xmax: -75.77316 ymax: 36.58965
## Geodetic CRS: NAD27
##   AREA PERIMETER CNTY_ CNTY_ID      NAME  FIPS FIPSNO CRESS_ID
## 1 0.114     1.442  1825    1825    Ashe 37009 37009      5
## 2 0.061     1.231  1827    1827  Alleghany 37005 37005      3
## 3 0.143     1.630  1828    1828    Surry 37171 37171     86
## 4 0.070     2.968  1831    1831 Currituck 37053 37053     27
## 5 0.153     2.206  1832    1832 Northampton 37131 37131     66
## 6 0.097     1.670  1833    1833 Hertford 37091 37091     46
##   BIR74 SID74 NWBIR74 BIR79 SID79 NWBIR79
## 1 1091     1     10  1364     0     19
## 2 487      0     10  542      3     12
## 3 3188     5     208  3616     6     260
## 4 508      1     123  830      2     145
## 5 1421     9    1066  1606     3    1197
## 6 1452     7    954  1838     5    1237
##           geometry
## 1 MULTIPOLYGON (((-81.47276 3...
## 2 MULTIPOLYGON (((-81.23989 3...
## 3 MULTIPOLYGON (((-80.45634 3...
## 4 MULTIPOLYGON (((-76.00897 3...
## 5 MULTIPOLYGON (((-77.21767 3...
## 6 MULTIPOLYGON (((-76.74506 3...

```

4.4.3 Construcción de mapas

Los mapas son muy útiles para transmitir información geoespacial. En esta sección se presentan ejemplos simples que demuestran el uso de algunos de los paquetes más comunes para la creación de mapas en R, tales como:

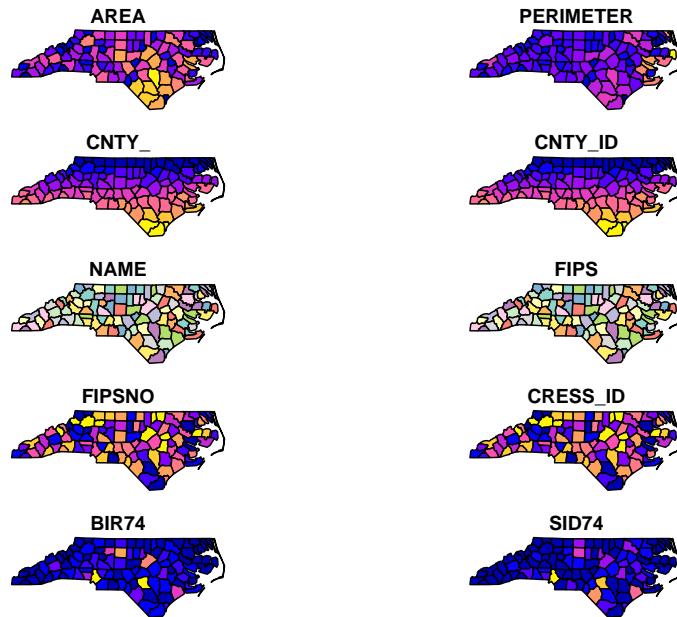
- **ggplot2**
- **leaflet**
- **mapview**
- **tmap**

A lo largo del resto del documento se mostrarán ejemplos más complejos que permiten visualizar resultados de diversas aplicaciones utilizando principalmente los paquetes **ggplot2** y **leaflet**.

4.4. INTEGRACIÓN DE CAPAS VECTORIALES DESDE SHAPEFILES115

- Un mapa del estado de **Carolina del Norte**, importado mediante el paquete **sf**, puede generarse de la siguiente manera:

```
plot(map)
```



4.4.3.1 Paquete ggplot2

ggplot2 es un paquete que permite crear gráficos basados en la **gramática de los gráficos**.

Esto significa que podemos construir una visualización utilizando la función **ggplot()** y los siguientes elementos:

1. **Datos a visualizar.**
2. **Formas geométricas** que representan los datos, como puntos o barras.
Estas se especifican mediante funciones **geom_***.
 - Ejemplo: **geom_point()** se usa para puntos.
 - Ejemplo: **geom_histogram()** se usa para histogramas.
3. **Estética de los objetos geométricos.**
La función **aes()** se emplea para mapear variables de los datos a las propiedades visuales de los objetos (color, tamaño, forma, posición).
4. **Elementos opcionales**, tales como escalas, títulos, etiquetas, leyendas y temas.

116 CHAPTER 4. VISUALIZACIÓN DE DATOS GEOGRÁFICOS CON GGMAP

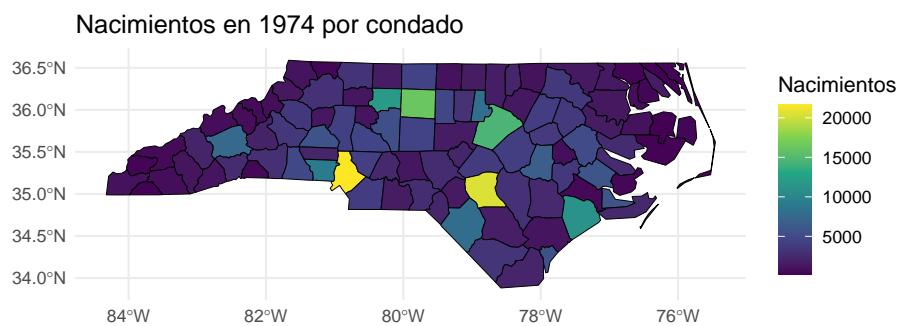
Podemos crear mapas utilizando la función `geom_sf()` y proporcionando un objeto de clase `sf`.

Si los datos disponibles son un objeto espacial de clase `SpatialPolygonsDataFrame`, podemos convertirlo fácilmente a un objeto `sf` con la función `st_as_sf()` del paquete `sf`.

- Por ejemplo, podemos crear un mapa de muertes súbitas infantiles en Carolina del Norte en 1974 (`SID74`) de la siguiente forma:

```
# Graficar por la variable nacimientos en 1974)
```

```
ggplot(map) +
  geom_sf(aes(fill = BIR74), color = "black") +
  scale_fill_viridis_c() +
  labs(title = "Nacimientos en 1974 por condado",
       fill = "Nacimientos") +
  theme_minimal()
```



- Veamos más mapas

```
library(patchwork)
p1 <- ggplot(map) +
```

4.4. INTEGRACIÓN DE CAPAS VECTORIALES DESDE SHAPEFILES117

```

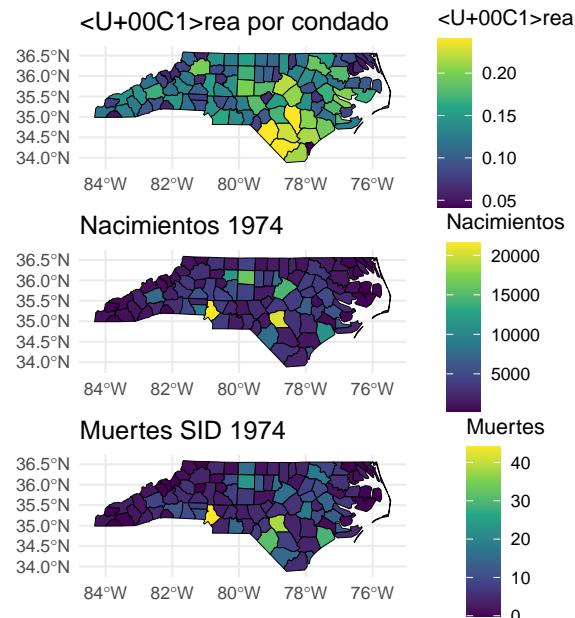
geom_sf(aes(fill = AREA), color = "black") +
scale_fill_viridis_c() +
labs(title = "Área por condado", fill = "Área")+
theme_minimal()

p2 <- ggplot(map) +
geom_sf(aes(fill = BIR74), color = "black") +
scale_fill_viridis_c() +
labs(title = "Nacimientos 1974", fill = "Nacimientos")+
theme_minimal()

p3 <- ggplot(map) +
geom_sf(aes(fill = SID74), color = "black") +
scale_fill_viridis_c() +
labs(title = "Muertes SID 1974", fill = "Muertes")+
theme_minimal()

# Combinar en un mismo layout
p1/p2/p3

```



- Para guardar un gráfico producido con **ggplot2**, podemos usar la función **ggsave()**. Otra alternativa es guardar el gráfico especificando un dispositivo de salida (por ejemplo, png, pdf), imprimir el gráfico y luego cerrar el dispositivo con **dev.off()**.

Ejemplo guardando como archivo **PNG**:

```
# Guardar gráfico como archivo PNG
png("plot.png")
ggplot(map) +
  geom_sf(aes(fill = SID74)) +
  scale_fill_viridis_c() +
  theme_bw()
dev.off()
```

- Además, existen paquetes que amplían las capacidades de **ggplot2**:
 - **ggridge**: permite crear gráficos animados.
 - **plotly**: permite crear gráficos interactivos.

Estos paquetes pueden combinarse con **ggplot2** para enriquecer la visualización de datos.

El paquete **plotly** permite transformar un gráfico de **ggplot2** en un objeto interactivo.

De esta forma, el usuario puede hacer zoom, mover el mapa, pasar el ratón sobre las regiones y explorar la información de manera dinámica.

- A continuación, se muestra un ejemplo con el shapefile de **Carolina del Norte**, visualizando la variable **SID74** (muertes súbitas infantiles en 1974):

```
library(sf)
library(ggplot2)
library(plotly)

# Crear gráfico con ggplot2
p <- ggplot(map) +
  geom_sf(aes(fill = SID74, text = paste("Condado:", NAME, "<br>SID74:", SID74)),
          color = "black") +
  scale_fill_viridis_c() +
  labs(title = "Muertes súbitas infantiles en Carolina del Norte (1974)",
       fill = "SID74") +
  theme_minimal()

# Convertir a interactivo con plotly
ggplotly(p, tooltip = "text")
```

- Por otro lado, realicemos el mapa de áreas silvestres en Montana” según la clasificación por estatus de delimitación

4.4. INTEGRACIÓN DE CAPAS VECTORIALES DESDE SHAPEFILES119

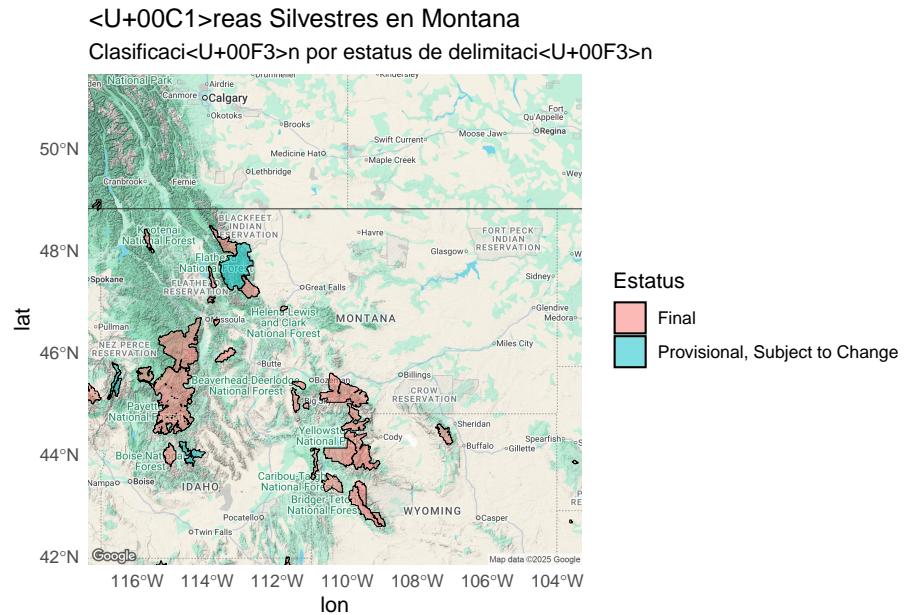
```
library(sf)
library(ggplot2)
library(ggmap)

# Leer shapefile
wild <- st_read("data/S_USA.Wilderness.shp")

## Reading layer `S_USA.Wilderness' from data source
##   `C:\Users\cdeor\OneDrive\Documentos\Books_CienciaDatos\rbook_dataviz\data\S_USA.Wilderness.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 447 features and 8 fields
## Geometry type: MULTIPOLYGON
## Dimension:      XY
## Bounding box:  xmin: -139.6362 ymin: 18.24841 xmax: -65.77081 ymax: 59.9997
## Geodetic CRS:  NAD83

# Descargar mapa base
montana_map <- get_googlemap(center = "Montana", zoom = 6, maptype = "terrain")

# mapa con ggplot
ggmap(montana_map) +
  geom_sf(data = wild, aes(fill = BOUNDARYST),
          inherit.aes = FALSE, color = "black", alpha = 0.5) +
  labs(title = "Áreas Silvestres en Montana",
       subtitle = "Clasificación por estatus de delimitación",
       fill = "Estatus") +
  theme_minimal()
```



4.4.3.2 Paquete `tmap`

El paquete `tmap` se utiliza para generar mapas temáticos con gran flexibilidad. Los mapas se crean usando la función `tm_shape()` y añadiendo capas con las funciones `tm_*`().

Además, podemos crear mapas **estáticos** o **interactivos** según el modo:

- `tmap_mode("plot")`: mapa estático.
- `tmap_mode("view")`: mapa interactivo.

Por ejemplo, un mapa interactivo de la variable SID74 en Carolina del Norte puede crearse de la siguiente forma:

```
# install.packages("tmap")

library(tmap)
tmap_mode("view")
tm_shape(map) +
  tm_polygons("SID74")
```

4.4. INTEGRACIÓN DE CAPAS VECTORIALES DESDE SHAPEFILES121

4.4.3.3 Paquete leaflet

Leaflet es una librería JavaScript muy popular para crear mapas interactivos. El paquete **leaflet** en **R** facilita su integración y control, permitiendo crear mapas dinámicos directamente desde **R**.

Podemos crear un mapa básico con **leaflet()** y añadir capas usando funciones como:

- **addTiles()**: agregar un mapa de fondo.
- **addPolygons()**: añadir polígonos.
- **addLegend()**: añadir una leyenda.

Para que los mapas funcionen correctamente en **leaflet**, es necesario que los datos estén proyectados en **EPSG:4326 (WGS84)**.

Si los datos tienen otra proyección, podemos transformarlos con **st_transform()** del paquete **sf**.

- Continuando con los datos del **contado de Carolina del norte**

```
st_crs(map)
```

```
## Coordinate Reference System:  
##   User input: NAD27  
##   wkt:  
## GEOCRS["NAD27",  
##        DATUM["North American Datum 1927",  
##                 ELLIPSOID["Clarke 1866",6378206.4,294.978698213898,  
##                           LENGTHUNIT["metre",1]],  
##                 PRIMEM["Greenwich",0,  
##                           ANGLEUNIT["degree",0.0174532925199433]],  
##                 CS[ellipsoidal,2],  
##                   AXIS["latitude",north,  
##                         ORDER[1],  
##                         ANGLEUNIT["degree",0.0174532925199433]],  
##                   AXIS["longitude",east,  
##                         ORDER[2],  
##                         ANGLEUNIT["degree",0.0174532925199433]],  
##                 ID["EPSG",4267]]
```

- Esto significa que tu shapefile está proyectado en el **datum NAD27**, cuyo código estándar es **EPSG:4267**. Por lo que las coordenadas del shapefile de **NAD27 (EPSG:4267)** debemos transformarla **WGS84 (EPSG:4326)**, entonces

```
map_leaflet <- st_transform(map, 4326)
```

Luego, creamos una paleta de colores usando la función `colorNumeric()` y graficamos el mapa con `leaflet`. Para ello, usamos las funciones:

- `addTiles()`: agrega el mapa base.
- `addPolygons()`: agrega los polígonos, especificando:
 - el color del borde (`color`),
 - el color de relleno (`fillColor`),
 - la opacidad (`fillOpacity`),
 - y la leyenda (`addLegend()`)

```
# install.packages("leaflet")
library(leaflet)

pal <- colorNumeric("YlOrRd", domain = map_leaflet$SID74)

leaflet(map_leaflet) %>%
  addTiles() %>%
  addPolygons(
    color = "white",
    fillColor = ~ pal(SID74),
    fillOpacity = 1
  ) %>%
  addLegend(
    pal = pal,
    values = ~SID74,
    opacity = 1
  )
```

*# inicializar mapa con datos (objeto sf)
agregar mapa base (tiles de OpenStreetMap)
agregar polígonos al mapa
color del borde de los polígonos
color de relleno según paleta definida
opacidad del relleno (1 = totalmente opaco)
agregar leyenda
paleta de colores usada
variable representada en el mapa
opacidad de la leyenda*

Para guardar un mapa creado en formato **HTML**, podemos usar la función `saveWidget()` del paquete `htmlwidgets`.

Si deseamos guardar el mapa como una **imagen estática (PNG)**, primero lo guardamos en formato HTML con `saveWidget()` y luego capturamos la versión estática usando la función `webshot()` del paquete `webshot`.

4.4.3.4 Paquete `mapview`

El paquete `mapview` permite crear de manera muy rápida visualizaciones interactivas para explorar tanto las **geometrías espaciales** como las **variables**

4.4. INTEGRACIÓN DE CAPAS VECTORIALES DESDE SHAPEFILES123

asociadas a los datos.

Una de sus principales ventajas es la simplicidad: basta con llamar a la función `mapview()` indicando el objeto espacial y la variable a visualizar mediante el argumento `zcol`.

- Por ejemplo, podemos crear un mapa interactivo de la variable `SID74` en los **condados de Carolina del Norte** de la siguiente forma:

```
# install.packages("mapview")
library(mapview)

mapview(map, zcol = "SID74")
```

El paquete `mapview` resulta muy útil para inspeccionar datos espaciales de manera rápida, pero además permite personalizar los mapas añadiendo elementos como **leyendas**, **mapas de fondo** y múltiples capas sincronizadas.

- Por ejemplo, podemos crear un mapa con fondo de **CartoDB.DarkMatter** y colorear los polígonos usando la paleta "`YlOrRd`" del paquete **RColorBrewer** de la siguiente manera:

```
# install.packages("RColorBrewer")
library(RColorBrewer)

pal <- colorRampPalette(brewer.pal(9, "YlOrRd"))

mapview(map,
        zcol = "SID74",
        map.types = "CartoDB.DarkMatter",
        col.regions = pal
      )
```

- También podemos usar la función `sync()` para producir una vista en cuadrícula de **múltiples mapas sincronizados**, creados con `mapview` o `leaflet`. Esto permite explorar varias variables a la vez con **zoom** y **paneo sincronizados**.
- Por ejemplo, podemos crear mapas de muertes súbitas infantiles en **1974 (SID74)** y **1979 (SID79)**, y luego mostrarlos sincronizados de la siguiente manera:

```
# install.packages("leafsync")
library(leafsync)
```

```
m74 <- mapview(map, zcol = "SID74")
m79 <- mapview(map, zcol = "SID79")
m <- leafsync::sync(m74, m79)
```

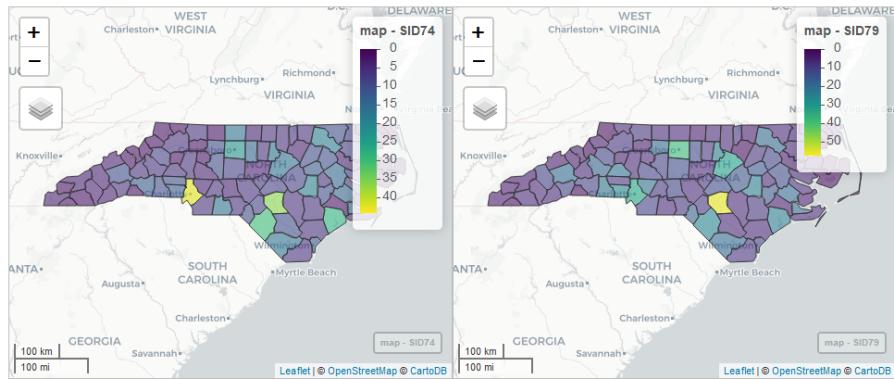


Figure 4.6: Mapas interactivos

Ejercicio para entregar

Considere el conjunto de datos violent_crimes.csv dentro del archivo RDataSets.zip, el cual abarca una gran variedad de delitos.

Realice visualizaciones geográficas para cada tipo de crimen aplicando los ítems estudiados en esta sección.

Verifique si existe algún patrón dentro de los delitos violentos que pueda explorarse visualmente.

Tenga en cuenta que los datos incluyen columnas de longitud, latitud, categoría del delito, fecha y hora.

Chapter 5

Análisis con datos faltantes y detección de atípicos

5.1 Manejo de datos faltantes

Los **datos faltantes** son uno de los temas menos abordados en la mayoría de los textos introductorios. Probablemente, esto se deba a que hasta hace poco abundaba más la teoría que las aplicaciones prácticas en su tratamiento. Sin embargo, con el crecimiento del análisis de datos, hoy es un asunto que no puede ignorarse, pues impacta directamente en la calidad de los resultados.

Las investigaciones más técnicas aún los consideran un reto, sobre todo desde el punto de vista matemático. Aun así, la relevancia práctica es clara: los datos faltantes están presentes en encuestas, mediciones de laboratorio, registros de salud, sistemas de monitoreo ambiental, entre muchos otros.

Esta sección presenta una introducción al tema y a las técnicas más efectivas para abordarlo. En términos generales, cuando los datos faltan, **la mejor manera de tratarlos es imputarlos, no ignorarlos**. Esto implica completar la información ausente con valores estimados que permitan mantener la coherencia y la validez de los análisis.

5.1.1 Consecuencias de los datos faltantes

- Dificultan la aplicación de modelos estadísticos.
- Reducen la potencia de los análisis.
- Pueden sesgar los resultados si no se tratan adecuadamente.

- En la práctica, han impedido la participación de individuos en encuestas o cuestionarios, lo que limita la representatividad de los estudios.

5.1.2 Tratamientos para los datos faltantes

Existen diferentes estrategias:

- **Eliminación de casos incompletos:** opción sencilla, pero que reduce el tamaño muestral y puede sesgar los resultados.
- **Imputación simple:** sustitución de los valores faltantes por medidas como la media, mediana o moda.
- **Imputación múltiple:** genera varios conjuntos de datos con imputaciones diferentes, permitiendo obtener estimaciones más robustas y reducir la incertidumbre.

En la práctica, **no hay un único método correcto**. Todo depende del contexto, del tipo de variable y de la magnitud del problema.

5.2 Manejo de datos atípicos

Los **datos atípicos (outliers)** son valores extremos que difieren significativamente del resto de las observaciones. Aunque en ocasiones corresponden a errores de medición, también pueden reflejar fenómenos reales que aportan información valiosa.

5.2.1 Consecuencias de los datos atípicos

- Distorsionan medidas estadísticas como la media y la desviación estándar.
- Afectan la estimación de parámetros en modelos de regresión.
- Pueden generar interpretaciones erróneas si no se identifican y analizan adecuadamente.

5.2.2 Tratamientos para los datos atípicos

- **Detección** mediante métodos gráficos (boxplot, histogramas, diagramas de dispersión) y técnicas estadísticas (z-score, IQR, modelos robustos).
- **Eliminación de outliers** cuando son claramente errores de medición o registros imposibles.

- **Transformaciones** (logarítmica, raíz cuadrada) que reducen la influencia de valores extremos.
 - **Métodos robustos** que limitan el impacto de los outliers sin eliminarlos (regresiones robustas, estimadores no paramétricos).
-

En conclusión, el análisis de datos faltantes y atípicos no debe verse como un paso opcional, sino como parte esencial del **preprocesamiento de datos**. La validez de cualquier modelo estadístico o de machine learning depende directamente de cómo se manejen estas problemáticas.

En este documento hemos introducido tanto las técnicas de imputación como las de detección y tratamiento de valores atípicos, ofreciendo un panorama que permite avanzar hacia análisis más confiables, precisos y con base científica.

5.3 Visualización de los datos faltantes

Para demostrar la visualización de patrones de **datos ausentes**, primero necesitamos crear algunos valores faltantes. Para **mostrar cómo utilizar la imputación múltiple en un escenario semirrealista**, vamos a crear una versión del conjunto de datos `mtcars` con algunos valores eliminados.

Configuramos `set.seed()` (para garantizar la **aleatoriedad determinista**) y creamos una nueva variable que mantenga nuestro conjunto de datos modificado.

1. Los datos se extrajeron de la revista *Motor Trend US* de 1974, y comprenden el **consumo de combustible y 10 aspectos del diseño y el rendimiento de 32 automóviles** (modelos de 1973 a 1974) (ver `mtcars`).

```
library(tidyverse)

set.seed(2)
miss_mtcars <- mtcars
```

- En primer lugar, vamos a crear **siete valores faltantes** en `drat` (alrededor del 20%), **cinco valores faltantes** en la columna `mpg` (alrededor del 15%), **cinco valores faltantes** en la columna `cyl`, **tres valores faltantes** en `wt` (alrededor del 10%), y **tres valores faltantes** en `vs`.

```
some_rows <- sample(1:nrow(miss_mtcars), 7)
miss_mtcars$drat[some_rows] <- NA

some_rows <- sample(1:nrow(miss_mtcars), 5)
miss_mtcars$mpg[some_rows] <- NA

some_rows <- sample(1:nrow(miss_mtcars), 5)
miss_mtcars$cyl[some_rows] <- NA

some_rows <- sample(1:nrow(miss_mtcars), 3)
miss_mtcars$wt[some_rows] <- NA

some_rows <- sample(1:nrow(miss_mtcars), 3)
miss_mtcars$vs[some_rows] <- NA
```

- Ahora, vamos a crear **cuatro valores faltantes** en qsec, pero sólo para los coches automáticos:

```
only_automatic <- which(miss_mtcars$am==0)
some_rows <- sample(only_automatic, 4)
miss_mtcars$qsec[some_rows] <- NA
```

- Ahora, observemos las primeras filas del nuevo conjunto de datos

```
head(miss_mtcars)
```

```
##          mpg cyl disp hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   NA 160 110 3.90 2.620 16.46 NA 1 4 4
## Mazda RX4 Wag 21.0     6 160 110 3.90 2.875 17.02  0 1 4 4
## Datsun 710    22.8   NA 108  93 3.85 2.320 18.61  1 1 4 1
## Hornet 4 Drive 21.4     6 258 110 3.08 3.215 19.44  1 0 3 1
## Hornet Sportabout 18.7     8 360 175 3.15 3.440 17.02  0 0 3 2
## Valiant       18.1     6 225 105   NA 3.460 20.22  1 0 3 1
```

```
str(miss_mtcars)
```

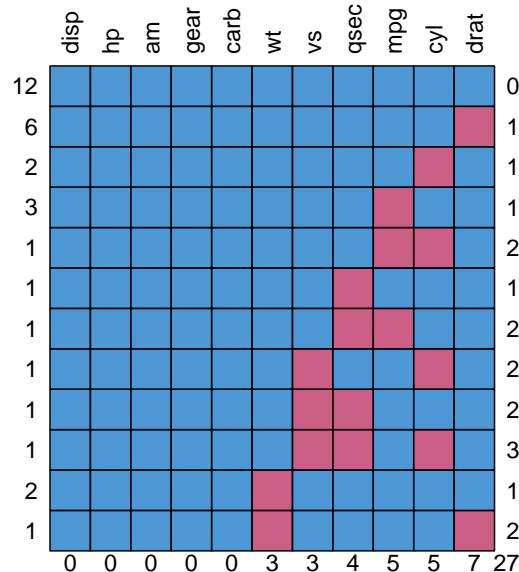
```
## 'data.frame': 32 obs. of 11 variables:
## $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 NA 19.2 ...
## $ cyl : num NA 6 NA 6 8 6 8 4 4 6 ...
## $ disp: num 160 160 108 258 360 ...
## $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num 3.9 3.9 3.85 3.08 3.15 NA 3.21 NA 3.92 3.92 ...
```

```
## $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num  16.5 17 18.6 19.4 17 ...
## $ vs   : num  NA 0 1 1 0 1 NA 1 1 1 ...
## $ am   : num  1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

2. Ahora vamos a **visualizar los datos faltantes**. La primera forma en que vamos a *visualizar el patrón de los datos faltantes* es utilizando la función `md.pattern` del paquete `mice` (que es también el paquete que usaremos para **imputar** nuestros datos faltantes).

```
library(mice)

md.pattern(miss_mtcars, plot = TRUE, rotate.names = TRUE)
```



```
##      disp hp am gear carb wt vs qsec mpg cyl drat
## 12     1  1  1    1   1  1  1    1   1  1  1  0
## 6      1  1  1    1   1  1  1    1   1  1  0  1
## 2      1  1  1    1   1  1  1    1   1  0  1  1
## 3      1  1  1    1   1  1  1    1   0  1  1  1
```

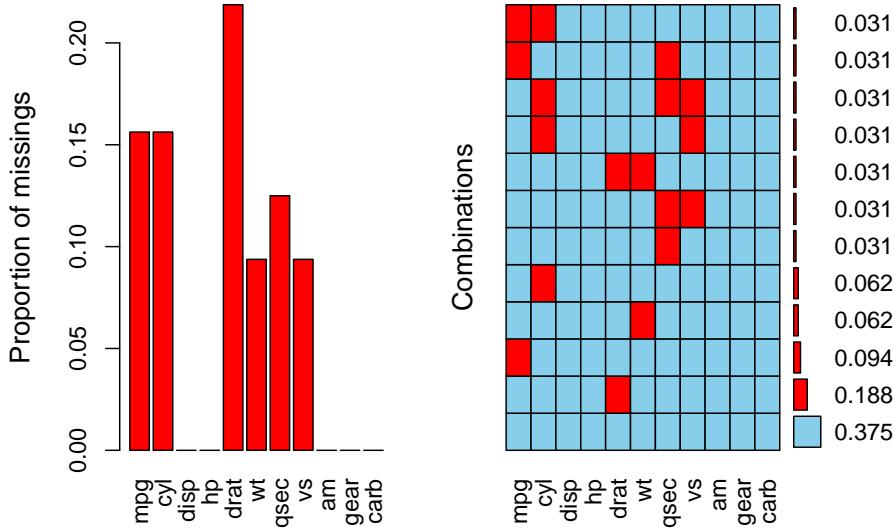
130 CHAPTER 5. ANÁLISIS CON DATOS FALTANTES Y DETECCIÓN DE ATÍPICOS

```
## 1    1 1 1    1    1 1 1    1    0 0    1 2
## 1    1 1 1    1    1 1 1    0 1 1    1 1
## 1    1 1 1    1    1 1 1    0 0 1    1 2
## 1    1 1 1    1    1 1 0    1 1 0    1 2
## 1    1 1 1    1    1 1 0    0 1 1    1 2
## 1    1 1 1    1    1 1 0    0 1 0    1 3
## 2    1 1 1    1    1 0 1    1 1 1    1 1
## 1    1 1 1    1    1 0 1    1 1 1    0 2
##    0 0 0    0    0 3 3    4 5 5    7 27
```

- **Columnas (arriba):** cada nombre corresponde a una **variable** del conjunto de datos (`disp`, `hp`, `am`, `gear`, `carb`, `wt`, `vs`, `mpg`, `cyl`, `drat`, `qsec`, etc.).
- **Cuadrícula de colores:**
 - **Azul** = valor **observado** (no faltante).
 - **Rosa** = valor **faltante** (`NA`).
 - Cada **fila** de la cuadrícula representa un **patrón único** de ausencia/presencia a través de todas las variables.
- **Números en el margen izquierdo:** indican la **frecuencia** (cuántas filas del dataset) que exhiben **ese patrón** específico.
 - Ejemplo: si ves “12” en la primera fila, significa “hay 12 observaciones con ese patrón” (a menudo, la fila superior suele ser el **patrón completo**: sin `NA`).
- Números en el **margen derecho**: muestran **cuántas variables faltan** en ese patrón (0, 1, 2, ...).
 - Sirve para distinguir patrones con 1, 2 o más variables ausentes.
- Números en la **base** (debajo de cada columna): total de **valores faltantes por variable**.
 - Útil para ver rápidamente **qué variable tiene más NA**.
 - En tu simulación, verás más `NA` en variables a las que se los inyectaste (p. ej., `drat`, `mpg`, `cyl`, `wt`, `vs`, y `qsec` para automáticos).

3. Ahora vamos a visualizar el patrón de datos faltantes gráficamente utilizando el paquete VIM, el cual debe instalar en caso de no contar con este.

```
library(VIM)
aggr(miss_mtcars, numbers=TRUE)
```

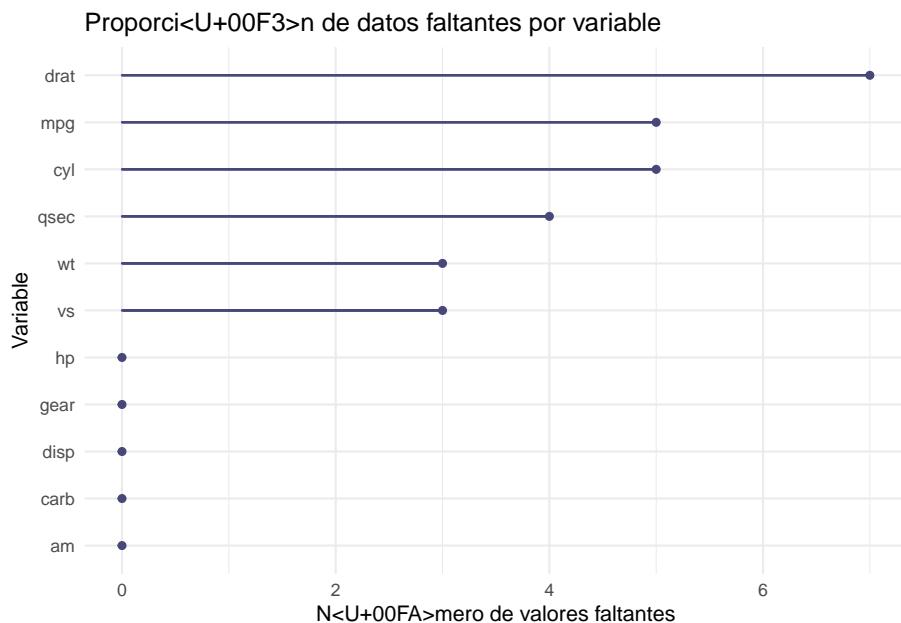


- A simple vista, esta representación nos muestra que la columna **drat** **representa la mayor proporción de datos faltantes por columnas**, seguida de **mpg, cyl, qsec, vs** y **wt**.
- El gráfico de la derecha nos muestra información similar a la salida de **md.pattern**. Esta representación, sin embargo, facilita identificar si hay algún **patrón sistemático de omisión**.
- **Interpretación de los colores:**
 - Celdas azules → datos no ausentes.
 - Celdas rojas → datos faltantes.
- **Números en la derecha del gráfico:** representan la **proporción de filas** que muestran ese patrón de datos faltantes.
- En este ejemplo, el **37.5% de las filas no contienen ningún tipo de dato faltante**.

4. Por otro lado, un gráfico de barras comparando la proporción de NA por variable

```
library(naniar)

# Gráfico de proporción de NA por variable
gg_miss_var(miss_mtcars) +
  ggplot2::labs(title = "Proporción de datos faltantes por variable",
                x = "Variable",
                y = "Número de valores faltantes")
```



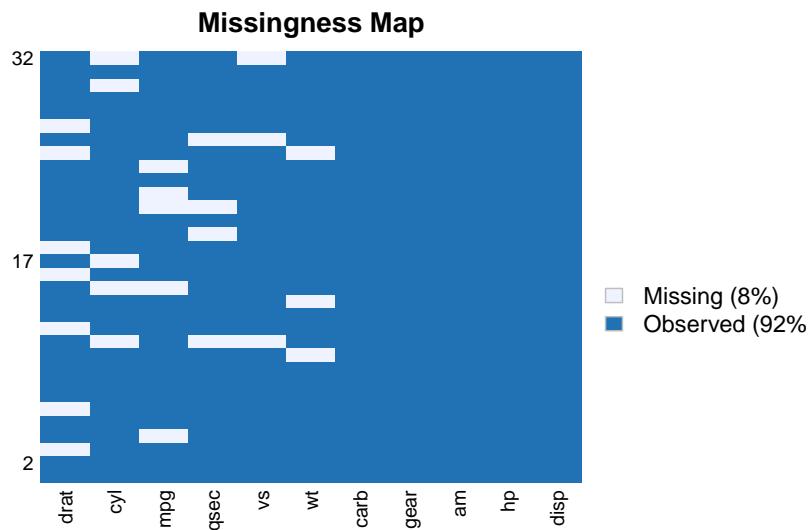
- Las variables con **mayor número de valores faltantes** son:
 - **qsec** y **drat** con **7** valores faltantes cada una.
 - **mpg** y **cyl** con **5** valores faltantes cada una.
- En un segundo nivel de afectación:
 - **wt** y **vs** con **3** valores faltantes cada una.
- El resto de las variables (**hp**, **gear**, **disp**, **carb**, **am**) presentan **muy pocos o ningún dato faltante**, lo que indica que no requieren un tratamiento prioritario.

En conclusión, el **patrón de ausencia no está distribuido de manera uniforme** en todas las variables.

- Existen **columnas críticas** como `qsec`, `drat`, `mpg` y `cyl`, que concentran la mayor parte de los datos ausentes.
 - Para aplicar un método de imputación, será necesario:
 - Evaluar si estas variables son relevantes para el análisis principal.
 - Considerar imputaciones robustas como la **imputación múltiple** en aquellas con mayor proporción de NA.
 - Posiblemente eliminar o transformar las observaciones si la proporción de ausentes supera un umbral crítico.
-

5. Otra forma de mirar los datos faltantes es usando el paquete de **Amelia**:

```
library(Amelia)
missmap(miss_mtcars)
```



- Los datos faltantes no están distribuidos al azar: se concentran en variables como `qsec`, `drat`, `cyl`, `mpg`, `vs` y `wt`.

- Existen patrones visibles: varias filas muestran **faltantes en más de una variable a la vez**, lo cual podría indicar un mecanismo de omisión sistemático.
 - Este tipo de visualización permite identificar:
 - **Observaciones con muchos faltantes** (filas con varias celdas blancas).
 - **Variables críticas** con más datos ausentes.
-

6. **Parallel boxplots:** El método VIM permite **comparar las distribuciones condicionales de una variable continua** según un conjunto de variables, con valores recodificados como *faltantes* o *no faltantes*, mediante múltiples **diagramas de caja paralelos**.

- Estos diagramas son útiles para explorar si una **variable continua** explica la distribución de valores faltantes en otra variable. En la siguiente figura se muestra un ejemplo con la variable *age* en los datos de *EU-SILC User Guide*.
- **Además del diagrama de caja estándar**, se muestran diagramas de caja agrupados por valores observados (*gris claro*) y faltantes (*gris oscuro*) en los diferentes componentes de ingresos.
 - El ancho de las cajas puede ser proporcional al tamaño del grupo o de tamaño igual.
 - En este ejemplo no es posible utilizar la primera opción porque la **proporción de valores faltantes es cercana a 0** para algunos componentes de ingresos.
- En muchos componentes, *la presencia de valores faltantes claramente depende de la magnitud de los valores de la edad*. Por ejemplo, los valores faltantes en la variable *py080n* (beneficios por desempleo) ocurren predominantemente para **individuos más jóvenes**. Esto indica situaciones **MAR (Missing At Random)** para los valores faltantes en estas variables, lo cual es información útil para los especialistas en la materia.
- **Scatterplots:** La implementación en VIM también incluye **boxplots para datos disponibles y faltantes en los márgenes del gráfico**.

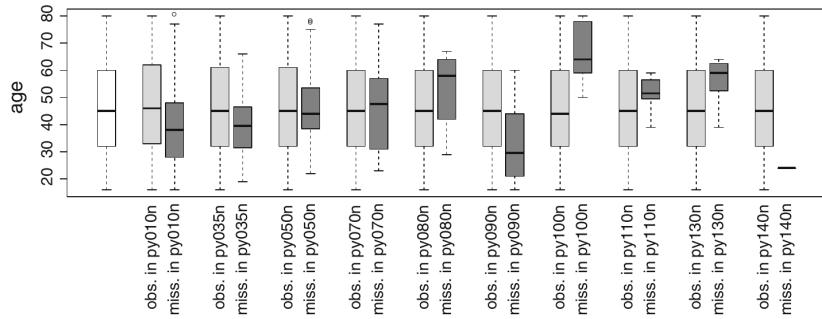
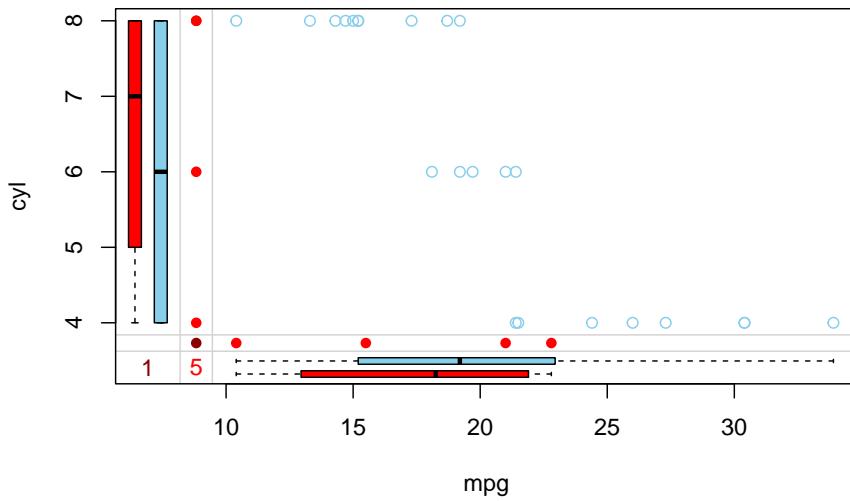


Figure 5.1: Boxplots paralelos

- Las **frecuencias de valores faltantes** en una o ambas variables se representan mediante números en la esquina inferior izquierda.
- Este gráfico será denominado en adelante `marginplot`.

```
marginplot(miss_mtcars[c(1,2)])
```



- El **diagrama de caja rojo** de la izquierda muestra la distribución del número de cilindros `cyl`, agrupados por datos faltantes para `mpg`, mientras que el **diagrama de caja azul** muestra esta distribución para datos

observados de `mpg`. Lo mismo ocurre con los gráficos de caja de `cyl` en la parte inferior del gráfico.

Si los datos tienen mecanismo **MCAR** (Missing Completely At Random), **se espera que los gráficos de caja rojos y azules sean muy similares**.

Para más información, ver el artículo asociado: *Exploring incomplete data using visualization techniques*.

7. Otra forma de identificar NAs por columnas es de la siguiente forma

```
miss_mtcars %>%
  summarise(across(everything(), ~ sum(is.na(.))), .names = "NA_{.col}"))

##   NA_mpg NA_cyl NA_disp NA_hp NA_drat NA_wt NA_qsec NA_vs NA_am
## 1      5      5       0     0       7      3       4      3     0
##   NA_gear NA_carb
## 1        0        0
```

5.4 Eliminación de datos faltantes

Uno de los métodos más utilizados por los científicos de datos para tratar valores faltantes es **omitir los casos que contienen NA y analizar únicamente las observaciones completas**. Este procedimiento se conoce como **eliminación por lista o análisis de casos completos**.

En R, se implementa con la función `na.omit()`, la cual elimina todas las filas que contengan uno o más valores faltantes.

Ventajas

Es un método **simple y rápido** de aplicar.

No requiere cálculos adicionales ni suposiciones complejas.

Desventajas

Si los datos son MCAR (Missing Completely At Random), la eliminación puede generar una reducción innecesaria del tamaño muestral, lo que afecta la precisión de las estimaciones y aumenta los errores estándar.

Si los datos no son MCAR, este método introduce sesgos importantes como medias, estimaciones de regresión, correlaciones. En consecuencia, la supresión de casos puede producir submuestras poco representativas y análisis engañosos.

Ejemplo con el conjunto de datos airquality

Tomaremos el conjunto de datos airquality, que contiene mediciones diarias de la calidad del aire en Nueva York de mayo a septiembre de 1973. Este dataset incluye variables con valores faltantes (NA).

El conjunto de datos representa 154 días consecutivos, por lo que la eliminación de filas completas puede afectar la continuidad temporal y, en consecuencia, cualquier análisis de series de tiempo.

1. Para este ejemplo, eliminaremos algunos puntos de datos del conjunto.
 - En el caso de **variables categóricas**, la sustitución de valores faltantes **no siempre es aconsejable**.
 - Algunas prácticas comunes consisten en sustituir los valores faltantes por la **moda** de la variable.
 - Sin embargo, esta práctica es **cuestionable** porque puede introducir sesgos y distorsionar la distribución real.
 - En este caso, como no hay variables categóricas con valores faltantes, no es necesario aplicar este tratamiento. Si fuera necesario, siempre es posible añadirlas nuevamente al conjunto de datos más adelante.
 - Podemos hacer una primera inspección de los datos y de sus valores faltantes con:

```
# Exploración del dataset airquality
data <- airquality
knitr::kable(head(data))
```

Ozone	Solar.R	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

- Agreguemos más datos faltantes

```
data[4:10, 3] <- rep(NA, 7)
data[1:5, 4] <- NA
```

```
data <- data[-c(5,6)]
summary(data)
```

	Ozone	Solar.R	Wind	Temp
## Min.	: 1.00	Min. : 7.0	Min. : 1.700	Min. : 57.00
## 1st Qu.:	18.00	1st Qu.:115.8	1st Qu.: 7.400	1st Qu.:73.00
## Median :	31.50	Median :205.0	Median : 9.700	Median :79.00
## Mean :	42.13	Mean :185.9	Mean : 9.806	Mean : 78.28
## 3rd Qu.:	63.25	3rd Qu.:258.8	3rd Qu.:11.500	3rd Qu.:85.00
## Max. :	168.00	Max. :334.0	Max. :20.700	Max. : 97.00
## NA's :	37	NA's :7	NA's :7	NA's :5

```
knitr::kable(head(data))
```

Ozone	Solar.R	Wind	Temp
41	190	7.4	NA
36	118	8.0	NA
12	149	12.6	NA
18	313	NA	NA
NA	NA	NA	NA
28	NA	NA	66

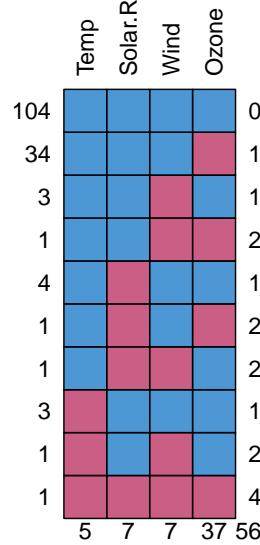
- Nótese que la variable **Ozone** es la que contiene **más puntos de datos faltantes**.
- A continuación, vamos a profundizar en los **patrones de datos ausentes**. Suponiendo que los datos sean **MCAR** (Missing Completely At Random), demasiados datos faltantes también pueden representar un problema.
- Como regla práctica, se considera que un **umbral máximo seguro es el 5% del total de datos** en conjuntos grandes. Si una variable o muestra presenta más del **5% de datos faltantes**, probablemente deba excluirse del análisis.
- Para verificarlo, revisamos:
 - Las **características (columnas)** con más del 5% de datos faltantes.
 - Las **muestras (filas)** en las que falta más del 5% de los datos.
- Para saber el conteo y el porcentaje de NAs usemos la función **pivot_longer**, este sirve para **reorganizar** esa tabla “ancha” en un formato **largo**

```
data %>%
  summarise(across(
    everything(),
    list(
      NA_count = ~ sum(is.na(.)),
      NA_percent = ~ mean(is.na(.)) * 100
    )
  )) %>%
  pivot_longer(
    everything(),
    names_to = c("Variable", "dummy", ".value"),
    names_sep = "_"
) %>%
  select(-dummy)
```

```
## # A tibble: 4 x 3
##   Variable count percent
##   <chr>     <int>   <dbl>
## 1 Ozone      37    24.2
## 2 Solar.R     7     4.58
## 3 Wind        7     4.58
## 4 Temp        5     3.27
```

- Vemos que a la variable `Ozone` le falta casi el **25% de los puntos de datos**, por lo que podríamos considerar la posibilidad de **eliminarla del análisis** o de **reunir más mediciones**. Las demás variables están por debajo del **umbral del 5%**, por lo que podemos mantenerlas en el análisis.
- Usemos la función `md.pattern()` para conocer mejor el **patrón de los datos que faltan**.

```
library(mice)
md.pattern(data, rotate.names = TRUE)
```



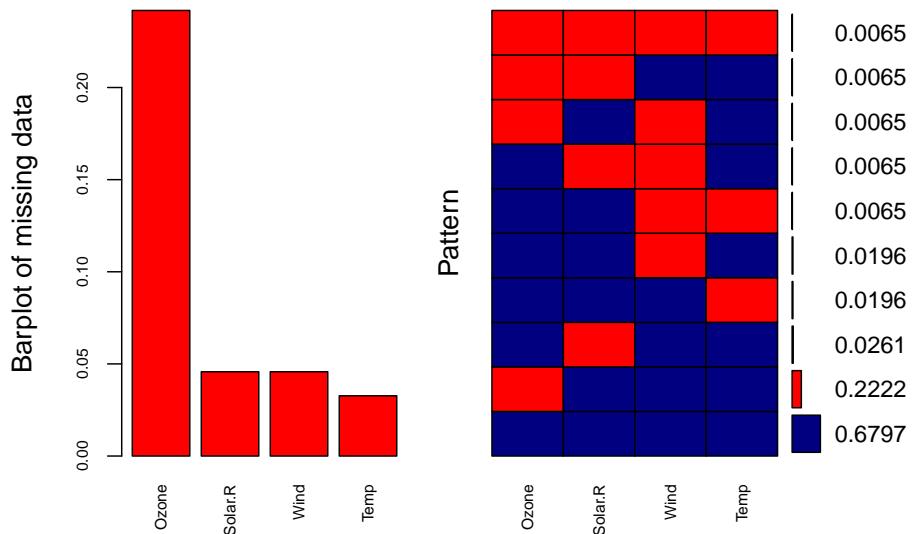
```
##      Temp Solar.R Wind Ozone
## 104     1       1    1     1   0
## 34      1       1    1     0   1
## 3       1       1    0     1   1
## 1       1       1    0     0   2
## 4       1       0    1     1   1
## 1       1       0    1     0   2
## 1       1       0    0     1   2
## 1       0       1    1     1   1
## 1       0       1    0     1   2
## 1       0       0    0     0   4
##      5       7    7    37  56
```

- La salida nos indica que **104 muestras están completas**, que a **34 muestras les falta sólo la medición de Ozone**, que a **4 muestras les falta sólo el valor de Solar.R**, y así sucesivamente.
- Una representación visual quizás más útil puede obtenerse utilizando el paquete VIM, de la siguiente manera:

```
library(VIM)

aggr_plot <- aggr(data, col=c('navyblue','red'),
                    numbers=TRUE, sortVars=TRUE,
```

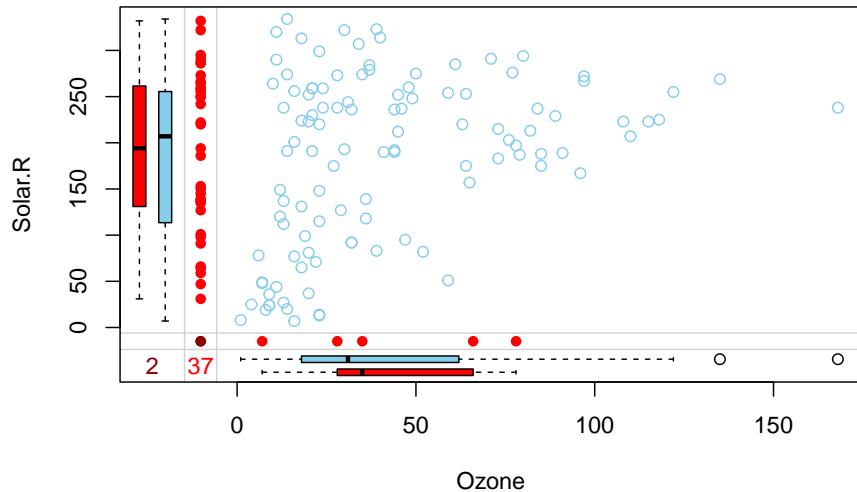
```
labels=names(data),cex.axis=.7,
gap=3, ylab=c("Barplot of missing data","Pattern"))
```



```
##  
## Variables sorted by number of missings:  
## Variable Count  
## Ozone 0.24183007  
## Solar.R 0.04575163  
## Wind 0.04575163  
## Temp 0.03267974
```

- El gráfico nos ayuda a entender que **casi el 70% de las muestras no presentan valores faltantes**, mientras que al **22%** les falta el valor de **Ozone** y las restantes muestran otros patrones de ausencia.
- A través de este enfoque, la situación parece un poco más clara. Otra aproximación visual útil es un **gráfico de caja especial**.

```
marginplot(data[, c(1, 2)])
```



- Aquí estamos limitados a trazar sólo **2 variables a la vez**, pero aun así podemos obtener algunas ideas interesantes.

- El **diagrama de caja rojo de la izquierda** muestra la distribución de **datos faltantes para Ozone**, mientras que el **diagrama de caja azul** muestra la distribución de los **puntos de datos restantes**.
- Lo mismo ocurre con los gráficos de caja de Solar.R en la parte inferior del gráfico.
- Si nuestra suposición de que los datos son **MCAR (Missing Completely At Random)** es correcta, entonces esperamos que **los gráficos de caja rojos y azules sean muy similares**.

2. A continuación, realizamos una simple función `na.omit()` para eliminar los casos que tienen NAs. Vemos que todas las filas que contenían algún NA en cualquier variable son eliminadas del dataframe.

```
data.omit <- na.omit(data)
head(data.omit)
```

```
##      Ozone Solar.R Wind Temp
## 12     16    256  9.7   69
## 13     11    290  9.2   66
## 14     14    274 10.9   68
## 15     18     65 13.2   58
```

```
## 16      14      334 11.5    64
## 17      34      307 12.0    66
```

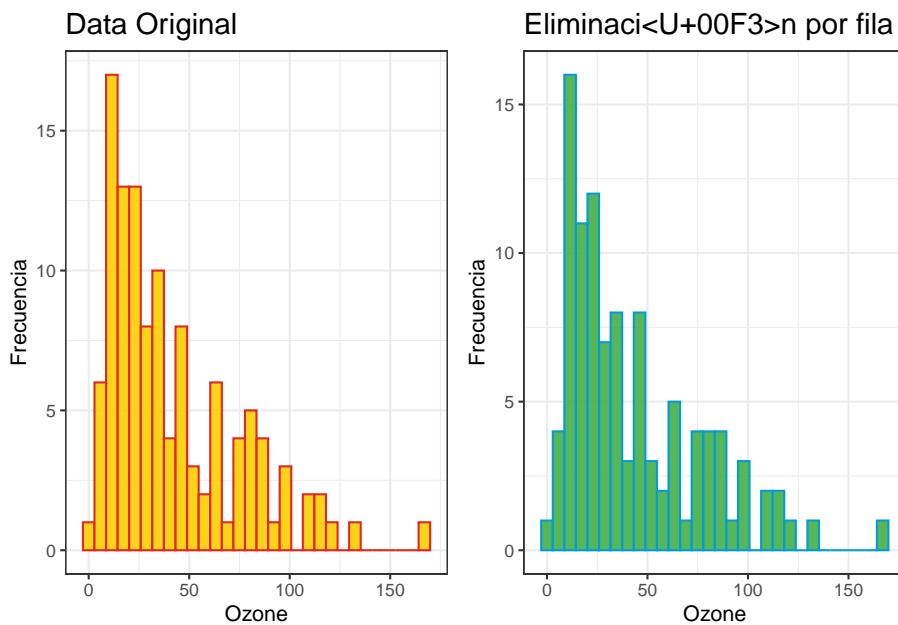
- Dibujemos los histogramas antes y después de la imputación/eliminación usando `ggplot`

```
library(ggplot2)
library(tidyverse)
library(gridExtra)

ggp1 <- ggplot(data.frame(value=data$Ozone), aes(x=value)) +
  geom_histogram(fill="#FBD000", color="#E52521", alpha=0.9) +
  ggtitle("Data Original ") +
  xlab('Ozone') + ylab('Frecuencia') +
  theme_bw() +
  theme(plot.title = element_text(size=15))

ggp2 <- ggplot(data.frame(value=data.omit$Ozone), aes(x=value)) +
  geom_histogram(fill="#43B047", color="#049CD8", alpha=0.9) +
  ggtitle("Eliminaci<U+00F3>n por fila") +
  xlab('Ozone') + ylab('Frecuencia') +
  theme_bw() +
  theme(plot.title = element_text(size=15))

grid.arrange(ggp1, ggp2, ncol = 2)
```



5.5 Imputación de datos con el promedio o mediana

La **imputación** es el proceso mediante el cual se reemplazan estos valores perdidos por estimaciones que permitan completar la base de datos y continuar con el análisis.

Existen múltiples técnicas de imputación. Entre las más sencillas y utilizadas están la imputación por **media** y la imputación por **mediana**.

5.5.1 La importancia de la distribución

Antes de imputar, es recomendable analizar la **distribución de los datos**:

- **Distribución aproximadamente normal (simétrica):** La media y la mediana son muy similares. En este caso, imputar con la **media** es razonable, ya que no introduce mucho sesgo y conserva el promedio de la variable.
- **Distribución asimétrica o con outliers:** La media se desplaza hacia los valores extremos, dejando de ser representativa. En este escenario, la **mediana** es más adecuada porque refleja mejor la tendencia central real y es robusta frente a valores atípicos.

En conclusión: la **forma de la distribución guía la elección** del método de imputación.

5.5.2 Imputación con la media

La imputación por la **media** consiste en reemplazar cada valor faltante por el promedio de los valores observados en la misma variable.

Ventajas de imputación con la media

Método rápido y fácil de implementar.

Mantiene el promedio de la variable en el conjunto de datos.

Adecuado si la distribución es normal y no hay outliers significativos.

Limitaciones de imputación con la media

Sensible a la presencia de valores extremos.

Reduce la variabilidad.

Puede distorsionar correlaciones con otras variables.

5.5.3 Imputación con la mediana

La imputación por la **mediana** consiste en reemplazar los valores faltantes por el valor central de los datos observados.

Ventajas de imputación con la mediana

Robusta frente a la presencia de valores atípicos.

Preserva mejor la tendencia central cuando la distribución es asimétrica.

Más confiable en variables sesgadas.

Limitaciones de imputación con la mediana

Al igual que la media, reduce la variabilidad.

Reemplaza todos los valores perdidos por un único valor fijo.

Puede alterar las relaciones estadísticas entre variables.

- Continuemos con el ejemplo de `airquality`:

```
data <- airquality

summary(data)

##      Ozone          Solar.R          Wind          Temp      
##  Min.   :  1.00   Min.   : 7.0   Min.   : 1.700   Min.   :56.00  
##  1st Qu.:18.00   1st Qu.:115.8  1st Qu.: 7.400   1st Qu.:72.00  
##  Median :31.50   Median :205.0  Median : 9.700   Median :79.00  
##  Mean   :42.13   Mean   :185.9  Mean   : 9.958   Mean   :77.88  
##  3rd Qu.:63.25   3rd Qu.:258.8  3rd Qu.:11.500   3rd Qu.:85.00  
##  Max.   :168.00  Max.   :334.0  Max.   :20.700   Max.   :97.00  
##  NA's   :37      NA's   :7      NA's   :1        NA's   :1      
##      Month         Day        
##  Min.   :5.000   Min.   : 1.0  
##  1st Qu.:6.000   1st Qu.: 8.0  
##  Median :7.000   Median :16.0  
##  Mean   :6.993   Mean   :15.8  
##  3rd Qu.:8.000   3rd Qu.:23.0  
##  Max.   :9.000   Max.   :31.0  
## 

data %>%
  summarise(across(
    everything(),
    list(
```

```

    NA_count   = ~ sum(is.na(.)),
    NA_percent = ~ mean(is.na(.)) * 100
)
)) %>%
pivot_longer(
  everything(),
  names_to = c("Variable", "dummy", ".value"),
  names_sep = "_"
) %>%
select(-dummy)

```

```

## # A tibble: 6 x 3
##   Variable count percent
##   <chr>     <int>   <dbl>
## 1 Ozone      37    24.2
## 2 Solar.R     7     4.58
## 3 Wind        0     0
## 4 Temp        0     0
## 5 Month       0     0
## 6 Day         0     0

```

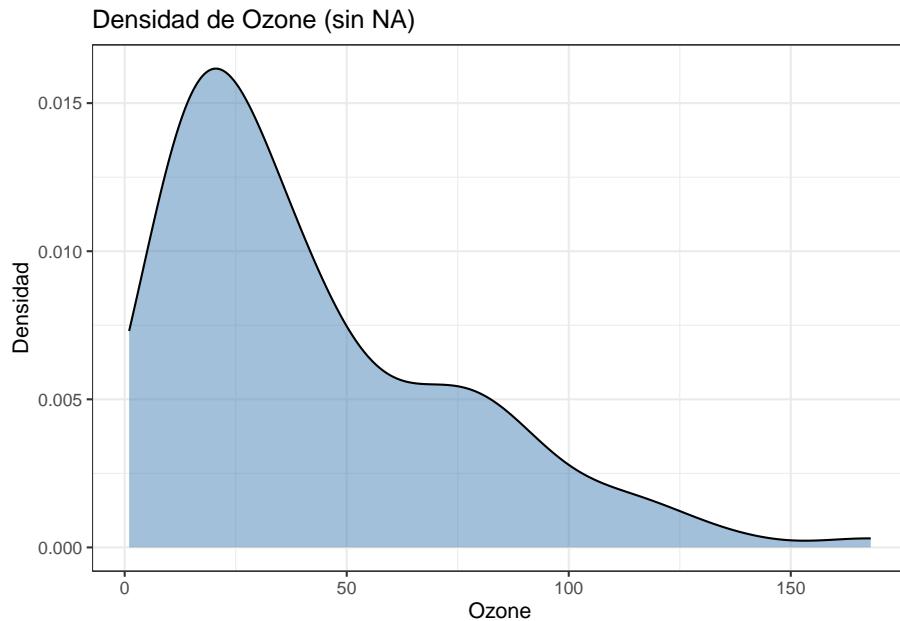
2. Haremos imputación de datos a la variable `Ozone`. Empecemos con un diagrama de densidad sin los datos faltantes

```

ozone_noNA <- na.omit(data$Ozone)

data.frame(Ozone = ozone_noNA) %>%
  ggplot(aes(x = Ozone)) +
  geom_density(fill = "steelblue", alpha = 0.5) +
  theme_bw() +
  labs(title = "Densidad de Ozone (sin NA)", x = "Ozone", y = "Densidad")

```



3. Hagamos una prueba de normalidad de los datos

```
ks.test(scale(ozone_noNA), 'pnorm')
```

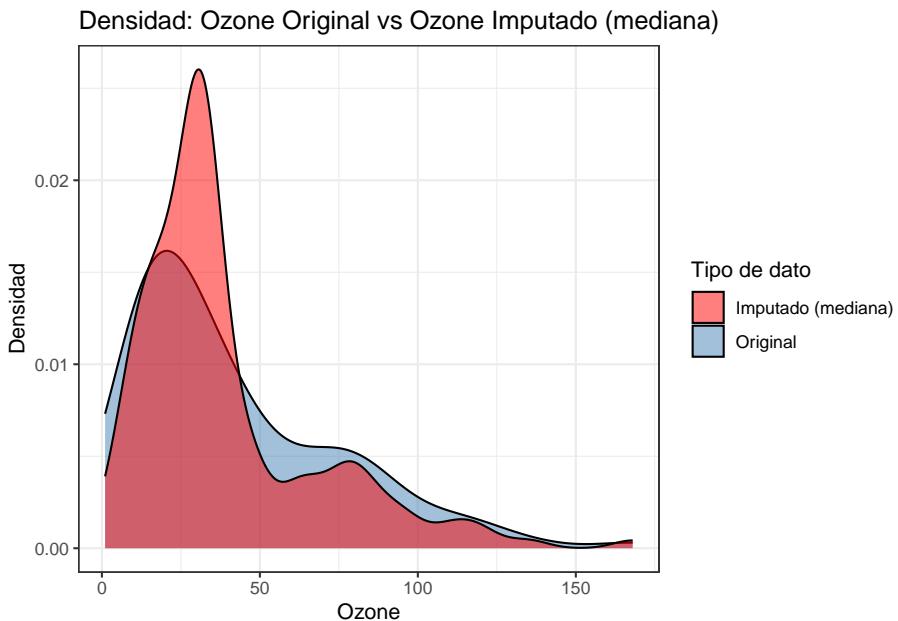
```
##  
##  Asymptotic one-sample Kolmogorov-Smirnov test  
##  
## data: scale(ozone_noNA)  
## D = 0.14799, p-value = 0.01243  
## alternative hypothesis: two-sided
```

- Como los datos de Ozone no tienen comportamiento normal, realizaremos la imputación de los datos con la mediana

```
new_data <- data %>%  
  mutate(Ozone_mediana = replace_na(as.numeric(Ozone), median(Ozone, na.rm = TRUE)))
```

4. Hagamos una comparación con la data original y los datos imputados usando gráficos de densidad

```
new_data %>%
  ggplot() +
  geom_density(aes(x = Ozone, fill = "Original"), alpha = 0.5) +
  geom_density(aes(x = Ozone_mediana, fill = "Imputado (mediana)"), alpha = 0.5) +
  theme_bw() +
  labs(title = "Densidad: Ozone Original vs Ozone Imputado (mediana)",
       x = "Ozone",
       y = "Densidad",
       fill = "Tipo de dato") +
  scale_fill_manual(values = c("Original" = "steelblue", "Imputado (mediana)" = "red"))
```



- La imputación con la **mediana** es adecuada en este caso porque la variable no sigue una distribución normal y está sesgada con outliers. Sin embargo, este método **reduce la variabilidad y distorsiona la forma real de la distribución** al concentrar valores en un solo punto.

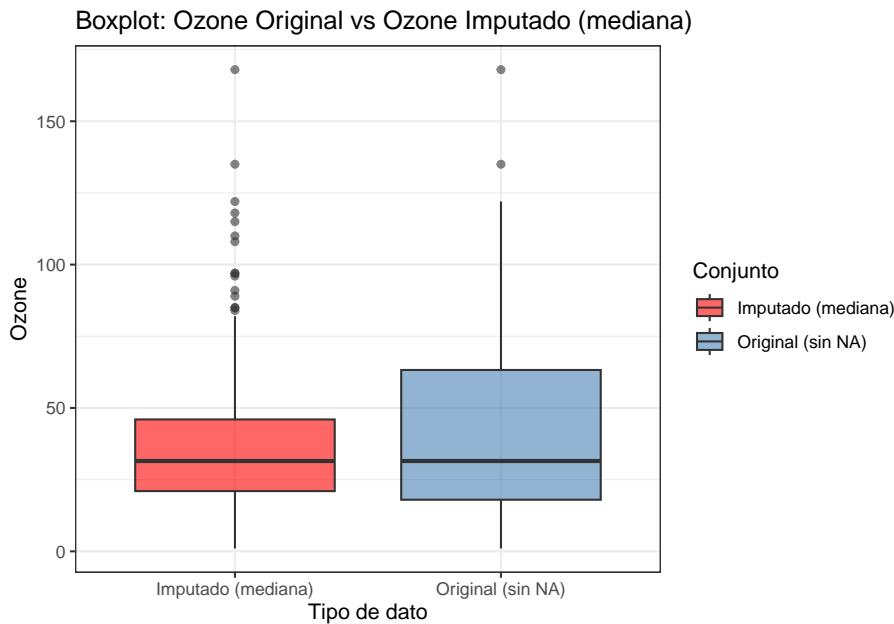
5. Ahora, hagamos la comparación con un diagrama de cajas y bigotes

```
new_data %>%
  select(Ozone, Ozone_mediana) %>%
  pivot_longer(cols = c(Ozone, Ozone_mediana),
               names_to = "Tipo",
               values_to = "Valor") %>%
```

```

mutate(Tipo = recode(Tipo,
                      Ozone = "Original (sin NA)",
                      Ozone_mediana = "Imputado (mediana")) %>%
  ggplot(aes(x = Tipo, y = Valor, fill = Tipo)) +
  geom_boxplot(alpha = 0.6) +
  scale_fill_manual(values = c("Original (sin NA)" = "steelblue",
                               "Imputado (mediana)" = "red")) +
  theme_bw() +
  labs(title = "Boxplot: Ozone Original vs Ozone Imputado (mediana)",
       x = "Tipo de dato",
       y = "Ozone",
       fill = "Conjunto")

```



- La imputación con la **mediana** es adecuada porque la variable no es normal y presenta outliers, pero sacrifica la variabilidad. El método es útil como aproximación robusta, aunque **no conserva del todo la distribución original**.

6. Hagamos una prueba para ver si las distribuciones difieren o no.

```

# Datos originales sin NA
ozone_noNA <- na.omit(data$Ozone)

```

```
# Prueba KS entre original e imputado
ks.test(ozone_noNA, new_data$Ozone_mediana)

##
##  Asymptotic two-sample Kolmogorov-Smirnov test
##
## data: ozone_noNA and new_data$Ozone_mediana
## D = 0.12092, p-value = 0.2897
## alternative hypothesis: two-sided
```

- La imputación de los valores faltantes de Ozone con la mediana no modificó significativamente la distribución de los datos en comparación con los valores originales.
7. Ahora, como en tu caso **imputaste con la mediana**, la comparación correcta es usar la **prueba de Wilcoxon de rangos** (también llamada *Mann-Whitney U test* cuando son muestras independientes)

```
# Prueba de Wilcoxon: compara las medianas de original vs imputado
wilcox.test(ozone_noNA, new_data$Ozone_mediana)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data: ozone_noNA and new_data$Ozone_mediana
## W = 8874, p-value = 1
## alternative hypothesis: true location shift is not equal to 0
```

- Esto confirma que la **imputación con la mediana preserva el centro de la distribución**. Aunque vimos que se reduce la **variabilidad** (la dispersión disminuye), el valor central de la distribución se mantiene intacto.

5.6 Imputación de datos con el paquete `mice`

5.6.1 Paquete MICE

El paquete **MICE** (*Multivariate Imputation by Chained Equations*) en **R** es una herramienta poderosa para la imputación de valores faltantes. A diferencia de métodos simples como la imputación por **media** o **mediana**, **mice** implementa la **imputación múltiple**, lo cual permite capturar la incertidumbre de los datos faltantes.

Cabe resaltar que

- Los datos faltantes representan incertidumbre.
- La imputación simple (media/mediana) reemplaza cada `NA` por un único valor, reduciendo la variabilidad.
- `mice` en cambio genera **m conjuntos imputados**, cada uno con valores distintos para los `NA`.
- Esto se hace modelando cada variable con faltantes a partir de las demás variables en el dataset (**cadena de ecuaciones**).
- Los resultados pueden combinarse para obtener inferencias robustas.

5.6.2 Parámetros principales de `mice`

- **method:** define el método de imputación.
 - "`mean`" → imputación con la media.
 - "`pmm`" (default) → *Predictive Mean Matching*.
 - "`cart`" → árboles de decisión.
 - "`rf`" → Random Forest.
- **m:** número de datasets imputados.
 - Default: `m = 5`.
 - Permite capturar la incertidumbre generando varias versiones del dataset.
- **maxit:** número de iteraciones.
 - Default: `maxit = 50`.
 - Controla cuántas veces se repite la imputación hasta converger.
- **seed:** semilla para garantizar reproducibilidad.
- **complete():** extrae un dataset completo imputado.

5.6.3 Métodos disponibles en `mice`

Algunos de los métodos más relevantes son:

- Numéricos:

- `mice.impute.mean` → imputación por la media.
- `mice.impute.norm` → regresión normal.
- `mice.impute.pmm` → *Predictive Mean Matching*.
- `mice.impute.cart` → árboles de decisión.
- `mice.impute.rf` → Random Forest.

- Categóricos:

- `mice.impute.logreg` → regresión logística.
- `mice.impute.polyreg` → regresión polinómica multinomial.
- `mice.impute.polr` → regresión logística ordinal.

- Otros:

- `mice.impute.2l.norm` → modelos jerárquicos.
- `mice.impute.passive` → imputación pasiva (variables derivadas).
- `mice.impute.jomoImpute` → modelos bayesianos mixtos.

5.6.4 Ventajas y limitaciones de MICE

Ventaja de `mice`

Captura la incertidumbre de los valores faltantes.

Compatible con múltiples tipos de variables (numéricas, categóricas, jerárquicas).

Permite métodos avanzados (PMM, RF, CART, regresión, etc.).

Limitaciones de `mice`

Más complejo que imputación simple.

Mayor costo computacional.

Requiere conocimiento estadístico para elegir métodos adecuados.

Comparación de métodos de imputación

Método	Ventajas	Limitaciones	Recomendación
Media	Simple, rápido	Sensible a outliers, reduce variabilidad	Distribuciones normales sin outliers
Mediana	Robusto, útil con sesgos	Reduce variabilidad, concentra valores	Distribuciones asimétricas o con outliers
PMM	Valores realistas, evita extremos	Más complejo, requiere iteraciones	Método recomendado en la mayoría de casos
RandomForest	Captura relaciones no lineales	Computacionalmente costoso	Datos con alta complejidad

5.6.5 Imputación de datos con emparejamiento predictivo medio

El método Predictive Mean Matching PMM es uno de los más usados en mice porque combina lo predictivo con lo realista.

- **¿Qué hace el método pmm?**

1. Para cada valor faltante, mice ajusta un **modelo de regresión** usando las demás variables como predictores.
2. Con ese modelo se **predice** el valor esperado del caso con el NA.
3. Se buscan en los datos observados los **k casos más cercanos** a esa predicción (vecinos).
4. El valor faltante se reemplaza con uno de esos valores **reales observados**, elegido al azar.

Las ventajas de este método PMM **utiliza aleatoriedad en la imputación para introducir variabilidad en las estimaciones de los valores faltantes**. Esto ayuda a evitar la **sobreestimación o subestimación sistemática de los valores imputados** y a capturar la incertidumbre asociada con la imputación de datos faltantes. La **introducción de aleatoriedad también puede ayudar a mejorar la robustez y generalización del modelo de imputación**, especialmente en situaciones donde los datos faltantes siguen patrones complejos o no aleatorios.

1. Realicemos la imputacion con los datos faltantes para Ozono y Solar.R pueden ser imputados por la función `mice()`.

```
library(mice)

# Imputación múltiple usando el paquete mice
imp <- mice(
  data,                      # datos con NA
  m = 5,                     # número de datasets imputados
  maxit = 50,                # iteraciones de imputación
  method = "pmm",            # método: Predictive Mean Matching
  seed = 500,                # semilla para reproducibilidad
  printFlag = FALSE # no mostrar mensajes en consola
)
```

2. Podemos recuperar el conjunto de datos completo utilizando la función `complete()`:

```
imp_df <- complete(imp)
head(imp_df)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5    1
## 2    36     118  8.0   72     5    2
## 3    12     149 12.6   74     5    3
## 4    18     313 11.5   62     5    4
## 5    14     237 14.3   56     5    5
## 6    28      82 14.9   66     5    6
```

3. Separemos los datos y realicemos un grafico de comparación para la variable `Ozone`:

```
# Separar originales e imputados (Ozone)
original_ozone <- data$Ozone[!is.na(data$Ozone)] # valores originales
imputed_only_ozone <- imp_df$Ozone[is.na(data$Ozone)] # solo imputados

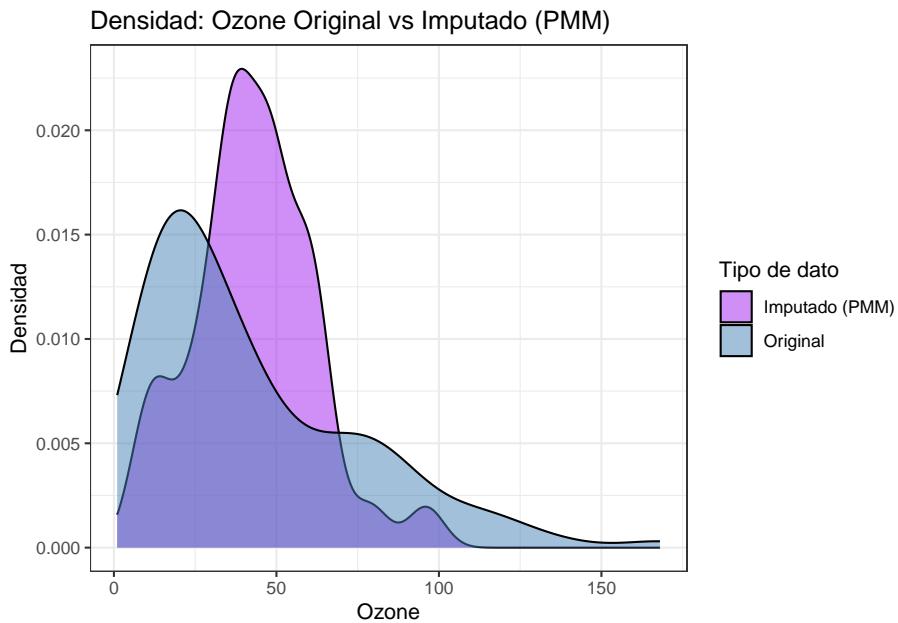
# Crear un data.frame combinado
ozone_df <- data.frame(
  Valor = c(original_ozone, imputed_only_ozone),
  Tipo  = c(rep("Original", length(original_ozone)),
            rep("Imputado (PMM)", length(imputed_only_ozone)))
)

# Gráfico de densidad
ggplot(ozone_df, aes(x = Valor, fill = Tipo)) +
  geom_density(alpha = 0.5) +
```

```

  labs(title = "Densidad: Ozone Original vs Imputado (PMM)",
       x = "Ozone",
       y = "Densidad",
       fill = "Tipo de dato") +
  theme_bw() +
  scale_fill_manual(values = c("Original" = "steelblue",
                               "Imputado (PMM)" = "purple"))

```



4. Realicemos las pruebas estadísticas:

```

# Prueba KS
ks_result <- ks.test(original_ozone, imputed_only_ozone)
print(ks_result)

##
## Results of Hypothesis Test
## -----
## 
## Alternative Hypothesis: two-sided
## 
## Test Name: Exact two-sample Kolmogorov-Smirnov test
## 
## Data: original_ozone and imputed_only_ozone

```

156 CHAPTER 5. ANÁLISIS CON DATOS FALTANTES Y DETECCIÓN DE ATÍPICOS

```
##  
## Test Statistic: D = 0.2912395  
##  
## P-value: 0.009919647  
  
# Prueba Wilcoxon  
wilcox_result <- wilcox.test(original_ozone, imputed_only_ozone)  
print(wilcox_result)  
  
##  
## Results of Hypothesis Test  
## -----  
##  
## Null Hypothesis: location shift = 0  
##  
## Alternative Hypothesis: True location shift is not equal to 0  
##  
## Test Name: Wilcoxon rank sum test with continuity correction  
##  
## Data: original_ozone and imputed_only_ozone  
##  
## Test Statistic: W = 1796.5  
##  
## P-value: 0.1369046
```

Como las pruebas realizadas, la imputación con la mediana es la mas optima, hasta el momento.

5.7 Imputación de datos con modelos de regresión

Para esta parte realizaremos un modelo de regresión usando `random forest`

```
# Imputación múltiple con Random Forest  
imput <- mice(  
  data, # datos con NA  
  m = 5, # número de datasets imputados  
  maxit = 50, # iteraciones de imputación  
  method = "rf", # método: Random Forest  
  seed = 500, # semilla para reproducibilidad  
  printFlag = FALSE # no mostrar mensajes en consola  
)
```

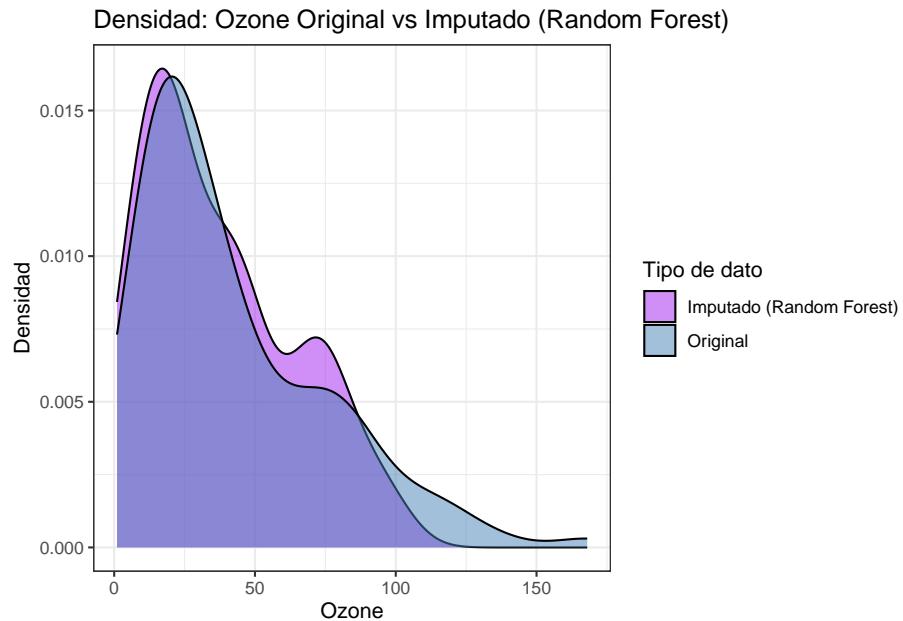
```
imp_df2 <- complete(imput)
head(imp_df2)
```

	Ozone	Solar.R	Wind	Temp	Month	Day
## 1	41	190	7.4	67	5	1
## 2	36	118	8.0	72	5	2
## 3	12	149	12.6	74	5	3
## 4	18	313	11.5	62	5	4
## 5	21	99	14.3	56	5	5
## 6	28	299	14.9	66	5	6

```
# Separar originales e imputados (Ozone)
original_ozone <- data$Ozone[!is.na(data$Ozone)]      # valores originales
imputed_only_ozone <- imp_df2$Ozone[is.na(data$Ozone)] # solo imputados

# Crear un data.frame combinado
ozone_df2 <- data.frame(
  Valor = c(original_ozone, imputed_only_ozone),
  Tipo  = c(rep("Original", length(original_ozone)),
            rep("Imputado (Random Forest)", length(imputed_only_ozone)))
)

# Gráfico de densidad
ggplot(ozone_df2, aes(x = Valor, fill = Tipo)) +
  geom_density(alpha = 0.5) +
  labs(title = "Densidad: Ozone Original vs Imputado (Random Forest)",
       x = "Ozone",
       y = "Densidad",
       fill = "Tipo de dato") +
  theme_bw() +
  scale_fill_manual(values = c("Original" = "steelblue",
                               "Imputado (Random Forest)" = "purple"))
```



```
# Prueba KS
ks_result <- ks.test(original_ozone, imputed_only_ozone)
print(ks_result)
```

```
##
## Results of Hypothesis Test
## -----
## 
## Alternative Hypothesis: two-sided
## 
## Test Name: Exact two-sample Kolmogorov-Smirnov test
## 
## Data: original_ozone and imputed_only_ozone
## 
## Test Statistic: D = 0.0999534
## 
## P-value: 0.8542215
```

```
# Prueba Wilcoxon
wilcox_result <- wilcox.test(original_ozone, imputed_only_ozone)
print(wilcox_result)
```

```
##
```

```

## Results of Hypothesis Test
## -----
## 
## Null Hypothesis:           location shift = 0
## 
## Alternative Hypothesis:   True location shift is not equal to 0
## 
## Test Name:                 Wilcoxon rank sum test with continuity correction
## 
## Data:                      original_ozone and imputed_only_ozone
## 
## Test Statistic:            W = 2287
## 
## P-value:                   0.5492801

```

Con todo lo realizado, podemos ver que el método de regresión `random forest`, nos da una mejor imputación con respecto a la mediana y el método `pmm`.

5.8 Detección de datos atípicos

5.8.1 Valores atípicos

Un **valor atípico** es una observación que se aleja considerablemente de las demás, es decir,
un punto de datos que difiere significativamente del resto.

Según *Enderlein (1987)*, los valores atípicos son aquellos que se desvían tanto de las demás observaciones que podría suponerse que provienen de un **mecanismo de muestreo diferente**.

5.8.1.1 Importancia del contexto

Una observación debe compararse siempre con otras obtenidas **bajo el mismo fenómeno** antes de calificarla como atípica.

Por ejemplo:

- Una persona con **200 cm de altura** podría considerarse un valor atípico si se compara con la población general.
- Sin embargo, **no sería atípica** si se compara con la población de **jugadores de baloncesto**.

Esto ilustra que la definición de atipicidad **depende del contexto**.

5.8.1.2 Métodos de detección en R

Existen diversos enfoques para detectar valores atípicos, que van desde técnicas descriptivas hasta pruebas **formales**:

- **Técnicas sencillas** (estadística descriptiva):
 - mínimo y máximo,
 - histograma,
 - boxplot,
 - percentiles.
- **Métodos más avanzados**:
 - Filtro de **Hampel**,
 - Test de **Grubbs**,
 - Test de **Dixon**,
 - Pruebas de **Rosner**.

5.8.1.3 Consideraciones prácticas

- Algunas pruebas estadísticas **requieren ausencia de valores atípicos** para obtener conclusiones válidas.
- La **eliminación de valores atípicos** no siempre es recomendable: debe hacerse con **precaución** y justificación, pues podría eliminar información valiosa del fenómeno estudiado.

5.8.2 Estadísticas descriptivas

El primer paso para detectar valores atípicos en **R** es comenzar con algunos **estadísticos descriptivos**, en particular con el **mínimo** y el **máximo**.

En **R**, esto puede hacerse fácilmente con la función:

```
summary
```

Para ilustrar el proceso, utilizaremos el dataset `mpg`, asociado al consumo de combustible de **1999 a 2008** para **38 modelos de coches populares** (ver `ggplot2::mpg`).

Este conjunto de datos es útil porque contiene variables numéricas de interés (como `hwy` y `cty`) en las que se pueden identificar posibles valores atípicos mediante estadísticos descriptivos.

1. Carguemos los datos

```
dat <- ggplot2::mpg
head(dat)

## # A tibble: 6 x 11
##   manufacturer model displ year cyl trans drv cty hwy fl
##   <chr>        <chr>  <dbl> <int> <int> <chr> <chr> <int> <int> <chr>
## 1 audi         a4      1.8  1999     4 auto(~ f      18    29 p
## 2 audi         a4      1.8  1999     4 manua~ f      21    29 p
## 3 audi         a4      2    2008     4 manua~ f      20    31 p
## 4 audi         a4      2    2008     4 auto(~ f      21    30 p
## 5 audi         a4      2.8  1999     6 auto(~ f      16    26 p
## 6 audi         a4      2.8  1999     6 manua~ f      18    26 p
## # i 1 more variable: class <chr>

summary(dat$hwy)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##      12.00 18.00 24.00 23.44 27.00 44.00
```

- El mínimo y el máximo son, respectivamente, el primer y el último valor de `hwy`. Como alternativa, también pueden calcularse con las funciones `min()` y `max()`

```
min(dat$hwy)
```

```
## [1] 12
```

```
max(dat$hwy)
```

```
## [1] 44
```

- Un claro *error de codificación, como un peso de 786 kg (1733 libras) para un humano*, ya se detectará fácilmente con esta técnica tan sencilla.

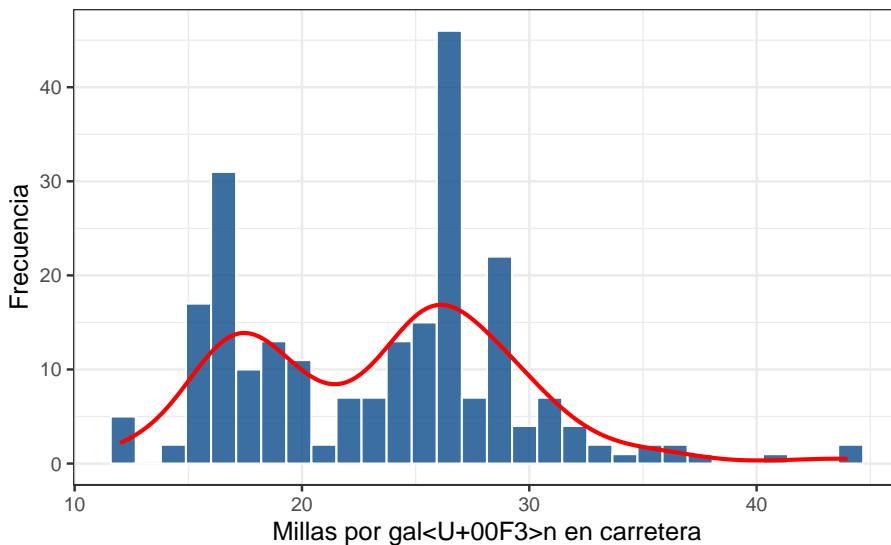
2. Un histograma es otra forma básica para detectar datos atípicos

```
library(ggplot2)

ggplot(dat, aes(x = hwy)) +
  geom_histogram(
    bins = 30,
    fill = "#0c4c8a",
    color = "white",      # bordes blancos para contraste
    alpha = 0.8
  ) +
  geom_density(
    aes(y = ..count..),
    color = "red",
    size = 1,
    alpha = 0.6
  ) +
  labs(
    title = "Distribución de consumo en carretera (hwy)",
    subtitle = "Datos del dataset mpg (ggplot2)",
    x = "Millas por galón en carretera",
    y = "Frecuencia"
  ) +
  theme_bw(base_size = 13) +
  theme(
    plot.title = element_text(face = "bold"),
    plot.subtitle = element_text(size = 10, color = "gray40")
  )
```

Distribución de consumo en carretera (hwy)

Datos del dataset mpg (ggplot2)



- Con base en este histograma, se evidencia que hay un par de observaciones más altas que todas las demás.
3. Además de los histogramas, los boxplots también son útiles para detectar posibles valores atípicos. Considere nuevamente a manera de ejemplo la columna hwy.

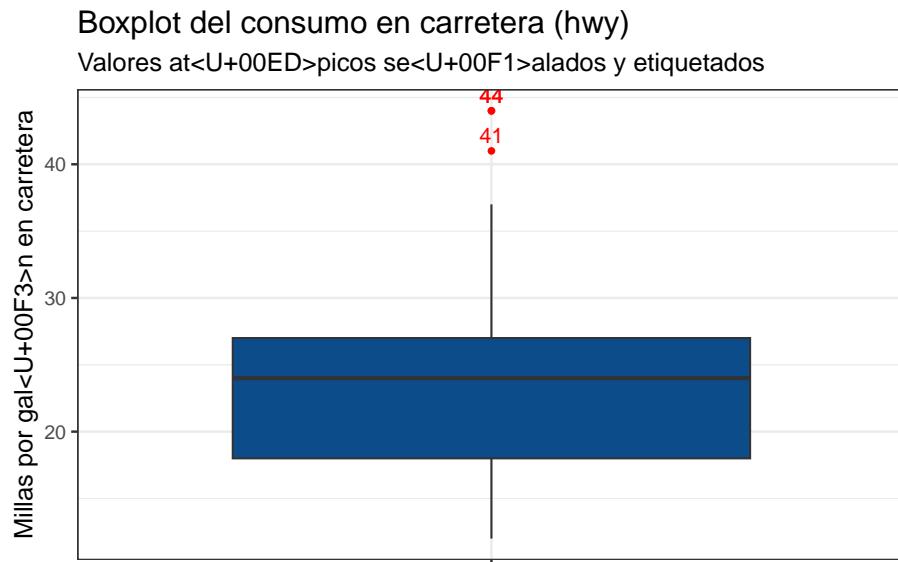
```
# Detectamos los outliers usando la regla del IQR
outliers <- dat %>%
  mutate(id = row_number()) %>% # identificador de fila
  group_by() %>%
  mutate(
    Q1 = quantile(hwy, 0.25),
    Q3 = quantile(hwy, 0.75),
    IQR = Q3 - Q1,
    lower = Q1 - 1.5 * IQR,
    upper = Q3 + 1.5 * IQR
  ) %>%
  filter(hwy < lower | hwy > upper)

# Boxplot con etiquetas de valores atípicos
ggplot(dat, aes(x = "", y = hwy)) +
  geom_boxplot(fill = "#0c4c8a", outlier.colour = "red", outlier.shape = 16) +
  geom_text(
```

```

data = outliers,
aes(label = hwy),      # mostramos el valor
color = "red",
vjust = -0.5
) +
labs(
  title = "Boxplot del consumo en carretera (hwy)",
  subtitle = "Valores atípicos señalados y etiquetados",
  x = "",
  y = "Millas por galón en carretera"
) +
theme_bw(base_size = 13)

```



4. También es posible extraer los valores de los posibles valores atípicos basándose en el criterio IQR gracias a la función `boxplot.stats()$out`:

```
boxplot.stats(dat$hwy)$out
```

```
## [1] 44 44 41
```

- Como puede ver, en realidad hay 3 puntos considerados como posibles valores atípicos; 2 observaciones con un valor de 44 y 1 observación con un valor de 41. Gracias a la función `which()` es posible extraer el número de fila correspondiente a estos valores atípicos:

```
out <- boxplot.stats(dat$hwy)$out
out_ind <- which(dat$hwy %in% c(out))
out_ind
```

```
## [1] 213 222 223
```

5. Con esta información, ahora puede *volver fácilmente a las filas específicas del conjunto de datos para verificarlas*, o imprimir todas las variables para estos valores atípicos:

```
dat[out_ind, ]
```

```
## # A tibble: 3 x 11
##   manufacturer model  displ  year   cyl trans drv   cty   hwy fl
##   <chr>        <chr>  <dbl> <int> <int> <chr> <chr> <int> <int> <chr>
## 1 volkswagen   jetta    1.9  1999     4 manu~ f      33    44 d
## 2 volkswagen   new b~   1.9  1999     4 manu~ f      35    44 d
## 3 volkswagen   new b~   1.9  1999     4 auto~ f      29    41 d
## # i 1 more variable: class <chr>
```

5.8.3 Percentiles

Este método de **detección de valores atípicos** se basa en los **percentiles**. Con este enfoque, todas las **observaciones que se encuentren fuera del intervalo formado por los percentiles 2.5 y 97.5** se considerarán como **posibles valores atípicos**:

$$I_{out} = [q_{0.025}, q_{0.975}]^C$$

- También pueden considerarse **otros percentiles** (por ejemplo, el 1 y el 99, o el 5 y el 95) para construir el intervalo.
- Los valores de los percentiles inferior y superior (y, por lo tanto, los **límites inferior y superior del intervalo**) pueden calcularse con la función `quantile()` en **R**.

```
bounds <- dat %>%
  summarise(
    lower_bound = quantile(hwy, 0.025, na.rm = TRUE),
    upper_bound = quantile(hwy, 0.975, na.rm = TRUE)
  )
bounds
```

```
## # A tibble: 1 x 2
##   lower_bound upper_bound
##       <dbl>      <dbl>
## 1         14     35.2
```

- Según este método, todas las observaciones por debajo de 14 y por encima de 35.175 se considerarán posibles valores atípicos. Los números de fila de las observaciones fuera del intervalo pueden extraerse de la siguiente manera

```
outlier_ind <- dat %>%
  mutate(row_id = row_number()) %>%
  filter(hwy < bounds$lower_bound | hwy > bounds$upper_bound) %>%
  pull(row_id)
```

```
outlier_ind
```

```
## [1] 55 60 66 70 106 107 127 197 213 222 223
```

```
dat[outlier_ind, "hwy"]
```

```
## # A tibble: 11 x 1
##       hwy
##   <int>
## 1    12
## 2    12
## 3    12
## 4    12
## 5    36
## 6    36
## 7    12
## 8    37
## 9    44
## 10   44
## 11   41
```

```
dat[outlier_ind, ]
```

```
## # A tibble: 11 x 11
##   manufacturer model displ year cyl trans drv cty hwy fl
##   <chr>        <chr> <dbl> <int> <int> <chr> <int> <int> <chr>
## 1 dodge        dakota~  4.7  2008     8 auto~ 4          9    12 e
## 2 dodge        durango~ 4.7  2008     8 auto~ 4          9    12 e
```

```

## 3 dodge      ram ~ 4.7 2008     8 auto~ 4      9    12 e
## 4 dodge      ram ~ 4.7 2008     8 manu~ 4      9    12 e
## 5 honda      civic 1.8 2008     4 auto~ f      25   36 r
## 6 honda      civic 1.8 2008     4 auto~ f      24   36 c
## 7 jeep       gran~ 4.7 2008     8 auto~ 4      9    12 e
## 8 toyota     coro~ 1.8 2008     4 manu~ f      28   37 r
## 9 volkswagen jetta 1.9 1999     4 manu~ f      33   44 d
## 10 volkswagen new ~ 1.9 1999     4 manu~ f      35   44 d
## 11 volkswagen new ~ 1.9 1999     4 auto~ f      29   41 d
## # i 1 more variable: class <chr>

```

- Hay 11 valores atípicos potenciales según el método de los percentiles. Para reducir este número, puede establecer los percentiles en 1 y 99:

```

bounds <- dat %>%
  summarise(
    lower_bound = quantile(hwy, 0.01, na.rm = TRUE),
    upper_bound = quantile(hwy, 0.99, na.rm = TRUE)
  )

outlier_ind <- dat %>%
  mutate(row_id = row_number()) %>%
  filter(hwy < bounds$lower_bound | hwy > bounds$upper_bound) %>%
  pull(row_id)

dat[outlier_ind,]

## # A tibble: 3 x 11
##   manufacturer model  displ year   cyl trans drv   cty   hwy fl
##   <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr>
## 1 volkswagen   jetta   1.9  1999     4 manu~ f      33   44 d
## 2 volkswagen   new b~  1.9  1999     4 manu~ f      35   44 d
## 3 volkswagen   new b~  1.9  1999     4 auto~ f      29   41 d
## # i 1 more variable: class <chr>

```

5.8.4 Filtro de Hampel

Otro método, conocido como **filtro de Hampel**, consiste en considerar como valores atípicos aquellos **valores que se encuentran fuera del intervalo formado por la mediana, más o menos 3 desviaciones absolutas de la mediana (MAD)**:

$$I = [\tilde{X} - 3 \cdot MAD, \tilde{X} + 3 \cdot MAD]$$

donde

- **MAD** es la *desviación absoluta de la mediana*.
- Se define como la mediana de las desviaciones absolutas respecto a la mediana:

$$MAD = k \cdot \text{median}(|X_i - \tilde{X}|)$$

Aquí, $\tilde{X} = \text{median}(X)$ es la mediana de los datos. - El parámetro k corresponde al **factor de escala**, que por defecto en R se toma como $k = 1$.

- Para aplicar este método en **R**, primero se establecen los **límites del intervalo** con las funciones `median()` y `mad()`. Los valores que se encuentren fuera de este rango se consideran **atípicos**.

```
# Calcular límites inferior y superior con tidyverse
hampel_bounds <- dat %>%
  summarise(lower_bound = median(hwy, na.rm = TRUE) - 3 * mad(hwy, constant = 1, na.rm = TRUE),
            upper_bound = median(hwy, na.rm = TRUE) + 3 * mad(hwy, constant = 1, na.rm = TRUE))

hampel_bounds

## # A tibble: 1 x 2
##   lower_bound upper_bound
##       <dbl>      <dbl>
## 1         9        39
```

- Según este método, *todas las observaciones por debajo de 9 y por encima de 39 se considerarán como posibles valores atípicos*. Los números de fila de las observaciones que están fuera del intervalo pueden entonces extraerse de la siguiente manera

```
# Extraer los índices y registros atípicos
outliers <- dat %>%
  mutate(row_id = row_number()) %>%
  filter(hwy < hampel_bounds$lower_bound | hwy > hampel_bounds$upper_bound) %>%
  pull(row_id)

outliers

## [1] 213 222 223
```

```
dat[outliers,]

## # A tibble: 3 x 11
##   manufacturer model  displ  year cyl trans drv     cty     hwy fl
##   <chr>        <chr>  <dbl> <int> <int> <chr> <chr> <int> <int> <chr>
## 1 volkswagen   jetta    1.9  1999     4 manu~ f       33     44 d 
## 2 volkswagen   new b~   1.9  1999     4 manu~ f       35     44 d 
## 3 volkswagen   new b~   1.9  1999     4 auto~ f       29     41 d 
## # i 1 more variable: class <chr>
```

5.8.5 Prueba de Grubbs

La prueba de Grubbs permite **detectar si el valor más alto o más bajo de un conjunto de datos es un valor atípico**. Este test analiza un valor atípico a la vez (ya sea el valor máximo o el mínimo), y por ello las hipótesis nula y alternativa se formulan así:

- H_0 : El valor más alto/bajo **no es un valor atípico**.
- H_1 : El valor más alto/bajo **es un valor atípico**.

La interpretación sería

- Si el valor p es inferior al nivel de significancia elegido (generalmente $\alpha = 0.05$), se **rechaza H_0** y se concluye que el valor más bajo o más alto es un valor atípico.
- Si el valor p es mayor o igual a α , **no se rechaza H_0** , lo que significa que el valor extremo no puede considerarse un atípico.

Importante

la prueba de Grubbs **no es apropiada para tamaños de muestra $n \leq 6$**

1. Para realizar la prueba de Grubbs en R, utilizamos la función **grubbs.test()** del paquete **outliers**. La función recibe distintos argumentos, entre ellos el parámetro **type**:

- **type = 10**: prueba si el valor máximo es un valor atípico.
- **type = 11**: prueba si tanto el mínimo como el máximo son valores atípicos.
- **type = 20**: prueba si hay dos valores atípicos en una cola.

```
library(outliers)
test <- grubbs.test(dat$hwy)
test

##
##  Grubbs test for one outlier
##
## data:  dat$hwy
## G = 3.45274, U = 0.94862, p-value = 0.05555
## alternative hypothesis: highest value 44 is an outlier
```

- El valor p obtenido es aproximadamente **0.056**. Al nivel de significación del 5% ($\alpha = 0.05$), **no rechazamos la hipótesis nula** de que el valor más alto (44) **no es un valor atípico**.

En otras palabras, **no tenemos evidencia suficiente para afirmar que 44 es un valor atípico**.

- Por defecto, la prueba se realiza sobre el **valor más alto** (como se muestra en la salida de R). La hipótesis alternativa sería: *el valor más alto (44) es un valor atípico*.
2. Si se desea realizar la prueba para el **valor más bajo**, basta con añadir el argumento:

```
test <- grubbs.test(dat$hwy, opposite = TRUE)
test
```

```
##
##  Grubbs test for one outlier
##
## data:  dat$hwy
## G = 1.92122, U = 0.98409, p-value = 1
## alternative hypothesis: lowest value 12 is an outlier
```

- El valor p es 1. Al nivel de significación del 5%, **no rechazamos la hipótesis de que el valor más bajo 12 no es un valor atípico**.
3. A modo de ilustración, **sustituiremos ahora una observación por un valor más extremo y realizaremos la prueba** de Grubbs en este nuevo conjunto de datos. Reemplazaremos el 34th por un valor de 212

```

dat[34, "hwy"] <- 212

test <- grubbs.test(dat$hwy)
test

##
## Grubbs test for one outlier
##
## data: dat$hwy
## G = 13.72240, U = 0.18836, p-value < 2.2e-16
## alternative hypothesis: highest value 212 is an outlier

```

El valor p es menor que la significancia. Al nivel de significancia del 5%, concluimos que el valor más alto 212 es un valor atípico.

5.8.6 Prueba de Dixon

Al igual que la prueba de **Grubbs**, la prueba de **Dixon** se utiliza para **comprobar si un único valor bajo o alto es un valor atípico**. Por lo tanto, si se sospecha que hay más de un valor atípico, la prueba debe realizarse en cada valor atípico sospechoso de forma individual.

La prueba de **Dixon** es especialmente útil para muestras de pequeño tamaño (generalmente $n \leq 25$).

Para realizar la prueba de Dixon en R, se utiliza la función `dixon.test()` del paquete **outliers**. Sin embargo, es importante destacar que:

- La prueba sólo puede aplicarse a conjuntos de datos pequeños.
 - R restringe su aplicación a conjuntos de entre 3 y 30 observaciones.
1. En este ejemplo, limitamos el análisis a las **20 primeras observaciones** del conjunto de datos.

```

subdat <- dat[1:20, ]
test <- dixon.test(subdat$hwy)
test

##
## Dixon test for outliers
##
## data: subdat$hwy
## Q = 0.57143, p-value = 0.006508
## alternative hypothesis: lowest value 15 is an outlier

```

- Los resultados muestran que el **valor más bajo, 15, es un valor atípico**, con valor p menor que la significancia del 5%.

2. Para comprobar el valor más alto, basta con añadir el argumento `opuesto = TRUE` a la función `dixon.test()`.

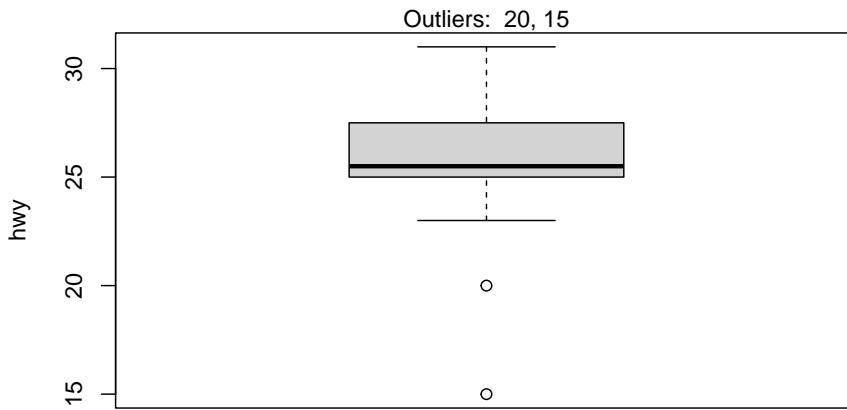
```
test <- dixon.test(subdat$hwy, opposite = TRUE)
test
```

```
##
## Dixon test for outliers
##
## data: subdat$hwy
## Q = 0.25, p-value = 0.8582
## alternative hypothesis: highest value 31 is an outlier
```

- Los resultados muestran que el valor más alto 31 no es un valor atípico ($p - valor = 0.8582$).

3. Es una buena práctica comprobar siempre los resultados de la prueba estadística de valores atípicos con el diagrama de caja para asegurarse de que hemos comprobado todos los valores atípicos potenciales

```
out <- boxplot.stats(subdat$hwy)$out
boxplot(subdat$hwy, ylab = "hwy")
mtext(paste("Outliers: ", paste(out, collapse = ", ")))
```



- A partir del boxplot, vemos que también podríamos aplicar la prueba de Dixon sobre el valor 20 además del valor 15 realizado anteriormente.
4. Esto puede hacerse encontrando el número de fila del valor mínimo, excluyendo este número de fila del conjunto de datos y aplicando finalmente la prueba de Dixon a este nuevo conjunto de datos

```

remove_ind <- which.min(subdat$hwy)
subsubdat <- subdat[-remove_ind, ]
tail(subsubdat)

## # A tibble: 6 x 11
##   manufacturer model  displ  year   cyl trans drv     cty     hwy fl
##   <chr>        <chr>  <dbl> <int> <int> <chr> <chr> <int> <int> <chr>
## 1 audi         a4    3.1  2008     6 auto~ 4      17     25 p 
## 2 audi         a4    3.1  2008     6 manu~ 4      15     25 p 
## 3 audi         a6    2.8  1999     6 auto~ 4      15     24 p 
## 4 audi         a6    3.1  2008     6 auto~ 4      17     25 p 
## 5 audi         a6    4.2  2008     8 auto~ 4      16     23 p 
## 6 chevrolet    c1500~  5.3  2008     8 auto~ r      14     20 r 
## # i 1 more variable: class <chr>

```

```
test <- dixon.test(subsubdat$hwy)
test

##
## Dixon test for outliers
##
## data: subsubdat$hwy
## Q = 0.44444, p-value = 0.1297
## alternative hypothesis: lowest value 20 is an outlier
```

- Los resultados muestran que el segundo valor más bajo, 20, no es un valor atípico ($p - valor = 0.13$)

5.8.7 Prueba de Rosner

La prueba de Rosner para detectar valores atípicos tiene las siguientes ventajas. **Se utiliza para detectar varios valores atípicos a la vez** (a diferencia de la prueba de Grubbs y Dixon, que debe realizarse de forma iterativa para detectar múltiples valores atípicos), y está diseñado para evitar el problema del enmascaramiento, en el que un valor atípico cercano a otro atípico puede pasar desapercibido. A diferencia de la prueba de Dixon, hay que tener en cuenta que la prueba de Rosner es **más apropiada cuando el tamaño de la muestra es grande** ($n \geq 20$). Por tanto, volvemos a utilizar el conjunto de datos inicial dat, que incluye 234 observaciones.

1. Para realizar la prueba de Rosner utilizamos la función `rosnerTest()` del paquete EnvStats. Esta función requiere al menos 2 argumentos: **los datos y el número de presuntos valores atípicos k** (con $k = 3$ como **número de presuntos valores atípicos por defecto**). Para este ejemplo, establecemos que el número de presuntos valores atípicos sea igual a 3, tal y como sugiere el número de posibles valores atípicos esbozado en el `boxplot` al principio del artículo.

```
library(EnvStats)
test <- rosnerTest(dat$hwy, k = 3)
```

2. Los resultados interesantes se ofrecen en la tabla `all.stats`

```
test$all.stats

##   i   Mean.i      SD.i Value Obs.Num      R.i+1 lambda.i+1 Outlier
## 1 0 24.21795 13.684345    212       34 13.722399   3.652091    TRUE
## 2 1 23.41202  5.951835     44       213  3.459098   3.650836   FALSE
## 3 2 23.32328  5.808172     44       222  3.559936   3.649575   FALSE
```

- Basándonos en la prueba de Rosner, vemos que sólo hay un valor atípico (véase la columna de valores atípicos), y que es la observación 34 (véase `Obs.Num`) con un valor de 212 (véase `Value`).

5.9 Tratamiento de datos atípicos

Una vez identificados los valores atípicos y habiendo decidido enmendar la situación según la naturaleza del problema, puede considerar uno de los siguientes enfoques.

- **Imputación:** Este método se ha tratado en detalle en la discusión sobre el tratamiento de los valores faltantes.
- **Capping:** Para los valores que se encuentran fuera de los límites de $1.5 \cdot IQR$ podríamos poner un tope sustituyendo las observaciones que se encuentran fuera del límite inferior por el valor del 5^{th} percentil y las que se encuentran por encima del límite superior, por el valor del 95^{th} percentil. A continuación se muestra un código de ejemplo que logra esto

```
x <- dat$hwy
qnt <- quantile(x, probs=c(.25, .75), na.rm = T) #Quartiles
caps <- quantile(x, probs=c(.05, .95), na.rm = T) #5th,95th Percentiles

H <- 1.5 * IQR(x, na.rm = T)
x[x < (qnt[1] - H)] <- caps[1]
x[x > (qnt[2] + H)] <- caps[2]
head(x)

## [1] 29 29 31 30 26 26
```

Predicción

En otro enfoque, los **valores atípicos pueden sustituirse por valores faltantes (NA)** y luego pueden predecirse considerándolos como una variable de respuesta. Ya hemos hablado de cómo predecir los valores faltantes en la sección anterior.

Ejercicio

Realice un EDA para el conjunto de datos (Diabetes Dataset). Este dataset consta de variables médicas predictoras (independientes) y una variable objetivo (dependiente: resultado).

Sustituya todos los valores nulos “ausentes” por NA y realice el respectivo análisis de datos faltantes.

176 CHAPTER 5. ANÁLISIS CON DATOS FALTANTES Y DETECCIÓN DE ATÍPICOS

Exprese en R el siguiente código escrito en Python que sustituye los ceros por NA:

Aplique imputación de datos usando el paquete mice en R con los siguientes métodos:

```
method = "pmm"  
method = "norm.predict"  
method = "norm.nob"  
method = "norm"
```

Identifique valores atípicos en cada variable del dataset utilizando técnicas estadísticas vistas en clase (Imputación/Capping/Predicción, test de Rosner, Dixon, Grubbs, Hampel, Percentiles, Boxplots, Histogramas y Descriptivos). Finalmente, realice imputación de los datos atípicos con base en lo desarrollado en el ítem anterior.

Bibliography

Moraga, P. (2019). *Geospatial Health Data: Modeling and Visualization with R-INLA and Shiny*. CRC Press.

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

Xie, Y. (2025). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.43.